

Дон Гриффитс

Дэвид Гриффитс

Head First

Программирование

для Android



Создавай
прибыльные
приложения



Избегай
недопустимых
активностей



Узнай, как
материальный
дизайн может
изменить
ТВОЮ ЖИЗНЬ



Стань
настоящим
мастером

Познакомься
со службой
позиционирования



Развлекайся
с Android
Support Libraries

O'REILLY®

ПИТЕР®

Head First Android Development

Wouldn't it be dreamy if there were a book on developing Android apps that was easier to understand than the space shuttle flight manual? I guess it's just a fantasy...



Dawn Griffiths
David Griffiths

O'REILLY®

Beijing ■ Cambridge ■ Köln ■ Sebastopol ■ Tokyo

Head First

Программирование для Android

Хорошо бы найти книгу,
которая научила бы меня
разрабатывать приложения
для Android и при этом была
понятнее руководства для пилотов
космического корабля.
Как жаль, что это только мечты...

Дон Гриффитс
Дэвид Гриффитс



 **ПИТЕР®**

Москва · Санкт-Петербург · Нижний Новгород · Воронеж
Киев · Екатеринбург · Самара · Минск

2016

Дон Гриффитс, Дэвид Гриффитс
Head First. Программирование для Android

Серия «*Head First O'Reilly*»

Перевел с английского *Е. Матвеев*

Заведующая редакцией	<i>Ю. Сергиенко</i>
Ведущий редактор	<i>Н. Римицан</i>
Художник	<i>С. Заматевская</i>
Корректор	<i>С. Беляева</i>
Верстка	<i>Н. Лукьянова</i>

ББК 32.973.2-018.2

УДК 004.451

Гриффитс Дон, Гриффитс Дэвид

Г85 Head First. Программирование для Android. — СПб.: Питер, 2016. — 704 с.: ил. — (Серия «Head First O'Reilly»).

ISBN 978-5-496-02171-5

Система Android покорила мир. Все хотят иметь планшет или смартфон, а устройства на базе Android пользуются невероятной популярностью. В этой книге мы научим вас разрабатывать собственные приложения, а также покажем, как построить простое приложение и запустить его на виртуальном устройстве Android. Вы узнаете, как структурировать приложения, познакомитесь с дизайном интерфейсов, научитесь создавать базы данных, заставите работать свои приложения на любых смартфонах и планшетах. Попутно будут рассмотрены основные компоненты приложений Android — такие, как активности и макеты. Все, что от вас потребуется, — некоторые базовые знания Java.

12+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ISBN 978-1449362188 англ.

© 2016 Piter Press, Ltd.

Authorized Russian translation of the English edition of Head First Android Development, ISBN 9781449362188 © 2015 Dawn Griffiths and David Griffiths.

This translation is published and sold by permission of O'Reilly Media, Inc., the owner of all rights to publish and sell the same.

ISBN 978-5-496-02171-5

© Перевод на русский язык ООО Издательство «Питер», 2016

© Издание на русском языке, оформление ООО Издательство «Питер», 2016

© Серия «Head First O'Reilly», 2016

Права на издание получены по соглашению с O'Reilly. Все права защищены. Никакая часть данной книги не может быть воспроизведена в какой бы то ни было форме без письменного разрешения владельцев авторских прав.

ООО «Питер Пресс», 192102, Санкт-Петербург, ул. Андреевская (д. Волкова), 3, литер А, пом. 7Н.

Налоговая льгота — общероссийский классификатор продукции ОК 034-2014, 58.11.12.000 —

Книги печатные профессиональные, технические и научные.

Подписано в печать 24.12.15. Формат 84x108/16. Бумага писчая. Усл. п. л. 73,920. Тираж 1200. Заказ 0000.

Отпечатано в соответствии с предоставленными материалами в ООО «ИПК Парето-Принт».

170546, Тверская область, Промышленная зона Боровлево-1, комплекс № 3А, www.pareto-print.

Посвящается нашим друзьям и семье.
Спасибо вам за любовь и поддержку.

Об авторах

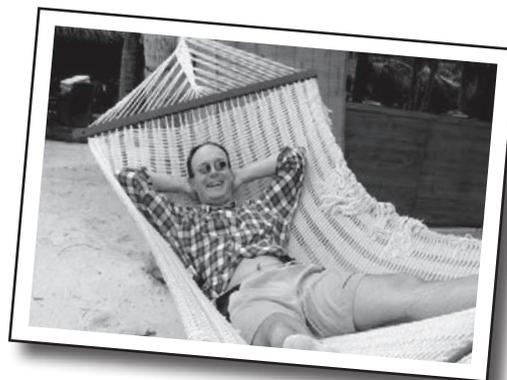


Дон
Гриффитс

Дон Гриффитс начала с изучения математики в одном из ведущих университетов Великобритании, где получила диплом с отличием. Затем она продолжила карьеру в области разработки программного обеспечения; ее опыт работы в IT-отрасли составляет 20 лет.

Прежде чем браться за книгу *Head First. Программирование для Android*, Дон написала уже три книги из серии Head First (*Head First Statistics*, *Head First 2D Geometry* и *Head First C*), а также участвовала в работе над другими книгами этой серии.

Когда Дон не работает над книгами серии Head First, она обычно совершенствует свое мастерство тай цзи, увлекается чтением, бегом, плетением кружев и кулинарией. Но больше всего ей нравится проводить время с ее замечательным мужем Дэвидом.



Дэвид
Гриффитс

Дэвид Гриффитс увлекся программированием в 12 лет, после документального фильма о работе Сеймура Пейперта. В 15 лет он написал реализацию языка программирования LOGO, созданного Пейпертом. После изучения теоретической математики в университете он начал писать программы для компьютеров и статьи в журналах. Он работал преподавателем гибких методологий разработки, разработчиком и дежурным по гаражу (хотя и в другом порядке). Дэвид пишет программы на 10 языках и прозу на одном. А когда он не занят программированием, литературной работой или преподаванием, он проводит свободное время в путешествиях со своей очаровательной женой — и соавтором — Дон.

До *Head First. Программирование для Android* Дэвид написал еще три книги из серии Head First: *Head First Rails*, *Head First Programming* и *Head First C*.

Наши микроблоги Twitter доступны по адресу <https://twitter.com/HeadFirstDroid>.

Содержание (сводка)

	Введение	23
1	Первые шаги. <i>С головой в пучину</i>	33
2	Построение интерактивных приложений. <i>Приложения, которые что-то делают</i>	71
3	Множественные активности и интен­ты. <i>Предъявите свой интен­т</i>	105
4	Жизненный цикл активности. <i>Из жизни активностей</i>	147
5	Пользовательский интерфейс. <i>Представление начинается</i>	195
6	Списковые представления и адаптеры. <i>Обо всем по порядку</i>	259
7	Фрагменты. <i>Модульная структура</i>	301
8	Вложенные фрагменты. <i>Укрощение фрагментов</i>	357
9	Панели действий. <i>В поисках короткого пути</i>	397
10	Выдвижные панели. <i>Подальше положишь...</i>	429
11	Базы данных SQLite. <i>Работа с базами данных</i>	469
12	Курсоры и асинхронные задачи. <i>Подключение к базам данных</i>	503
13	Службы. <i>К вашим услугам</i>	573
14	Материальное оформление. <i>Жизнь в материальном мире</i>	629
	Приложение I. Исполнительная среда. <i>Исполнительная среда Android</i>	681
	Приложение II. ADB. <i>Android Debug Bridge</i>	685
	Приложение III. Эмулятор. <i>Эмулятор Android</i>	691
	Приложение IV. Остатки. <i>Десять важнейших тем (которые мы не рассмотрели)</i>	694

Содержание (настоящее)

Введение

Ваш мозг и Android. Вы учитесь — готовитесь к экзамену. Или пытаетесь освоить сложную техническую тему. Ваш мозг пытается оказать вам услугу. Он старается сделать так, чтобы на эту очевидно несущественную информацию не тратились драгоценные ресурсы. Их лучше потратить на что-нибудь важное. Так как же заставить его изучить программирование для Android?

Для кого написана эта книга?	24
Мы знаем, о чем вы думаете	25
И мы знаем, о чем думает ваш мозг	25
Метапознание: наука о мышлении	27
Вот что сделали МЫ	28
Примите к сведению	30
Научные редакторы	31
Благодарности	32

1 Первые Шаги

С головой в пучину

Система Android покорила мир. Все хотят иметь планшет или смартфон, а устройства на базе Android пользуются невероятной популярностью. В этой книге мы научим вас разрабатывать собственные приложения, а также покажем, как построить простое приложение и запустить его на виртуальном устройстве Android. Попутно будут рассмотрены основные компоненты приложений Android — такие как активности и макеты. Все, что от вас потребуется — некоторые базовые знания Java...



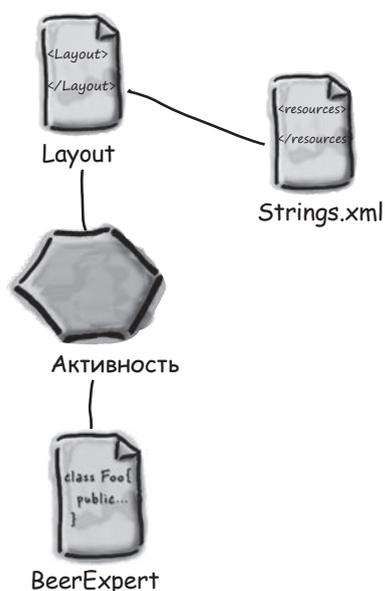
Добро пожаловать в мир Android	34
Платформа Android в разрезе	35
Вот что мы сейчас сделаем	36
Среда разработки	37
Установите Java	38
Построение простого приложения	39
Давайте построим простое приложение	40
Построение простого приложения (продолжение)	41
Активности и макеты: с высоты птичьего полета	44
Построение простого приложения (продолжение)	45
Только что вы создали свое первое Android-приложение	47
Android Studio создает всю структуру папок за вас	48
Полезные файлы в проекте	49
Редактирование кода в Android Studio	50
Запуск приложения в эмуляторе Android	55
Создание виртуального устройства Android	56
Запуск приложения в эмуляторе	59
Информация о ходе запуска отображается на консоли	60
Тест-драйв	61
Что же только что произошло?	62
Модификация приложения	63
Что содержит макет?	64
activity_main.xml состоит из двух элементов	65
Файл макета содержит ссылку на строку, а не саму строку	66
Заглянем в файл strings.xml	67
Ваш инструментарий Android	70



2 Построение интерактивных приложений

Приложения, которые что-то делают

Обычно приложение должно реагировать на действия пользователя. Из этой главы вы узнаете, как существенно повысить интерактивность ваших приложений. Мы покажем, как заставить приложение делать что-то в ответ на действия пользователя и как заставить активность и макет общаться друг с другом, как старые знакомые. Попутно вы больше узнаете о том, как на самом деле работает Android; мы расскажем о R — неприметном сокровище, которое связывает все воедино.



В этой главе мы построим приложение для выбора пива	72
Что нужно сделать	73
Создание проекта	74
Мы создали активность и макет по умолчанию	75
Добавление компонентов в визуальном редакторе	76
В <code>activity_find_beer.xml</code> появилась новая кнопка	77
Подробнее о коде макета	78
Изменения в XML...	80
...отражаются в визуальном редакторе	81
Использование строковых ресурсов вместо жестко запрограммированного текста	82
Внесение изменений в макет для использования строковых ресурсов	83
Добавление значений в список	85
Передача массива строк раскрывающемуся списку	86
Тест-драйв раскрывающегося списка	86
Кнопка должна что-то делать	87
Как заставить кнопку вызвать метод	88
Как выглядит код активности	89
Добавление в активность метода <code>onClickFindBeer()</code>	90
Метод <code>onClickFindBeer()</code> должен что-то делать	91
Обновление кода активности	93
Первая версия активности	95
Что делает этот код	96
Построение вспомогательного класса Java	98
Код активности, версия 2	101
Что происходит при выполнении кода	102
Тест-драйв	103
Ваш инструментарий Android	104

Множественные активности и интенты

Предъявите свой интент

Для большинства приложений одной активности недостаточно. До настоящего момента мы рассматривали приложения с одной активностью; для простых приложений это нормально. Однако в более сложной ситуации одна активность попросту не справляется со всеми делами. Мы покажем вам, как строить приложения с несколькими активностями и как организовать взаимодействие между активностями с использованием интентов. Также вы узнаете, как использовать интенты за пределами приложения и как выполнять действия при помощи активностей других приложений на вашем устройстве. Внезапно перед вами открываются совершенно новые перспективы...



Приложение может содержать несколько активностей	106
Структура приложения	107
Создание проекта	107
Обновление макета	108
Обновление strings.xml...	109
Создание второй активности и макета	110
Знакомьтесь: файл манифеста Android	112
Интент — разновидность сообщения	114
Использование интента для запуска второй активности	115
Что происходит при запуске приложения	116
Передача текста второй активности	118
Обновление свойств надписи	119
putExtra() включает в интент дополнительную информацию	120
Обновление кода CreateMessageActivity	123
Использование информации из интента в ReceiveMessageActivity	124
Что происходит при щелчке на кнопке Send Message	125
Как работают приложения Android	127
Создание интента с указанием действия	129
Изменение интента для использования действия	130
Что происходит при выполнении кода	131
Как Android использует фильтр интентов	134
Запуск приложения на РЕАЛЬНОМ устройстве	137
Что происходит при вызове createChooser()	141
Изменение кода выбора активности	143
Тест-драйв	144
Если подходящих активностей НЕТ	145
Ваш инструментарий Android	146

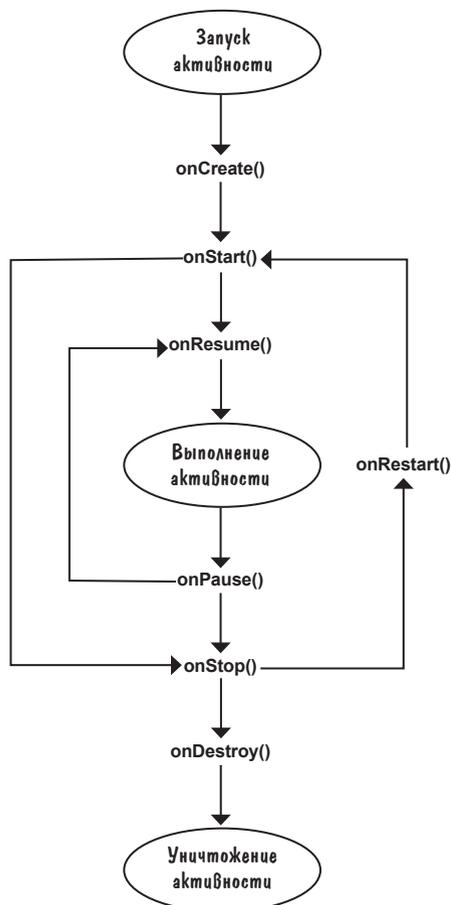


4 Жизненный цикл активности

Из жизни активностей

Активности образуют основу любого Android-приложения.

Ранее вы видели, как создавать активности и как организовать запуск одной активности из другой с использованием интента. Но что при этом происходит, если заглянуть поглубже? В этой главе более подробно рассматривается жизненный цикл активностей. Что происходит при создании или уничтожении активностей? Какие методы вызываются, когда активность становится видимой и появляется на переднем плане, и какие методы вызываются, когда активность теряет фокус и скрывается? И как выполняются операции сохранения и восстановления состояния активности?

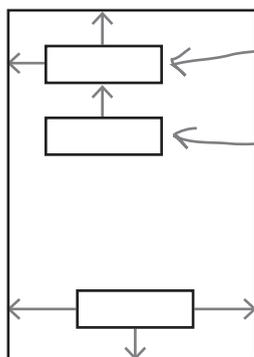


Как на самом деле работают активности?	148
Приложение Stopwatch	150
Разметка макета приложения Stopwatch	151
Как работает код активности	153
Добавление кода кнопок	154
Метод runTimer()	155
Объекты Handler позволяют планировать выполнение кода	156
Полный код runTimer()	157
Полный код StopwatchActivity	158
Что происходит при запуске приложения	160
Поворот экрана изменяет конфигурацию устройства	164
От рождения до смерти: состояния активности	165
Жизненный цикл активности: от создания до уничтожения	166
Активность наследует свои методы жизненного цикла	167
Что делать при изменении конфигурации?	168
Что происходит при запуске приложения	171
Жизненный цикл активности: видимость	175
Необходимо реализовать еще два метода жизненного цикла	176
Обновленный код StopwatchActivity	179
Что происходит при запуске приложения	180
Жизненный цикл активности: видимость	183
Прекращение отсчета времени при приостановке активности	186
Что происходит при запуске приложения	187
Тест-драйв	188
Полный код активности	189
Краткое руководство по методам жизненного цикла	193
Ваш инструментарий Android	194

5 Пользовательский интерфейс

Представление начинается

Давайте честно признаем: создавать хорошие макеты нужно уметь. Если вы строите приложения, которые должны использоваться людьми, необходимо позаботиться о том, чтобы эти макеты выглядели в точности так, как вам нужно. До настоящего момента мы едва затронули тему создания макетов; пришло время разобраться поглубже. Мы познакомим вас с другими типами макетов, которые могут использоваться в программах, после чего будет представлен обзор основных компонентов графического интерфейса и способов их использования. К концу главы вы увидите, что несмотря на внешние различия, у всех макетов и компонентов графического интерфейса больше общего, чем кажется на первый взгляд.



Представления могут размещаться относительно родительского макета...

...или относительно других представлений.

Три ключевых макета: относительный, линейный и табличный	197
Позиционирование представлений относительно родительского макета	200
Позиционирование представлений относительно других представлений	202
Атрибуты для позиционирования представлений относительно других представлений	203
Относительный макет: итоги	205
В линейных макетах представления выводятся в одну строку или столбец	206
Начало настройки линейного макета	209
Назначение веса одному представлению	211
Назначение весов нескольким представлениям	212
Атрибут android:gravity: список значений	214
Другие допустимые значения атрибута android:layout-gravity	216
Полная разметка линейного макета	217
Линейный макет: итоги	218
Добавление представлений в табличный макет	222
Создание табличного макета	223
Строка 0: добавление представлений в конкретные строки и столбцы	225
Строка 1: представление занимает несколько столбцов	226
Строка 2: представление занимает несколько столбцов	227
Полный код табличного макета	228
Табличный макет: итоги	229
Ваш инструментарий Android	257

6 Списковые представления и адаптеры

Обо всем по порядку

Хотите знать, как лучше организовать Android-приложение?

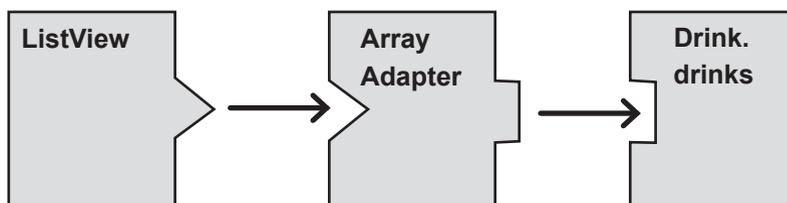
Мы рассмотрели основные структурные элементы, используемые при построении приложений; теперь пора привести знания в порядок. В этой главе мы покажем, как взять разрозненные идеи и превратить их в классное приложение. Мы покажем, как списки данных могут стать основой структуры вашего приложения и что связывание списков позволяет создавать мощные и удобные приложения. Попутно вы в общих чертах узнаете, как при помощи слушателей событий и адаптеров сделать ваше приложение более динамичным.

Вывести начальный экран со списком всех команд.

Вывести подробную информацию по каждому блюду.

Вывести подробную информацию по каждому напитку.

Каждое приложение начинается с идей	260
Проведите классификацию идей: верхний уровень, категории, детализация/редактирование	261
Навигация по активностям	262
Навигация с использованием списковых представлений	263
Построим приложение Starbuzz	264
Активность детализации с информацией о напитке	265
Структура приложения Starbuzz	266
Макет верхнего уровня содержит изображение и список	270
Полный код макета верхнего уровня	272
Обработка щелчков компонентом ListView	273
Полный код TopLevelActivity	275
Создание списковой активности	281
Связывание списковых представлений с массивами при помощи адаптера массива	283
Добавление адаптера массива в DrinkCategoryActivity	284
Что происходит при выполнении кода	285
Как мы обрабатывали щелчки в TopLevelActivity	288
Полный код DrinkCategoryActivity	290
Активность детализации выводит данные из одной записи	291
Обновление представлений	293
Код DrinkActivity	295
Тест-драйв	298
Ваш инструментарий Android	300



Фрагменты

1

Модульная структура

Вы уже умеете создавать приложения, которые работают одинаково независимо от устройства, на котором они запускаются.

Но что, если ваше приложение должно выглядеть и вести себя по-разному в зависимости от того, где оно запущено — на телефоне или планшете? В этой главе мы покажем, как в приложении выбрать наиболее подходящий макет по размерам экрана устройства. Также вы познакомитесь с фрагментами — механизмом создания модульных программных компонентов, которые могут повторно использоваться разными активностями.

Структура приложения Workout	305
Класс Workout	307
Добавление фрагмента в проект	308
Добавление фрагмента в макет активности	311
Передача идентификатора фрагменту	312
Присваивание идентификатора	313
Жизненный цикл фрагмента	315
Ваш фрагмент наследует методы жизненного цикла	316
Создание фрагмента со списком	320
Создание спискового фрагмента	322
Использование ArrayAdapter для заполнения ListView	323
Обновленный код WorkoutListFragment	324
Включение фрагмента WorkoutListFragment в макет MainActivity	325
Связывание списка с детализацией	327
Логическое отделение фрагмента от активности	328
Добавление интерфейса к списковому фрагменту	329
Реализация интерфейса активностью	330
Фрагменты должны поддерживать кнопку возврата	331
Транзакции фрагментов	333
Обновленный код MainActivity	334
Тест-драйв	335
Поворот устройства нарушает работу приложения	336
Код WorkoutDetailFragment	337
Телефон и планшет	338
Структура приложения для планшета и телефона	339
Выбор имен папок	341
Макет MainActivity для телефона	347
Полный код DetailActivity	351
Использование различий в макетах	352
Обновленный код MainActivity	353
Тест-драйв	354
Ваш инструментарий Android	355

Значит, фрагмент будет содержать только списковое представление. Интересно... Для активностей, содержащих только список, мы использовали активность ListActivity. Нет ли чего-нибудь похожего для фрагментов?



Вложенные фрагменты

Укращение фрагментов

Вы уже видели, что использование фрагментов в активностях способствует повторному использованию кода и делает приложения более гибкими. В этой главе мы покажем, как вложить один фрагмент внутрь другого. Вы научитесь пользоваться диспетчером дочерних фрагментов для укращения строптивных транзакций фрагментов. А попутно вы узнаете, почему так важно знать различия между активностями и фрагментами.

Создание вложенных фрагментов	358
Жизненные циклы фрагментов и активностей похожи...	359
Код StopwatchFragment	364
Макет StopwatchFragment	367
Добавление фрагмента в WorkoutDetailFragment	369
Добавление фрейма в месте нахождения фрагмента	370
Отображение фрагмента в коде Java	371
getManager() создает транзакции на уровне активности	372
Вложенным фрагментам — вложенные транзакции	373
Отображение фрагмента в методе onCreateView() родителя	374
Полный код WorkoutDetailFragment	375
Почему при нажатии кнопки происходит сбой?	377
Обратимся к разметке макета StopwatchFragment	378
Атрибут onClick вызывает методы активности, а не фрагмента	379
Удаление атрибутов onClick из макета фрагмента	380
Реализация OnClickListener фрагментом	381
Связывание OnClickListener с кнопками	383
Код StopwatchFragment	384
При повороте устройства активность создается заново	387
Код WorkoutDetailFragment	390
Тест-драйв	391
Ваш инструментарий Android	396

Посмотрите-ка — транзакции фрагментов. Я должна немедленно применить их.



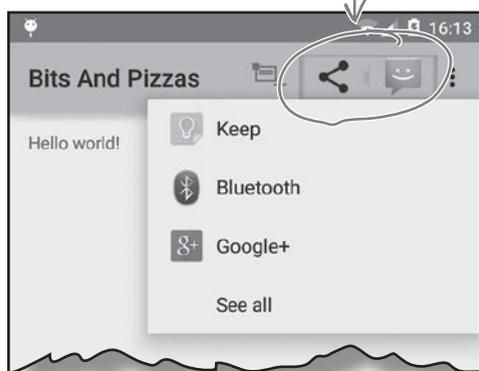
9

Панели Действий

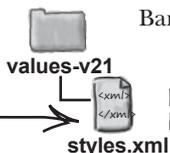
В поисках короткого пути

Все мы предпочитаем короткие пути к цели. В этой главе вы узнаете, как ускорить выполнение команд ваших приложений при помощи панелей действий. Вы узнаете, как запускать другие активности из элементов действий на панели действий, как передавать данные другим приложениям при помощи провайдера передачи информации и как перемещаться в иерархии приложения с использованием кнопки Вверх на панели действий. Попутно вы узнаете, как обеспечить единый стиль внешнего вида и поведения приложений с использованием тем оформления, и познакомитесь с пакетом библиотеки поддержки Android.

Так выглядит действие передачи информации на панели действий. Если щелкнуть на нем, на экране появляется список приложений, которые могут использоваться для передачи информации.



API 21? Идеальное совпадение...



Name: AppTheme
Parent: Theme.Material.Light

Хорошее приложение имеет четкую структуру	398
Типы навигации	399
Начнем с панели действий	400
Библиотеки поддержки Android	401
Библиотеки поддержки можно включить в ваш проект	402
Использование современных тем в приложении themes	403
Применение темы в AndroidManifest.xml	404
Определение стилей в файлах стилевых ресурсов	405
Назначение темы по умолчанию в styles.xml	406
Что происходит при запуске приложения	407
Добавление элементов действий на панель действий	408
Файл ресурсов меню	409
Атрибут меню showAsAction	410
Добавление нового элемента действия	411
Создание OrderActivity	414
Запуск OrderActivity элементом действия Create Order	415
Полный код MainActivity.java	416
Передача информации с панели действий	418
Добавление провайдера в файл menu_main.xml	419
Информация задается при помощи интента	420
Полный код MainActivity.java	421
Код MainActivity.java (продолжение)	422
Навигация с кнопкой Вверх	423
Назначение родителя активности	424
Добавление кнопки Вверх	425
Тест-драйв	426
Ваш инструментарий Android	427

10

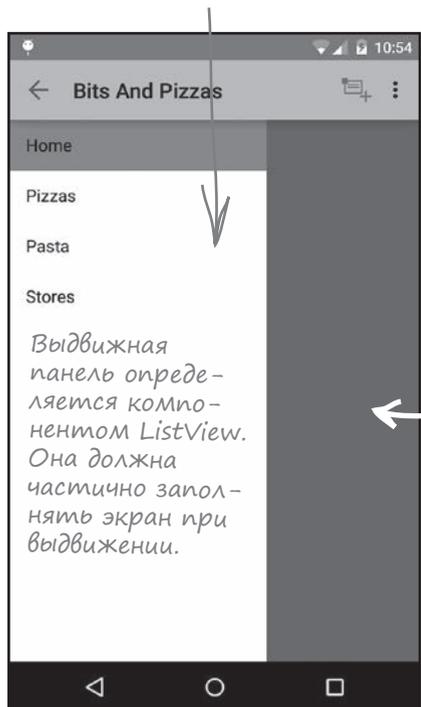
ВЫДВИЖНЫЕ ПАНЕЛИ

Подальше положишь...

С приложением намного приятнее работать, когда в нем хорошо организована навигация. В этой главе мы представим

выдвижные панели (drawers) — чтобы открыть их, следует провести пальцем по экрану или прикоснуться к значку на панели действий. Вы увидите, как использовать выдвижные панели для вывода списка ссылок, ведущих к основным разделам приложения. Также вы увидите, как переключение фрагментов упрощает переход к этим разделам и ускоряет их отображение.

Информация выводится во фрейме, при этом она заполняет экран. В настоящий момент фрейм частично скрыт выдвижной панелью.



Возвращаемся к приложению Pizza	430
Подробнее о выдвижных панелях	431
Структура приложения Pizza	432
Создание фрагмента TopFragment	433
Создание фрагмента PizzaFragment	434
Создание фрагмента PastaFragment	435
Создание фрагмента StoresFragment	436
Добавление DrawerLayout	437
Полная разметка activity_main.xml	438
Инициализация списка на выдвижной панели	439
OnClickListener и обработка щелчков на вариантах спискового представления	440
Метод selectItem() — что было сделано	443
Изменение заголовка на панели действий	444
Закрытие выдвижной панели	445
Обновленный код MainActivity.java	446
Открытие и закрытие выдвижной панели	448
Использование ActionBarDrawerToggle	449
Изменение элементов панели действий во время выполнения	450
Обновленный код MainActivity.java	451
Управление открытием и закрытием выдвижной панели	452
Синхронизация состояния ActionBarDrawerToggle	453
Обновленный код MainActivity.java	454
Обработка изменений конфигурации	457
Реакция на изменения в стеке возврата	458
Назначение меток фрагментам	459
Поиск фрагмента по метке	460
Полный код MainActivity.java	461
Тест-драйв	467
Ваш инструментарий Android	468

11

Базы данных SQLite

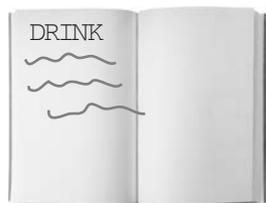
Работа с базами данных

Какая бы информация ни использовалась в приложении — рекордные счета или тексты сообщений в социальной сети, — эту информацию необходимо где-то хранить.

В Android для долгосрочного хранения данных обычно используется база данных SQLite. В этой главе вы узнаете, как создать базу данных, добавить в нее таблицы и заполнить данными, — все это делается при помощи удобных вспомогательных объектов SQLite. Затем будет показано, как выполнить безопасное обновление структуры базы данных и как вернуться к предыдущей версии в случае необходимости.



onCreate()



SQLite database

Name: "starbuzz"
Version: 1

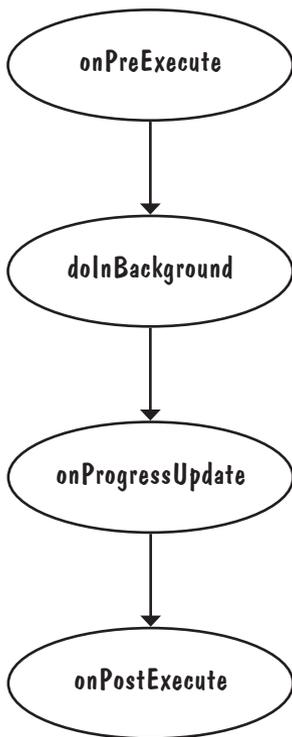
Возвращение в Starbuzz	470
Android хранит информацию в базах данных SQLite	471
Android включает классы SQLite	472
Текущая структура приложения Starbuzz	473
Переход на работу с базой данных	474
Помощник SQLite управляет базой данных	475
Помощник SQLite	475
Создание помощника SQLite	476
Внутри базы данных SQLite	478
Таблицы создаются командами SQL	479
Вставка данных методом insert()	480
Обновление записей методом update()	481
Сложные условия	482
Код StarbuzzDatabaseHelper	483
Что делает код помощника SQLite	484
А если структура базы данных изменится?	487
Номера версий баз данных SQLite	488
Обновление базы данных: сводка	489
История продолжается...	490
Как помощник SQLite принимает решения	491
Метод onUpgrade()	492
Метод onDowngrade()	493
Обновление базы данных	494
Обновление существующей базы данных	497
Переименование таблиц	498
Полный код помощника SQLite	499
Что происходит при выполнении кода	501
Ваш инструментарий Android	502

12

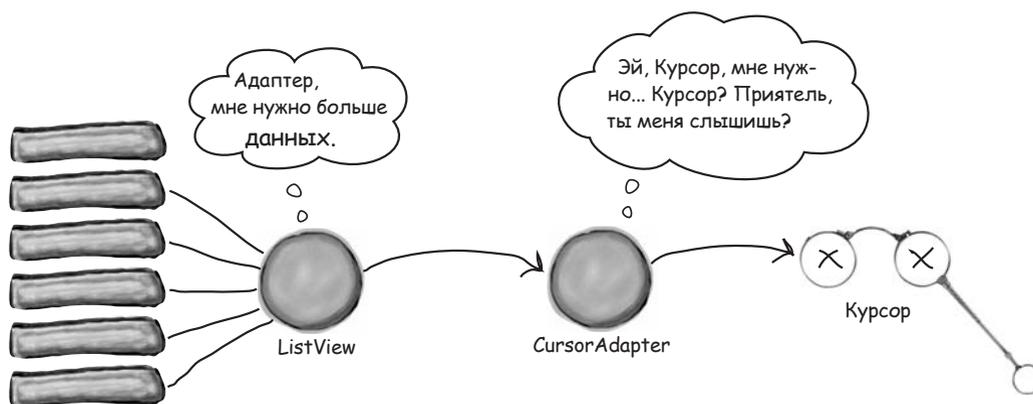
Курсоры и асинхронные задачи

Подключение к базам данных

Как же подключиться из приложения к базе данных SQLite? В предыдущей главе было показано, как создать базу данных SQLite с использованием помощника SQLite. Пора сделать следующий шаг — узнать, как работать с базой данных из активностей. В этой главе вы узнаете, как использовать курсоры для получения информации из базы данных, как перемещаться по набору данных с использованием курсора и как получить данные из курсора. Затем вы узнаете, как использовать адаптеры курсоров для их связывания со списковыми представлениями. В завершение мы покажем, как написание эффективного многопоточного кода с объектами AsyncTask ускоряет работу приложений.



Текущий код DrinkActivity	506
Определение таблицы и столбцов	510
Упорядочение данных в запросах	512
Функции SQL в запросах	513
Код DrinkActivity	522
Текущий код DrinkCategoryActivity	525
Код DrinkActivity	545
Обновленный код TopLevelActivity.java	556
Метод onPreExecute()	563
Метод doInBackground()	564
Метод onProgressUpdate()	565
Метод onPostExecute()	566
Класс AsyncTask	567
Код DrinkActivity.java	569
Ваш инструментарий Android	572



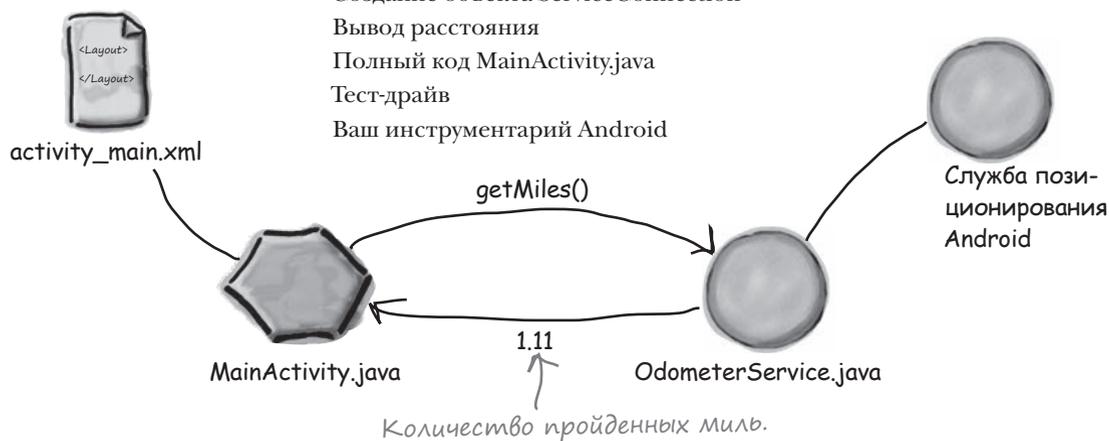
13

СЛУЖБЫ

К вашим услугам

Существуют операции, которые должны выполняться постоянно. Например, если вы запустили воспроизведение музыкального файла в приложении-проигрывателе, вероятно, музыка не должна останавливаться при переключении на другое приложение. В этой главе вы узнаете, как использовать службы для подобных ситуаций, а заодно научитесь пользоваться некоторыми встроенными службами Android. Служба уведомлений поможет вам держать пользователей в курсе дел, а при помощи службы позиционирования пользователь сможет узнать, где он находится.

Запускаемая служба	575
Класс IntentService — взгляд издалека	577
Объявление службы в AndroidManifest.xml	580
Службы запускаются вызовом startService()	582
Полный код DelayedMessageService.java	586
Использование службы уведомлений	589
Запуск активности из оповещения	591
Отправка уведомлений с использованием службы уведомлений	593
Полный код DelayedMessageService.java	594
Создание проекта Odometer	603
Определение Binder	605
Четыре ключевых метода класса Service	607
Получение информации о местонахождении	608
Добавление LocationListener в код службы	609
Регистрация LocationListener	610
Передача пройденного расстояния активности	611
Полный код OdometerService.java	612
Обновление AndroidManifest.xml	614
Обновление макета MainActivity	618
Создание объекта ServiceConnection	619
Вывод расстояния	621
Полный код MainActivity.java	622
Тест-драйв	626
Ваш инструментарий Android	627

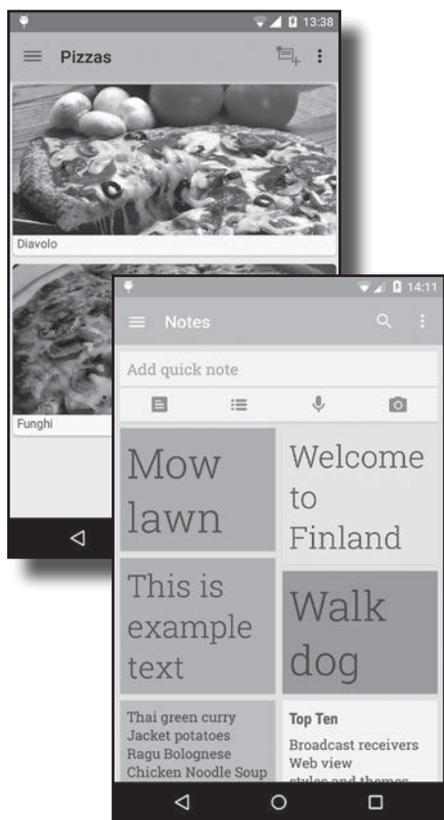


14

Материальное оформление

Жизнь в материальном мире

В API уровня 21 компания Google представила концепцию материального оформления. В этой главе вы узнаете, что такое материальное оформление и как реализовать его принципы в ваших приложениях. Мы начнем с карточек, которые могут повторно использоваться в приложениях для обеспечения целостности оформления. Затем будет рассмотрен компонент RecyclerView — хороший друг спискового представления. Попутно вы узнаете, как создавать адаптеры и как полностью изменить внешний вид RecyclerView всего двумя строками кода.



Знакомство с материальным оформлением	630
Структура приложения Pizza	632
Создание представлений CardView	635
Полная разметка card_captioned_image.xml	636
Создание базового адаптера	638
Определение класса ViewHolder	639
Создание ViewHolder	640
Добавление данных в карточки	642
Полный код CaptionedImagesAdapter.java	643
Создание RecyclerView	644
Добавление RecyclerView в макет	645
Код PizzaMaterialFragment.java	646
Размещение представлений в RecyclerView	647
Полный код PizzaMaterialFragment.java	649
Использование нового фрагмента PizzaMaterialFragment в MainActivity	650
Что происходит при выполнении кода	651
Создание PizzaDetailActivity	659
Что должен делать код PizzaDetailActivity.java	660
Код PizzaDetailActivity.java	661
Выбор вариантов в RecyclerView	663
Прослушивание событий представлений в адаптере	664
Повторное использование адаптеров	665
Добавление интерфейса в адаптер	666
Реализация слушателя в PizzaMaterialFragment.java	668
Информацию — на передний план	671
Полная разметка fragment_top.xml	676
Полный код TopFragment.java	677
Ваш инструментарий Android	679



Исполнительная среда Android

Приложения Android должны работать на устройствах с маломощными процессорами и ограниченной памятью. Java-приложения могут расходовать много памяти. Кроме того, если бы приложения выполнялись на виртуальной машине Java (JVM, Java Virtual Machine), на маломощных устройствах запуск приложения мог бы занимать много времени. Чтобы избежать этих проблем, система Android вместо JVM использует для запуска своих приложений другую виртуальную машину, которая называется ART (Android runtime). В этом приложении вы узнаете, как ART удается обеспечить нормальное выполнение Java-приложений на компактном маломощном устройстве.



Android Debug Bridge

В этой книге для всех задач Android-программирования использовались средства среды разработки. Однако в некоторых случаях инструменты командной строки попросту более эффективны — например, если Android Studio упорно отказывается видеть ваше устройство Android, а вы *знаете*, что оно есть. В этой главе мы познакомимся с **Android Debug Bridge (сокращенно adb)** — утилитой командной строки, предназначенной для взаимодействия с эмулятором или устройствами Android.



Эмулятор Android

У вас никогда не возникало ощущение, что эмулятора приходится ждать целую вечность? Безусловно, эмулятор Android полезен. С его помощью можно увидеть, как приложение будет работать на других устройствах — помимо физических устройств, имеющихся у вас. Но иногда он кажется слегка... заторможенным. В этом приложении мы объясним, почему эмулятор кажется медленным. Более того, мы приведем несколько полезных советов для ускорения его работы.

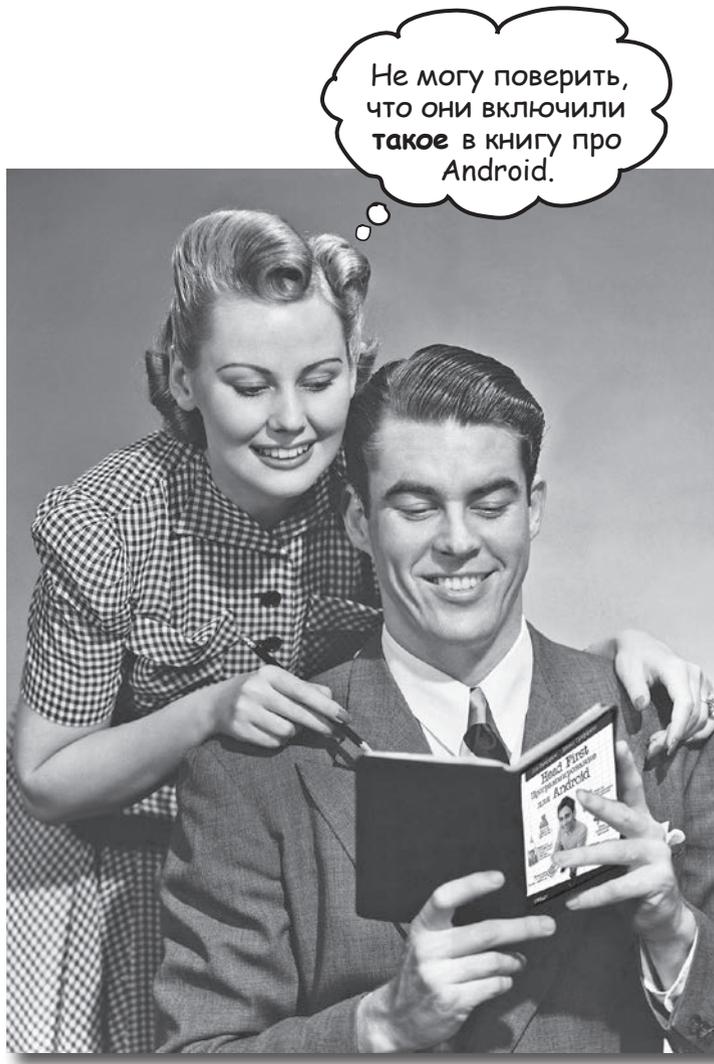


Десять важнейших тем (которые мы не рассмотрели)

Но и это еще не все. Осталось еще несколько тем, о которых, как нам кажется, вам следует знать. Делать вид, что их не существует, было бы неправильно — как, впрочем, и выпускать книгу, которую поднимет разве что культурист. Прежде чем откладывать книгу, ознакомьтесь с этими **лакомыми кусочками**, которые мы оставили напоследок.

Как работать с этой книгой

Введение



В этом разделе мы ответим на насущный вопрос: «Так почему они включили ТАКОЕ в книгу по программированию для Android?»»

Для кого написана эта книга?

Если вы ответите «да» на все следующие вопросы:

- 1 Вы уже умеете программировать на Java?
- 2 Вы хотите достичь мастерства в области разработки приложения для Android, создать следующий бестселлер в области программных продуктов, заработать целое состояние и купить собственный остров?
- 3 Вы предпочитаете заниматься практической работой и применять полученные знания вместо того, чтобы выслушивать нудные многочасовые лекции?

Ладно, здесь мы слегка хвятили через край. Но ведь нужно с чего-то начинать, верно?

тогда эта книга для вас.

Кому эта книга не подойдет?

Если вы ответите «да» на один из следующих вопросов:

- 1 Вам нужен краткий вводный курс или справочник по разработке приложений для Android?
- 2 Вы скорее пойдете к зубному врачу, чем опробуете что-нибудь новое? Вы считаете, что в книге по Android не должно быть веселых человечков, а если читатель будет помирать со скуки — еще лучше?

...эта книга *не* для вас.

*[Замечание от отдела продаж:
вообще-то эта книга для любого,
у кого есть деньги... и если что —
PayPal тоже подойдет.]*



Мы знаем, о чем вы думаете

«Разве серьезные книги по программированию для Android *такие?*»

«И почему здесь столько рисунков?»

«Можно ли так чему-нибудь *научиться?*»

И мы знаем, о чем думает ваш мозг

Мозг жаждет новых впечатлений. Он постоянно ищет, анализирует, *ожидает* чего-то необычного. Он так устроен, и это помогает нам выжить.

Как наш мозг поступает со всеми обычными, повседневными вещами? Он всеми силами пытается оградиться от них, чтобы они не мешали его *настоящей* работе — сохранению того, что действительно *важно*. Мозг не считает нужным сохранять скучную информацию. Она не проходит фильтр, отсекающий «очевидно несущественное».

Но как же мозг *знает*, что важно? Представьте, что вы выехали на прогулку и вдруг прямо перед вами появляется тигр. Что происходит в вашей голове и теле?

Активизируются нейроны. Вспыхивают эмоции. Происходят химические реакции. И тогда ваш мозг понимает...

Конечно, это важно! Не забывать!

А теперь представьте, что вы находитесь дома или в библиотеке — в теплом, уютном месте, где тигры не водятся. Вы учитесь — готовитесь к экзамену. Или пытаетесь освоить сложную техническую тему, на которую вам выделили неделю... максимум десять дней.

И тут возникает проблема: ваш мозг пытается оказать вам услугу. Он старается сделать так, чтобы на эту *очевидно* несущественную информацию не тратились драгоценные ресурсы. Их лучше потратить на что-нибудь важное. На тигров, например. Или на то, что к огню лучше не прикасаться. Или что на лыжах не стоит кататься в футболке и шортах.

Нет простого способа сказать своему мозгу: «Послушай, мозг, я тебе, конечно, благодарен, но какой бы скучной ни была эта книга и пусть мой датчик эмоций сейчас на нуле, я *хочу* запомнить то, что здесь написано».



Эта книга для тех, кто хочет учиться.

Как мы что-то узнаем? Сначала нужно это «что-то» *понять*, а потом *не забыть*. Затолкать в голову побольше фактов недостаточно. Согласно новейшим исследованиям в области когнитивистики, нейробиологии и психологии обучения, для усвоения материала требуется что-то большее, чем простой текст на странице. Мы знаем, как заставить ваш мозг работать.

Основные принципы серии «Head First»:

Наглядность. Графика запоминается лучше, чем обычный текст, и значительно повышает эффективность восприятия информации (до 89 % по данным исследований). Кроме того, материал становится более понятным. **Текст размещается на рисунках**, к которым он относится, а не под ними или на соседней странице — и вероятность успешного решения задач, относящихся к материалу, повышается вдвое.

Разговорный стиль изложения. Недавние исследования показали, что при разговорном стиле изложения материала (вместо формальных лекций) улучшение результатов на итоговом тестировании достигает 40 %. Рассказывайте историю, вместо того чтобы читать лекцию. Не относитесь к себе слишком серьезно. Что привлечет ваше внимание: занимательная беседа за столом или лекция?

Активное участие читателя. Пока вы не начнете напрягать извилины, в вашей голове ничего не произойдет. Читатель должен быть заинтересован в результате; он должен решать задачи, формулировать выводы и овладевать новыми знаниями. А для этого необходимы упражнения и каверзные вопросы, в решении которых задействованы оба полушария мозга и разные чувства.

Привлечение (и сохранение) внимания читателя. Ситуация, знакомая каждому: «Я очень хочу изучить это, но засыпаю на первой странице». Мозг обращает внимание на интересное, странное, притягательное, неожиданное. Изучение сложной технической темы не обязано быть скучным. Интересное узнается намного быстрее.

Обращение к эмоциям. Известно, что наша способность запоминать в значительной мере зависит от эмоционального сопереживания. Мы запоминаем то, что нам безразлично. Мы запоминаем, когда что-то чувствуем. Нет, сантименты здесь ни при чем: речь идет о таких эмоциях, как удивление, любопытство, интерес и чувство «Да я крут!» при решении задачи, которую окружающие считают сложной — или когда вы понимаете, что разбираетесь в теме лучше, чем всезнайка Боб из технического отдела.

Метапознание: наука о мышлении

Если вы действительно хотите быстрее и глубже усваивать новые знания — задумайтесь над тем, как вы думаете. Учитесь учиться.

Мало кто из нас изучает теорию метапознания во время учебы. Нам *положено* учиться, но нас редко этому *учат*.

Но раз вы читаете эту книгу, то, вероятно, вы хотите освоить программирование для Android, и по возможности быстрее. Вы хотите *запомнить* прочитанное, а для этого абсолютно необходимо сначала *понять* прочитанное.

Чтобы извлечь максимум пользы из учебного процесса, нужно заставить ваш мозг воспринимать новый материал как Нечто Важное. Критичное для вашего существования. Такое же важное, как тигр. Иначе вам предстоит бесконечная борьба с вашим мозгом, который всеми силами уклоняется от запоминания новой информации.

Как же УБЕДИТЬ мозг, что программирование не менее важно, чем голодный тигр?

Есть способ медленный и скучный, а есть быстрый и эффективный. Первый основан на тупом повторении. Всем известно, что даже самую скучную информацию *можно* запомнить, если повторять ее снова и снова. При достаточном количестве повторений ваш мозг прикидывает: «Вроде бы несущественно, но раз одно и то же повторяется *столько* раз... Ладно, уговорил».

Быстрый способ основан на **повышении активности мозга** и особенно на сочетании разных ее *видов*. Доказано, что все факторы, перечисленные на предыдущей странице, помогают вашему мозгу работать на вас. Например, исследования показали, что размещение слов *внутри* рисунков (а не в подписях, в основном тексте и т. д.) заставляет мозг анализировать связи между текстом и графикой, а это приводит к активизации большего количества нейронов. Больше нейронов — выше вероятность того, что информация будет сочтена важной и достойной запоминания.

Разговорный стиль тоже важен: обычно люди проявляют больше внимания, когда они участвуют в разговоре, так как им приходится следить за ходом беседы и высказывать свое мнение. Причем мозг совершенно не интересуется, что вы «разговариваете» с книгой! С другой стороны, если текст сух и формален, то мозг чувствует то же, что чувствуете вы на скучной лекции в роли пассивного участника. Его клонит в сон.

Но рисунки и разговорный стиль — это только начало.



Вот что сделали МЫ

Мы использовали **рисунки**, потому что мозг лучше приспособлен для восприятия графики, чем текста. С точки зрения мозга рисунок стоит 1024 слова. А когда текст комбинируется с графикой, мы внедряем текст прямо в рисунки, потому что мозг при этом работает эффективнее.

Мы используем **избыточность**: повторяем одно и то же несколько раз, применяя разные средства передачи информации, обращаемся к разным чувствам — и все для повышения вероятности того, что материал будет закодирован в нескольких областях вашего мозга.

Мы используем концепции и рисунки несколько **неожиданным** образом, потому что мозг лучше воспринимает новую информацию. Кроме того, рисунки и идеи обычно имеют **эмоциональное содержание**, потому что мозг обращает внимание на биохимию эмоций. То, что заставляет нас **чувствовать**, лучше запоминается — будь то **шутка**, **удивление** или **интерес**.

Мы используем **разговорный стиль**, потому что мозг лучше воспринимает информацию, когда вы участвуете в разговоре, а не пассивно слушаете лекцию. Это происходит и при **чтении**.

В книгу включены многочисленные упражнения, потому что мозг лучше запоминает, когда вы что-то делаете. Мы постарались сделать их непростыми, но интересными — то, что предпочитает большинство читателей.

Мы совместили **несколько стилей обучения**, потому что одни читатели предпочитают пошаговые описания, другие стремятся сначала представить «общую картину», а третьим хватает фрагмента кода. Независимо от ваших личных предпочтений полезно видеть несколько вариантов представления одного материала.

Мы постарались задействовать **оба полушария вашего мозга**; это повышает вероятность усвоения материала. Пока одна сторона мозга работает, другая часто имеет возможность отдохнуть; это повышает эффективность обучения в течение продолжительного времени.

А еще в книгу включены **истории** и упражнения, отражающие другие точки зрения. Мозг глубже усваивает информацию, когда ему приходится оценивать и выносить суждения.

В книге часто встречаются **вопросы**, на которые не всегда можно дать простой ответ, потому что мозг быстрее учится и запоминает, когда ему приходится что-то делать. Невозможно накачать **мышцы**, наблюдая за тем, как занимаются **другие**. Однако мы позаботились о том, чтобы усилия читателей были приложены в **верном** направлении. Вам не придется ломать голову над невразумительными примерами или разбираться в сложном, перенасыщенном техническим жаргоном или слишком лаконичном тексте.

В историях, примерах, на картинках используются **люди** — потому что вы тоже **человек**. И ваш мозг обращает на людей больше внимания, чем на неодушевленные **предметы**.



Вырежьте и прикрепите на холодильник.

Что можете сделать ВЫ, чтобы заставить свой мозг повиноваться

Мы свое дело сделали. Остальное за вами. Эти советы станут отправной точкой; прислушайтесь к своему мозгу и определите, что вам подходит, а что не подходит. Попробуйте новое.

- 1 Не торопитесь. Чем больше вы поймете, тем меньше придется запоминать.**
Просто читать недостаточно. Когда книга задает вам вопрос, не переходите к ответу. Представьте, что кто-то *действительно* задает вам вопрос. Чем глубже ваш мозг будет мыслить, тем скорее вы поймете и запомните материал.
- 2 Выполняйте упражнения, делайте заметки.**
 Мы включили упражнения в книгу, но выполнять их за вас не собираемся. И не *разглядывайте* упражнения. **Берите карандаш и пишите.** Физические действия *во время* учения повышают его эффективность.
- 3 Читайте врезки.**
 Это значит: читайте все. **Врезки – часть основного материала!** Не пропускайте их.
- 4 Не читайте другие книги после этой перед сном.**
 Часть обучения (особенно перенос информации в долгосрочную память) происходит после того, как вы откладываете книгу. Ваш мозг не сразу усваивает информацию. Если во время обработки поступит новая информация, часть того, что вы узнали ранее, может быть потеряна.
- 5 Говорите вслух.**
 Речь активизирует другие участки мозга. Если вы пытаетесь что-то понять или получше запомнить, произнесите вслух. А еще лучше, попробуйте объяснить кому-нибудь другому. Вы будете быстрее усваивать материал и, возможно, откроете для себя что-то новое.
- 6 Пейте воду. И побольше.**
 Мозг лучше всего работает в условиях высокой влажности. Дегидратация (которая может наступить еще до того, как вы почувствуете жажду) снижает когнитивные функции.
- 7 Прислушивайтесь к своему мозгу.**
 Следите за тем, когда ваш мозг начинает уставать. Если вы начинаете поверхностно воспринимать материал или забываете только что прочитанное, пора сделать перерыв.
- 8 Чувствуйте!**
 Ваш мозг должен знать, что материал книги действительно *важен*. Переживайте за героев наших историй. Придумывайте собственные подписи к фотографиям. Поморщиться над неудачной шуткой все равно лучше, чем не почувствовать ничего.
- 9 Пишите побольше кода!**
 Освоить разработку Android-приложений можно только одним способом: **писать побольше кода**. Именно этим мы и будем заниматься в книге. Программирование – искусство, и добиться мастерства в нем можно только практикой. Для этого у вас будут все возможности: в каждой главе приведены упражнения, в которых вам придется решать задачи. Не пропускайте их – работа над упражнениями является важной частью процесса обучения. К каждому упражнению приводится решение – не бойтесь **заглянуть в него**, если окажетесь в тупике! (Споткнуться можно даже о маленький камешек.) По крайней мере постарайтесь решить задачу, прежде чем заглядывать в решение. Обязательно добейтесь того, чтобы ваше решение заработало, прежде чем переходить к следующей части книги.

Примите к сведению

Это учебник, а не справочник. Мы намеренно убрали из книги все, что могло бы помешать изучению материала, над которым вы работаете. И при первом чтении книги начинать следует с самого начала, потому что книга предполагает наличие у читателя определенных знаний и опыта.

Предполагается, что у вас уже есть опыт программирования на языке Java.

Мы будем строить приложения Android с использованием Java и XML. Предполагается, что вы уже знакомы с языком программирования Java. Если вы еще *никогда* не писали программы на Java, прочитайте *Head First Java*, прежде чем браться за эту книгу.

Мы начинаем строить приложения с первой главы.

Хотите — верьте, хотите — нет, но даже если вы никогда не программировали для Android, вы все равно можете сходу взяться за создание приложений. Заодно вы познакомитесь с Android Studio, основной интегрированной средой разработки для Android.

Примеры создавались для обучения.

Во время работы над книгой мы построим несколько разных приложений. Некоторые из них очень малы, чтобы вы могли сосредоточиться на конкретных аспектах Android. Другие, более крупные приложения показывают, как разные компоненты работают в сочетании друг с другом. Мы не будем доводить до конца все части всех приложений, но ничто не мешает вам экспериментировать с ними самостоятельно — это часть учебного процесса. Исходный код всех приложений доступен по адресу: <https://tinyurl.com/HeadFirstAndroid>.

Упражнения ОБЯЗАТЕЛЬНЫ.

Упражнения являются частью основного материала книги. Одни упражнения способствуют запоминанию материала, другие помогают лучше понять его, третьи ориентированы на его практическое применение. *Не пропускайте упражнения.*

Повторение применяется намеренно.

У книг этой серии есть одна принципиальная особенность: мы хотим, чтобы вы *действительно хорошо* усвоили материал. И чтобы вы запомнили все, что узнали. Большинство справочников не ставит своей целью успешное запоминание, но это не справочник, а учебник, поэтому некоторые концепции излагаются в книге по несколько раз.

Упражнения «Мозговой штурм» не имеют ответов.

В некоторых из них правильного ответа вообще нет, в других вы должны сами решить, насколько правильны ваши ответы (это является частью процесса обучения). В некоторых упражнениях «Мозговой штурм» приводятся подсказки, которые помогут вам найти нужное направление.

Научные редакторы

Эдвард



Эдвард Ю Шун Вон увлекается программированием с того момента, когда он написал свою первую строку кода Haskell в 2006-м. В настоящее время он работает над событийным управлением торговыми операциями в центре Лондона. Он щедро делится своей страстью к разработке с сообществами London Java Community и Software Craftmanship Community. В свободное время Эдварда можно найти на футбольном поле или за подготовкой игровых обзоров для YouTube (@arkangelofkaos).

Тони Уильямс — разработчик с опытом программирования на языке Java и создания приложений для Android.

Благодарности

Нашему редактору:

Большое спасибо нашему редактору **Меган Бланшетт**, взявшей под опеку серию Head First. Ее мнение и оценки были чрезвычайно полезными. Вы не представляете, сколько раз она говорила, что наши слова состоят из правильных букв, но не всегда в правильном порядке.

Спасибо **Берту Бэйтсу**, который помог нам отказаться от старых учебников и поделился своими мыслями. Информация, полученная от Берта, значительно улучшила эту книгу.

Меган Бланшетт



Группе O'Reilly:

Огромное спасибо **Майку Хендриксону**, который поверил нам и предложил написать книгу; **Кортни Нэш** за все содействие на ранних стадиях работы над книгой; **группе предварительного выпуска** за подготовку ранних версий этой книги. Также мы хотим поблагодарить **Мелани Ярбро**, **Джасмин Квитин** и остальных участников производственной группы, столь умело руководивших процессом производства.

Семье, друзьям и коллегам:

Написание книги из серии Head First — затея непростая, и этот раз не стал исключением. Вероятно, эта книга не вышла бы в свет, если бы не доброта и поддержка наших друзей и родственников. Отдельное спасибо **Энди П. Стиву**, **Колину**, **Жаки**, **Анджеле**, **Полу Б.**, **маме**, **папе**, **Карлу**, **Робу** и **Лоррейн**.

Без кого эта книга не появилась бы:

Наша группа технического рецензирования прекрасно справилась со своей задачей: поддерживать нас на правильном пути и обеспечить актуальность материала. Мы также благодарны всем, кто поделился своим мнением по поводу ранних версий книги. Как нам кажется, книга от этого стала намного, намного лучше.

И наконец, спасибо **Кэти Сьерра** и **Берту Бэйтсу** за создание этой замечательной серии книг.

1 Первые шаги

С ГОЛОВОЙ В ПУЧИНУ



Система Android покорила мир. Все хотят иметь планшет или смартфон, а устройства на базе Android пользуются невероятной популярностью. В этой книге мы научим вас разрабатывать собственные приложения, а также покажем, как построить простое приложение и запустить его на виртуальном устройстве Android. Попутно будут рассмотрены основные компоненты приложений Android — такие как активности и макеты. Все, что от вас потребуется — некоторые базовые знания Java...

Добро пожаловать в мир Android

Android — самая популярная мобильная платформа в мире. Согласно последним опросам, в мире свыше *миллиарда* активных Android-устройств, и их количество продолжает стремительно расти. Android — полнофункциональная платформа с открытым кодом на базе Linux, разрабатываемая компанией Google. Это мощная платформа разработки, включающая все необходимое для построения современных приложений из кода Java и XML. Более того, построенные приложения могут устанавливаться на множестве разных устройств — телефонах, планшетах и не только. Что же собой представляет типичное Android-приложение?

Макеты определяют, как будут выглядеть экраны приложения

Типичное Android-приложение состоит из одного или нескольких экранов. Внешний вид каждого экран определяется при помощи **макета**. Макеты обычно состоят из разметки XML и могут включать компоненты графических интерфейсов: кнопки, текстовые поля, подписи и т. д.

Макеты сообщают Android, как будут выглядеть экраны вашего приложения.

Код Java определяет, что приложение должно делать

Макеты определяют только *внешний вид* приложения. Чтобы определить, что приложение должно *делать*, разработчик пишет код Java. Специальный класс Java, называемый **активностью**, решает, какой макет следует использовать, и описывает, как приложение должно реагировать на действия пользователя. Например, если в макете присутствует кнопка, вы должны написать для активности код Java, определяющий, что будет происходить при нажатии этой кнопки.

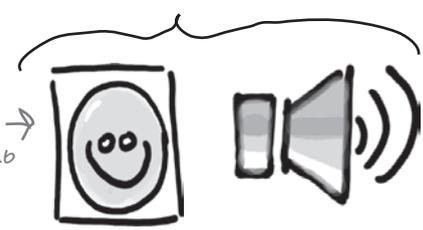
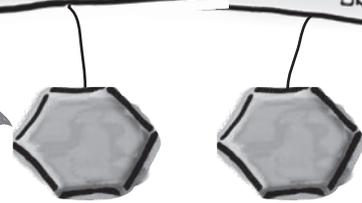
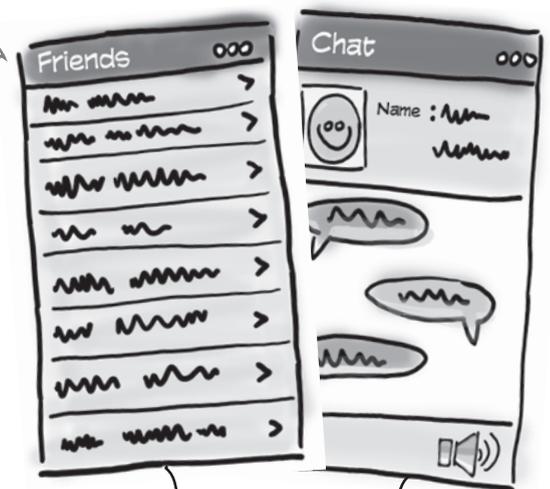
Активности определяют, что приложение должно делать.

Иногда не обойтись без дополнительных ресурсов

Кроме кода Java и макетов, в Android-приложения можно добавить любые дополнительные файлы. На самом деле приложение Android — всего лишь набор файлов в заранее определенных каталогах. При построении приложения все эти файлы собираются воедино, и вы получаете приложение, которое можно запустить на вашем устройстве.

Ресурсы могут включать звуковые и графические файлы.

Наши Android-приложения будут состоять из кода Java и XML. Некоторые вещи будут объясняться по ходу дела, но чтобы извлечь пользу из этой книги, читатель должен неплохо разбираться в Java.



Платформа Android в разрезе

Платформа Android состоит из множества компонентов. В нее входят базовые приложения (например, Контакты), набор программных интерфейсов (API) для управления внешним видом и поведением приложения, а также множество вспомогательных файлов и библиотек поддержки. Структура, которая образуется из этих компонентов, выглядит примерно так:

Android включает несколько базовых приложений — таких, как Контакты, Календарь, Карты и браузер.

При построении приложений вам доступны те же API, которые используются базовыми приложениями. При помощи этих API вы управляете внешним видом и поведением своих приложений.

Под инфраструктурой приложений располагается уровень библиотек C и C++. Для работы с ними используются API.

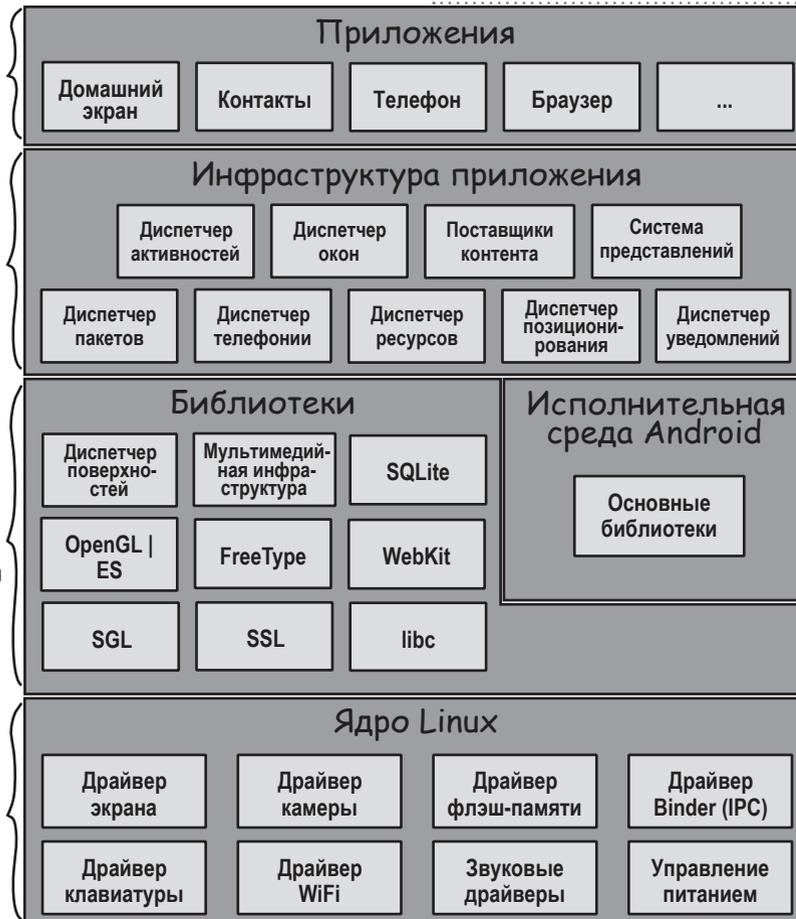
В самом основании системы лежит ядро Linux. В Android оно обеспечивает работу драйверов, а также таких базовых сервисов, как безопасность и управление памятью.



РАССЛАБЬТЕСЬ

Не огорчайтесь, если все это покажется вам слишком сложным.

Сейчас мы всего лишь даем общий обзор компонентов, входящих в платформу Android. Разные компоненты будут подробно описаны далее.



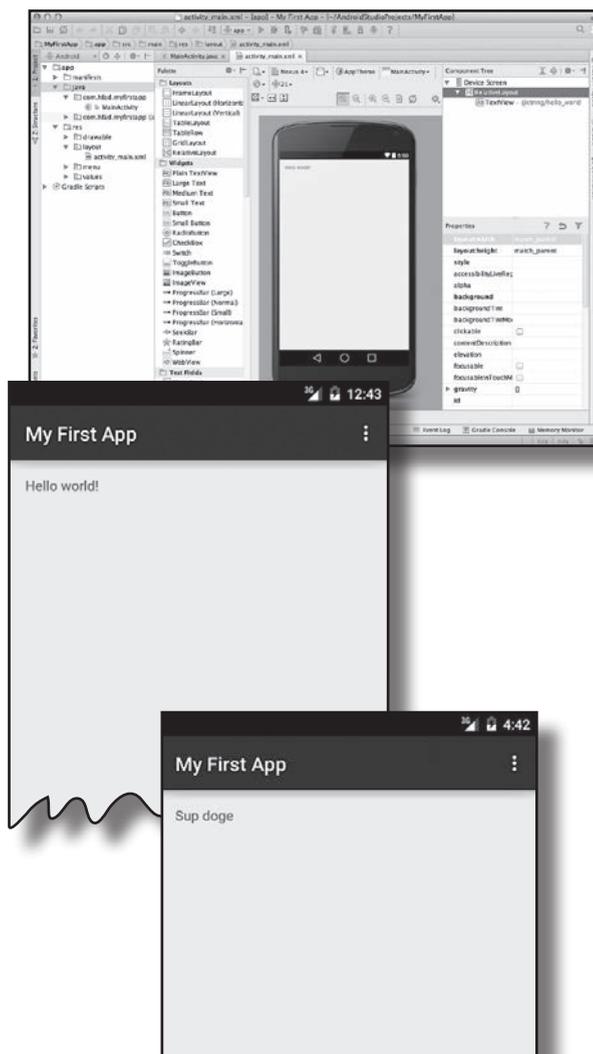
Исполнительная среда Android включает набор базовых библиотек, реализующих большую часть языка программирования Java. Каждое Android-приложение выполняется в отдельном процессе.

К счастью, доступ к замечательным библиотекам Android предоставляется через API в инфраструктуре приложений, и для создания замечательных Android-приложений вы будете использовать именно эти API. Чтобы взяться за работу, вам понадобятся только некоторые знания Java и замысел интересного приложения.

Вот что мы сейчас сделаем

Давайте сходу возьмемся за дело и построим простейшее Android-приложение. Для этого необходимо выполнить лишь несколько действий:

- 1 **Подготовка среды разработки.**
Необходимо установить среду Android Studio, включающую все необходимое для разработки Android-приложений.
- 2 **Построение простейшего приложения.**
Мы создадим в Android Studio простое приложение, которое будет выводить текст на экран.
- 3 **Запуск приложения в эмуляторе Android.**
Мы воспользуемся встроенным эмулятором, чтобы увидеть приложение в действии.
- 4 **Изменение приложения.**
Наконец, мы внесем несколько изменений в приложение, созданное на шаге 2, и снова запустим его.



Часто Задаваемые Вопросы

В: Все Android-приложения пишутся на языке Java?

О: Android-приложения также можно разрабатывать и на других языках, но большинство разработчиков предпочитает Java.

В: Насколько хорошо нужно знать Java для разработки Android-приложений?

О: Необходимо иметь опыт работы с Java SE. Если вы чувствуете, что потеряли форму, мы рекомендуем книгу Кэти Сьерра и Берта Бейтса *Head First Java*.

В: Нужно ли мне знать о Swing и AWT?

О: Даже если у вас нет опыта программирования настольных графических интерфейсов на Java, не беспокойтесь — в Android не используется ни Swing, ни AWT.

Вы находитесь здесь.

первые шаги

Среда разработки

Java — самый популярный язык, используемый для разработки Android-приложений. Устройства на базе Android не запускают файлы `.class` и `.jar`. Вместо этого для повышения скорости и эффективности использования аккумуляторов Android-устройства используют собственные оптимизированные форматы компилированного кода. Это означает, что вы не сможете воспользоваться обычной средой разработки на языке Java — вам также понадобятся специальные инструменты для преобразования откомпилированного кода в формат Android, установки его на Android-устройствах и отладки приложения, когда оно заработает. Все необходимое содержится в **Android SDK**. Посмотрим, что в него входит.

□	Подготовка среды
□	Построение приложения
□	Запуск приложения
□	Изменение приложения

Android SDK

Пакет Android Software Development Kit (SDK) содержит библиотеки и инструменты, необходимые для разработки Android-приложений:



Android Studio — специализированная версия IntelliJ IDEA

IntelliJ IDEA — одна из самых популярных интегрированных сред разработки (IDE) для программирования на Java. Android Studio — версия IDEA, которая включает версию Android SDK и дополнительные инструменты графических интерфейсов, упрощающие разработку приложений.

Кроме редактора и доступа к инструментам и библиотекам из Android SDK, Android Studio предоставляет шаблоны, упрощающие создание новых приложений и классов, а также средства для выполнения таких операций, как упаковка приложений и их запуск.

Установите Java

Android Studio — среда разработки на языке Java, поэтому на вашем компьютере должна быть установлена правильная версия Java. Сначала проверьте системные требования Android Studio и определите, какие версии Java Development Kit (JDK) и Java Runtime Edition (JRE) вам понадобятся. Системные требования можно просмотреть здесь:

<http://developer.android.com/sdk/index.html#Requirements>

Когда вы будете знать, какие версии JDK и JRE вам понадобятся, загрузите и установите их отсюда:

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

Затем установите Android Studio

Когда Java успешно заработает, загрузите Android Studio отсюда:

<https://developer.android.com/sdk/installing/index.html?pkg=studio>

На этой странице также размещены инструкции по установке. Выполните их, чтобы установить Android Studio на вашем компьютере. Когда установка будет завершена, откройте Android Studio и выполните инструкции по добавлению новейших инструментов SDK и библиотек поддержки. Когда все будет сделано, на экране появляется заставка Android Studio. Все готово для построения вашего первого Android-приложения.

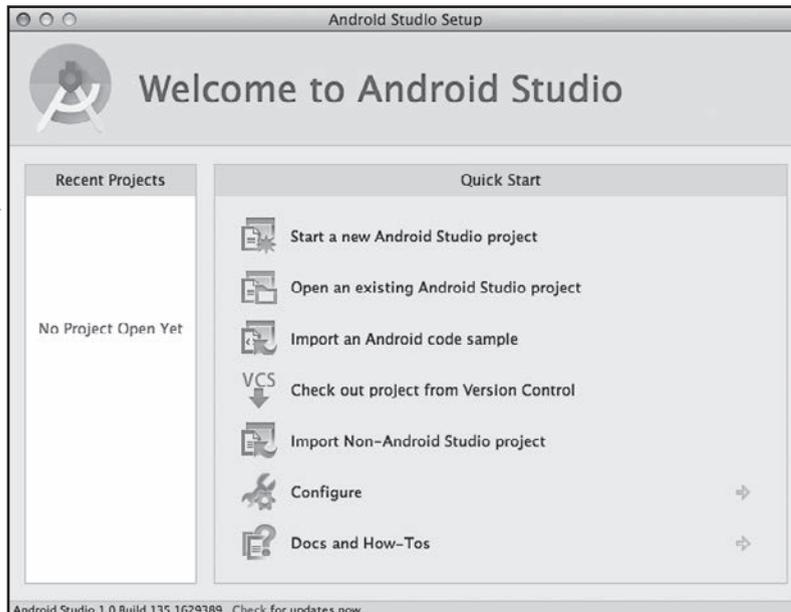
- Подготовка среды
- Построение приложения
- Запуск приложения
- Изменение приложения

Oracle и Google иногда меняют свои URL-адреса. Если эти не работают, выполните поиск.

Если этот URL-адрес изменится, поищите Android Studio на сайте developer.android.com.

Мы не приводим инструкции по установке, потому что они довольно быстро устаревают. Следуйте инструкциям из электронной документации, и все будет нормально.

Заставка Android Studio включает список возможных действий.



Часто задаваемые вопросы

В: Вы говорите, что мы будем использовать Android Studio для построения Android-приложений. А это обязательно?

О: Строго говоря, Android-приложения можно строить и без Android Studio. Все, что для этого необходимо — это инструменты для написания и компиляции Java-кода, а также некоторые специализированные инструменты для преобразования откомпилированного кода в форму, которая может запускаться на Android-устройствах.

В: Значит, я могу использовать другую среду разработки?

О: Android Studio — официальная среда разработки Android, но среда Eclipse также весьма популярна. Дополнительную информацию можно найти здесь: <https://developer.android.com/tools/sdk/eclipse-adt.html>.

В: Возможно ли создавать Android-приложения без IDE?

О: Возможно, но это увеличит объем работы. Большинство Android-приложений в настоящее время строятся с использованием программы сборки *gradle*. Проекты *gradle* можно создавать и строить с использованием текстового редактора и командной строки.

В: Программы сборки? Выходит, *gradle* — что-то вроде ANT?

О: Они похожи, но *gradle* обладает намного большими возможностями, чем ANT. *Gradle* может компилировать и устанавливать код, как и ANT, но при этом также использует Maven для загрузки всех сторонних библиотек, необходимых для работы вашего кода. *Gradle* также использует язык сценариев

Groovy, а это означает, что с использованием *gradle* легко создаются достаточно сложные сценарии построения.

В: Большинство приложений строится с использованием *gradle*? Но вы же говорили, что многие разработчики используют Android Studio?

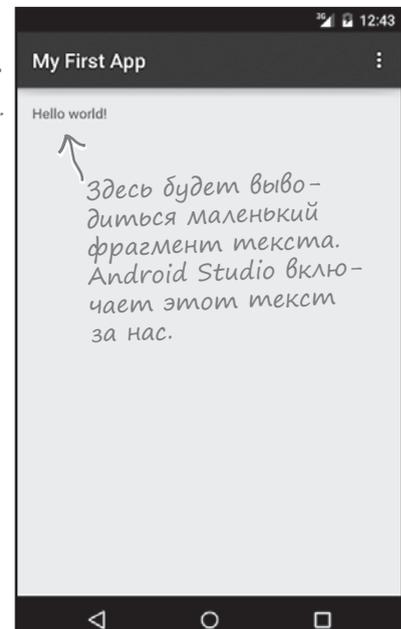
О: Android Studio предоставляет графический интерфейс для работы с *gradle*, а также с другими инструментами построения макетов, чтения журнальных данных и отладки.

Построение простого приложения

Итак, среда разработки подготовлена, и можно переходить к созданию вашего первого Android-приложения. Вот как оно будет выглядеть:

Имя приложения.

Приложение очень простое, но для самого первого Android-приложения этого вполне достаточно.



Этот шаг завершен,
мы его вычеркиваем.



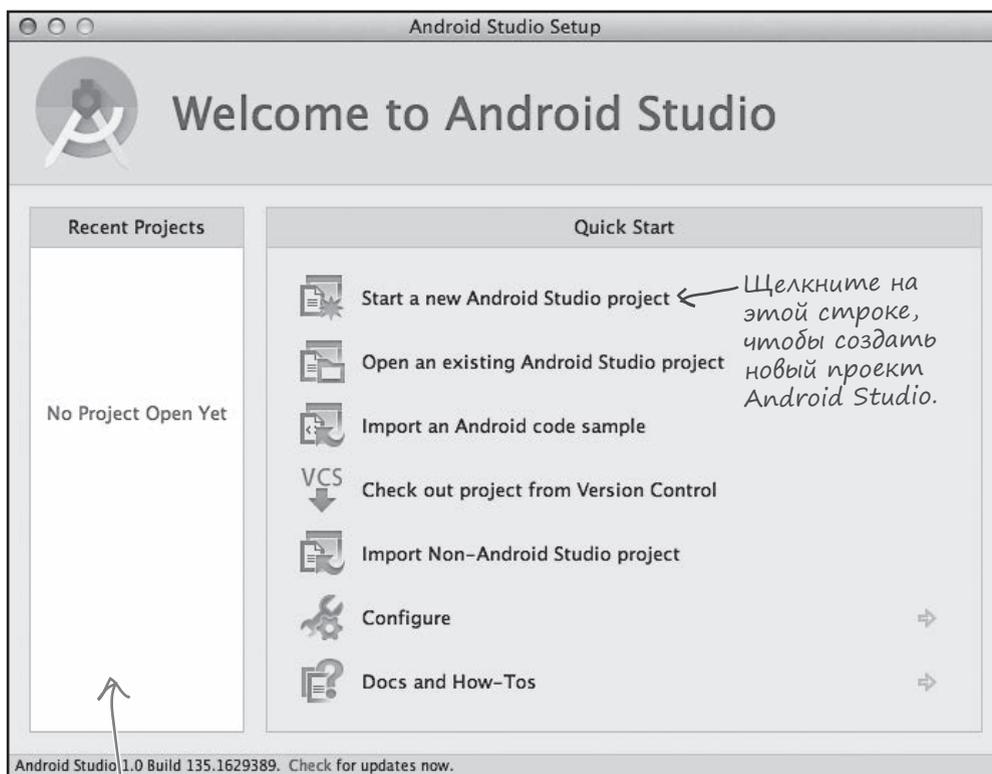
- Подготовка среды
- Построение приложения
- Запуск приложения
- Изменение приложения

Давайте построим простое приложение

Каждый раз, когда вы создаете новое приложение, для него необходимо создать новый проект. Убедитесь в том, что среда Android Studio открыта, и повторяйте за нами.

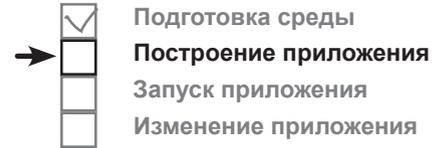
1. Создание нового проекта

На заставке Android Studio перечислены некоторые возможные операции. Мы хотим создать новый проект; щелкните на строке “Start a new Android Studio project”.



Здесь будет выводиться список всех созданных вами проектов. Это наш первый проект, поэтому эта область пока пуста.

Построение простого приложения (продолжение)



2. Настройка проекта

Теперь необходимо настроить конфигурацию приложения: указать, как оно будет называться, какой домен компании будет использоваться и где должны храниться его файлы. Android Studio использует домен компании и имя приложения для формирования имени пакета, которое будет использоваться вашим приложением. Например, если присвоить приложению имя “My First App” и указать домен компании “hfad.com”, то Android Studio сформирует имя пакета `com.hfad.myfirstapp`. Имя пакета играет очень важную роль в Android, потому что оно используется Android-устройствами для *однозначной идентификации приложения*.

Введите имя приложения “My First App”, домен компании “hfad.com”, и подтвердите местоположение по умолчанию. Щелкните на кнопке Next.



Будьте осторожны!

Имя пакета должно оставаться неизменным

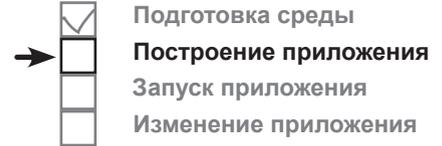
на протяжении всего срока жизни приложения.

Это уникальный идентификатор вашего приложения, который используется для управления версиями программы.

Мастер строит имя пакета из имени приложения и домена компании.

Create New Project
 New Project
 Android Studio
 Configure your new project
 Имя приложения отображается в Google Play Store, а также в других местах.
 Application name: My First App
 Company Domain: hfad.com
 Package name: com.hfad.myfirstapp Edit
 Project location: /Users/dawng/AndroidStudioProjects/MyFirstApp ...
 В этой папке будут храниться все файлы вашего приложения.
 Cancel Previous Next Finish

Построение простого приложения (продолжение)



3. Выбор уровня API

Теперь необходимо указать, какие уровни API системы Android будут использоваться вашим приложением. Уровни API увеличиваются с выходом каждой очередной версии Android. Если только вы не хотите, чтобы приложение работало только на самых новых устройствах, стоит выбрать один из более старых уровней API. Здесь мы выбираем API уровня 15; это означает, что приложение сможет работать на большинстве устройств. Кроме того, наша версия приложения создается только для телефонов и планшетов, поэтому флажки остальных вариантов так и остаются снятыми. Когда это будет сделано, щелкните на кнопке Next.

← Дополнительная информация о разных уровнях API приведена на следующей странице.

Минимальный уровень SDK — наименьшая версия, которая будет поддерживаться вашим приложением. Приложение будет работать на устройствах с API этого уровня и выше. На устройствах с API более низкого уровня оно работать не будет.



Версии Android под увеличительным стеклом



Вероятно, вам не раз доводилось слышать, как применительно к Android упоминаются разные «вкусные» названия: Ice Cream Sandwich (сэндвич с мороженым), Jelly Bean (мармеладная конфета), KitKat и Lollipop (леденец). Что это за кондитерская?

Каждой версии Android присваивается номер и кодовое имя. Номер версии определяет конкретную версию Android (например, 5.0), тогда как кодовое имя представляет собой чуть более общее «дружественное» имя, которое может объединять сразу несколько версий Android. Под «уровнем API» понимается версия API, используемых приложением. Например, Android версии 5.0 соответствует уровень API 21.

Версия	Кодовое имя	Уровень API
1.0		1
1.1		2
1.5	Cupcake	3
1.6	Donut	4
2.0	Eclair	5
2.0.1	Eclair	6
2.1	Eclair	7
2.2.x	Froyo	8
2.3 — 2.3.2	Gingerbread	9
2.3.2 — 2.3.7	Gingerbread	10
3.0	Honeycomb	11
3.1	Honeycomb	12
3.2	Honeycomb	13
4.0 — 4.0.2	Ice Cream Sandwich	14
4.0.3-4.0.4	Ice Cream Sandwich	15
4.1	Jelly Bean	16
4.2	Jelly Bean	17
4.3	Jelly Bean	18
4.4	KitKat	19
4.4	KitKat (with wearable extensions)	20
5.0	Lollipop	21

Сейчас эти версии уже не встречаются.

На большинстве устройств используется один из этих уровней API.

При разработке Android-приложений необходимо учитывать, с какими версиями Android должно быть совместимо ваше приложение. Если вы укажете, что приложение совместимо только с самой последней версией SDK, может оказаться, что оно не запускается на очень многих устройствах. Информацию о процентном распределении версий по устройствам можно найти здесь: <https://developer.android.com/about/dashboards/index.html>.

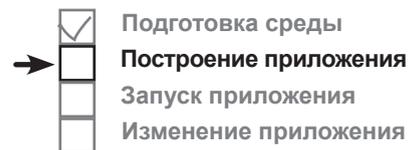
Активности и макеты: с высоты птичьего полета

Далее вам будет предложено добавить активность в ваш проект. Каждое Android-приложение состоит из экранов, а каждый экран состоит из активности и макета.

Активность — одна четко определенная операция, которую может выполнить пользователь. Например, в приложении могут присутствовать активности для составления сообщения электронной почты, поиска контакта или создания снимка. Активности обычно ассоциируются с одним экраном и программируются на Java.

Макет описывает внешний вид экрана. Макеты создаются в виде файлов в разметке XML и сообщают Android, где располагаются те или иные элементы экрана.

Рассмотрим подробнее, как взаимодействуют активности и макеты.



Макеты определяют способ представления пользовательского интерфейса.

Активности определяют действия.

1 Устройство запускает приложение и создает объект активности.

2 Объект активности задает макет.

3 Активность приказывает Android вывести макет на экран.

4 Пользователь взаимодействует с макетом, отображаемым на устройстве.

5 Активность реагирует на эти взаимодействия, выполняя код приложения.

6 Активность обновляет содержимое экрана...

7 ...и пользователь видит это на устройстве.



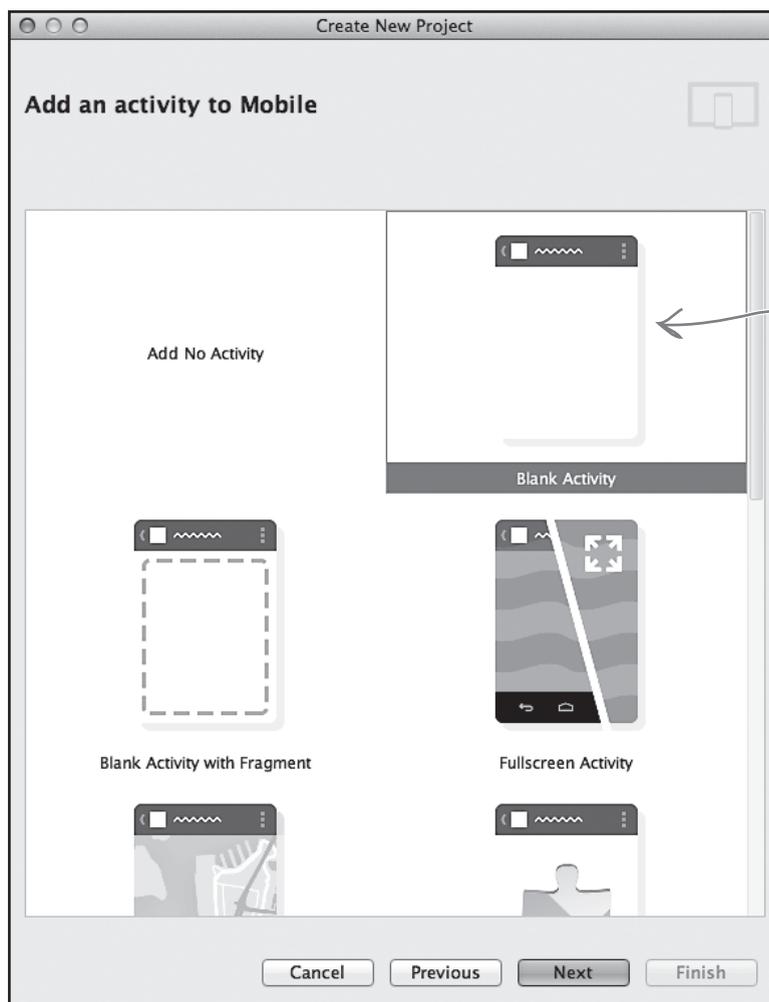
Теперь, когда вы чуть больше знаете о том, что собой представляют активности и макеты, мы пройдем два последних шага мастера и прикажем ему создать простейшую активность и макет.

Построение простого приложения (продолжение)

4. Создание активности

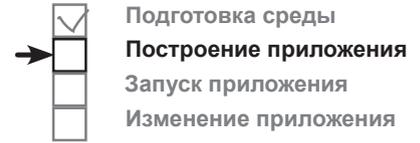
На следующем экране представлен набор шаблонов, которые могут использоваться для создания активности и макета. Вы должны выбрать один из них. Так как в нашем приложении будут использоваться простейшая активность и макет, выберите вариант Blank Activity и щелкните на кнопке Next.

- Подготовка среды
- Построение приложения**
- Запуск приложения
- Изменение приложения



Есть и другие типы активностей, которые можно выбрать в этом окне, но на этот раз следует выбрать вариант Blank Activity.

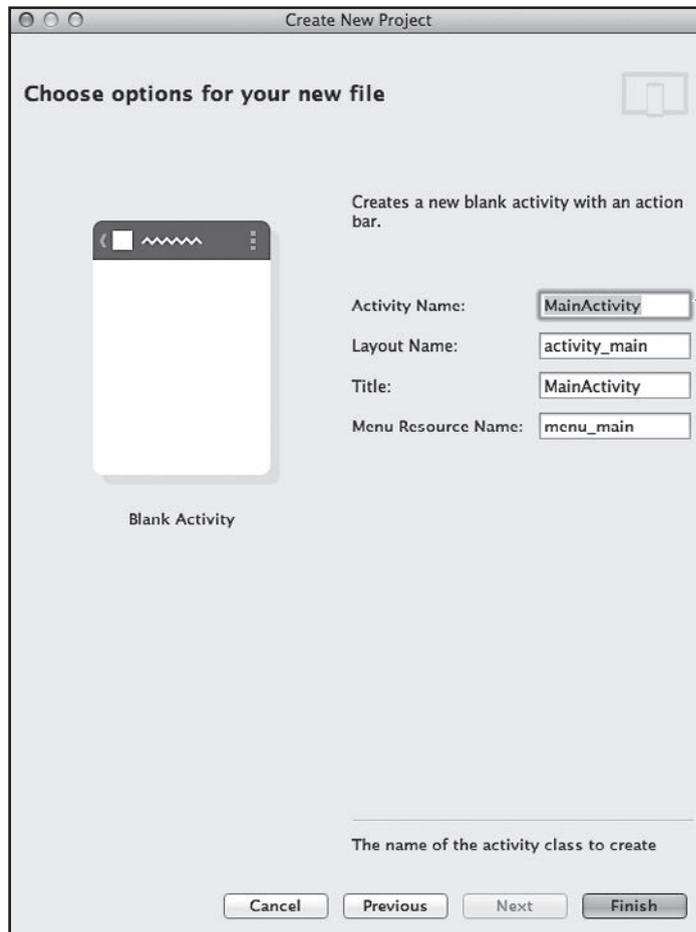
Построение простого приложения (продолжение)



5. Настройка активности

Теперь среда разработки предложит выбрать имена для активности и макета экрана. Вам также придется указать текст заголовка экрана и имя ресурса меню. Введите имя активности “MainActivity” и имя макета “activity_main”. Активность представляет собой класс Java, а макет — файл с разметкой XML, поэтому для введенных нами имен будет создан файл класса Java с именем *MainActivity.java* и файл XML с именем *activity_main.xml*.

Когда вы щелкнете на кнопке Finish, Android Studio построит приложение.



Присвойте активности имя “MainActivity”, а макету — имя “activity_main”. Для остальных параметров подтвердите значения по умолчанию.

Только что вы создали свое первое Android-приложение

- Подготовка среды
- Построение приложения
- Запуск приложения
- Изменение приложения

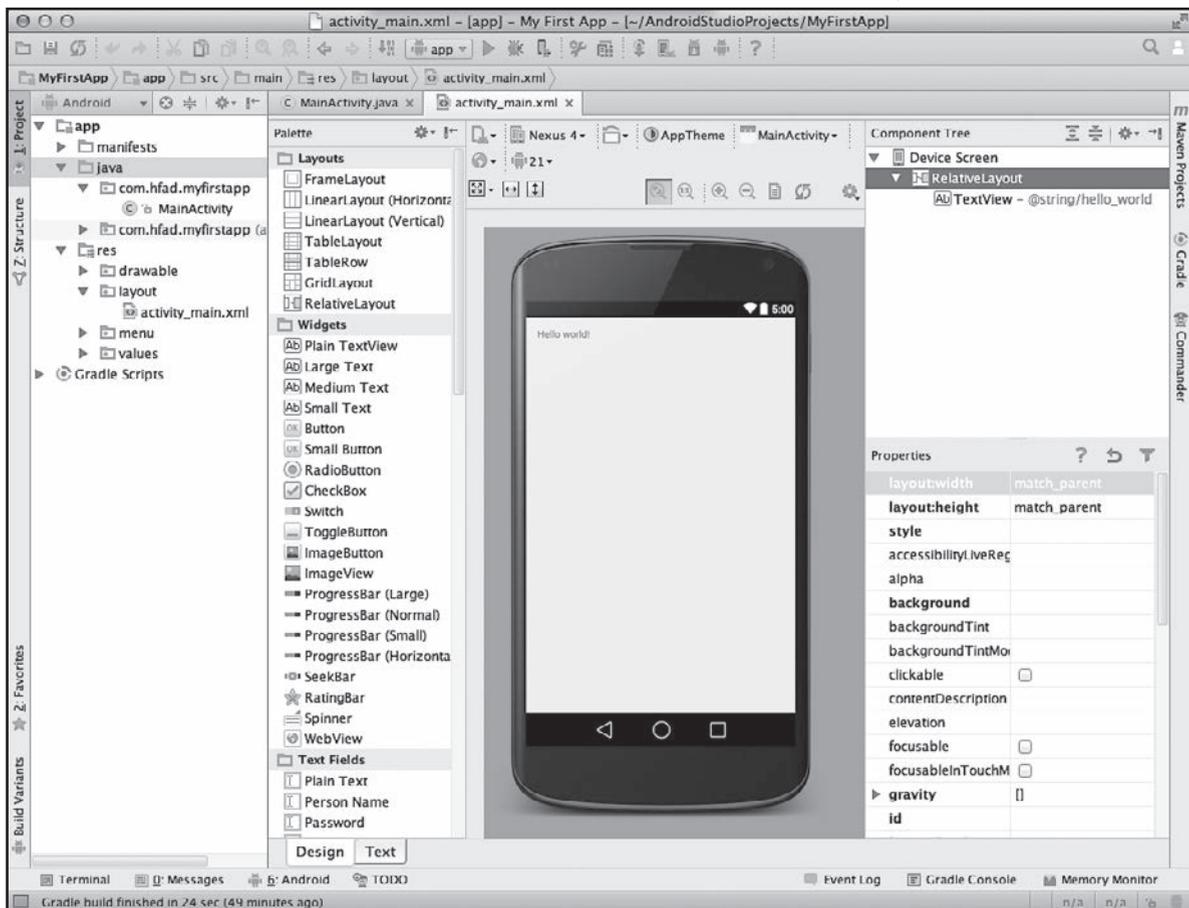
Итак, что же произошло?

- ★ **Мастер Android Studio создал для вашего приложения проект, параметры которого были настроены в соответствии с вашими указаниями.**
Вы определили, с какими версиями Android должно быть совместимо приложение, а мастер создал все файлы и папки, необходимые для простейшего работоспособного приложения.
- ★ **Мастер создал базовую активность и базовый макет с шаблонным кодом.**
Шаблонный код включает разметку XML для макета и код Java для активности; он выводит текст "Hello world!" в макете. Вы можете изменить этот код.

Когда вы завершите создание проекта, пройдя все шаги работы с мастером, Android Studio автоматически отобразит проект в среде разработки.

Вот как выглядит наш проект на текущий момент (не беспокойтесь, если сейчас все выглядит слишком сложно — на нескольких ближайших страницах мы все объясним):

Так выглядит проект в Android Studio.



Android Studio создает всю структуру папок за вас

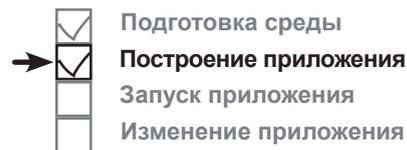
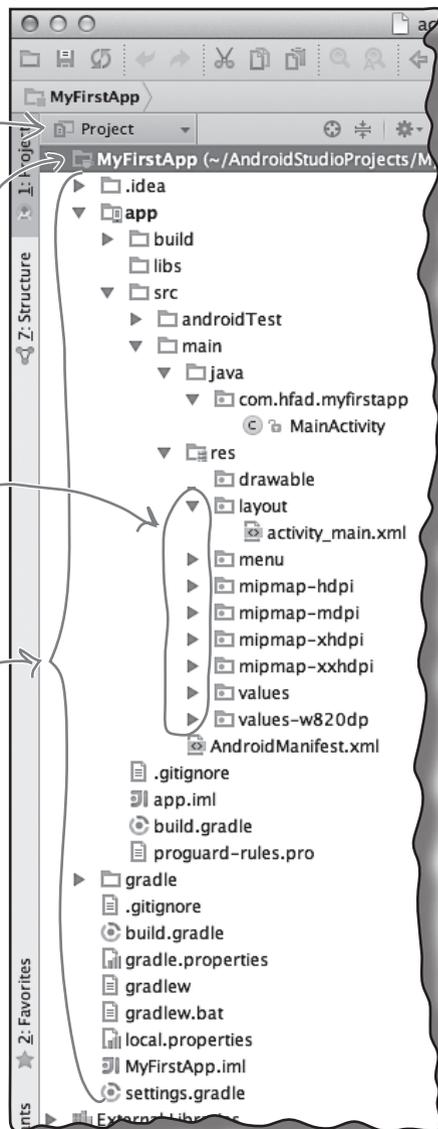
Android-приложение в действительности представляет собой набор файлов, размещенных в четко определенной структуре папок; Android Studio создает все эти папки за вас при создании нового приложения. Если вас интересует, как выглядит эта структура папок, проще всего посмотреть ее на панели у левого края окна Android Studio. На ней отображаются все проекты, открытые в настоящее время. Чтобы свернуть или развернуть содержимое папки, щелкните на стрелке слева от значка папки.

Выберите здесь проект, чтобы увидеть входящие в него файлы и папки.

Здесь выводится имя проекта.

Щелкайте на стрелках, чтобы свернуть или развернуть содержимое папок.

Все эти файлы и папки входят в ваш проект.



В структуре папок присутствуют файлы разных типов

Посмотрев структуру папок, вы увидите, что мастер создал за вас папки и файлы разных типов:

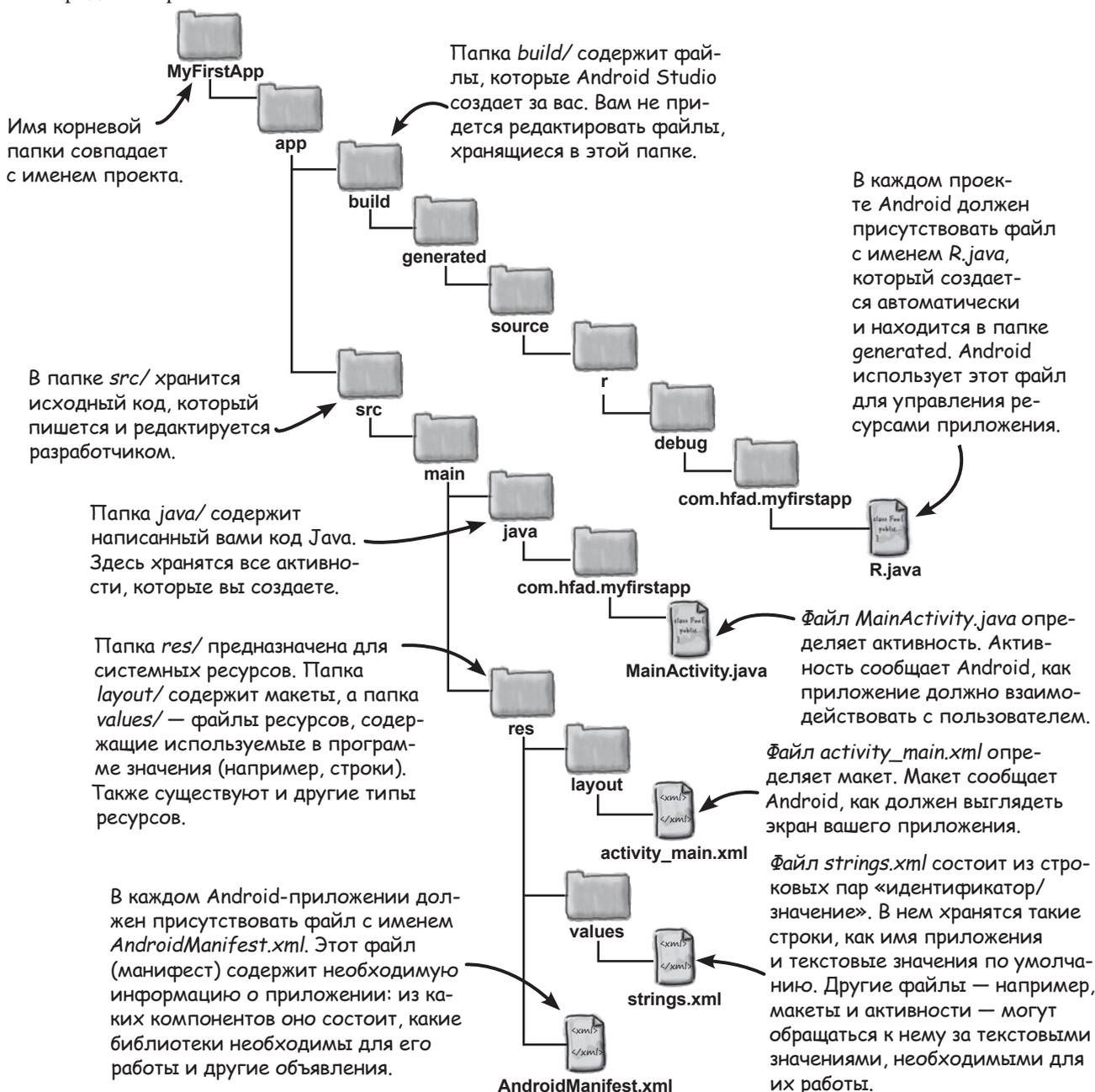
- ★ **Исходные файлы Java и XML**
Файлы активности и макета, которые были созданы за вас мастером.
- ★ **Файлы Java, сгенерированные Android**
Дополнительные файлы Java, которые Android Studio тоже генерирует автоматически. Вносить в них изменения вам не придется.
- ★ **Файлы ресурсов**
К этой категории относятся файлы изображений на значках по умолчанию, стили, которые могут использоваться вашим приложением, и все общие строковые данные, к которым может обращаться приложение.
- ★ **Библиотеки Android**
В окне мастера вы указали минимальную версию SDK, с которой должно быть совместимо ваше приложение. Android Studio включает в приложение библиотеки Android, актуальные для этой версии.
- ★ **Файлы конфигурации**
Файлы конфигурации сообщают Android, что содержит приложение и как его следует выполнять.

Давайте внимательнее присмотримся к некоторым ключевым файлам и папкам в мире приложений Android.

Полезные файлы в проекте

Проекты Android Studio используют систему сборки gradle для компиляции и развертывания приложений. Проекты gradle имеют стандартную структуру. Некоторые ключевые файлы и папки, с которыми вам предстоит работать:

- Подготовка среды
- Построение приложения
- Запуск приложения
- Изменение приложения



- Подготовка среды
- Построение приложения
- Запуск приложения
- Изменение приложения

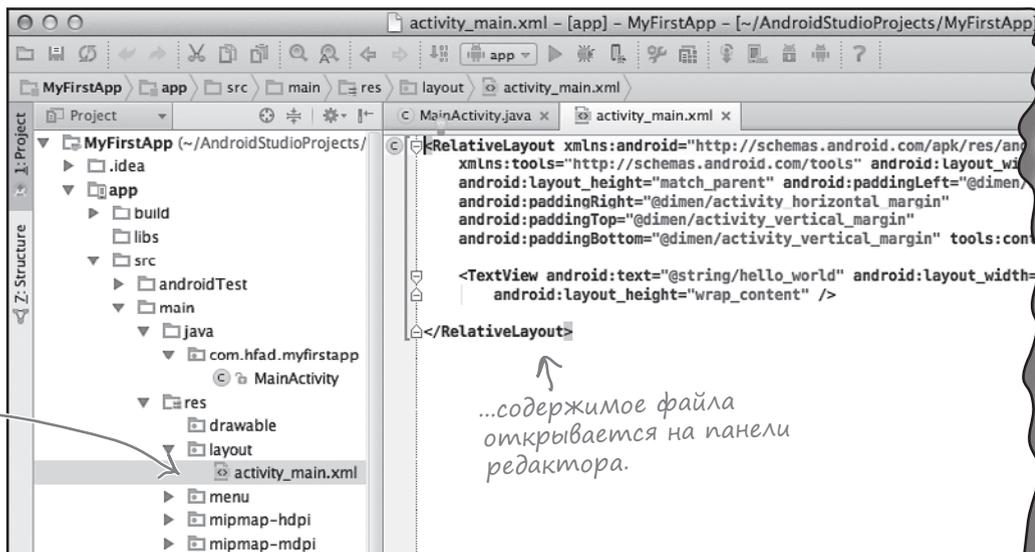
Редактирование кода в Android Studio

Для просмотра и изменения файлов используются различные редакторы Android Studio. Сделайте двойной щелчок на файле, с которым вы хотите работать; его содержимое появляется в середине окна Android Studio.

Редактор кода

Большинство файлов отображается в редакторе кода. По сути это обычный текстовый редактор, но с поддержкой таких дополнительных возможностей, как цветное выделение синтаксиса и проверка кода.

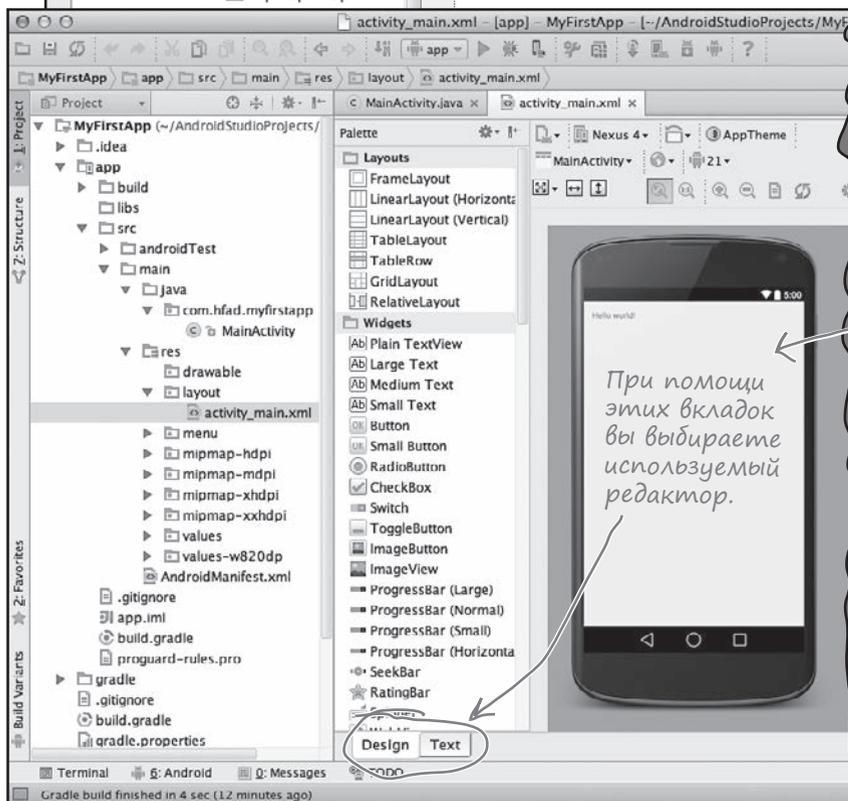
Сделайте двойной щелчок на файле...



...содержимое файла открывается на панели редактора.

Визуальный редактор

При редактировании макета появляется дополнительная возможность: вместо редактирования разметки XML можно использовать визуальный редактор. Визуальный редактор позволяет перетащить компоненты графического интерфейса на макет и расположить их так, как вы считаете нужным. Редактор кода и визуальный редактор обеспечивают разные представления одного файла, и вы можете переключаться между ними по своему усмотрению.



При помощи этих вкладок вы выбираете используемый редактор.

Это разные представления одного файла. В одном представлении отображается код, а в другом — визуальный результат.

КТО И ЧТО ДЕЛАЕТ?

Ниже приведен фрагмент кода из файла макета, сгенерированного Android Studio. Да, мы знаем, что вы еще ни разу не видели код макета, и все же попробуйте соединить каждое из описаний в нижней части страницы с правильной строкой кода. Мы уже провели одну линию, чтобы вам было проще взяться за дело.

activity_main.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    tools:context=".MainActivity">

    <TextView
        android:text="@string/hello_world"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</RelativeLayout>
```

Добавить отступы у краев экрана.

Добавить графический компонент `TextView` (надпись) для вывода текста.

Включить перенос текста по горизонтали и вертикали.

Вывести значение ресурсной строки с именем `hello_world`.

Назначить высоту и ширину макета по размерам экрана устройства.

КТО И ЧТО ДЕЛАЕТ?

РЕШЕНИЕ

Ниже приведен фрагмент кода из файла макета, сгенерированного Android Studio. Да, мы знаем, что вы еще ни разу не видели код макета, и все же попробуйте соединить каждое из описаний в нижней части страницы с правильной строкой кода. Мы уже провели одну линию, чтобы вам было проще взяться за дело.

activity_main.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    tools:context=".MainActivity">

    <TextView
        android:text="@string/hello_world"
        {
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />

</RelativeLayout>
```

Добавить отступы у краев экрана.

Добавить графический компонент `TextView` (надпись) для вывода текста.

Включить перенос текста по горизонтали и вертикали.

Вывести значение ресурсной строки с именем `hello_world`.

Назначить высоту и ширину макета по размерам экрана устройства.

КТО И ЧТО ДЕЛАЕТ?

А теперь посмотрим, удастся ли вам сделать то же с кодом активности. Это условный код, а не тот код, который Android Studio генерирует за вас. Соедините каждое описание с правильной строкой кода.

MainActivity.java

```
package com.hfad.myfirstapp;

import android.os.Bundle;
import android.app.Activity;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Имя пакета.

Классы Android, используемые в MainActivity.

Указывает, какой макет должен использоваться.

Реализация метода onCreate () из класса Activity. Этот метод вызывается при первом создании активности.

MainActivity расширяет класс Android android.app.Activity.

КТО И ЧТО ДЕЛАЕТ? РЕШЕНИЕ

А теперь посмотрим, удастся ли вам сделать то же с кодом активности. Это условный код, а не тот код, который Android Studio генерирует за вас. Соедините каждое описание с правильной строкой кода.

MainActivity.java

```
package com.hfad.myfirstapp;

import android.os.Bundle;
import android.app.Activity;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Имя пакета.

Классы Android, используемые в MainActivity.

Указывает, какой макет должен использоваться.

Реализация метода onCreate () из класса Activity. Этот метод вызывается при первом создании активности.

MainActivity расширяет класс Android android.app.Activity.

Запуск приложения в эмуляторе Android

Итак, вы увидели, как Android-приложение выглядит в Android Studio, и в общих чертах представили, как работает система в целом. Но *на самом деле* вам хочется увидеть, как работает приложение, верно?

В том, что касается запуска приложений, есть пара вариантов. Вариант первый — запустить приложение на физическом устройстве. Но что, если у вас нет такого устройства под рукой? Или вы хотите узнать, как оно будет выглядеть на устройстве другого типа, которого у вас вообще нет?

Альтернативное решение — воспользоваться **эмулятором Android**, встроенным в Android SDK. Эмулятор позволяет создать одно или несколько **виртуальных устройств Android** (Android Virtual Device, AVD) и запустить приложение в эмуляторе *так, словно оно выполняется на физическом устройстве*.

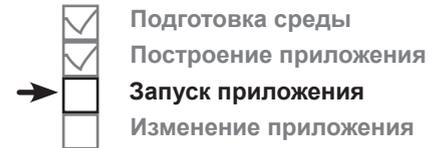
Как же выглядит эмулятор?

Перед вами AVD в эмуляторе Android. Оно выглядит как телефон, работающий на вашем компьютере.

Эмулятор представляет собой приложение, точно воссоздающее аппаратное окружение устройства Android: от центрального процессора и памяти до звуковых микросхем и экрана. Эмулятор построен на базе существующего эмулятора QEMU, похожего на другие виртуальные машины, с которыми вам, возможно, доводилось работать — такие, как VirtualBox или VMWare.

Внешний вид и поведение AVD зависят от заданных вами параметров. На иллюстрации AVD эмулирует Nexus 4, поэтому эмулятор выглядит и ведет себя так, словно на вашем компьютере имеется внутреннее устройство Nexus 4.

Давайте создадим AVD, чтобы вы могли увидеть свое приложение, выполняемое в эмуляторе.



Эмулятор Android позволяет запустить приложение на виртуальном устройстве Android (AVD). AVD ведет себя практически так же, как и физическое Android-устройство. Вы можете создать сразу несколько AVD для разных типов устройств.



После создания AVD вы увидите свое приложение, выполняемое на нем. Android Studio запускает эмулятор автоматически.

Как и физический телефон, AVD необходимо разблокировать перед началом использования. Просто щелкните на значке с замком и перетащите вверх.

Создание виртуального устройства Android

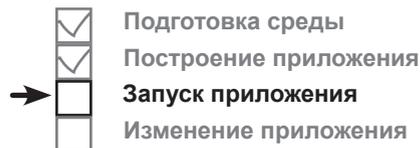
Создание AVD в Android Studio состоит из нескольких шагов. Мы создадим AVD для Nexus 4 с уровнем API 21, чтобы вы могли видеть, как ваше приложение выглядит и ведет себя на устройствах этого типа. Последовательность действий остается более или менее постоянной для любого типа устройств.

Откройте AVD Manager

В диспетчере AVD Manager вы сможете создавать новые AVD, а также просматривать и редактировать уже созданные виртуальные устройства. Чтобы запустить его, выберите в меню Tools пункт Android и выберите AVD Manager.

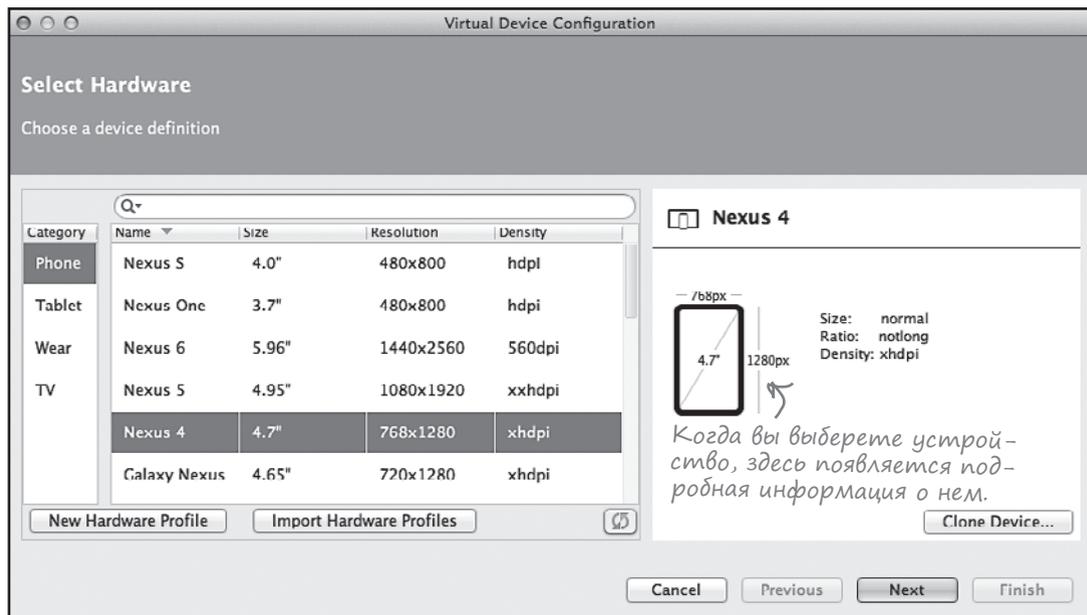
Если вы еще не создали ни одного виртуального устройства, открывается окно с предложением создать его. Щелкните на кнопке “Create a virtual device”.

Щелкните на кнопке “Create a virtual device”, чтобы создать AVD.

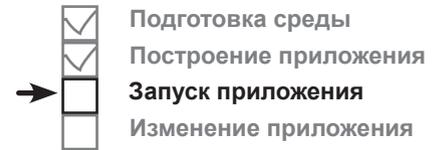


Выберите тип устройства

На следующем экране вам будет предложено выбрать определение устройства — то есть тип устройства, который будет эмулировать AVD. Давайте посмотрим, как будет выглядеть наше приложение на телефоне Nexus 4. Выберите в меню Category пункт Phone, затем выберите в списке Nexus 4. Щелкните на кнопке Next.



Создание виртуального устройства Android (продолжение)

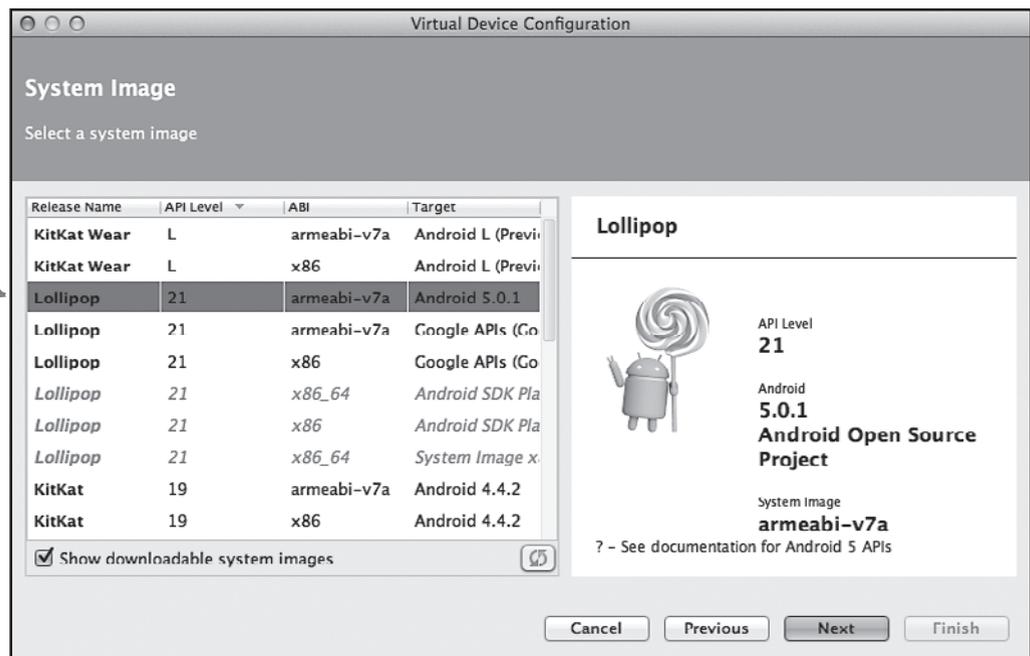


Выберите образ системы

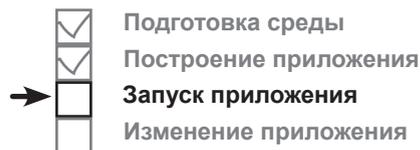
Далее следует выбрать образ системы, то есть установленную версию операционной системы Android. Вы можете выбрать версию Android, которая должна поддерживаться AVD, и тип процессора (ARM или x86).

Вы должны выбрать образ системы для уровня API, совместимого с создаваемым приложением. Например, если вы хотите, чтобы приложение работало на минимальном уровне API 15, выберите образ системы с уровнем API *не менее* 15. Мы будем использовать образ системы для уровня API 21. Выберите строку Lollipop/21/armeabi-v7a с целевой системой Android 5.0.1. Щелкните на кнопке Next.

Если этот образ системы не установлен на вашем компьютере, вам будет предоставлена возможность загрузить его.



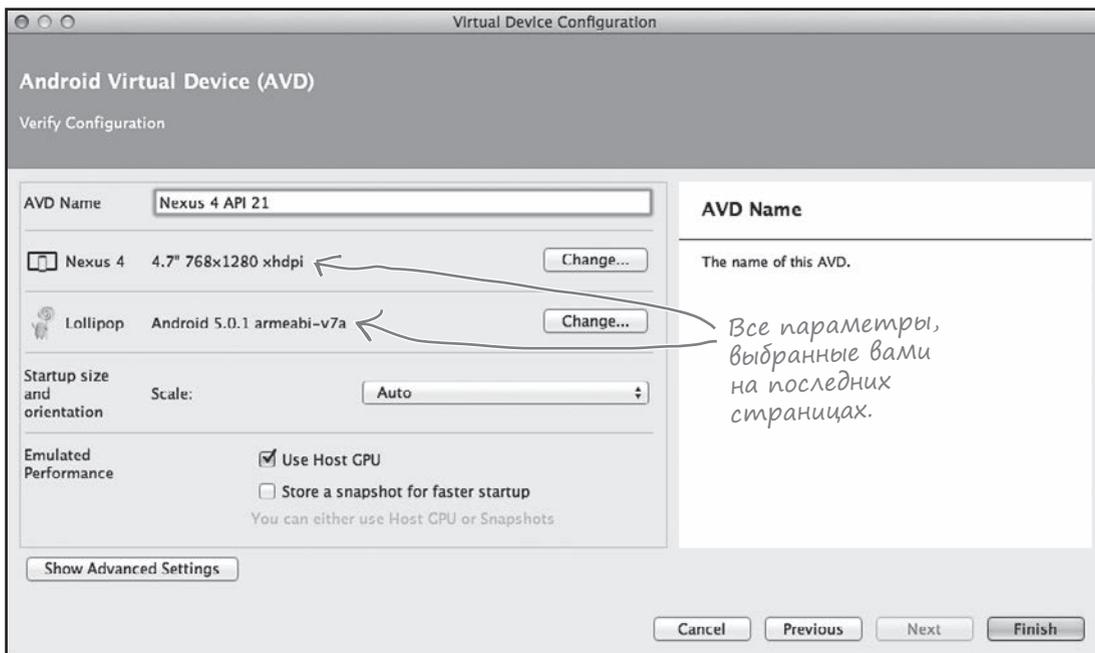
Процесс создания AVD будет продолжен на следующей странице.



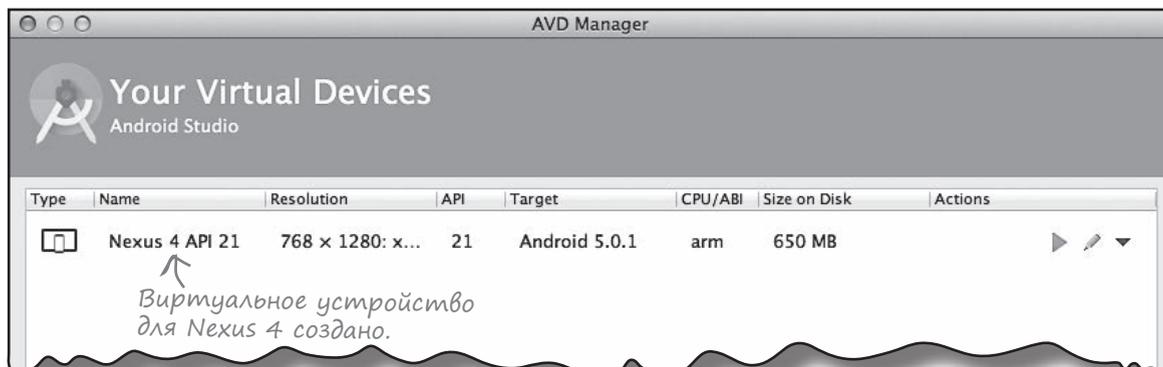
Создание виртуального устройства Android (продолжение)

Проверка конфигурации AVD

На следующем экране вам будет предложено подтвердить конфигурацию AVD. На нем приведена сводка параметров, выбранных вами на нескольких последних экранах, а также предоставляется возможность изменить их. Подтвердите значения и щелкните на кнопке Finish.



AVD Manager создает AVD и, когда виртуальное устройство будет создано, отображает его в списке устройств. Теперь AVD Manager можно закрыть.



Запуск приложения в эмуляторе

Теперь, когда вы создали виртуальное устройство, используйте его для запуска приложения. Для этого выберите команду “Run ‘app’” из меню Run. Когда вам будет предложено выбрать устройство, убедитесь в том, что установлен переключатель “Launch emulator” с только что созданным вами виртуальным устройством Nexus 4 AVD. Щелкните на кнопке ОК.

Пока вы терпеливо ожидаете появления AVD, посмотрим, что происходит при выполнении команды Run.

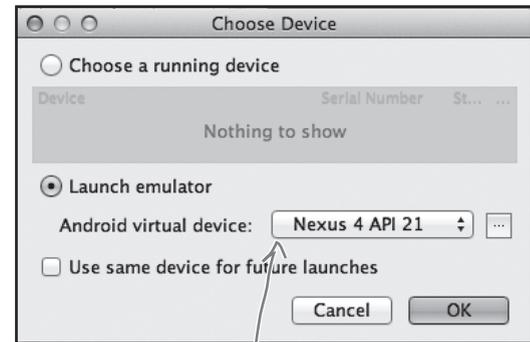
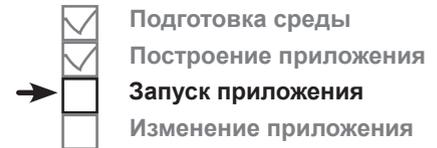
Компиляция, упаковка, развертывание и запуск

Команда Run не просто запускает приложение. Она также выполняет все подготовительные операции, необходимые для запуска приложения:



- 1 **Файлы с исходным кодом Java компилируются в байт-код.**
- 2 **Создается пакет Android-приложения, или файл APK.**
Файл APK включает откомпилированные файлы Java, а также все библиотеки и ресурсы, необходимые для работы приложения.
- 3 **Если эмулятор не выполняется в настоящий момент, он запускается с AVD.**

- 4 **Когда эмулятор будет запущен, а AVD активизируется, файл APK передается на AVD и устанавливается.**
- 5 **AVD запускает главную активность, связанную с приложением.**
Ваше приложение отображается на экране AVD. Теперь ничто не мешает тому, чтобы проверить его в работе.

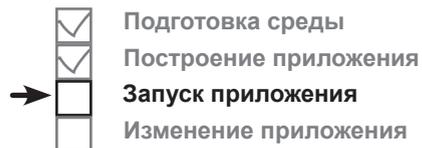


Виртуальное устройство Android, которое мы только что создали.

Файл APK — файл пакета приложения Android. По сути это архив JAR или ZIP с приложением Android.

будьте терпеливы

Информация о ходе запуска отображается на консоли



Запуск эмулятора с AVD может занимать немало времени — обычно *несколько минут*. К счастью, за ходом операции можно проследить на консоли Android Studio. На консоли выводится подробный отчет о том, что делает система сборки gradle, а если в процессе запуска возникнут какие-либо ошибки — они будут выделены в тексте.

Консоль располагается в нижней части экрана Android Studio:

Пока эмулятор запускается, вы можете заняться чем-нибудь полезным: скажем, вышиванием или приготовлением обеда.



Вот что выводится в окне консоли при запуске приложения:

Android Studio запускает эмулятор с AVD Nexus4, только что созданным нами виртуальным устройством Android.

```
Waiting for device.
/Applications/ad-bundle-mac/sdk/tools/emulator -avd Nexus_4_API_21 -netspeed full -netdelay none
Device connected: emulator-5554
Device Nexus_4_API_21 [emulator-5554] is online, waiting for processes to start up..
Device is ready: Nexus_4_API_21 [emulator-5554]
Target device: Nexus_4_API_21 [emulator-5554]
Uploading file
    local path: /Users/dawng/AndroidStudioProjects/MyFirstApp/app/build/outputs/apk/app-debug.apk
    remote path: /data/local/tmp/com.hfad.myfirstapp
Installing com.hfad.myfirstapp
DEVICE SHELL COMMAND: pm install -r "/data/local/tmp/com.hfad.myfirstapp"
pkg: /data/local/tmp/com.hfad.myfirstapp
Success
Launching application: com.hfad.myfirstapp/com.hfad.myfirstapp.MainActivity.
DEVICE SHELL COMMAND: am start -n "com.hfad.myfirstapp/com.hfad.myfirstapp.MainActivity" -a android.intent.action.MAIN -c android.intent.category.LAUNCHER
Starting: Intent { act=android.intent.action.MAIN cat=[android.intent.category.LAUNCHER] cmp=com.hfad.myfirstapp/.MainActivity }
```

Виртуальное устройство Android запущено и готово к работе.

Загрузка и установка файла APK.

Наконец, наше приложение начинает работу с запуска его главной активности — той самой, которую мастер сгенерировал за вас.



Тест-драйв

Итак, посмотрим, что же на самом деле происходит на экране при запуске приложения.

Сначала эмулятор запускается в отдельном окне. Эмулятор довольно долго загружает AVD, а потом через некоторое время в AVD появляется экран блокировки.

- Подготовка среды
- Построение приложения
- Запуск приложения
- Изменение приложения

Эмулятор запускается...

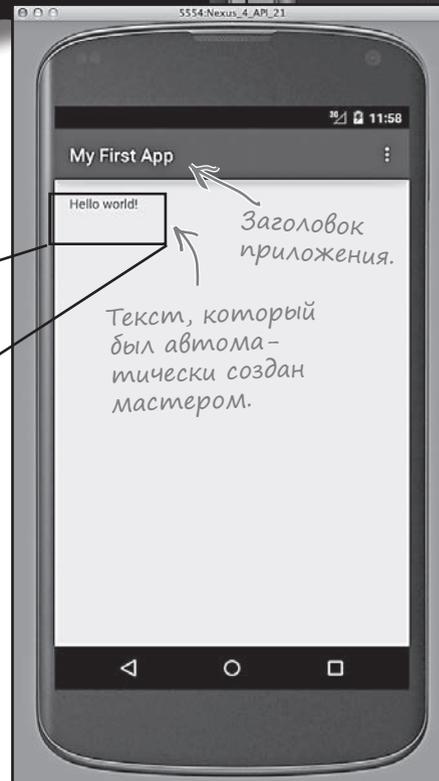


...а это AVD с экраном блокировки. Он выглядит и работает точно так же, как и на реальном устройстве Nexus 4.



Когда вы снимете блокировку с экрана AVD, махнув вверх на изображении замка, появляется только что созданное вами приложение. Имя приложения отображается в верхней части экрана, а текст по умолчанию "Hello world!" выводится в основной области.

Hello world!



Заголовок приложения.

Текст, который был автоматически создан мастером.

Android Studio создает текст "Hello world!" автоматически, без дополнительных приказаний с вашей стороны.

Приложение выполняется в AVD.

Что же только что произошло?

Разобьем то, что происходит при запуске приложения, на несколько этапов:



- Подготовка среды
- Построение приложения
- Запуск приложения
- Изменение приложения

- 1 Android Studio запускает эмулятор, загружает AVD и устанавливает приложение.
- 2 Когда приложение запустится, на базе `MainActivity.java` создается объект активности.
- 3 Активность указывает, что она использует макет `activity_main.xml`.
- 4 Активность приказывает Android вывести макет на экран. Текст “Hello world!” появляется на экране.



Часто задаваемые вопросы

В: Вы упомянули о том, что при создании файла APK исходный код Java компилируется в байт-код и добавляется в APK. Вероятно, имеется в виду, что он компилируется в байт-код Java, не так ли?

О: Да, но это еще не все. В Android все работает несколько иначе.

Принципиальное отличие заключается в том, что в Android ваш код не выполняется в обычной виртуальной машине Java. Он выполняется в исполнительной среде Android (Android Runtime, ART), а на более старых устройствах — в предшественнике ART, который называется Dalvik. Таким образом, вы пишете исходный код Java, компилируете его в файлы `.class` при помощи компилятора Java, после чего файлы `.class` объединяются в один файл в формате DEX, содержащий более компактный и эффективный байт-код. После этого ART выполняет код DEX. Более подробная информация приведена в приложении А.

В: Как все сложно. Почему бы просто не использовать обычную виртуальную машину Java?

О: ART может преобразовать байт-код DEX в платформенный код, который может выполняться прямо на процессоре Android-устройства. Такое решение значительно ускоряет выполнение приложения и сокращает энергопотребление.

В: Виртуальная машина Java действительно настолько снижает эффективность?

О: Да, потому что в Android каждое приложение выполняется в отдельном процессе. При использовании обычных виртуальных машин Java потребовалось бы намного больше памяти.

В: Нужно ли создавать AVD заново при каждом создании нового приложения?

О: Нет — единожды созданное виртуальное устройство можно будет использовать для любых приложений. Возможно, вы захотите создать несколько AVD для тестирования приложений в разных ситуациях. Например, можно создать AVD для планшетных компьютеров, чтобы понять, как приложение будет выглядеть и работать на устройствах этой категории.

Модификация приложения

На нескольких последних страницах мы создали простейшее Android-приложение и увидели, как оно выполняется в эмуляторе. Теперь мы займемся усовершенствованием только что созданного приложения.

Сейчас приложение выводит стандартный текст “Hello world!”, включенный в него мастером. Мы заменим этот текст, чтобы приложение приветствовало пользователя как-то иначе. Что же для этого нужно сделать? Чтобы получить ответ на этот вопрос, отступим на шаг назад и посмотрим, как в настоящее время строится приложение.

Приложение состоит из одной активности и одного макета

Во время построения приложения мы сообщили Android Studio, как следует настроить его, а мастер сделал все остальное. Мастер сгенерировал базовую активность, а также макет по умолчанию.

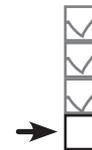
Активность управляет тем, что делает приложение

Android Studio создает за нас активность с именем `MainActivity.java`. Активность определяет, что приложение **делает** и как оно должно реагировать на действия пользователя.

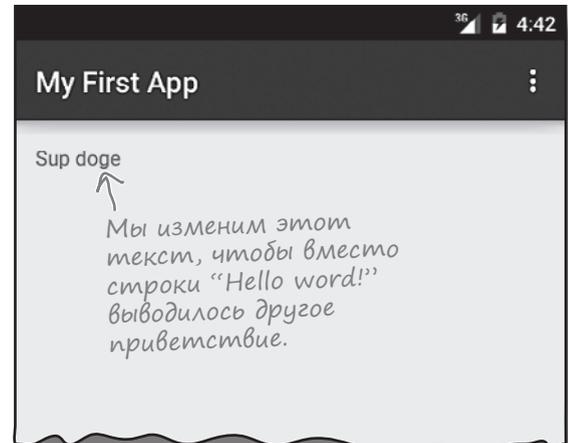
Макет управляет внешним видом приложения

Файл `MainActivity.java` указывает, что он использует макет с именем `activity_main.xml`, который среда Android Studio сгенерировала за нас. Макет определяет, как должно **выглядеть** приложение.

Итак, мы собираемся изменить внешний вид приложения, изменяя выводимый им текст. Это означает, что нам придется иметь дело с компонентом Android, управляющим внешним видом приложения. Значит, нужно поближе познакомиться с *макетом*.



- Подготовка среды
- Построение приложения
- Запуск приложения
- Изменение приложения



Активность определяет, что делает приложение и как оно должно взаимодействовать с пользователем.



MainActivity.java



activity_main.xml

Макет определяет, как должно выглядеть приложение.

Что содержит макет?

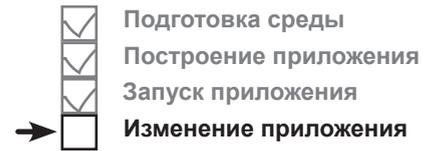
Мы собираемся изменить текст “Hello world!”, который среда Android Studio создала за нас, поэтому начнем с файла макета `activity_main.xml`. Если он еще не открыт в редакторе, откройте его — найдите файл в папке `app/src/main/res/layout` и сделайте на нем двойной щелчок.

Визуальный редактор

Есть два способа просмотра и редактирования файлов макетов в Android Studio: в **визуальном редакторе** и в **редакторе кода**.

В визуальном редакторе текст “Hello world!” отображается в макете, как и следовало ожидать. Но как выглядит разметка XML, обеспечивающая вывод этого текста?

Давайте посмотрим. Для этого нужно переключиться в редактор кода.



Редактор кода

Если выбрать редактор кода, в окне отображается содержимое файла `activity_main.xml`. Присмотримся к нему внимательнее.



activity_main.xml состоит из двух элементов

Ниже приведен код из файла `activity_main.xml`, сгенерированного Android Studio.

- Подготовка среды
- Построение приложения
- Запуск приложения
- Изменение приложения

```

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    ...
    tools:context=".MainActivity" >

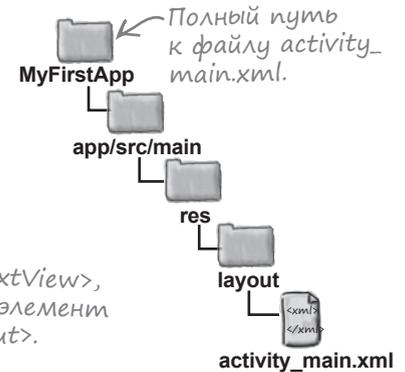
    <TextView
        android:text="@string/hello_world"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</RelativeLayout>
    
```

Этот элемент <RelativeLayout>

Здесь идет разметка XML, сгенерированная Android Studio, но для нас она пока интереса не представляет.

Элемент <TextView>, вложенный в элемент <RelativeLayout>.



Код состоит из двух элементов.

Первый элемент — `<RelativeLayout>` — приказывает Android выводить компоненты макета в относительных позициях. Например, элемент `<RelativeLayout>` может использоваться для выравнивания компонентов по центру макета, выравнивания по нижнему краю экрана Android-устройства или позиционирования относительно других компонентов.

Второй элемент — `<TextView>` — используется для вывода текста. Он вложен в элемент `<RelativeLayout>` и в нашем примере используется для вывода сгенерированного текста "Hello world!".

Ключевая часть кода в элементе `<TextView>` находится в первой строке. Вы ничего не замечаете?

```

<TextView
    android:text="@string/hello_world"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
    
```

Элемент `TextView` описывает текст в макете.

Ничего не заметили в этой строке?



Будьте осторожны!

Android Studio иногда отображает значения ссылок вместо реального кода.

Например, вместо реального кода `"@string/hello_world"` может отображаться текст «Hello world!». Все такие подстановки должны выделяться цветом в редакторе кода; если щелкнуть на них или навести указатель мыши, то откроется настоящий код.

```

<TextView
    android:text="Hello world!"
    android:text="@string/hello_world"
    android:layout_height="wrap_content" />
    
```

Файл макета содержит ссылку на строку, а не саму строку

Ключевой частью элемента `<TextView>` является первая строка:

```
android:text="@string/hello_world" />
```

Запись `android:text` означает, что речь идет о свойстве `text` элемента `<TextView>`, а конструкция определяет, какой текст должен выводиться в макете. Но почему используется синтаксис `"@string/hello_world"` вместо простого `"Hello world!"`? И что это вообще значит?

Начнем с первой части, `@string`. Она просто приказывает Android найти текстовое значение в файле строковых ресурсов. В нашем примере Android Studio создает файл строковых ресурсов с именем `strings.xml`, который находится в папке `app/src/main/res/values`. Вторая часть, `hello_world`, приказывает Android **получить значение ресурса с именем `hello_world`**. Таким образом, `@string/hello_world` означает: «Найти строковый ресурс с именем `hello_world` и использовать связанное с ним текстовое значение».

Вывести текст...

```
android:text="@string/hello_world" />
```

...строкового ресурса `hello_world`.

Слишком сложно. Почему бы не включить в `activity_main.xml` обычный текст? Разве это не проще?

Одна важнейшая причина: локализация

Допустим, вы создали приложение, которое пользовалось большим успехом в локальном магазине Google Play Store. Но вы не хотите ограничиваться одной страной или языком — приложение должно быть доступно для пользователей из других стран, говорящих на других языках.

Выделение текстовых значений в `strings.xml` существенно упрощает решение подобных задач. Вместо того, чтобы изменять жестко запрограммированные текстовые значения в множестве разных файлов, достаточно заменить файл `strings.xml` его локализованной версией. Использование файла `strings.xml` в качестве центрального хранилища текстовых значений также упрощает глобальные изменения в тексте в масштабах всего приложения. Если директор потребует изменить текст в приложении из-за того, что компания сменила свое название, достаточно ограничиться файлом `strings.xml`.

Размещайте строковые значения в `strings.xml` вместо того, чтобы жестко программировать их. `strings.xml` — файл ресурсов, используемый для хранения пар «имя / значение строки». Макеты и активности могут обращаться к строковым значениям по имени.



Заглянем в файл strings.xml

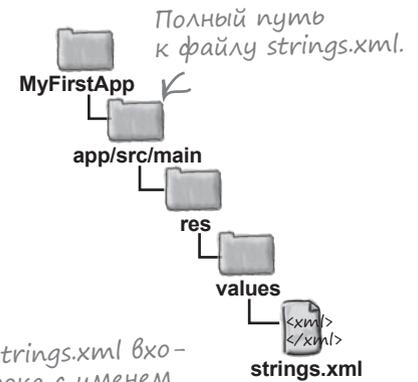
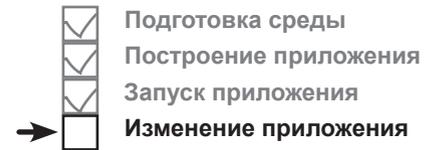
Среда Android Studio автоматически создала файл строковых ресурсов с именем *strings.xml*; давайте посмотрим, содержит ли этот файл ресурс `hello_world`. Найдите его в папке `app/src/main/res/values` на панели структуры проекта и откройте двойным щелчком.

Вот как выглядит код в *strings.xml*:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">My First App</string>
  <string name="hello_world">Hello world!</string>
  <string name="action_settings">Settings</string>
</resources>
```

Как видно из листинга, в файле присутствует строка, которая, похоже, нам и нужна. Она описывает строковый ресурс с именем `hello_world` и значением “Hello world!”:

```
<string name="hello_world">Hello world!</string>
```



В файл *strings.xml* входит строка с именем `hello_world` и значением “Hello world!”

Отредактируйте файл strings.xml, чтобы изменить выводимый текст

Итак, давайте изменим текст, выводимый приложением. Если вы этого еще не сделали ранее, найдите файл *strings.xml* на панели структуры проекта в Android Studio и сделайте на нем двойной щелчок, чтобы открыть его.

Ниже приведен код из файла. Найдите строку с именем “`hello_world`” и замените текстовое значение “Hello world!” другим — например, “Sup doge”:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">My First App</string>
  <string name="hello_world">Hello world! Sup doge</string>
  <string name="action_settings">Settings</string>
</resources>
```

Замените значение “Hello world!” на “Sup doge”!

После изменения файла перейдите в меню `File` и выберите команду `Save All`, чтобы сохранить изменения.



Строковые ресурсы под увеличительным стеклом

`strings.xml` — файл ресурсов по умолчанию. В этом файле хранятся строки в виде пар «имя/значение» для последующих обращений к ним из приложения. Строковые ресурсы имеют следующий формат:

Элемент `<resources>` идентифицирует содержимое файла как ресурсы.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">My First App</string>
    <string name="hello_world">Hello world!</string>
    <string name="action_settings">Settings</string>
</resources>
```

Элемент `<string>` определяет пару «имя/значение» как строковый ресурс.

Существуют два признака, по которым Android распознает `strings.xml` как файл строковых ресурсов:

- ★ **Файл хранится в папке `app/src/main/res/values`.**
Файлы XML, которые хранятся в этой папке, содержат простые значения — строки, определения цветов и т.д.
- ★ **Файл содержит элемент `<resources>`, который в свою очередь содержит один или несколько элементов `<string>`.**
Из формата самого файла следует, что он представляет собой файл ресурсов для хранения строк. Элемент `<resources>` сообщает Android, что файл содержит ресурсы, а элемент `<string>` идентифицирует каждый строковый ресурс.

Это означает, что файл строковых ресурсов не обязан называться `strings.xml`; ему можно присвоить другое имя или разбить строковые ресурсы по нескольким файлам.

Каждая пара «имя/значение» имеет формат

```
<string name="имя_строки">значение_строки</string>
```

где `имя_строки` — идентификатор строки, а `значение_строки` — собственно строковое значение.

Для обращения к значению строки из макета используется синтаксис вида

```
"@string/имя_строки" ← Имя строки, значение которой требуется получить.
```

Префикс `"string"` приказывает Android искать строковый ресурс с заданным именем.



Тест-драйв

После того как файл будет отредактирован, попробуйте снова запустить приложение в эмуляторе — выберите команду “Run ‘app’” из меню Run. На этот раз приложение должно вывести сообщение “Sup doge” вместо “Hello world!”.

- Подготовка среды
- Построение приложения
- Запуск приложения
- Изменение приложения

Обновленная версия приложения в эмуляторе.



Часто задаваемые вопросы

В: Текстовые значения обязательно хранить в файле строковых ресурсов — таком, как *strings.xml*?

О: Это не обязательно, но при обнаружении жестко запрограммированных текстовых значений Android выдает предупреждение. На первый взгляд может показаться, что строковые ресурсы требуют дополнительных усилий, но они значительно упрощают выполнение таких задач, как локализация. Кроме того, проще с самого начала использовать строковые ресурсы вместо того, чтобы переходить на них потом.

В: Как выделение строковых значений упрощает локализацию?

О: Допустим, вы хотите, чтобы в приложении по умолчанию использовался английский язык, но если на устройстве выбран французский язык, приложение переключалось на французский. Вместо того, чтобы жестко программировать разные языки в приложении, достаточно создать два разных файла строковых ресурсов: один для английского и другой для французского текста.

В: Как приложение определяет, какой файл следует использовать?

О: Сохраните файл строковых ресурсов по умолчанию (для английского языка) в папке *app/src/main/res/values*, а файл строковых ресурсов для французского языка — в отдельной папке с именем *app/src/main/res/values-fr*. Если на устройстве включен французский язык, приложение использует строки из папки *app/src/main/res/values-fr*. Если на устройстве включен любой другой язык, то оно использует строки из *app/src/main/res/values*.

В: Код макета, сгенерированный Android Studio, несколько отличается от того, что приведен в примерах книги. Это важно?

О: Разметка XML, генерируемая Android Studio, может слегка различаться в зависимости от используемой версии. Беспокоиться об этом не нужно — в дальнейшем вы все равно научитесь создавать код макета самостоятельно и сможете заменить большую часть кода, сгенерированного Android Studio.



Ваш инструментарий Android

Глава 1 подходит к концу. В ней ваш инструментарий дополнился основными концепциями программирования для Android.

Весь код для этой главы можно загрузить по адресу <https://tinyurl.com/HeadFirstAndroid>.

КЛЮЧЕВЫЕ МОМЕНТЫ



- Версии Android характеризуются номером версии, уровнем API и кодовым именем.
- Android Studio — специализированная версия среды IntelliJ IDEA, интегрированная с пакетом Android Software Development Kit (SDK) и системой сборки gradle.
- Типичное Android-приложение состоит из активностей, макетов и файлов ресурсов.
- Макеты описывают внешний вид приложения. Они хранятся в папке `app/src/main/res/layout`.
- Активности описывают то, что делает приложение, и как оно взаимодействует с пользователем. Созданные вами активности хранятся в папке `app/src/main/java`.
- Файл `strings.xml` содержит пары «имя/значение» для строк. Они используются для вынесения конкретных текстовых значений из макетов и активностей, а также для поддержки локализации.
- Файл `AndroidManifest.xml` содержит информацию о самом приложении. Этот файл находится в папке `app/src/main`.
- AVD — виртуальное устройство Android (Android Virtual Device). AVD выполняется в эмуляторе Android и моделирует физическое устройство Android.
- APK — пакет приложения Android, аналог JAR-файла для приложений Android. Файл содержит байт-код приложения, библиотеки и ресурсы. Установка приложения на устройстве осуществляется установкой его пакета APK.
- Приложения Android выполняются в отдельных процессах с использованием исполнительной среды Android (ART).
- Элемент `RelativeLayout` используется для размещения компонентов графического интерфейса в относительных позициях в макете.
- Элемент `TextView` используется для вывода текста.

2 построение интерактивных приложений

Приложения, которые что-то делают



Интересно, что
будет, если нажать
кнопку
«катапульта»?

Обычно приложение должно реагировать на действия пользователя. В этой главе вы узнаете, как существенно повысить интерактивность ваших приложений. Мы покажем, как заставить приложение делать что-то в ответ на действия пользователя и как заставить активность и макет общаться друг с другом, как старые знакомые. Попутно вы больше узнаете о том, как на самом деле работает Android; мы расскажем о R — неприметном сокровище, которое связывает все воедино.

В этой главе мы построим приложение для выбора пива

В главе 1 вы узнали, как создать простейшее приложение при помощи мастера New Project в Android Studio и как изменить текст, отображаемый в макете. Но когда вы создаете Android-приложение обычно это приложение должно что-то *делать*.

В этой главе мы покажем, как создать приложение, взаимодействующее с пользователем. В приложении Beer Adviser пользователь выбирает вид пива, который он предпочитает, щелкает на кнопке и получает список рекомендуемых сортов.

Приложение имеет следующую структуру:

1 Макет определяет, как будет выглядеть приложение.

Он состоит из трех компонентов графического интерфейса:

- раскрывающегося списка значений, в котором пользователь выбирает нужный вид пива;
- кнопки, которая при нажатии возвращает подборку сортов пива;
- надписи для вывода сортов пива.

2 Файл strings.xml включает все строковые ресурсы, необходимые макету, — например, текст надписи на кнопке, входящей в макет.

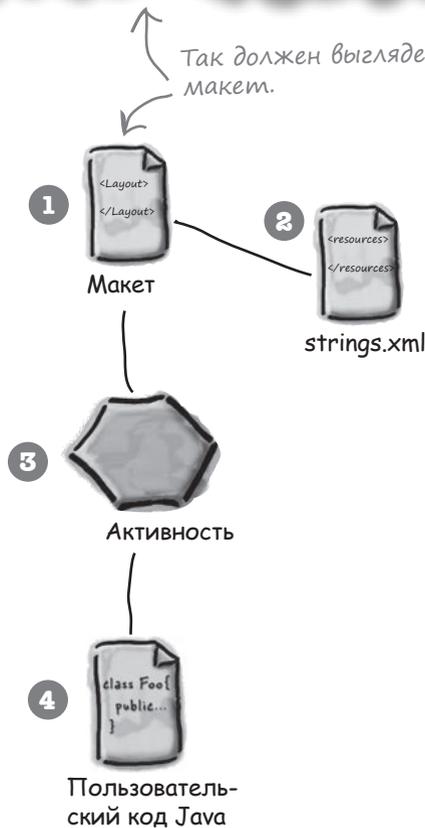
3 Активность определяет, как приложение должно взаимодействовать с пользователем. Она получает вид пива, выбранный пользователем, и использует его для вывода списка сортов, которые могут представлять интерес для пользователя. Для решения этой задачи используется вспомогательный класс Java.

4 Класс Java, содержащий логику приложения.

Класс включает метод, который получает вид пива в параметре и возвращает список сортов пива указанного типа. Активность вызывает метод, передает ему вид пива и использует полученный ответ.



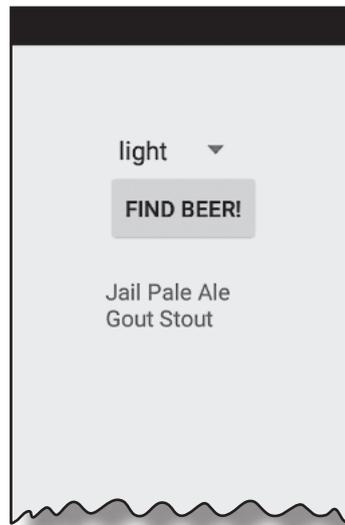
Так должен выглядеть макет.



Что нужно сделать

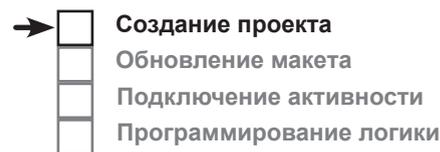
Итак, приступим к построению приложения Beer Adviser. Работа состоит из нескольких шагов (все они будут подробно рассмотрены в этой главе):

- 1 **Создание проекта.**
Мы создаем совершенно новое приложение, для которого нужно будет создать новый проект. Как и в предыдущей главе, в этот проект достаточно включить базовый макет и активность.
- 2 **Обновление макета.**
Когда основная структура приложения будет готова, следует отредактировать макет и включить в него все компоненты графического интерфейса, необходимые для работы приложения.



- 3 **Связывание макета с активностью.**
Макет создает только визуальное оформление. Чтобы приложение могло выполнять разумные действия, необходимо связать макет с кодом Java в активности.
- 4 **Программирование логики приложения.**
Мы добавим в приложение класс Java, который будет возвращать пользователю правильные сорта пива в зависимости от их выбора.

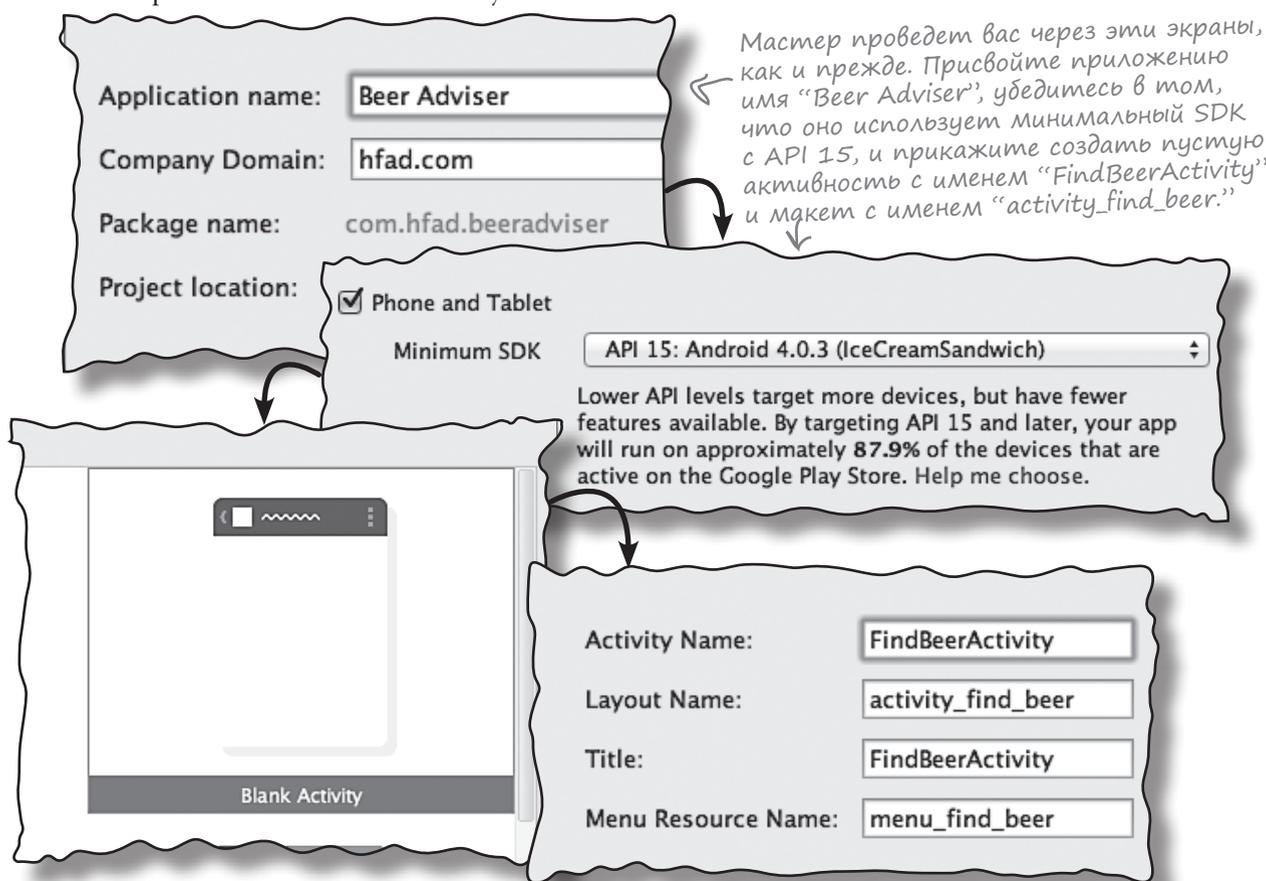




Создание проекта

Работа начинается с создания нового приложения (это делается почти так же, как в предыдущей главе):

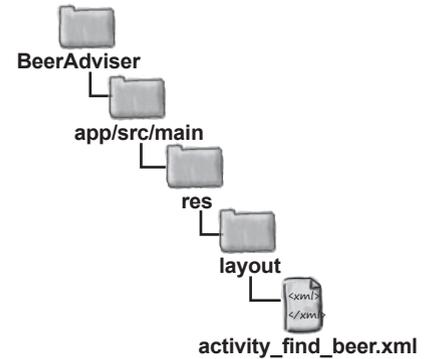
- ★ Откройте Android Studio и выберите на заставке строку “Start a new Android Studio project”. Запускается мастер, уже знакомый вам по главе 1.
- ★ По запросу мастера введите имя приложения “Beer Adviser”; убедитесь в том, что в окне сгенерировано имя пакета `com.hfad.beeradviser`.
- ★ Чтобы приложение работало на большинстве телефонов и планшетов, выберите минимальную версию SDK с API 15 и проследите за тем, чтобы флажок “Phone and Tablet” был установлен. Это означает, что на любом телефоне или планшете, на котором выполняется приложение, должна быть установлена как минимум версия API 15. Большинство устройств на базе Android соответствует этому критерию.
- ★ Выберите пустую активность в качестве активности по умолчанию. Присвойте ей имя “FindBeerActivity”, а макету — имя “activity_find_beer”. Подтвердите значения по умолчанию для текста заголовка (Title) и имени ресурса меню (Menu Resource Name), так как в этом приложении они не используются.



Мы создали активность и макет по умолчанию

Когда вы щелкнете на кнопке Finish, среда Android Studio создаст новый проект, содержащий активность *FindBeerActivity.java* и макет *activity_find_beer.xml*. Начнем с редактирования файла макета. Для этого перейдите в папку *app/src/main/res/layout* и откройте файл *activity_find_beer.xml*.

Как и в предыдущем приложении, мастер создал макет по умолчанию с элементом `<TextView>`, содержащим текст "Hello world!":



Разметка XML макета

```

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    tools:context=".FindBeerActivity">
  
```

Эти элементы относятся к макету в целом. Они определяют его ширину и высоту, а также величину отступов от краев макета.

```

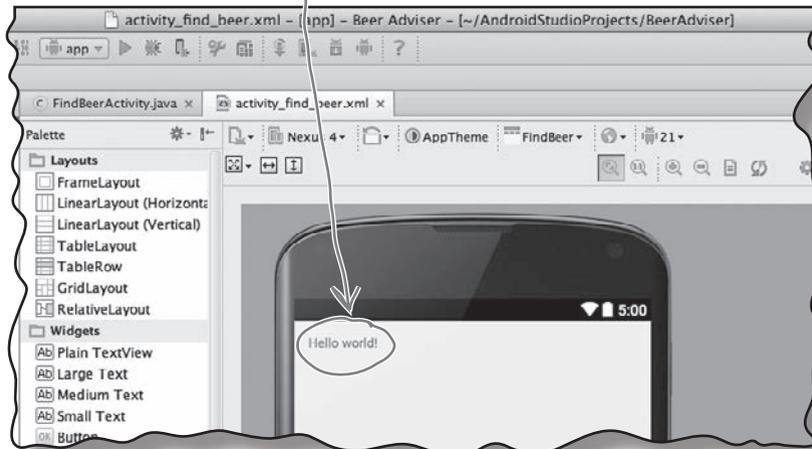
<TextView
    android:text="@string/hello_world"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
  
```

```

</RelativeLayout>
  
```

Визуальный редактор

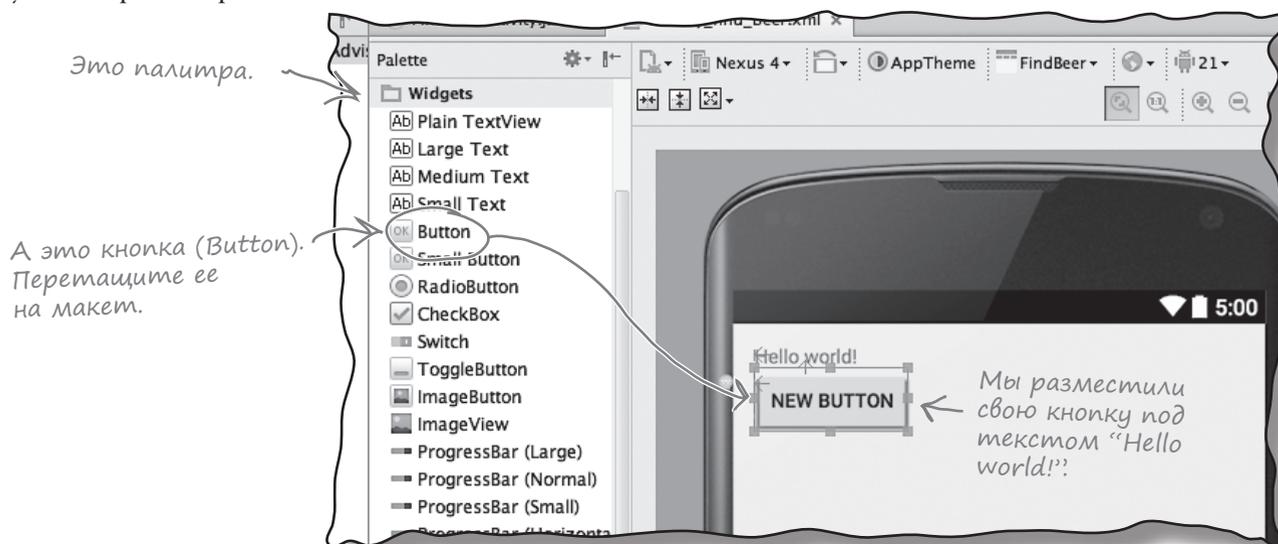
Элемент `<TextView>` из разметки XML отображается в визуальном редакторе.



- Создание проекта
- Обновление макета
- Подключение активности
- Программирование логики

Добавление компонентов в визуальном редакторе

Добавить компоненты графического интерфейса в макет можно двумя способами: в разметке XML или в визуальном редакторе. Начнем с добавления кнопки в визуальном редакторе. Слева от визуального редактора располагается палитра с компонентами графического интерфейса, которые можно перетаскивать мышью на макет. Просмотрите раздел Widgets и найдите в нем компонент кнопки (Button). Щелкните на нем и перетащите на макет в визуальном редакторе.



Изменения, внесенные в визуальном редакторе, отражаются в XML

Такое перетаскивание компонентов графического интерфейса является удобным способом обновления макета. Переключившись на редактор кода, вы увидите, что в результате добавления кнопки в визуальном редакторе в файле появилось несколько строк кода:

```

...
<TextView
    android:text="@string/hello_world"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/textView" />
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="New Button"
    android:id="@+id/button"
    android:layout_below="@+id/textView"
    android:layout_alignLeft="@+id/textView" />
...
    
```

Код, добавленный визуальным редактором, зависит от того, где вы разместили кнопку. Возможно, ваша разметка макета будет отличаться от нашей; не беспокойтесь — скоро мы ее изменим.

Новый элемент Button описывает кнопку, которую вы перетащили на макет. Он будет более подробно рассмотрен ниже.

Элементу TextView, который вы видели ранее, присвоен идентификатор.

В activity_find_beer.xml появилась новая кнопка

Редактор добавил новый элемент `<Button>` в файл `activity_find_beer.xml`:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="New Button"
    android:id="@+id/button"
    android:layout_below="@+id/textView"
    android:layout_alignLeft="@+id/textView" />
```

В мире Android кнопка нажимается пользователем, чтобы инициировать какое-либо действие. Кнопка обладает свойствами, управляющими ее позицией, размером, внешним видом и методами активности, которые она должна вызывать. Эти свойства существуют не только у кнопок — они есть и у других компонентов графического интерфейса, включая надписи.

Кнопки и надписи — subclasses одного класса Android View

Тот факт, что кнопки и надписи имеют так много общих свойств, вполне логичен — оба компонента наследуют от одного класса Android **View**. Более подробные описания свойств будут приведены ниже, а пока рассмотрим несколько типичных примеров.

android:id

Имя, по которому идентифицируется компонент. Свойство ID используется для управления работой компонента из кода активности, а также для управления размещением компонентов в макете:

```
android:id="@+id/button"
```

android:text

Свойство сообщает Android, какой текст должен выводиться в компоненте. В случае `<Button>` это текст, выводимый на кнопке:

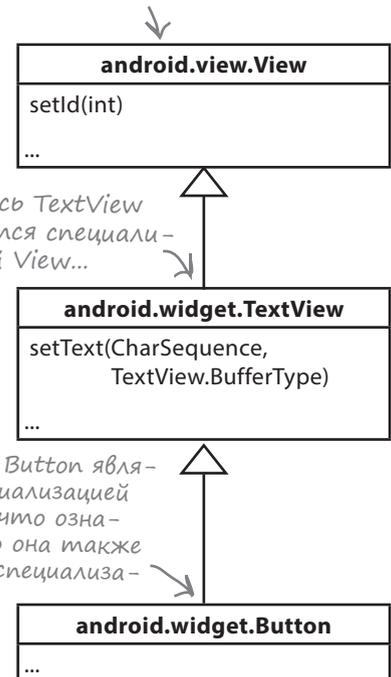
```
android:text="New Button"
```

android:layout_width, android:layout_height

Эти свойства задают базовую ширину и высоту компонента. Значение `"wrap_content"` означает, что размеры компонента должны подбираться по размерам содержимого:

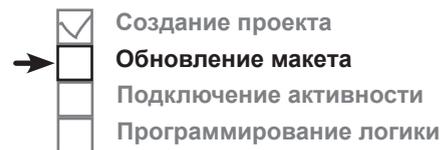
```
android:layout_width="wrap_content"
android:layout_height="wrap_content"
```

Класс View содержит множество разных методов. Они будут рассмотрены позднее в книге.



Надпись TextView является специализацией View...

...а кнопка Button является специализацией TextView, что означает, что она также является специализацией View.



Подробнее о коде макета

Давайте внимательнее рассмотрим код макета и разобьем его так, чтобы происходящее стало более понятным (не беспокойтесь, если ваш код выглядит немного иначе — просто следите за логикой):

Элемент
RelativeLayout

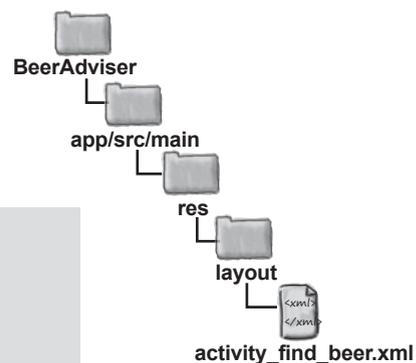
```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    tools:context=".FindBeerActivity">
```

Надпись

```
<TextView
    android:text="@string/hello_world"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/textView" />
```

Кнопка

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="New Button"
    android:id="@+id/button"
    android:layout_below="@+id/textView"
    android:layout_alignLeft="@+id/textView" />
```



```
</RelativeLayout>
```

← Закрывает элемент RelativeLayout.

Элемент RelativeLayout

Код макета начинается с элемента-`<RelativeLayout>`. Элемент `<RelativeLayout>` сообщает Android, что компоненты графического интерфейса в макете должны отображаться относительно друг друга. Например, вы можете приказать, чтобы один компонент отображался слева от другого, или они должны быть выровнены по некоторой общей линии. В нашем примере кнопка располагается прямо под надписью, поэтому кнопка отображается относительно надписи.

← Также существуют и другие способы размещения компонентов графического интерфейса. Вскоре вы о них узнаете.

Элемент TextView

Внутри элемента `<RelativeLayout>` первым идет элемент надписи `<TextView>`:

```
...
<TextView
    android:text="@string/hello_world"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/textView" />
...
```

Никакие свойства, указывающие местонахождение надписи в макете, не были заданы, поэтому по умолчанию Android выводит ее в левом верхнем углу экрана. Обратите внимание: надписи присваивается идентификатор `textView`. Вы поймете, зачем он нужен, когда мы перейдем к следующему элементу.

Элемент Button

Содержимое элемента `<RelativeLayout>` завершает элемент кнопки `<Button>`:

```
...
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="New Button"
    android:id="@+id/button"
    android:layout_below="@+id/textView"
    android:layout_alignLeft="@+id/textView" />
...
```

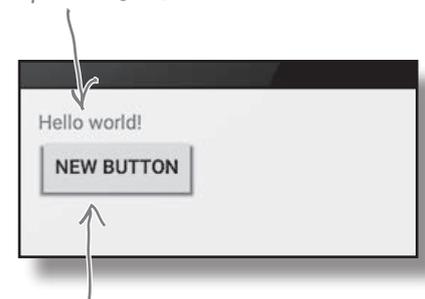
При добавлении кнопки в макет мы разместили ее так, что она находится под надписью, а левый край кнопки выравнивается по левому краю надписи. Кнопка размещается *относительно* надписи, и этот факт отражен в разметке XML:

```
android:layout_below="@+id/textView"
android:layout_alignLeft="@+id/textView"
```

Также возможны другие варианты разметки XML, достигающей того же визуального эффекта. Например, приведенный фрагмент XML указывает, что *кнопка размещается под надписью*. Также мы могли воспользоваться эквивалентной командой, указывающей, что *надпись размещается над кнопкой*.

В режиме относительного размещения `<RelativeLayout>` компоненты графического интерфейса размещаются относительно друг друга.

По умолчанию надпись размещается в левом верхнем углу.

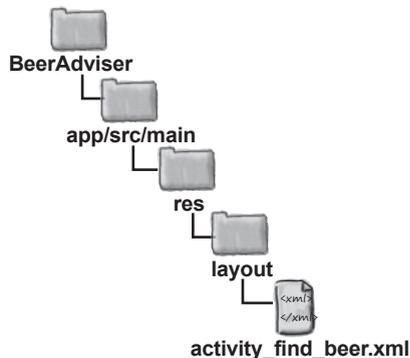


Свойства кнопки заданы так, что она отображается под надписью, а ее левый край выравнивается по вертикали с левым краем надписи.

Изменения в XML...

Вы уже видели, как изменения, вносимые в визуальном редакторе, отражаются в разметке XML макета. Также справедливо и обратное: все изменения, вносимые в разметке XML макета, отражаются в визуальном редакторе.

Попробуйте заменить код из файла `activity_find_beer.xml` следующим фрагментом:



```

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    tools:context=".FindBeerActivity" >
    
```

Раскрывающийся список значений в системе Android. Компонент предназначен для выбора одного значения из представленного набора.

```

<Spinner
    android:id="@+id/color"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="37dp" />
    
```

Элемент `<Spinner>` создает раскрывающийся список значений.

Кнопка размещается под раскрывающимся списком и выравнивается с ним по левому краю.

```

<Button
    android:id="@+id/find_beer"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/color"
    android:layout_below="@+id/color"
    android:text="Button" />
    
```

Надпись размещается под кнопкой и выравнивается по левому краю кнопки.

```

<TextView
    android:id="@+id/brands"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/find_beer"
    android:layout_below="@+id/find_beer"
    android:layout_marginTop="18dp"
    android:text="TextView" />
    
```

```

</RelativeLayout>
    
```



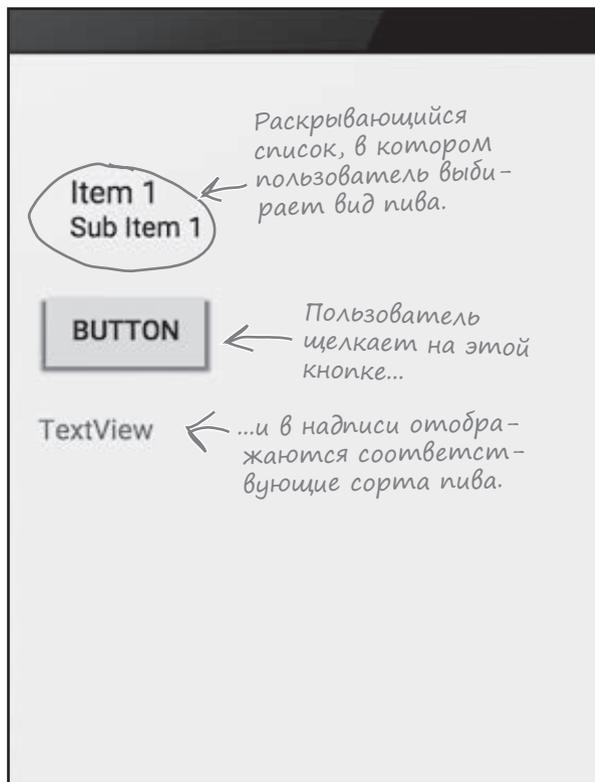
Задание!

Замените содержимое `activity_find_beer.xml` показанной разметкой XML.

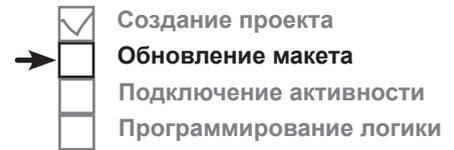
...отражаются в визуальном редакторе

После внесения изменений в XML макета перейдите в визуальный редактор. Вместо макета с надписью и расположенной под ней кнопкой должен отображаться макет, в котором надпись отображается под кнопкой.

Над кнопкой располагается **раскрывающийся список** (spinner). Если коснуться его, на экране появляется список, из которого пользователь выбирает одно значение.



Мы показали, как добавлять компоненты графического интерфейса в макет в визуальном редакторе и как добавлять их в разметку XML. Скорее всего, для получения желаемых результатов вы будете чаще работать с XML напрямую, без использования визуального редактора. Дело в том, что прямое редактирование XML позволяет более точно управлять макетом, а также сокращает зависимость разработчика от среды разработки.



Раскрывающийся список предоставляет пользователю набор значений, из которого пользователь выбирает один вариант.

Компоненты графического интерфейса — кнопки, раскрывающиеся списки, надписи — обладают похожими атрибутами, так как все они являются специализациями View. Классы всех этих компонентов наследуют от одного класса Android View.

Использование строковых ресурсов вместо жестко запрограммированного текста

Прежде чем запускать приложение, необходимо внести еще одно изменение. На данный момент тексты кнопки и надписи задаются жестко запрограммированными строковыми значениями. Как упоминалось в главе 1, желательно заменить их ссылками на файл строковых ресурсов *strings.xml*. И хотя такая замена не является строго необходимой, это полезная привычка. Использование файла строковых ресурсов для статического текста упрощает создание версий приложения на других языках, а если вдруг потребуется изменить формулировки во всем приложении, достаточно будет внести изменения в одном централизованном месте. Откройте файл *app/src/main/res/values/strings.xml*. При переходе в режим XML он должен выглядеть примерно так:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="app_name">Beer Adviser</string>
    <string name="hello_world">Hello world!</string>
    <string name="action_settings">Settings</string>

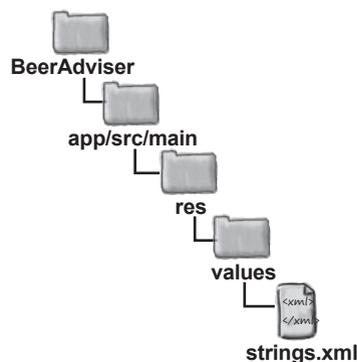
</resources>
```

Эти строки созданы средой Android Studio.

Прежде всего удалите ресурс “hello_world”, так как он нам больше не нужен. Добавьте новый ресурс с именем “find_beer” и значением “Find Beer!”. Когда это будет сделано, добавьте новый ресурс с именем “brands”, но пока не вводите значение.

Новый код должен выглядеть примерно так:

```
...
<string name="app_name">Beer Adviser</string>
<string name="hello_world">Hello world!</string>
<string name="action_settings">Settings</string>
<string name="find_beer">Find Beer!</string>
<string name="brands"></string>
...
```



Необходимо удалить строковый ресурс hello_world и добавить два новых ресурса с именами find_beer и brands.

Внесение изменений в макет для использования строковых ресурсов

Теперь изменим элементы кнопки и надписи в разметке XML макета, чтобы они использовали два только что добавленных строковых ресурса. Откройте файл `activity_find_beer.xml` и внесите следующие изменения:

- ★ Замените строку `android:text="Button"` строкой `android:text="@string/find_beer"`.
- ★ Замените строку `android:text="TextView"` строкой `android:text="@string/brands"`.

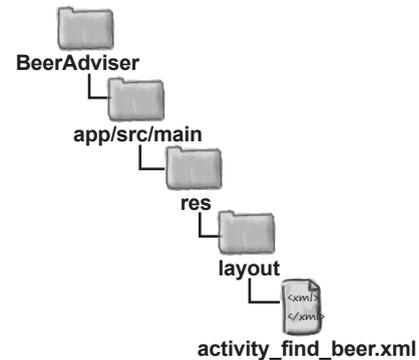
...

```
<Spinner
    android:id="@+id/color"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="37dp" />
```

```
<Button
    android:id="@+id/find_beer"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/color"
    android:layout_below="@+id/color"
    android:text="@string/find_beer" />
```

```
<TextView
    android:id="@+id/brands"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/find_beer"
    android:layout_below="@+id/find_beer"
    android:layout_marginTop="18dp"
    android:text="@string/brands" />
```

...



← На кнопке выводится значение строкового ресурса `find_beer`.

← Значение строкового ресурса `brands` выводится в надписи. В настоящее время надпись пуста, но все будущие изменения строкового значения будут отражаться автоматически.



Посмотрим, что же получилось

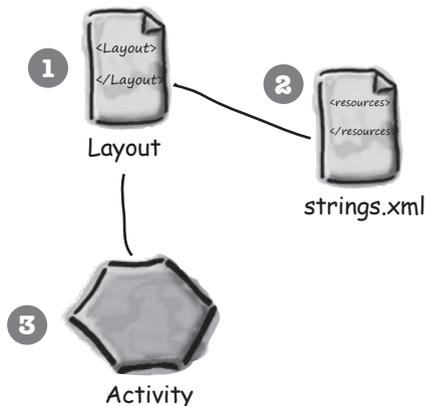
Над приложением еще придется поработать, но давайте посмотрим, как выглядит результат на текущий момент. Сохраните внесенные изменения и выберите команду “Run ‘app’” из меню Run. Когда вам будет предложено выбрать способ запуска, выберите запуск в эмуляторе. Терпеливо подождите, пока приложение загрузится. Рано или поздно оно появится на экране.

Попробуйте прикоснуться к раскрывающемуся списку. Возможно, это и не очевидно, но при прикосновении на экране должен появиться список значений — просто мы еще не добавили в него ни одного значения.

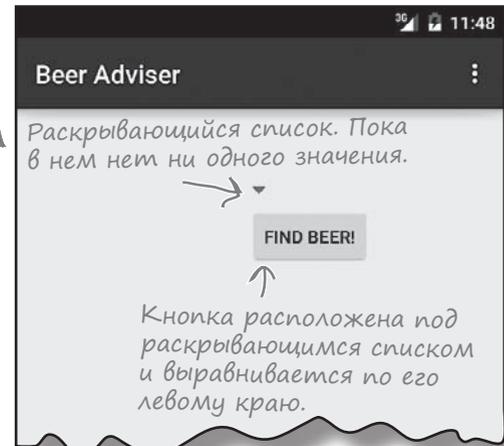
Что мы успели сделать

Ниже кратко перечислены основные действия, которые были выполнены на настоящий момент:

- 1 Мы создали макет, определяющий внешний вид приложения. Макет включает в себя раскрывающийся список, кнопку и надпись.
- 2 Файл `strings.xml` включает необходимые строковые ресурсы. Мы добавили текст для кнопки и пустую строку для сортов пива.
- 3 Активность определяет, как приложение должно взаимодействовать с пользователем. Среда Android Studio сгенерировала базовую активность, но она пока так и осталась в исходном виде.



- Создание проекта
- Обновление макета
- Подключение активности
- Программирование логики



Часть Задаваемые Вопросы

В: При запуске приложения макет несколько отличается от того, как он выглядел в визуальном редакторе. Почему?

О: Визуальный редактор очень старается правильно показать, как будет выглядеть макет, но у него есть свои ограничения. Например, в нашей разметке XML указано, что раскрывающийся список должен использовать горизонтальное выравнивание по центру, но в визуальном редакторе это может быть неочевидно.

На практике всегда лучше работать прямо с XML. Разметка дает более точное представление о том, что происходит, и предоставляет более точные средства контроля.

В: Кажется, в макете была еще и надпись?

О: Надпись есть, но пока она не содержит никакого текста, поэтому вы ее не видите. Она появится позднее в этой главе, когда мы настроим надпись для вывода текста.

Добавление значений в список

На данный момент макет включает раскрывающийся список, но в этом списке нет никаких данных. Чтобы список приносил пользу, в нем должен отображаться список значений. Пользователь выбирает в этом списке то значение, которое ему нужно. Список значений для раскрывающегося списка можно определить практически так же, как мы определим текст на кнопке и надписи: нужно создать для него **ресурс**. До сих пор в файле `strings.xml` определялись одиночные строковые значения. Все, что нам нужно, — это определить *массив* строковых значений и передать ссылку на него раскрывающемуся списку.

Добавление ресурса массива очень похоже на добавление ресурса строки

Как вы уже знаете, для добавления строкового ресурса в файл `strings.xml` необходимо выполнить следующие действия:

```
<string name="имя_строки">значение</string>
```

где `имя_строки` — идентификатор строки, а `значение` — собственно строковое значение.

Синтаксис добавления массива строк выглядит так:

```
<string-array name="имя_массива_строк" > ← Имя массива.
    <item>значение1</item>
    <item>значение2</item>
    <item>значение3</item>
    ...
</string-array>
```

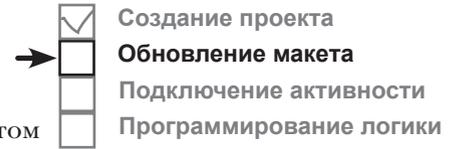
} Значения в массиве. Добавьте столько, сколько потребуется.

где `имя_массива` — имя массива, а `значение1`, `значение2`, `значение3` — отдельные строковые значения, входящие в массив.

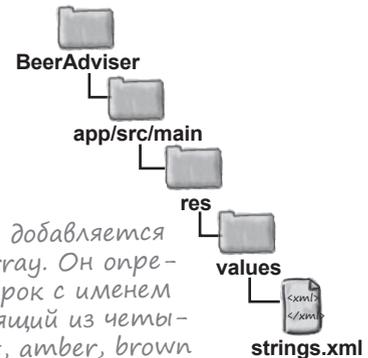
Давайте включим ресурс `string-array` в наше приложение. Откройте файл `strings.xml` и добавьте в него следующий фрагмент:

```
...
<string name="brands"></string>
<string-array name="beer_colors">
    <item>light</item>
    <item>amber</item>
    <item>brown</item>
    <item>dark</item>
</string-array>
</resources>
```

В файл `strings.xml` добавляется элемент `string-array`. Он определяет массив строк с именем `beer_colors`, состоящий из четырех значений: `light`, `amber`, `brown` и `dark`.



Ресурсы — непрограммные данные (например, графика или строки), используемые в приложении.



Передача массива строк раскрывающемуся списку

Для обращения к массиву строк в макете используется синтаксис, сходный с синтаксисом получения строкового значения. Вместо конструкции

"@string/имя_строки"

используется конструкция

"@array/имя_массива"

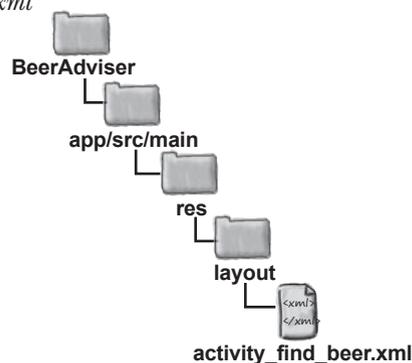
Используйте @string для обращения к строке и @array — для обращения к массиву.

где имя_массива — имя массива.

Используем эту ссылку в макете. Откройте файл макета `activity_find_beer.xml` и добавьте в элемент `<Spinner>` атрибут `entries`:

```
...
<Spinner
...
    android:layout_marginTop="37dp"
    android:entries="@array/beer_colors" />
...
```

Это означает "данные раскрывающегося списка берутся из массива `beer_colors`".



Тест-драйв раскрывающегося списка

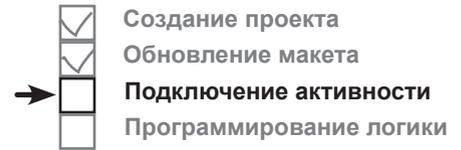
Посмотрим, как эти изменения отразились на приложении. Сохраните изменения и запустите приложение. Результат должен выглядеть примерно так:

- Создание проекта
- Обновление макета
- Подключение активности
- Программирование логики

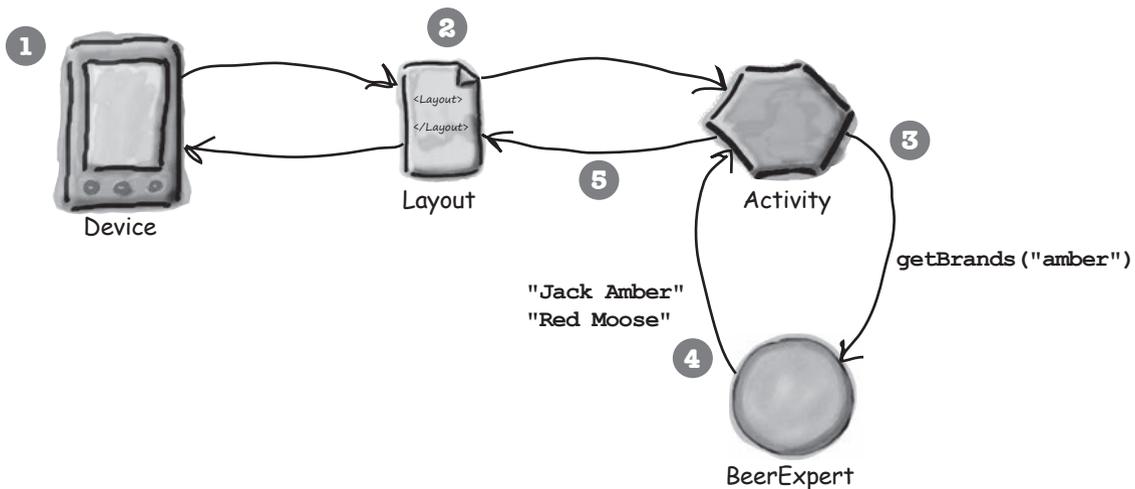
The screenshots show the 'Beer Adviser' app interface. The first screenshot shows the default state with a spinner set to 'light' and a 'FIND BEER!' button. The second screenshot shows the spinner menu expanded, listing 'light', 'amber', 'brown', and 'dark'. The third screenshot shows the spinner set to 'amber' after a selection.

Кнопка должна что-то делать

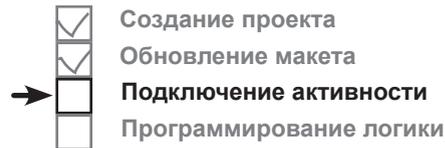
Пока что мы добавили в макет новые компоненты графического интерфейса и заполнили раскрывающийся список массивом значений. Теперь нужно добиться того, чтобы приложение реагировало на значение, выбранное в раскрывающемся списке. Оно должно работать приблизительно так:



- 1 Пользователь выбирает вид пива в раскрывающемся списке. Пользователь щелкает на кнопке, чтобы найти сорта пива данного вида.
- 2 Макет указывает, какой метод активности должен вызываться при щелчке на кнопке.
- 3 Метод активности получает от раскрывающегося списка выбранное значение (вид пива) и передает его методу `getBrands()` класса Java с именем `BeerExpert`.
- 4 Метод `getBrands()` класса `BeerExpert` находит сорта пива, соответствующие заданному виду, и возвращает их активности в виде объекта `ArrayList` со строковыми данными.
- 5 Активность получает ссылку на надпись из макета и присваивает ее свойству `text` список подходящих сортов. Информация отображается на устройстве.



Для начала нужно добиться того, чтобы при нажатии кнопки вызывался метод.



Как заставить кнопку вызывать метод

Если вы добавляете в макет кнопку, то скорее всего, когда пользователь щелкает на этой кнопке, в приложении что-то должно происходить. Но для этого необходимо, чтобы при щелчке на кнопке вызывался некий метод вашей активности.

Чтобы щелчок на кнопке приводил к вызову метода активности, необходимо внести изменения в двух файлах:

- ★ **Изменения в файле макета `activity_find_beer.xml`.**
Необходимо указать, какой метод активности должен вызываться при щелчке на кнопке.
- ★ **Изменения в файле активности `FindBeerActivity.java`.**
Необходимо написать метод, который будет вызываться при щелчке.

Начнем с макета.

`onClick` и метод, вызываемый при щелчке

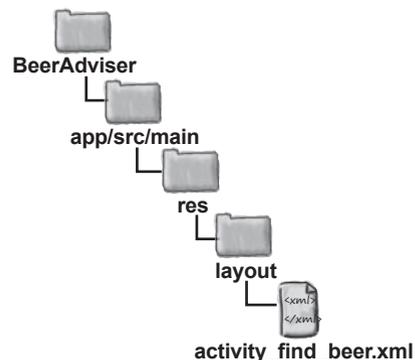
Чтобы сообщить Android, какой метод должен вызываться при щелчке на кнопке, достаточно всего одной строки разметки XML. Все, что для того нужно — добавить атрибут `android:onClick` в элемент `<button>` и указать имя вызываемого метода:

```
android:onClick="имя_метода"
```

← Это означает “когда пользователь щелкает на компоненте, вызвать метод активности с именем `имя_метода`”.

Посмотрим, как это делается. Откройте файл макета `activity_find_beer.xml` и добавьте в элемент `<button>` новую строку XML, которая сообщает, что при щелчке на кнопке должен вызываться метод `onClickFindBeer()`:

```
...
<Button
    android:id="@+id/find_beer"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/color"
    android:layout_below="@+id/color"
    android:text="@string/find_beer"
    android:onClick="onClickFindBeer" />
...
```



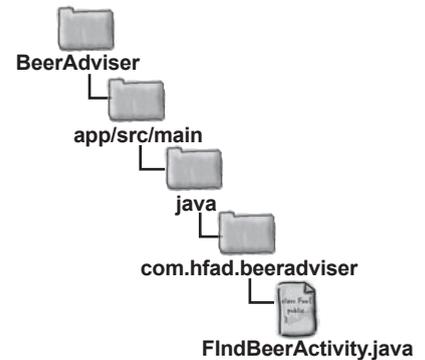
← При щелчке на кнопке вызывать метод `onClickFindBeer()` активности. Мы создадим этот метод на нескольких ближайших страницах.

После внесения изменений сохраните файл.

Теперь макет знает, какой метод активности следует вызвать; но мы еще должны написать сам метод. Давайте посмотрим, как выглядит активность.

Как выглядят код активности

В процессе создания проекта для приложения мы приказали мастеру сгенерировать простейшую активность с именем `FindBeerActivity`. Код этой активности хранится в файле `FindBeerActivity.java`. Откройте этот файл — перейдите в папку `app/src/main/java` и сделайте на нем двойной щелчок.



Вы увидите, что среда Android Studio сгенерировала за вас довольно большой объем кода Java. Вместо того, чтобы подробно анализировать весь сгенерированный код, мы просто предложим заменить его кодом, приведенным ниже. Дело в том, что большая часть кода активности, сгенерированного Android Studio, не обязательна, а мы хотим сосредоточиться на фундаментальных аспектах Android, а не на особенностях отдельной среды разработки. Итак, удалите код, который сейчас находится в файле `FindBeerActivity.java`, и замените его следующим кодом:

```
package com.hfad.beeradviser;
```

```
import android.os.Bundle;
import android.app.Activity;
```

```
public class FindBeerActivity extends Activity {
```

```
@Override
```

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_find_beer);
}
```

```
}
```

Класс расширяет класс `Android Activity`.

Метод `onCreate()` вызывается при исходном создании активности.

Метод `setContentView` сообщает Android, какой макет использует активность. В данном случае это макет `activity_find_beer`.

Этот код — все, что необходимо для создания простейшей активности. Как видите, в нем создается класс, который расширяет класс `android.app.Activity` и реализует метод `onCreate()`.

Все активности должны расширять класс `Activity`. Класс `Activity` содержит набор методов, которые превращают обычный класс Java в полноценную активность Android.

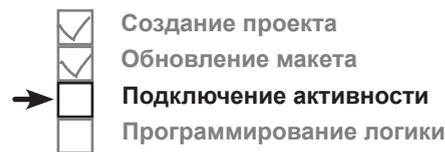
Все активности также должны реализовать метод `onCreate()`. Метод `onCreate()` вызывается при создании объекта активности и используется для настройки основных параметров — например, выбора макета, с которым связывается активность. Это делается при помощи метода `setContentView()`. В приведенном примере вызов `setContentView(R.layout.activity_find_beer)` сообщает Android, что эта активность использует макет `activity_find_beer`.

На предыдущей странице мы добавили атрибут `onClick` к кнопке в макете и присвоили ему значение `onClickFindBeer`. Теперь нужно добавить этот метод в активность, чтобы он вызывался при нажатии кнопки. Таким образом, активность будет реагировать на нажатия пользователем кнопки в интерфейсе.



`onClickFindBeer()`

Добавление в активность метода `onClickFindBeer()`



Метод `onClickFindBeer()` должен иметь строго определенную сигнатуру; в противном случае он не будет вызываться при щелчке на кнопке, указанной в макете. Он имеет следующую форму:

```
public void onClickFindBeer(View view) {
```

Метод должен быть объявлен открытым (`public`).

Метод должен возвращать `void`.

Метод должен иметь один параметр с типом `View`.

Если метод имеет другую сигнатуру, он не будет реагировать на прикосновение пользователя к кнопке. Дело в том, что Android незаметно для пользователя ищет открытый метод, возвращающий `void`, имя которого совпадает с именем метода, указанного в разметке XML макета.

Параметр `View` на первый взгляд кажется несколько странным, но для его присутствия имеется веская причина. Он определяет компонент графического интерфейса, инициировавший вызов метода (в данном случае это кнопка). Как упоминалось ранее, компоненты графического интерфейса — такие, как кнопки и надписи, — все являются специализациями `View`.

Итак, обновим наш код активности. Добавьте в код активности метод `onClickFindBeer()`, приведенный ниже:

Мы используем ... этот класс, поэтому его необходимо импортировать.

```
import android.view.View;
```

```
public class FindBeerActivity extends Activity {
```

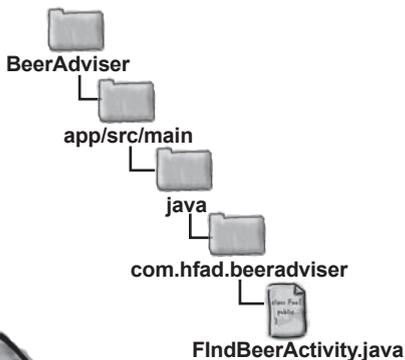
```
...
```

```
//Call when the user clicks the button
```

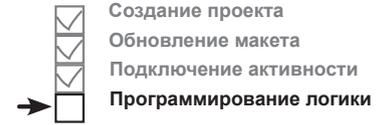
```
public void onClickFindBeer(View view) {
```

```
}
```

Добавьте метод `onClickFindBeer()` в `FindBeerActivity.java`.



Метод `onClickFindBeer()` должен что-то делать



Итак, мы создали в активности метод `onClickFindBeer()`. Далее нужно позаботиться о том, чтобы при выполнении этого метода что-то происходило. Приложение должно выводить подборку сортов пива, соответствующих виду, выбранному пользователем.

Для этого необходимо сначала получить ссылки на оба компонента графического интерфейса в макете — раскрывающийся список и надпись. С помощью этих ссылок мы сможем получить значение, выбранное в списке (вид пива), и вывести текст в надписи.

Использование `findViewById()` для получения ссылки на компонент

Для получения ссылки на два компонента графического интерфейса можно воспользоваться методом `findViewById()`. Метод `findViewById()` получает идентификатор компонента в виде параметра и возвращает объект `View`. Далее остается привести возвращаемое значение к правильному типу компонента (например, `TextView` или `Button`).

Посмотрим, как метод `findViewById()` используется для получения ссылки на надпись с идентификатором `brands`:

```
TextView brands = (TextView) findViewById(R.id.brands);
```

brands имеет тип TextView, поэтому ссылка приводится к этому типу.

Присмотритесь к тому, как задается идентификатор надписи. Вместо того, чтобы передавать имя компонента, мы передаем идентификатор вида `R.id.brands`. Что это означает? Что такое `R`?

`R.java` — специальный файл Java, который генерируется инструментарием Android при создании или построении приложения. Он находится в папке `app/build/generated/source/r/debug` вашего проекта — внутри папки, имя которой совпадает с именем пакета приложения. Android использует `R` для отслеживания ресурсов, используемых в приложении; среди прочего, этот класс позволяет получать ссылки на компоненты графического интерфейса из кода активности.

Открыв файл `R.java`, вы увидите, что он содержит серию внутренних классов, по одному для каждого типа ресурсов. Обращение к каждому ресурсу этого типа осуществляется через внутренний класс. Скажем, `R` включает внутренний класс с именем `id`, а в этот внутренний класс входит значение `static final brands`. Строка кода

```
(TextView) findViewById(R.id.brands);
```

использует это значение для получения ссылки на надпись `brands`.

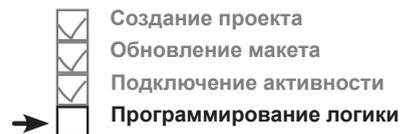
Нас интересует специализация View с идентификатором brands.

`R` — специальный класс Java, который позволяет получать ссылки на ресурсы в приложении.



РАССЛАБЬТЕСЬ

`R.java` генерируется за вас. Вы не сможете изменять код, находящийся в `R`, но полезно знать, что он существует.



Получив ссылку на объект View, вы можете вызывать его методы

Метод `findViewById()` предоставляет Java-версию компонента графического интерфейса. Это означает, что вы можете читать и задавать свойства компонента при помощи методов, предоставляемых классом Java. Давайте разберемся подробнее.

Назначение текста в компоненте TextView

Как вы уже видели, для получения ссылки на компонент надписи в Java используется синтаксис

```
TextView brands = (TextView) findViewById(R.id.brands);
```

При выполнении этой строки кода создается объект класса `TextView` с именем `brands`. После этого вы можете вызывать методы этого объекта `TextView`. Допустим, вы хотите, чтобы в надписи `brands` отображался текст “Gottle of geer”. Класс `TextView` содержит метод `setText()`, используемый для задания свойства `text`. Он используется следующим образом:

```
brands.setText("Gottle of geer");
```

← Объекту `TextView` с именем `brands` назначается текст “Gottle of geer”.

Получение выбранного значения в раскрывающемся списке

Вы также можете получить ссылку на раскрывающийся список; это делается практически так же, как для надписи. Снова используется метод `findViewById()`, но на этот раз результат приводится к типу `Spinner`:

```
Spinner color = (Spinner) findViewById(R.id.color);
```

Вы получаете объект `Spinner` и можете вызывать его методы. Например, вот как происходит получение текущего выбранного варианта в списке и преобразование его к типу `String`:

```
String.valueOf(color.getSelectedItem())
```

← Получает выбранный вариант в списке и преобразует его в `String`.

Конструкция

```
color.getSelectedItem()
```

возвращает обобщенный объект Java. Дело в том, что значения раскрывающегося списка не обязаны быть объектами `String` — это могут быть, например, изображения. В нашем случае известно, что значения представляют собой объекты `String`, поэтому мы используем метод `String.valueOf()` для преобразования выбранного варианта из `Object` в `String`.

Обновление кода активности

Вы уже знаете достаточно для того, чтобы написать часть кода метода `onClickFindBeer()`. Вместо того, чтобы писать весь необходимый код за один подход, начнем с получения варианта, выбранного в раскрывающемся списке, и отображения его в надписи.



Развлечения с магнитами

Кто-то написал метод `onClickFindBeer()` для нашей новой активности на холодильнике, заменив пустые места магнитами. К сожалению, от сквозняка магниты упали на пол. Сможете ли вы снова собрать код метода?

Метод должен получать вид пива, выбранный в раскрывающемся списке, и выводить его в надписи.

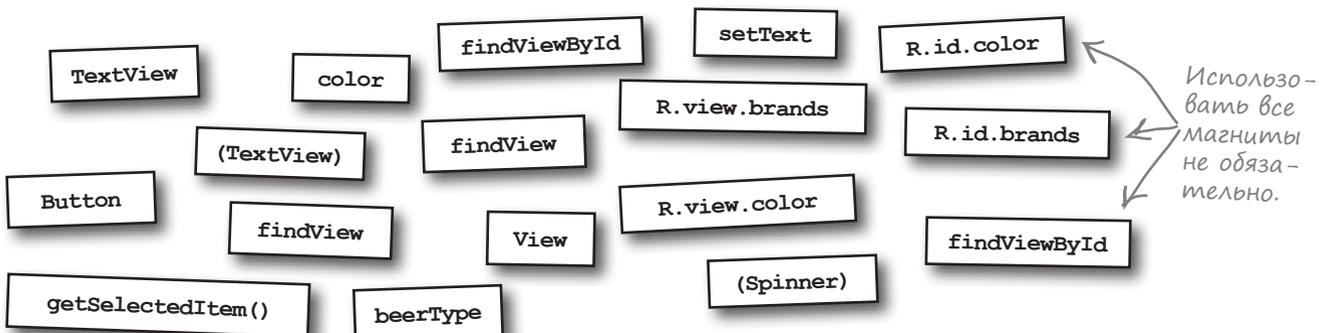
```
//Вызывается при щелчке на кнопке
public void onClickFindBeer(.....view) {

    //Получить ссылку на TextView
    ..... brands = ..... ( ..... );

    //Получить ссылку на Spinner
    Spinner ..... = ..... ( ..... );

    //Получить вариант, выбранный в Spinner
    String ..... = String.valueOf(color. ....);

    //Вывести выбранный вариант
    brands.....(beerType);
}
```





Развлечения с магнитами. Решение

Кто-то написал метод `onClickFindBeer()` для нашей новой активности на холодильнике, заменив пустые места магнитами. К сожалению, от сквозняка магниты упали на пол. Сможете ли вы снова собрать код метода?

Метод должен получать вид пива, выбранный в раскрывающемся списке, и выводить его в надписи.

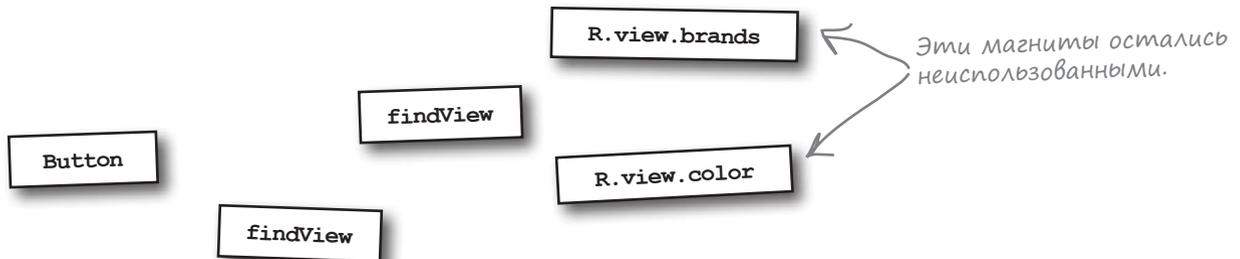
```
//Вызывается при щелчке на кнопке
public void onClickFindBeer( View ..... view) {

    //Получить ссылку на TextView
    TextView brands = (TextView) ..... findViewById ( R.id.brands );

    //Получить ссылку на Spinner
    Spinner color ..... = (Spinner) ..... findViewById ( R.id.color );

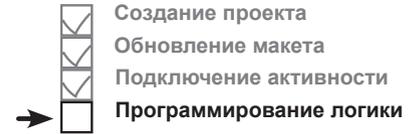
    //Получить вариант, выбранный в Spinner
    String beerType ..... = String.valueOf(color. .... getSelectedItem() ..... );

    //Вывести выбранный вариант
    brands. setText (beerType);
}
```



Первая версия активности

Наш хитроумный план заключается в том, чтобы строить активность поэтапно и тестировать ее на каждом этапе. В итоге активность должна получить значение, выбранное в раскрывающемся списке, вызвать метод вспомогательного класса Java, а затем вывести подходящие сорта пива. От первой версии требуется совсем немного: она должна убедиться в том, что выбранный вариант правильно читается из списка. Ниже приведен код активности вместе с методом, который мы собрали воедино на предыдущей странице. Внесите эти изменения в *FindBeerActivity.java* и сохраните файл:

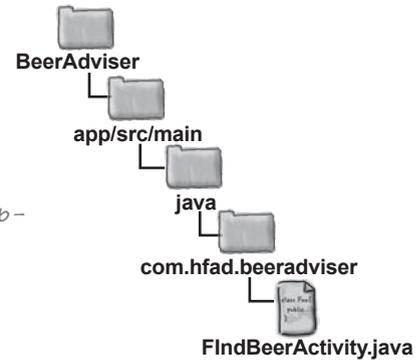


```
package com.hfad.beeradviser;

import android.os.Bundle;
import android.app.Activity;
import android.view.View;
import android.widget.Spinner;
import android.widget.TextView;

public class FindBeerActivity extends Activity {
```

Мы используем эти дополнительные классы.



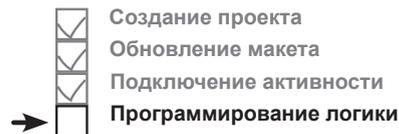
```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_find_beer);
    }
```

← Этот метод не изменялся.

```
    //Вызывается при щелчке на кнопке
    public void onClickFindBeer(View view) {
        //Получить ссылку на TextView
        TextView brands = (TextView) findViewById(R.id.brands);
        //Получить ссылку на Spinner
        Spinner color = (Spinner) findViewById(R.id.color);
        //Получить вариант, выбранный в Spinner
        String beerType = String.valueOf(color.getSelectedItem());
        //Вывести выбранный вариант
        brands.setText(beerType);
    }
}
```

findViewById возвращает объект *View*. Его необходимо преобразовать к правильному подтипу *View*.

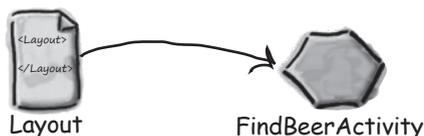
getSelectedItem возвращает *Object*. Этот объект необходимо преобразовать в *String*.



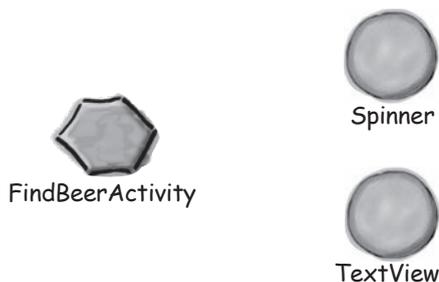
Что делает этот код

Прежде чем переходить к тестированию приложения, разберемся, что же делает этот код.

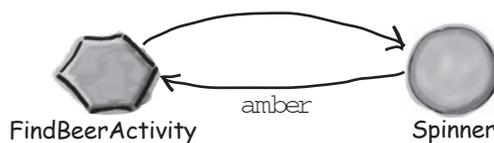
- 1 **Пользователь выбирает вид пива в раскрывающемся списке и щелкает на кнопке Find Beer. Это приводит к вызову метода `public void onClickFindBeer(View)` активности. Макет сообщает, какой метод активности должен вызываться при щелчке на кнопке, при помощи свойства `android:onClick` кнопки.**



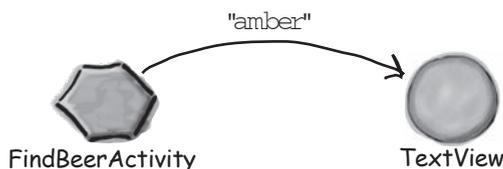
- 2 **Чтобы получить ссылки на компоненты графического интерфейса `TextView` и `Spinner`, активность вызывает метод `findViewById()`.**



- 3 **Активность получает текущее выбранное значение в раскрывающемся списке и преобразует его в `String`.**



- 4 **Затем активность задает свойство `text` компонента `TextView`, чтобы текущий вариант из списка отображался в надписи.**

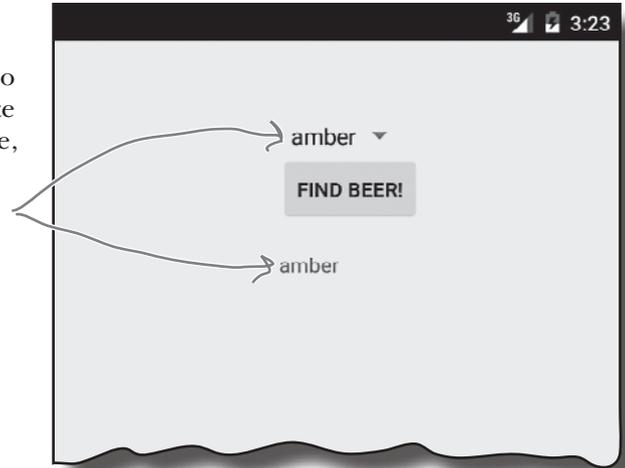




Тест-драйв

Внесите изменения в файл активности, сохраните его и запустите приложение. Теперь при щелчке на кнопке Find Beer значение варианта, выбранного в списке, выводится в надписи.

Выбранный вид пива отображается в надписи.



Часто задаваемые вопросы

В: Я добавил строку в файл `strings.xml`, но не вижу ее в `R.java`. Почему ее там нет?

О: Android Studio генерирует `R.java` при сохранении внесенных изменений. Если вы добавили ресурс, но не видите его в `R.java`, убедитесь в том, что изменения были сохранены. `R.java` также обновляется при построении приложения. Приложение строится при запуске, так что запуск приложения также приведет к обновлению `R.java`.

В: Похоже, наш раскрывающийся список состоит только из статических значений, хранящихся в массиве строк. Могу ли я изменять эти значения из программного кода?

О: Можете, но такой способ сложнее простого использования статического набора. Позднее мы покажем, как взять под полный контроль значения, отображаемые в компонентах (в том числе и в раскрывающемся списке).

В: Объект какого типа возвращается вызовом `getSelectedItem()`?

О: Он объявлен с типом `Object`. Так как для представления значений используется массив строк, фактическим возвращаемым значением в данном случае является `String`.

В: В данном случае? Почему не всегда?

О: С раскрывающимися списками можно выполнять и более сложные операции, чем простое отображение текста. Например, раскрывающийся список может отображать рядом с каждым значением значок. Тот факт, что `getSelectedItem()` возвращает `Object`, предоставляет большую гибкость.

В: Важен ли выбор имени `onClickFindBeer`?

О: Важно лишь то, чтобы имя метода в коде активности соответствовало имени, использованному в атрибуте `onClick` кнопки в макете.

В: Почему мы заменили код активности, сгенерированный Android Studio?

О: Такие среды разработки, как Android Studio, включают много служебных функций и инструментов, способных обеспечить значительную экономию времени. Они генерируют большой объем кода, и иногда этот код бывает полезным. Но во время изучения нового языка или среды разработки, на наш взгляд, лучше сосредоточиться на основах языка, а не на коде, который автоматически генерируется средой разработки. Такой подход поможет вам лучше разобраться в сути происходящего, а затем использовать полученные знания независимо от используемой среды разработки.

Построение вспомогательного класса Java

Как упоминалось в начале главы, приложение Beer Adviser решает, какие сорта пива рекомендовать пользователю, при помощи вспомогательного класса Java. Этот класс Java содержит самый обычный код Java, и ничто в нем не указывает на то, что этот код используется Android-приложением.

Спецификация вспомогательного класса Java

Вспомогательный класс Java должен удовлетворять следующим требованиям:

- ★ Класс должен принадлежать пакету `com.hfad.beeradviser`.
- ★ Классу должно быть присвоено имя `BeerExpert`.
- ★ Класс должен предоставлять один метод `getBrands()`, который получает желательный вид пива (в виде `String`) и возвращает контейнер `List<String>` с рекомендуемыми сортами.

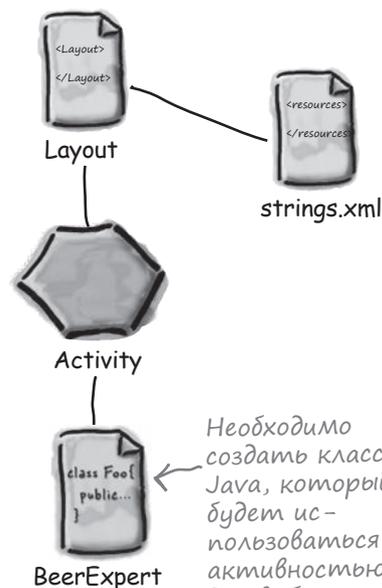
Построение и тестирование класса Java

Классы Java бывают чрезвычайно сложными, в них могут быть задействованы вызовы нетривиальной логики приложения. Либо постройте собственную версию класса, либо воспользуйтесь нашей готовой версией, приведенной ниже:

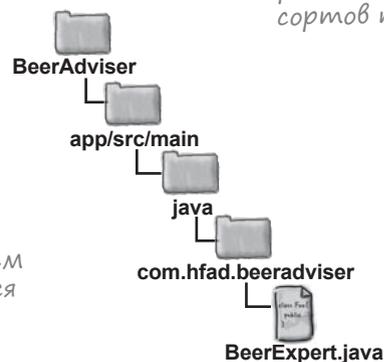
```
package com.hfad.beeradviser;
import java.util.ArrayList;
import java.util.List;
```

```
public class BeerExpert {
    List<String> getBrands(String color) {
        List<String> brands = new ArrayList<String>();
        if (color.equals("amber")) {
            brands.add("Jack Amber");
            brands.add("Red Moose");
        } else {
            brands.add("Jail Pale Ale");
            brands.add("Gout Stout");
        }
        return brands;
    }
}
```

Обычный код Java — в нем нет ничего, относящегося к специфике Android.



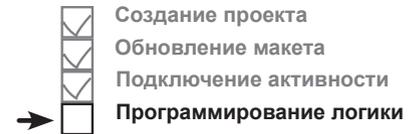
Необходимо создать класс Java, который будет использоваться активностью для выбора рекомендуемых сортов пива.



Задача!

Добавьте класс `BeerExpert` в свой проект. Выделите пакет `com.hfad.beeradviser` в папке `app/src/main/java` и выполните команду `File→New...→Java Class`. Новый класс будет создан в этом пакете.

Активность дополняется вызовом метода вспомогательного класса Java для получения НАСТОЯЩИХ рекомендаций



Во второй версии активности мы доработаем метод `onClickFindBeer()`, чтобы он вызывал метод класса `BeerExpert` для получения рекомендаций. Все необходимые изменения содержат только традиционный код Java. Вы можете попытаться написать код и запустить приложение самостоятельно, или же переверните страницу и повторяйте за нами.

Возьми в руку карандаш



Доработайте активность, чтобы она вызывала метод `getBrands()` класса `BeerExpert` и выводила результаты в надписи.

```
package com.hfad.beeradviser;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.View;
import android.widget.Spinner;
import android.widget.TextView;
import java.util.List;

public class FindBeerActivity extends Activity {
    private BeerExpert expert = new BeerExpert();
    ...
    //Вызывается при щелчке на кнопке
    public void onClickFindBeer(View view) {
        //Получить ссылку на TextView
        TextView brands = (TextView) findViewById(R.id.brands);
        //Получить ссылку на Spinner
        Spinner color = (Spinner) findViewById(R.id.color);

        //Получить вариант, выбранный в Spinner
        String beerType = String.valueOf(color.getSelectedItem());

    }
}
```

← Мы добавили эту строку за вас.

← Для получения рекомендаций необходимо использовать класс `BeerExpert`, поэтому мы добавили и эту строку.

↑
Попробуйте дописать метод `onClickFindBeer()`.



Возьми в руку карандаш

Решение

Доработайте активность, чтобы она вызывала метод `getBrands()` класса `BeerExpert` и выводила результаты в надписи.

```
package com.hfad.beeradviser;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.View;
import android.widget.Spinner;
import android.widget.TextView;
import java.util.List;

public class FindBeerActivity extends Activity {
    private BeerExpert expert = new BeerExpert();
    ...
    //Вызывается при щелчке на кнопке
    public void onClickFindBeer(View view) {
        //Получить ссылку на TextView
        TextView brands = (TextView) findViewById(R.id.brands);
        //Получить ссылку на Spinner
        Spinner color = (Spinner) findViewById(R.id.color);
        //Получить вариант, выбранный в Spinner
        String beerType = String.valueOf(color.getSelectedItem());
```

```
//Получить рекомендации от класса BeerExpert
```

```
List<String> brandsList = expert.getBrands(beerType);
```

← Получить контейнер List с сортами пива.

```
StringBuilder brandsFormatted = new StringBuilder();
```

← Построить String по данным из List.

```
for (String brand : brandsList) {
```

```
    brandsFormatted.append(brand).append('\n');
```

← Каждый сорт выводится с новой строки.

```
}
```

```
//Display the beers
```

```
brands.setText(brandsFormatted);
```

← Вывести результаты в надписи.

```
    }
}
```

↑
Реализация `BeerExpert` содержит только традиционный код Java, поэтому не беспокойтесь, если ваш код немного отличается от нашего.

Код активности, версия 2

Ниже приведена полная версия кода активности. Внесите изменения в свою версию `FindBeerActivity.java`, убедитесь в том, что класс `BeerExpert` включен в проект, и сохраните изменения:

```
package com.hfad.beeradviser;
```

```
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.View;
import android.widget.Spinner;
import android.widget.TextView;
```

```
import java.util.List; ← В программе используется
                        этот класс.
```

```
public class FindBeerActivity extends Activity {
    private BeerExpert expert = new BeerExpert();
```

```
@Override
```

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_find_beer);
}
```

```
//Вызывается при щелчке на кнопке
```

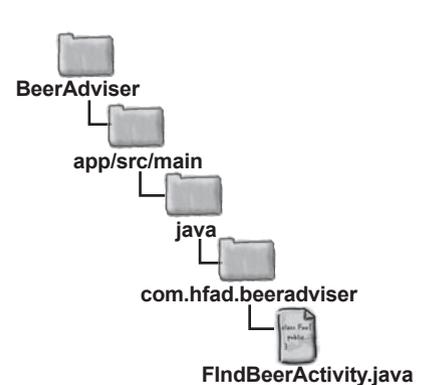
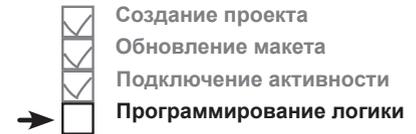
```
public void onClickFindBeer(View view) {
    //Получить ссылку на TextView
    TextView brands = (TextView) findViewById(R.id.brands);
    //Получить ссылку на Spinner
    Spinner color = (Spinner) findViewById(R.id.color);
    //Получить вариант, выбранный в Spinner
    String beerType = String.valueOf(color.getSelectedItem());
    //Получить рекомендации от класса BeerExpert
    List<String> brandsList = expert.getBrands(beerType);
    StringBuilder brandsFormatted = new StringBuilder();
    for (String brand : brandsList) {
        brandsFormatted.append(brand).append('\n');
    }
```

```
//Вывести сорта пива
```

```
brands.setText(brandsFormatted);
```

```
}
```

```
}
```



← Добавьте экземпляра `BeerExpert` как приватную переменную.

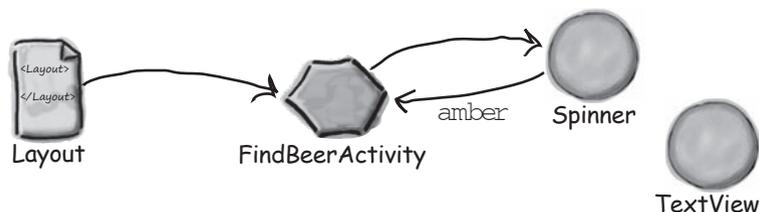
← Класс `BeerExpert` используется для получения набора сортов.

← Здесь строится объект `String`, в котором каждый сорт выводится с новой строки.

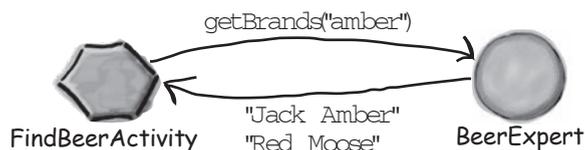
← Содержимое `String` отображается в надписи.

Что происходит при выполнении кода

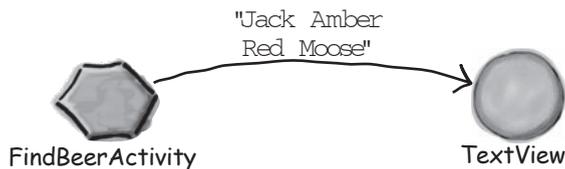
- 1 Когда пользователь щелкает на кнопке Find Beer, вызывается метод `onClickFindBeer()` из класса активности. Метод создает ссылку на раскрывающийся список и надпись и получает текущее значение, выбранное в списке.



- 2 Метод `onClickFindBeer()` вызывает метод `getBrands()` из класса `BeerExpert`, передавая ему вид пива, выбранный в раскрывающемся списке. Метод `getBrands()` возвращает список сортов пива.



- 3 Метод `onClickFindBeer()` форматирует список сортов и использует его для задания свойства `text` надписи.



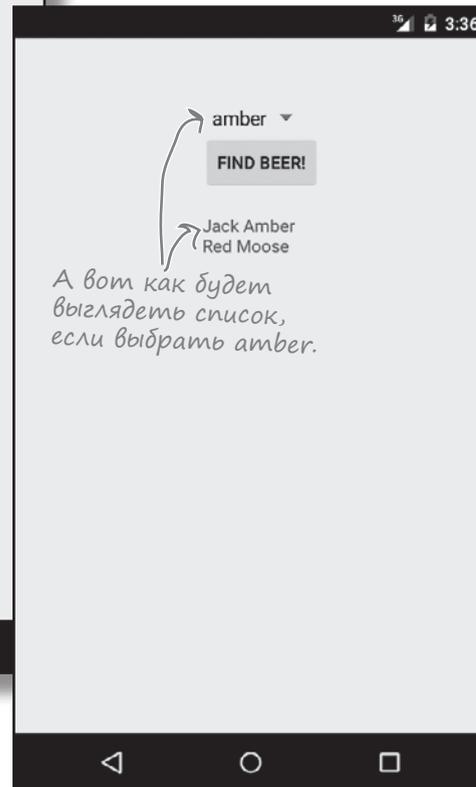


Тест-драйв

После того как приложение будет изменено, запустите его. Поэкспериментируйте, выбирая разные виды пива и щелкая на кнопке Find Beer.

построение интерактивных приложений

- Создание проекта
- Обновление макета
- Подключение активности
- Программирование логики



Когда вы выбираете разные виды пива и щелкаете на кнопке Find Beer, приложение при помощи класса BeerExpert выдает подборку подходящих сортов.



Ваш инструментарий Android

Глава 2 осталась позади, а ваш инструментарий пополнился средствами построения интерактивных приложений Android.

Весь код для этой главы можно загрузить по адресу <https://tinyurl.com/HeadFirstAndroid>.

КЛЮЧЕВЫЕ МОМЕНТЫ



- Элемент `Button` используется для добавления кнопки.
- Элемент `Spinner` используется для добавления раскрывающегося списка.
- Все компоненты графического интерфейса наследуют от класса `Android View`.
- Массив строковых значений создается конструкцией следующего вида:

```
<string-array name="array">
  <item>string1</item>
  ...
</string-array>
```

- Для обращения к `string-array` в макете используется синтаксис:

```
"@array/array_name"
```

- Чтобы при щелчке на кнопке вызывался метод, включите в макет следующий атрибут:

```
android:onClick="clickMethod"
```

При этом в активности должен существовать соответствующий метод:

```
public void clickMethod(View view) {
}
```

- Класс `R.java` генерируется средой. Он позволяет получать ссылки на макеты, компоненты графического интерфейса, строки и другие ресурсы в коде Java.
- Метод `findViewById()` возвращает ссылку на компонент.
- Метод `setText()` задает текст компонента.
- Метод `getSelectedItem()` возвращает вариант, выбранный в раскрывающемся списке.
- Чтобы добавить вспомогательный класс в проект Android, выполните команду `File→New...→Java Class`.

3 Множественные активности и интеннты

Предъявите свой интеннт

Я отправила интеннт с вопросом, кто может обработать мой ACTION_CALL, и мне сразу предложили целую кучу кандидатов.



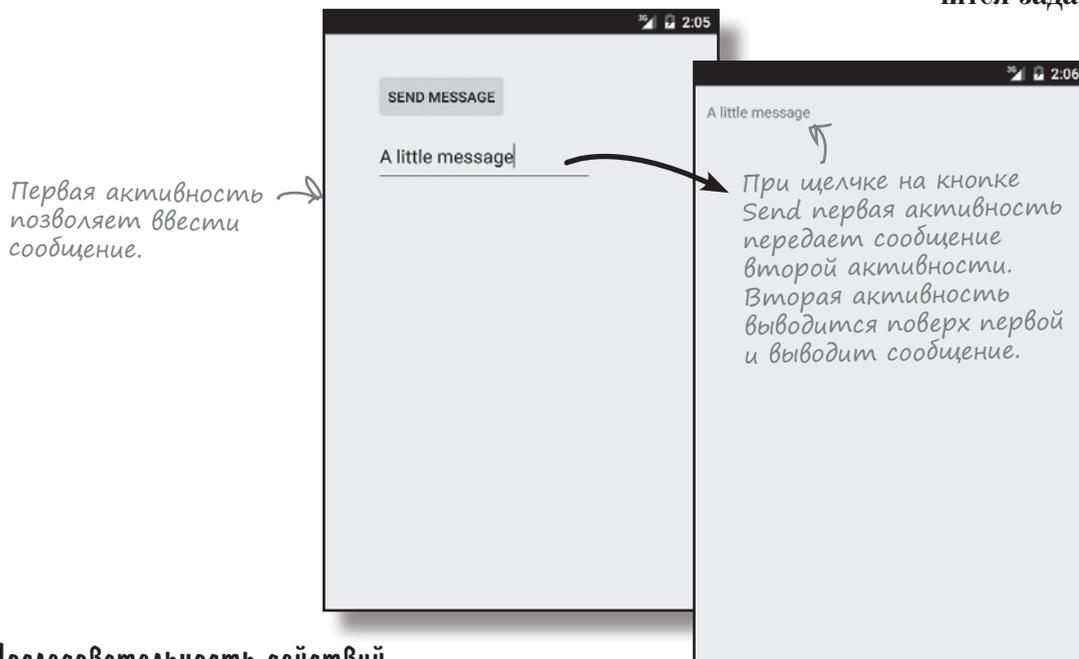
Для большинства приложений одной активности недостаточно. До настоящего момента мы рассматривали приложения с одной активностью; для простых приложений это нормально. Однако в более сложной ситуации одна активность попросту не справляется со всеми делами. Мы покажем вам, **как строить приложения с несколькими активностями** и как организовать взаимодействие между активностями с использованием **интеннтов**. Также вы узнаете, как использовать интеннты **за пределами приложения** и как **выполнять действия при помощи активностей других приложений на вашем устройстве**. Внезапно перед вами открываются совершенно новые перспективы...

Приложение может содержать несколько активностей

Ранее в книге мы говорили, что активность — одна четко определенная операция, которая может выполняться пользователем, — например, отображение списка рецептов. В очень простом приложении этого может быть достаточно.

Но обычно пользователю требуется выполнять *более* одной операции — например, не только выводить список рецептов, но и добавлять их в базу данных. В таких случаях в приложении используются разные активности: одна для отображения списка рецептов, а другая для добавления рецепта. Чтобы понять, как работает эта система, лучше всего опробовать ее в деле. Мы построим приложение с двумя активностями. Первая активность позволяет ввести текст сообщения. Щелчок на кнопке в первой активности запускает вторую активность, которой передается сообщение. Далее вторая активность выводит полученное сообщение.

Активность — одна целенаправленная операция, которая может выполняться пользователем. Если объединить несколько активностей для выполнения чего-то более сложного, получится задача.



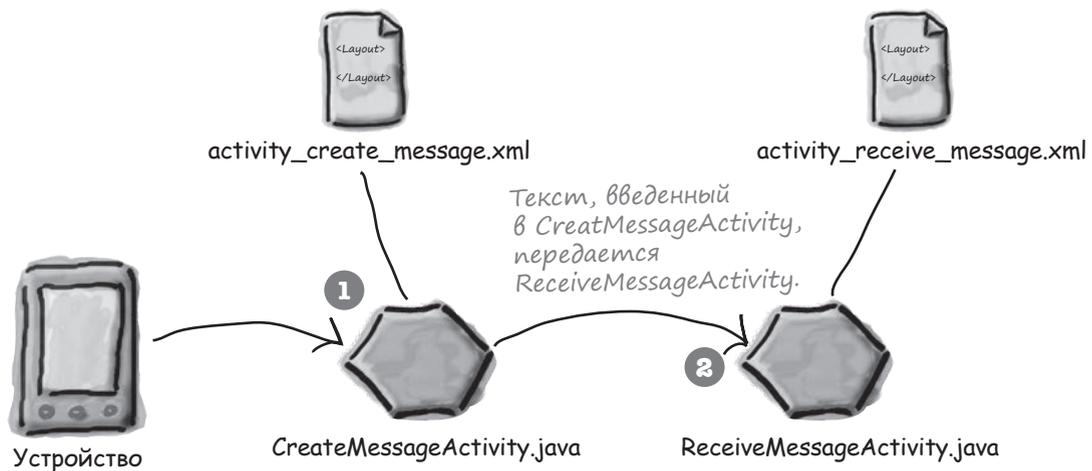
Последовательность действий

- 1 Создание базового приложения с одной активностью и макетом.
- 2 Добавление второй активности и макета.
- 3 Организация вызова второй активности из первой.
- 4 Организация передачи данных из первой активности во вторую.

Структура приложения

Приложение состоит из двух активностей и двух макетов.

- 1 **В начале работы приложения запускается активность `CreateMessageActivity`.**
Эта активность использует макет `activity_create_message.xml`.
- 2 **Пользователь щелкает на кнопке в `CreateMessageActivity`.**
Кнопка запускает активность `ReceiveMessageActivity`, которая использует макет `activity_receive_message.xml`.



Создание проекта

Проект приложения создается точно так же, как и в предыдущих главах. Создайте в Android Studio новый проект для приложения с именем “Messenger” и именем пакета `com.hfad.messenger`. Выберите минимальный уровень API 15, чтобы приложение работало на большинстве устройств. Чтобы ваш код не отличался от нашего, создайте пустую активность с именем “`CreateMessageActivity`” и макет с именем “`activity_create_message`”.

На следующей странице мы отредактируем макет активности.



Создание 1-й активности
Создание 2-й активности
Вызов 2-й активности
Передача данных

Обновление макета

Перед вами разметка XML из файла `activity_create_message.xml`. Мы убрали элемент `<TextView>`, автоматически сгенерированный Android Studio, и заменили его элементами `<Button>` и `<EditText>`. Элемент `<EditText>` создает текстовое поле с возможностью редактирования, которое может использоваться для ввода данных.

Приведите файл `activity_create_message.xml` в соответствие со следующей разметкой:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    tools:context=".CreateMessageActivity" >
```

```
<Button
    android:id="@+id/send"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentTop="true"
    android:layout_marginLeft="36dp"
    android:layout_marginTop="21dp"
    android:onClick="onSendMessage"
    android:text="@string/send" />
```

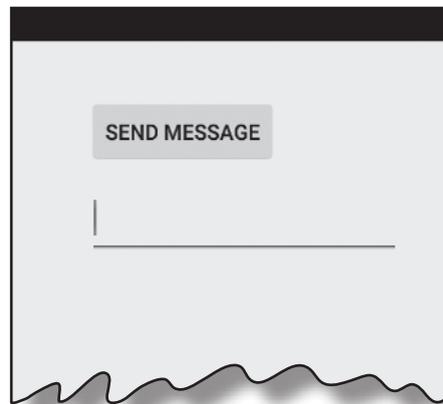
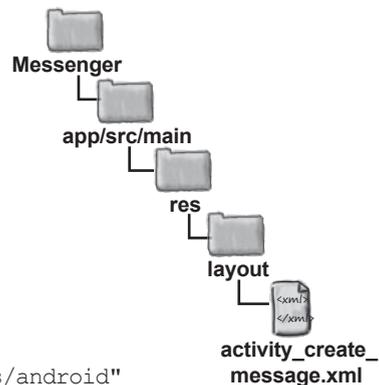
Заменили элемент `<TextView>`, сгенерированный Android Studio, элементами `<Button>` и `<EditText>`.

Щелчок на кнопке запускает метод `onSendMessage()` из активности.

```
<EditText
    android:id="@+id/message"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/send"
    android:layout_below="@+id/send"
    android:layout_marginTop="18dp"
    android:ems="10" />
```

Строковый ресурс.

Описывает ширину текстового поля `<EditText>`. Ширина поля должна быть достаточной для размещения 10 букв «М».



Элемент `<EditText>` определяет текстовое поле для ввода и редактирования текста. Класс наследует от того же класса Android View, что и классы других компонентов графического интерфейса, встречавшиеся нам до настоящего момента.

Обновление strings.xml...

Добавленной нами кнопке назначается текстовое значение `@string/send`. Это означает, что мы должны добавить в `strings.xml` строку с именем "send" и присвоить ей значение — текст, который должен отображаться на кнопке. Выполните следующие действия:

```
...
<string name="send">Send Message</string>
...
```

...и добавление метода в активность

Следующая строка элемента `<Button>`

```
android:onClick="onSendMessage"
```

означает, что метод `onSendMessage()` активности будет срабатывать при щелчке на кнопке. Давайте добавим этот метод в активность.

Откройте файл `CreateMessageActivity.java` и замените код, сгенерированный Android Studio, следующим:

```
package com.hfad.messenger;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;

public class CreateMessageActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_create_message);
    }

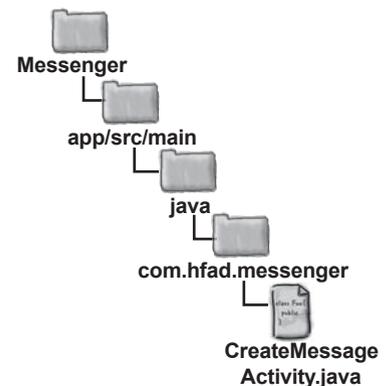
    //Вызвать onSendMessage() при щелчке на кнопке
    public void onSendMessage(View view) {
    }
}
```

Добавьте новый строковый ресурс с именем `send`. Мы присвоили ему значение `Send Message`, чтобы на кнопке выводился текст "Send Message".

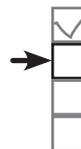
Мы заменяем код, который был сгенерирован Android Studio, так как большая часть этого кода не обязательна.

Метод `onCreate()` вызывается при создании активности.

Этот метод будет вызываться при щелчке на кнопке. Мы допишем тело метода далее в этой главе.



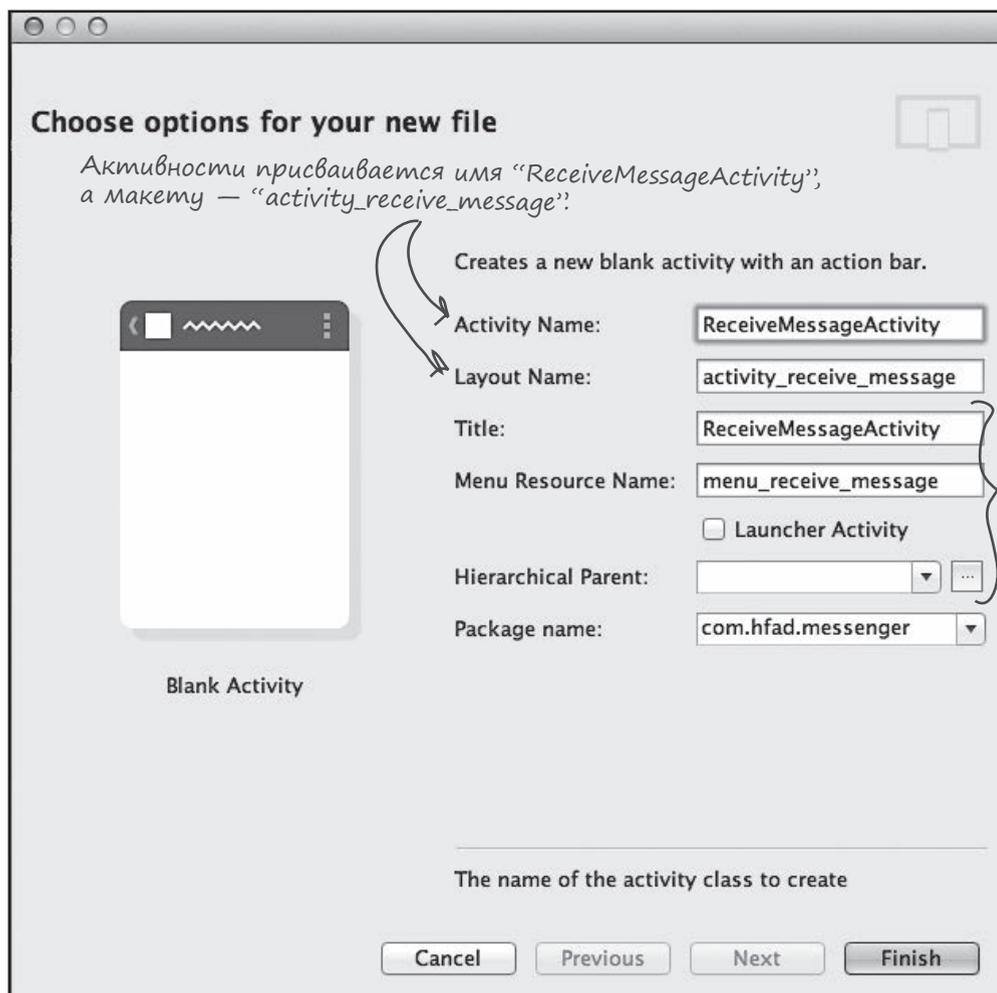
Итак, первая активность готова; переходим ко второй.



Создание второй активности и макета

В Android Studio имеется мастер для создания новых активностей и макетов в приложениях. По сути это упрощенная версия мастера, используемого при создании приложений; используйте его каждый раз, когда вам потребуется создать новую активность. Чтобы создать новую активность, выполните команду File → New → Activity и выберите вариант Blank Activity. На экране появляется окно, в котором вы сможете выбрать параметры новой активности.

Каждой создаваемой новой активности и макету необходимо присвоить имя. Задайте для активности имя “ReceiveMessageActivity”, а для макета — имя “activity_receive_message”. Убедитесь в том, что пакету присвоено имя “com.hfad.messenger”. Подтвердите остальные значения по умолчанию, а когда все будет готово — щелкните на кнопке Finish.

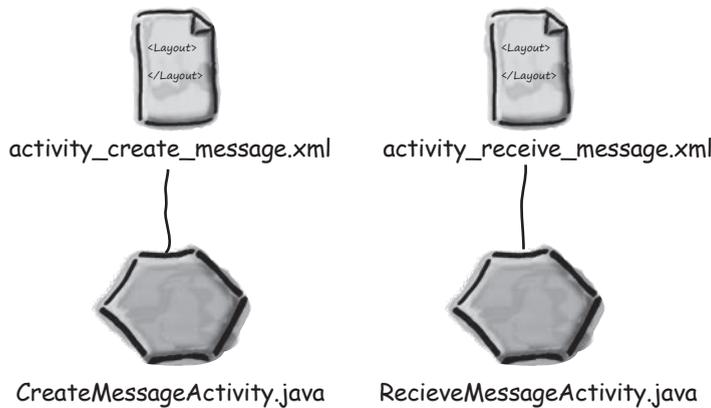


Что произошло?

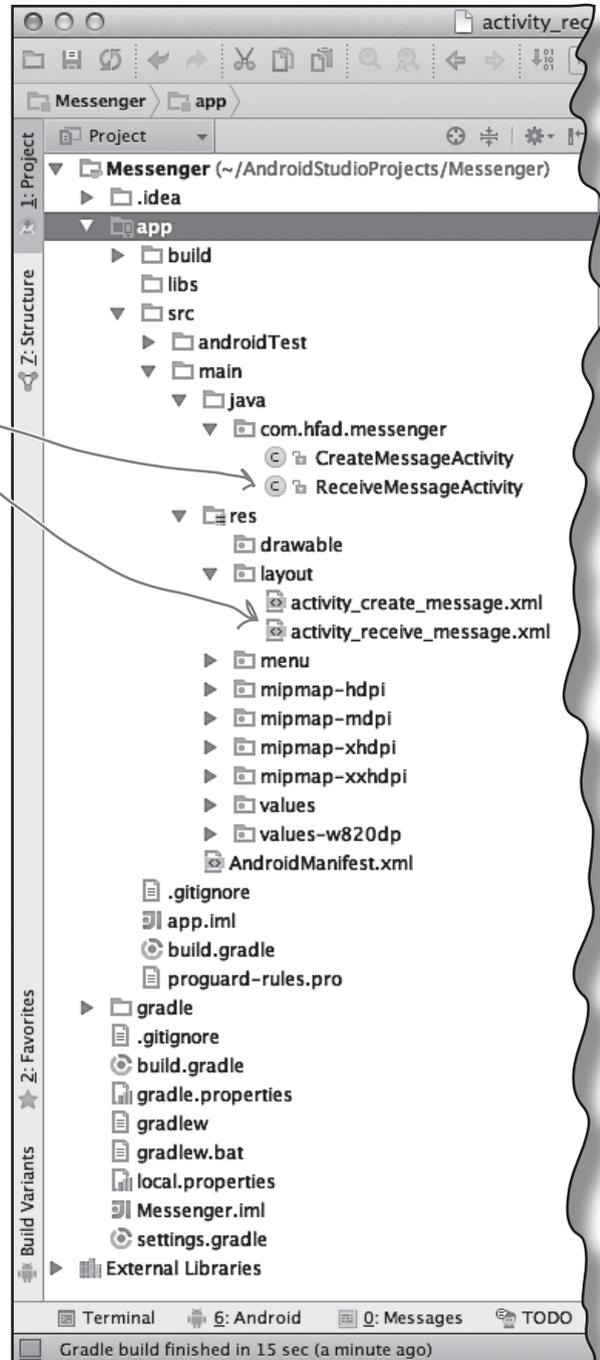
Когда вы щелкнули на кнопке Finish, Android Studio создает для вас новый файл активности вместе с новым макетом. Заглянув на панель структуры проекта, вы увидите, что в папке `app/src/main/java` появился новый файл с именем `ReceiveMessageActivity.java`, а в папке `app/src/main/res/layout` — файл с именем `activity_receive_message.xml`.

Новая активность и макет, которые мы только что создали. Теперь приложение содержит две активности и два макета.

Каждая активность использует свой отдельный макет. `CreateMessageActivity` использует макет `activity_create_message.xml`, а `ReceiveMessageActivity` — макет `activity_receive_message.xml`.



Android Studio также незаметно изменяет конфигурацию приложения в файле с именем `AndroidManifest.xml`. Давайте разберемся повнимательнее.





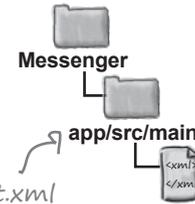
Создание 1-й активности
 Создание 2-й активности
 Вызов 2-й активности
 Передача данных

Знакомьтесь: файл манифеста Android

Каждое Android-приложение должно содержать файл с именем *AndroidManifest.xml*. Вы можете найти его в папке *app/src/main* вашего проекта. Файл *AndroidManifest.xml* содержит важнейшую информацию о приложении: какие активности оно содержит, какие библиотеки ему необходимы и другие объявления. Android создает этот файл при создании приложения. Если вы вспомните настройки, которые были выбраны при создании проекта, то часть содержимого этого файла может показаться знакомой.

Наш экземпляр *AndroidManifest.xml* выглядит так:

Файл *AndroidManifest.xml* находится в этой папке.



```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.hfad.messenger" >
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".CreateMessageActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".ReceiveMessageActivity"
            android:label="@string/title_activity_receive_message" >
        </activity>
    </application>
</manifest>
```

← Имя пакета, которое мы указали.

Android Studio означает приложению значок по умолчанию. Мы еще вернемся к этой теме позднее.

← Тема влияет на оформление приложения. Этот вопрос тоже будет рассмотрен позднее.

Первая активность, Create Message Activity.

Вторая активность, Receive Message Activity.

Будьте осторожны!
 Если вы разрабатываете Android-приложение без IDE, файл придется создать вручную.

← Это означает, что активность является главной активностью приложения.

↑ А это означает, что активность может использоваться для запуска приложения.

↑ Android Studio добавляет эти строки при добавлении второй активности.

Каждая активность должна быть объявлена

Все активности должны быть объявлены в файле *AndroidManifest.xml*. Если активность не объявлена в файле, то система не будет знать о его существовании. А если система не знает об активности, то активность не будет выполняться.

Активности объявляются в манифесте включением элемента `<activity>` в элемент `<application>`. Собственно, каждая активность в приложении должна иметь соответствующий элемент `<activity>`. Общий формат объявления выглядит так:

```
<application
...
...>
<activity
  android:name="имя_класса_активности"
  android:label="@string/метка_активности"
  ...
  ...>
...
</activity>
...
</application>
```

← Каждая активность должна быть объявлена в элементе `<application>`.

← Эта строка обязательна.

← Эта строка не обязательна, но Android Studio генерирует ее за нас.

← Активность также может обладать другими свойствами.



Активность

Следующая строка является обязательной и используется для определения имени класса активности:

```
android:name="имя_класса_активности"
```

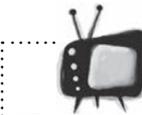
`имя_класса_активности` — имя класса с префиксом “.”, в данном случае `.ReceiveMessageActivity`. Имя класса снабжается префиксом “.”, потому что Android строит *полное* имя класса, объединяя имя класса с именем пакета.

Следующая строка не является обязательной; она задает метку активности, удобную для пользователя:

```
android:label="@string/метка_активности"
```

Метка выводится в верхней части экрана при выполнении активности. Если убрать ее, то Android будет использовать вместо нее имя приложения.

В объявление активности также могут включаться и другие свойства — например, разрешения безопасности или возможность использования активностями других приложений.



Будьте осторожны!

Вторая активность была объявлена

на автоматически, потому что она была добавлена с использованием мастера Android Studio.

Если вы добавляете новые активности вручную, то вам придется редактировать файл *AndroidManifest.xml* самостоятельно. Если вы используете другую среду разработки, может оказаться, что эта среда не включает информацию в манифест.

Интент — разновидность сообщения



- Создание 1-й активности
- Создание 2-й активности
- Вызов 2-й активности
- Передача данных

К настоящему моменту мы создали приложение с двумя активностями, каждая из которых имеет собственный макет. При запуске приложения будет выполняться наша первая активность, `CreateMessageActivity`. Следующий шаг — заставить `CreateMessageActivity` вызывать `ReceiveMessageActivity`, когда пользователь щелкает на кнопке `Send Message`.

Чтобы запустить одну активность из другой, воспользуйтесь **интентом**. Интент можно рассматривать как своего рода «намерение выполнить некую операцию». Это разновидность сообщений, позволяющая связать разнородные объекты (например, активности) на стадии выполнения. Если одна активность хочет запустить другую, она отправляет для этого интент системе Android. Android запускает вторую активность и передает ей интент.

Процедура создания и отправки интента состоит всего из двух строк кода. Для начала создайте интент:

```
Intent intent = new Intent(this, Target.class);
```

При создании интента указывается активность, которая должна его получить, — словно адрес, написанный на конверте.

Первый параметр сообщает Android, от какого объекта поступил интент; для обозначения текущей активности используется ключевое слово `this`. Во втором параметре передается имя класса активности, которая должна получить интент.

После того как интент будет создан, он передается Android следующим вызовом:

```
startActivity(intent);
```

`startActivity()` запускает активность, указанную в интенте...



Этот вызов приказывает Android запустить активность, определяемую интентом. При получении интента Android убеждается в том, что все правильно, и приказывает активности запуститься. Если найти активность не удалось, инициируется исключение `ActivityNotFoundException`.



Использование интента для запуска второй активности

А теперь применим эту схему на практике и используем интент для вызова `ReceiveMessageActivity`. Активность должна запускаться, когда пользователь щелкает на кнопке `Send Message`, поэтому мы добавляем две строки кода в метод `onSendMessage()`.

Внесите изменения, выделенные жирным шрифтом:

```
package com.hfad.messenger;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class CreateMessageActivity extends Activity {

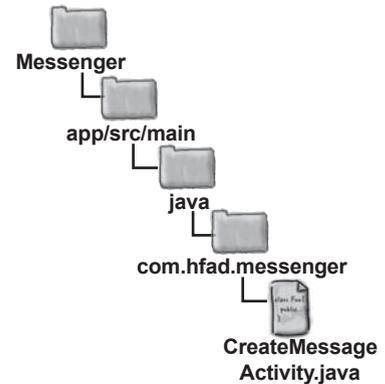
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_create_message);
    }

    // Вызвать onSendMessage() при щелчке на кнопке
    public void onSendMessage(View view) {
        Intent intent = new Intent(this, ReceiveMessageActivity.class);
        startActivity(intent);
    }
}
```

Необходимо импортировать класс интента `android.content.Intent`, так как он используется в `onSendMessage()`.

Этот метод не изменился.

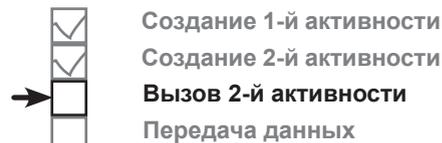
Запустить активность `ReceiveMessageActivity`.



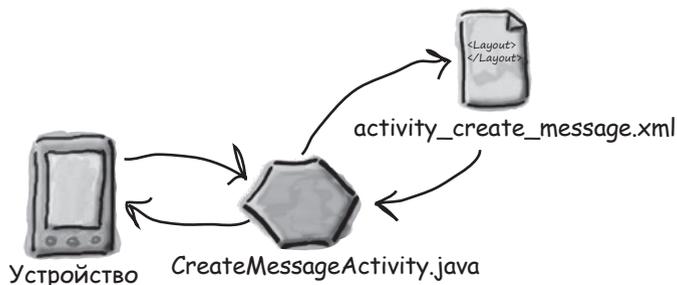
Что же произойдет, если запустить приложение?

Что происходит при запуске приложения

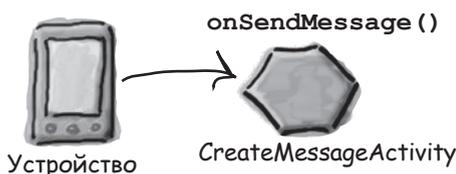
Прежде чем тестировать приложение, еще раз посмотрим, как будет функционировать версия, построенная нами к настоящему моменту:



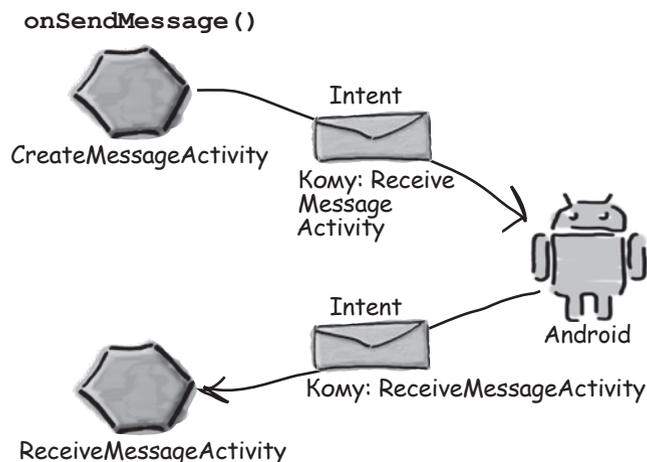
- 1** При запуске приложения начинает работать его главная активность `CreateMessageActivity`. В конфигурации запускаемой активности указано, что она использует макет `activity_create_message.xml`. Этот макет отображается в новом окне.



- 2** Пользователь щелкает на кнопке. Метод `onSendMessage()` класса `CreateMessageActivity` реагирует на щелчок.

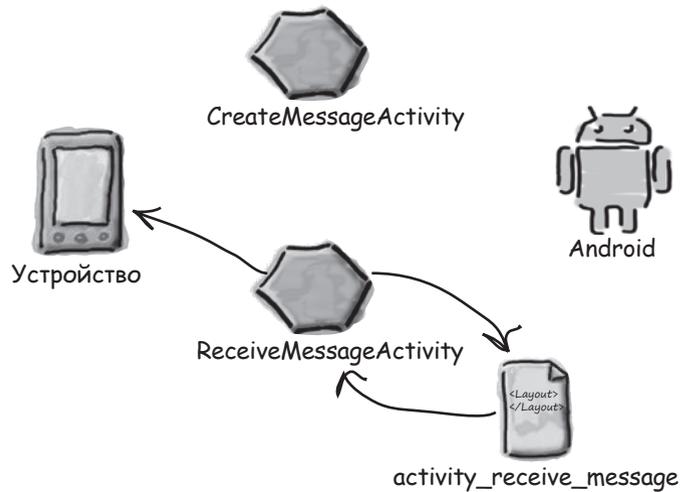


- 3** Метод `onSendMessage()` приказывает Android запустить активность `ReceiveMessageActivity` при помощи интента. Android убеждается в том, что интент правилен, и после этого приказывает `ReceiveMessageActivity` запуститься.



История продолжается...

- 4 При запуске активность `ReceiveMessageActivity` сообщает, что она использует макет `activity_receive_message.xml`; этот макет отображается в новом окне.



Тест-драйв

Сохраните изменения и запустите приложение. Активность `CreateMessageActivity` запускается, а при щелчке на кнопке `Send Message` она запускает `ReceiveMessageActivity`.



Создание 1-й активности
Создание 2-й активности
Вызов 2-й активности
Передача данных



Передача текста второй активности

К настоящему моменту мы запрограммировали активность `CreateMessageActivity` так, чтобы она запускала `ReceiveMessageActivity` при щелчке на кнопке `SendMessage`. Теперь необходимо обеспечить передачу текста из `CreateMessageActivity` в `ReceiveMessageActivity`, чтобы активность `ReceiveMessageActivity` могла вывести полученный текст. Для этого необходимо сделать три вещи:

- 1 Изменить макет `activity_receive_message.xml` так, чтобы он мог использоваться для вывода текста. Сейчас он представляет собой макет по умолчанию, сгенерированный мастером.
- 2 Обновить активность `CreateMessageActivity.xml`, чтобы она получала введенный пользователем текст. Этот текст должен быть добавлен в интент перед его отправкой.
- 3 Обновить код `ReceiveMessageActivity.java`, чтобы он выводил текст, отправленный в интенте.

Начнем с макета

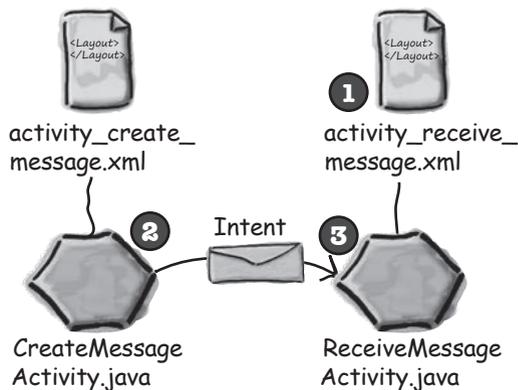
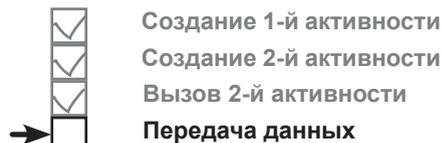
Вот как выглядит макет `activity_receive_message.xml`, сгенерированный Android Studio:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    tools:context="com.hfad.messenger.ReceiveMessageActivity">

    <TextView
        android:text="@string/hello_world"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</RelativeLayout>
```

← Надпись, которая сейчас присутствует в макете.



Упражнение

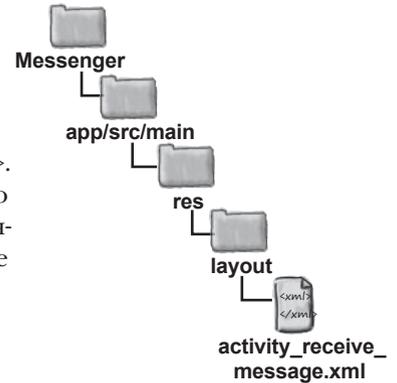
В макет необходимо внести пару изменений. Во-первых, нужно присвоить элементу `<TextView>` идентификатор `"message"`, чтобы к нему можно было обращаться из кода активности, а во-вторых, нужно заблокировать вывод текста `"Hello world!"`. Как изменить макет? Попробуйте сами, прежде чем переходить к следующей странице.

Обновление свойств надписи

В макет необходимо внести пару изменений.

Прежде всего необходимо назначить идентификатор элементу `<TextView>`. Идентификаторы должны назначаться всем компонентам графического интерфейса, с которыми вы хотите работать в коде активности, — идентификатор используется для обращения к компоненту из кода Java. Также необходимо отменить вывод в макете текста “Hello world!”.

Чтобы внести оба изменения, приведите макет к следующему виду:



```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    tools:context="com.hfad.messenger.ReceiveMessageActivity">
    <TextView
        android:id="@+id/message"
        android:text="@string/hello_world"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</RelativeLayout>
```

В этой строке элементу `<TextView>` назначается идентификатор `message`.

Удалите строку, в которой атрибуту `text` присваивается значение `@string/hello_world`.

Вместо того, чтобы удалять строку

```
android:text="@string/hello_world"
```

также можно было бы в файле `strings.xml` присвоить строковому ресурсу `hello_world` пустое значение. Мы не стали так делать, потому что единственный текст, который будет выводиться в этой надписи — это сообщение, передаваемое из `CreateMessageActivity`. Итак, после внесения изменений в макет можно переходить к работе с активностями.

Часть задаваемые вопросы

В: Обязательно ли использовать интен­ты? Разве я не могу просто создать экземпляр второй активности в коде первой активности?

О: Хороший вопрос, но... Нет, в Android так дела не делаются. Одна из причин заключается в том, что при передаче интен­тов Android знает последовательность запуска активностей. В частности, при нажатии кнопки Back на вашем устройстве Android будет точно знать, в какую точку следует вернуться.

putExtra() включает в интент дополнительную информацию

Вы уже видели, как создать новый интент командой

```
Intent intent = new Intent(this, Target.class);
```

В интент также можно добавить дополнительную информацию, которая должна передаваться получателю. В этом случае активность, получившая интент, сможет на него как-то среагировать. Для этого используется метод putExtra ()

```
intent.putExtra("сообщение", значение);
```

где сообщение – имя ресурса для передаваемой информации, а значение – само значение. Перегрузка метода putExtra () позволяет передавать значение многих возможных типов. Например, это может быть примитив (скажем, boolean или int), массив примитивов или String. Многократные вызовы putExtra () позволяют включить в интент несколько экземпляров дополнительных данных. Если вы решите действовать так, проследите за тем, чтобы каждому экземпляру было присвоено уникальное имя.

Как получить дополнительную информацию из интента

Впрочем, это еще не все. Когда Android приказывает ReceiveMessageActivity запуститься, активность должна каким-то образом получить дополнительную информацию, которая была отправлена CreateMessageActivity системе Android в интенте.

Существует пара удобных методов, которые помогают в решении этой задачи. Первый метод:

```
getIntent();
```

getIntent () возвращает интент, запустивший активность; из полученного интента можно прочитать любую информацию, отправленную вместе с ним. Конкретный способ чтения зависит от типа отправленной информации. Например, если вы знаете, что интент включает строковое значение с именем "message", используйте следующий вызов:

```
Intent intent = getIntent();
```

```
String string = intent.getStringExtra("message");
```

Конечно, из интента можно читать не только строковые значения. Например, вызов

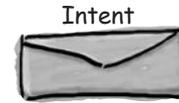
```
int intNum = intent.getIntExtra("name", default_value);
```

может использоваться для получения значения int с именем name. Параметр default_value указывает, какое значение int должно использоваться по умолчанию.



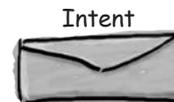
- Создание 1-й активности
- Создание 2-й активности
- Вызов 2-й активности
- Передача данных

Вызов putExtra() включает дополнительную информацию в отправляемое сообщение.



Кому: ReceiveMessageActivity
message: "Hello!"

В аргументе value могут передаваться значения разных типов. Полный перечень этих типов приведен в документации Google Android. Кроме того, Android Studio отображает список вариантов во время ввода кода.



Кому: ReceiveMessageActivity
message: "Hello!"

Получить переданную с интентом строку с именем "message".

У бассейна



Выловите из бассейна фрагменты кода и расставьте их в пустых строках *CreateMessageActivity.java*. Каждый фрагмент кода может использоваться только один раз, при этом все фрагменты использовать не обязательно. Ваша **задача** — сделать так, чтобы активность прочитала текст сообщения из `<EditText>` и добавила его в ИНТЕНТ.

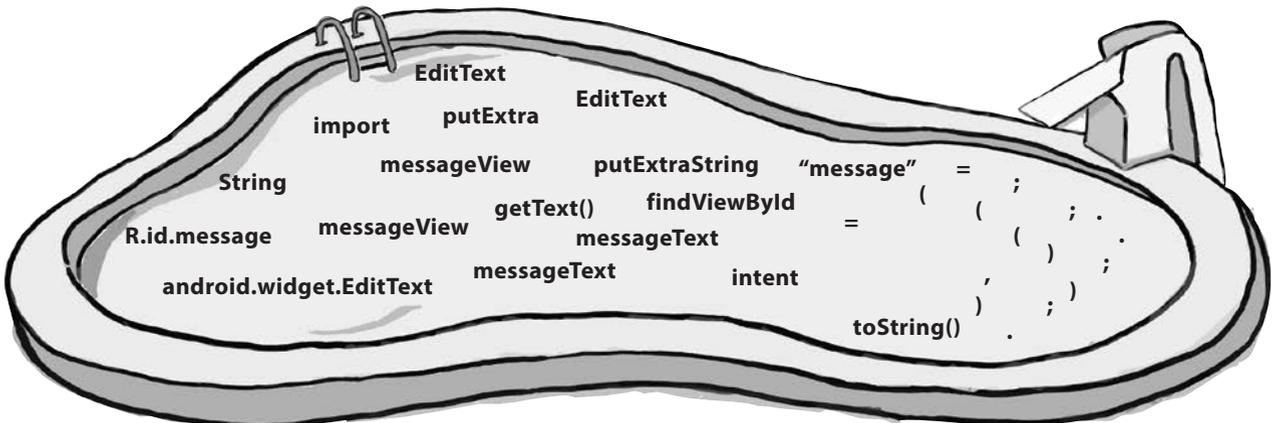
```
package com.hfad.messenger;

import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.view.View;
.....

public class CreateMessageActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_create_message);
    }

    //Вызвать onSendMessage() при щелчке на кнопке
    public void onSendMessage(View view) {
        .....
        .....
        Intent intent = new Intent(this, ReceiveMessageActivity.class);
        .....
        startActivity(intent);
    }
}
```



```
package com.hfad.messenger;

import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.view.View;
import android.widget.EditText;

public class CreateMessageActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_create_message);
    }

    //Вызвать onSendMessage() при щелчке на кнопке
    public void onSendMessage(View view) {
        EditText messageView = (EditText) findViewById(R.id.message);
        String messageText = messageView.getText().toString();
        Intent intent = new Intent(this, ReceiveMessageActivity.class);
        intent.putExtra("message", messageText);
        startActivity(intent);
    }
}
```

Необходимо импортировать класс EditText.



У бассейна. Решение

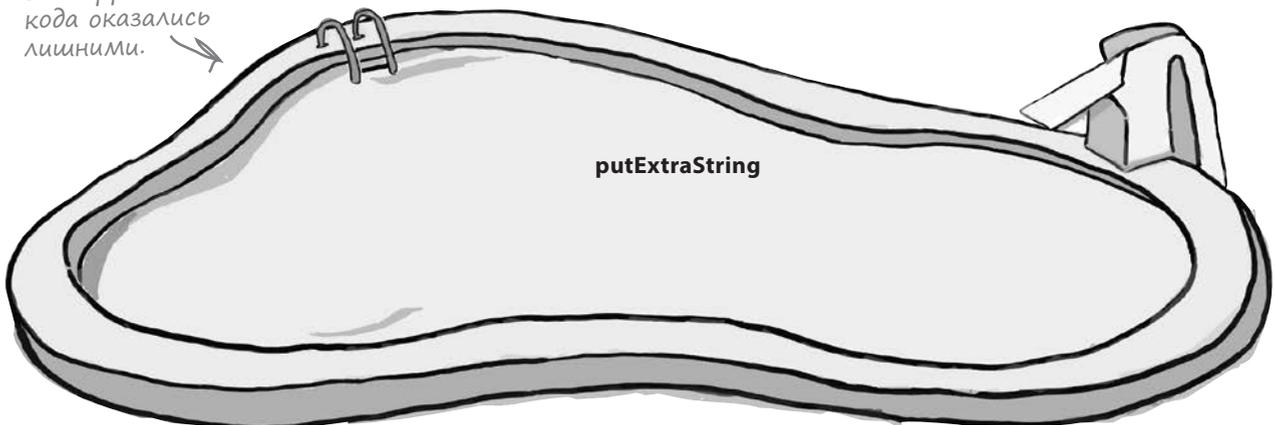
Выловите из бассейна фрагменты кода и расставьте их в пустых строках `CreateMessageActivity.java`. Каждый фрагмент кода может использоваться только один раз, при этом все фрагменты использовать не обязательно.

Ваша **задача** — сделать так, чтобы активность прочитала текст сообщения из `<EditText>` и добавила его в ИНТЕНТ.

Получить текст из текстового поля с идентификатором `message`.

Добавить текст в интент под именем "message".

Эти фрагменты кода оказались лишними.



Обновление кода CreateMessageActivity

Мы обновили код `CreateMessageActivity.java`, чтобы активность получала текст, введенный пользователем на экране, и добавляла его в интеннт. Ниже приведен полный код (не забудьте изменить свою версию и включить изменения, выделенные жирным шрифтом):

```

package com.hfad.messenger;

import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.view.View;
import android.widget.EditText;

public class CreateMessageActivity extends Activity {

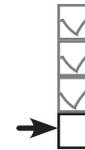
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_create_message);
    }

    //Вызвать onSendMessage() при щелчке на кнопке
    public void onSendMessage(View view) {
        EditText messageView = (EditText) findViewById(R.id.message);
        String messageText = messageView.getText().toString();
        Intent intent = new Intent(this, ReceiveMessageActivity.class);
        intent.putExtra(ReceiveMessageActivity.EXTRA_MESSAGE, messageText);
        startActivity(intent);
    }
}

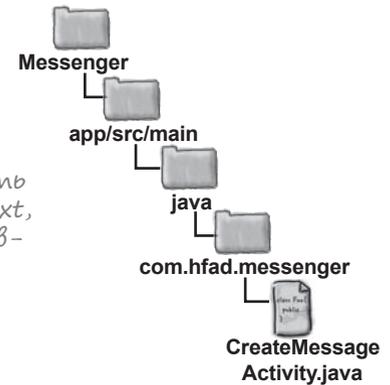
```

← Необходимо импортировать класс `android.widget.EditText`, используемый в коде активности.

↑
Запустим `ReceiveMessageActivity` при помощи интеннта.



- Создание 1-й активности
- Создание 2-й активности
- Вызов 2-й активности
- Передача данных



Получить текущий текст из `EditText`.

Мы создаем интеннт, а затем добавляем в него текст. В качестве имени дополнительной информации используется константа – в этом случае мы можем быть уверены в том, что `CreateMessageActivity` и `ReceiveMessageActivity` используют одну и ту же строку. Определение константы будет добавлено в `ReceiveMessageActivity` на следующей странице.

Теперь, когда активность `CreateMessageActivity` добавила в интеннт дополнительную информацию, необходимо прочитать эту информацию и использовать ее.

Использование информации из интента в ReceiveMessageActivity

Итак, мы запрограммировали в CreateMessageActivity добавление текста в интент; теперь нужно изменить ReceiveMessageActivity для использования передаваемого текста.

ReceiveMessageActivity будет отображать сообщение в своей надписи при создании активности. Так как метод onCreate() активности вызывается сразу же при ее создании, код будет добавлен в этот метод.

Чтобы получить сообщение из интента, мы сначала получим объект интента вызовом getIntent(), а затем — сами передаваемые данные вызовом getStringExtra().

Ниже приведен полный код ReceiveMessageActivity.java (замените код, сгенерированный Android Studio, и сохраните все изменения):

```
package com.hfad.messenger;

import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.widget.TextView;
```

← Необходимо импортировать классы Intent и TextView.

```
public class ReceiveMessageActivity extends Activity {

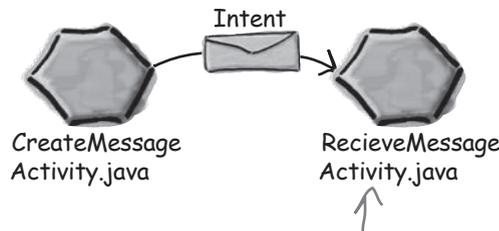
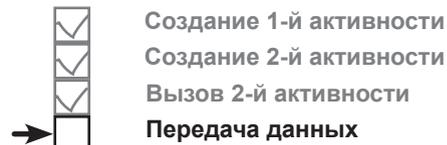
    public static final String EXTRA_MESSAGE = "message";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_receive_message);
        Intent intent = getIntent();
        String messageText = intent.getStringExtra(EXTRA_MESSAGE);
        TextView messageView = (TextView) findViewById(R.id.message);
        messageView.setText(messageText);
    }
}
```

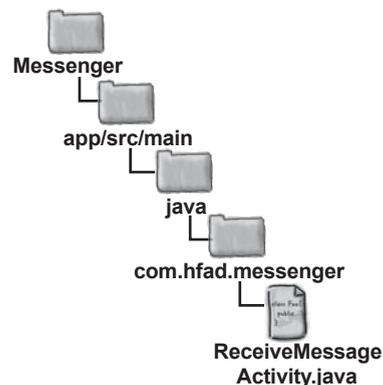
↑ Имя дополнительного значения, передаваемого в интенте.

↑ Добавить текст в надпись с идентификатором message.

Прежде чем тестировать приложение, еще раз пройдемся по коду и вспомним, что в нем происходит.



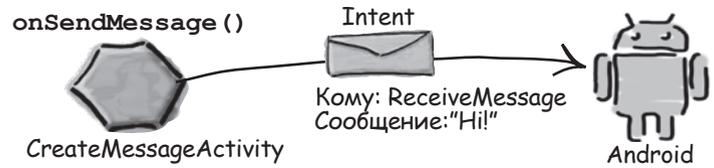
Теперь нужно запрограммировать обработку интента, полученного ReceiveMessageActivity.



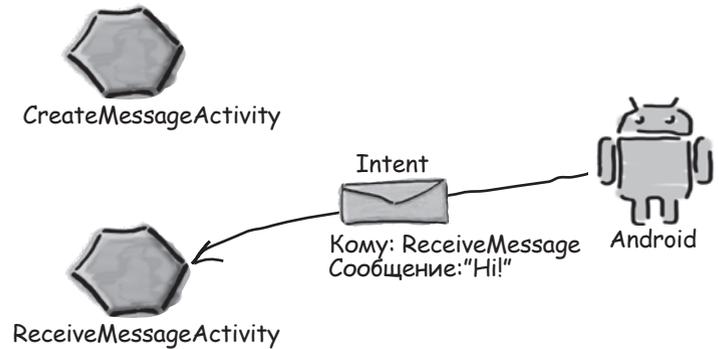
← Получить интент и извлечь из него сообщение вызовом getStringExtra().

Что происходит при щелчке на кнопке Send Message

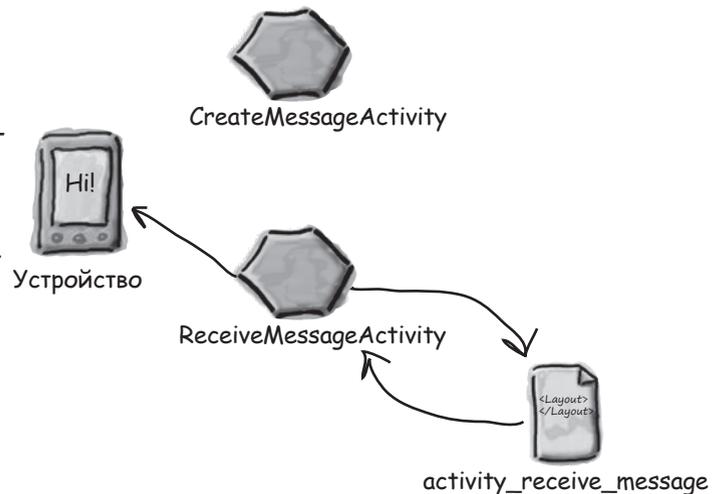
- 1** Когда пользователь щелкает на кнопке, вызывается метод `onSendMessage()`. Код метода `onSendMessage()` создает интент для запуска активности `ReceiveMessageActivity`, добавляет в интент текст сообщения и передает его Android вместе с инструкцией по запуску активности.



- 2** Android проверяет интент на правильность и приказывает `ReceiveMessageActivity` запуститься.



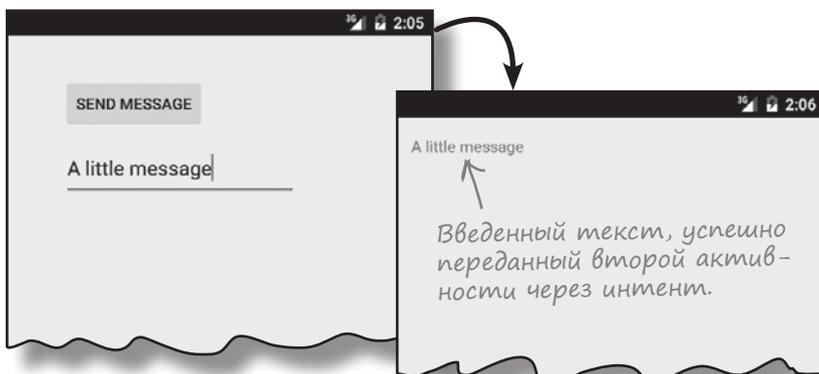
- 3** При запуске активность `ReceiveMessageActivity` указывает, что она использует макет `activity_receive_message.xml`. Этот макет отображается на устройстве. Активность изменяет макет и выводит в нем текст, полученный из интента.





Тест-драйв

Проверьте, что вы внесли изменения в обеих активностях, сохраните изменения и запустите приложение. Запускается активность `CreateMessageActivity`; если ввести текст и щелкнуть на кнопке `Send Message`, запускается `ReceiveMessageActivity`. Текст, введенный в первой активности, появляется в надписи.



- Создание 1-й активности
- Создание 2-й активности
- Вызов 2-й активности
- Передача данных



Оба приложения занимают весь экран — мы не показываем часть пустого пространства.

Приложение можно изменить так, чтобы сообщения отправлялись другим людям

Теперь, когда наше приложение научилось отправлять сообщения другой активности, его можно изменить так, чтобы оно отправляло сообщения другим людям. Для этого приложение интегрируется с другими приложениями, поддерживающими отправку сообщений и уже установленными на устройстве. В зависимости от того, какие приложения установлены у пользователя, можно организовать отправку сообщений из вашего приложения через Gmail, Google+, Facebook, Twitter...

Одну минутку! Сколько же работы придется проделать, чтобы наше приложение заработало со всеми этими приложениями? И как мне узнать, какие приложения установлены на чужих устройствах? Это несерьезно.

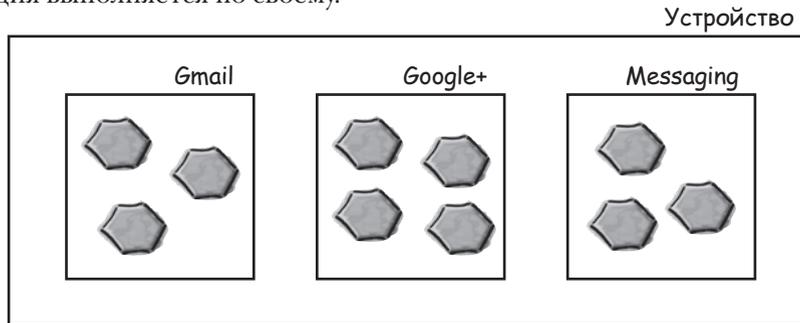
На самом деле все не так сложно, как может показаться, — благодаря особенностям архитектуры Android.

Помните, что говорилось в начале главы о задачах — цепочках из нескольких активностей? Так вот, **при этом совершенно не обязательно ограничиваться активностями вашего приложения.** С таким же успехом можно использовать активности *других* приложений.



Как работают приложения Android

Как вам уже известно, Android-приложения состоят из одной или нескольких активностей, а также других компонентов — например, макетов. Каждая активность представляет одну четко определенную операцию, которая может выполняться пользователем. Например, такие приложения, как Gmail, Google+, Сообщения, Facebook и Twitter, содержат активности, позволяющие отправлять сообщения, хотя в каждом приложении эта операция выполняется по-своему.



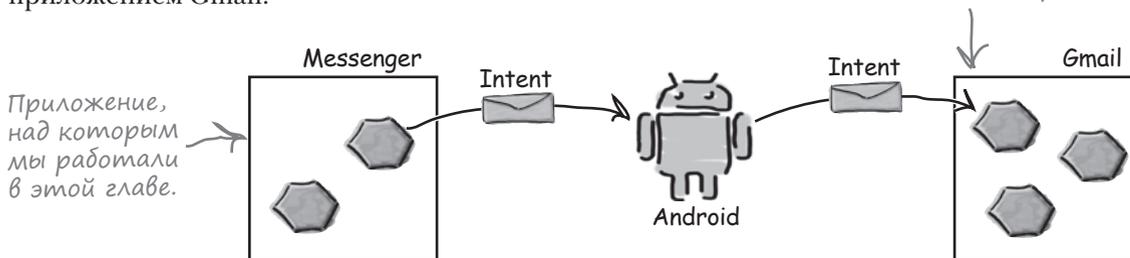
Каждое приложение состоит из активностей. Также существуют и другие компоненты, но сейчас нас интересуют только активности.

Интенты могут запускать активности из других приложений

Вы уже видели, как использовать интент для запуска второй активности из того же приложения. Первая активность передает интент Android; Android проверяет интент, а затем приказывает второй активности запуститься.

Этот принцип относится и к активностям других приложений. Активность вашего приложения передает интент Android, Android проверяет его, а затем приказывает второй активности запуститься — *несмотря на то, что эта активность находится в другом приложении*. Например, можно воспользоваться интентом для запуска активности Gmail, отправляющей сообщения, и передать ей текст, который нужно отправить. Вместо того, чтобы писать собственные активности для отправки электронной почты, можно воспользоваться готовым приложением Gmail.

Вы можете создать интент для запуска другой активности даже в том случае, если активность находится в другом приложении.



Это означает, что объединяя активности на устройстве в цепочку, вы можете строить приложения, обладающие существенно большей функциональностью.

Но мы не знаем, какие приложения установлены на устройстве

Прежде чем вызывать активности из других приложений, необходимо ответить на три вопроса:

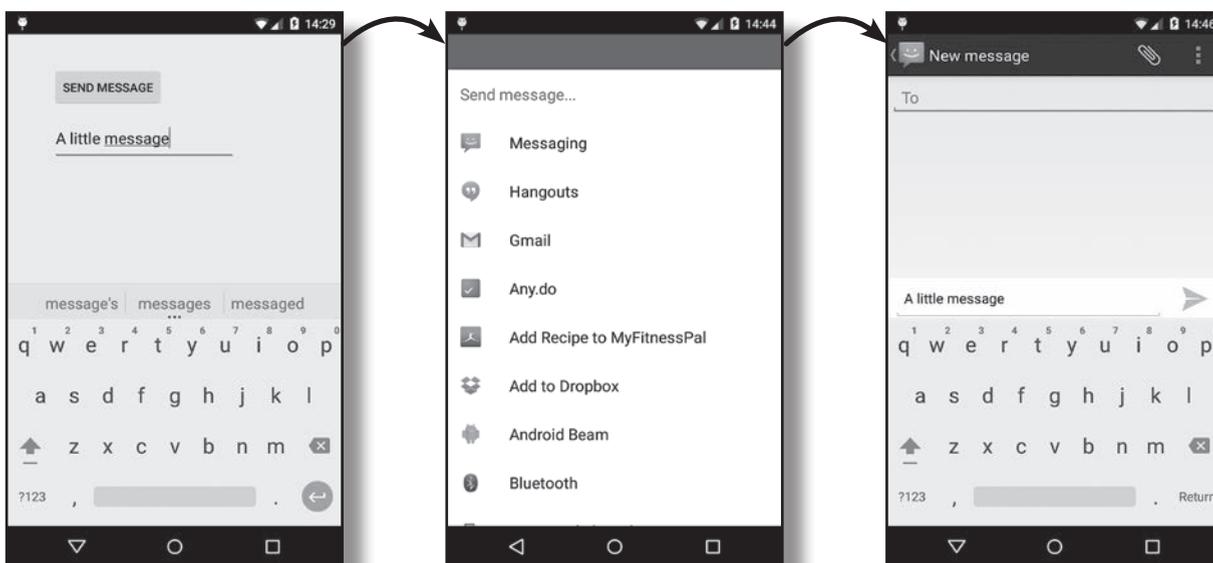
- ★ Как узнать, какие активности доступны на устройстве пользователя?
- ★ Как узнать, какие из этих активностей подходят для того, что мы собираемся сделать?
- ★ Как узнать, как использовать эти активности?

К счастью, все эти проблемы решаются при помощи **действий** (actions). Действия — стандартный механизм, при помощи которого Android узнает о том, какие стандартные операции могут выполняться активностями. Например, Android знает, что все активности, зарегистрированные для действия send, могут отправлять сообщения.

А теперь нужно научиться создавать интенды, использующие действия для получения набора активностей, которые могут использоваться для выполнения стандартных функций — например, для отправки сообщений.

Что мы собираемся сделать

- 1 **Создать интент с указанием действия.**
Интент сообщит Android, что вам нужна активность, умеющая отправлять сообщения. Интент будет включать текст сообщения.
- 2 **Разрешить пользователю выбрать используемое приложение.**
Скорее всего, на устройстве установлено сразу несколько приложений, способных отправлять сообщения, поэтому пользователь должен выбрать одно из них. Мы хотим, чтобы пользователь мог выбрать приложение каждый раз, когда он щелкает на кнопке Send Message.



Создание интента с указанием действия



Определение действия
Выбор активности

Ранее вы видели, как создать интент для запуска конкретной активности командой вида

```
Intent intent = new Intent(this, ReceiveMessageActivity.class);
```

Такие интенты называются **явными**; вы явно сообщаете Android, какой класс должна запустить система.

Если требуется выполнить некоторое действие и вас не интересует, какой активностью оно будет выполнено, создайте **неявный интент**. При этом вы сообщаете Android, какое действие нужно выполнить, а все подробности по выбору активности, выполняющей это действие, поручаются Android.

Как создать интент

Для создания интента с указанием действия применяется следующий синтаксис:

```
Intent intent = new Intent(действие);
```

где действие — тип действия, выполняемого активностью. Android предоставляет целый ряд стандартных вариантов действий. Например, действие `Intent.ACTION_DIAL` используется для набора номера, `Intent.ACTION_WEB_SEARCH` — для выполнения веб-поиска, а `Intent.ACTION_SEND` — для отправки сообщений. Итак, если вы хотите создать интент для отправки сообщения, используйте команду следующего вида:

```
Intent intent = new Intent(Intent.ACTION_SEND);
```

Добавление дополнительной информации

После определения действия в интент можно включить дополнительную информацию. Допустим, вы хотите добавить текст, который образует тело отправляемого сообщения. Задача решается следующим фрагментом кода:

```
intent.setType("text/plain");  
intent.putExtra(Intent.EXTRA_TEXT, текст);
```

где текст — отправляемый текст. Вызов сообщает Android, что активность должна уметь обрабатывать данные с типом данных MIME “text/plain”, а также передает сам текст.

Если потребуется добавить несколько видов дополнительной информации, используйте многократные вызовы метода `putExtra()`. Например, если вы хотите также указать тему сообщения, используйте вызов вида

```
intent.putExtra(Intent.EXTRA_SUBJECT, тема);
```

где тема — тема сообщения.

↑
Мы сообщили интен-ту, для какого класса он предназначен, — а если мы этого не знаем?

О том, какие действия активностей можно использовать в программах и какую дополнительную информацию они поддерживают, можно узнать в справочных материалах для разработчиков Android: <http://tinyurl.com/n57qb5>.

← Эти атрибуты актуальны для `Intent.ACTION_SEND`, а не для всех возможных действий.

← Если информация о теме не актуальна для конкретного приложения, оно просто проигнорирует эту информацию. С другой стороны, любое приложение, которое умеет ее использовать, так и поступит.



Изменение интента для использования действия

Мы изменим файл `CreateMessageActivity.java` так, чтобы в нем создавался неявный интент для использования действия отправки. Внесите изменения, выделенные жирным шрифтом, и сохраните свою работу:

```

package com.hfad.messenger;

import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.view.View;
import android.widget.EditText;

public class CreateMessageActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_create_message);
    }

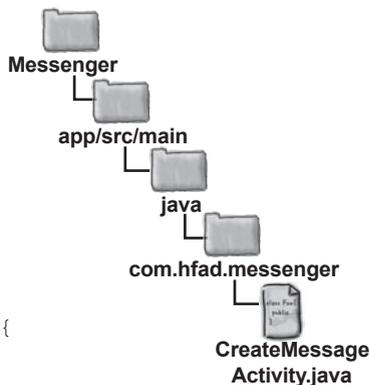
    //Вызвать onSendMessage() при щелчке на кнопке
    public void onSendMessage(View view) {
        EditText messageView = (EditText) findViewById(R.id.message);
        String messageText = messageView.getText().toString();
        Intent intent = new Intent(this, ReceiveMessageActivity.class);
        intent.putExtra(ReceiveMessageActivity.EXTRA_MESSAGE, messageText);
        Intent intent = new Intent(Intent.ACTION_SEND);
        intent.setType("text/plain");
        intent.putExtra(Intent.EXTRA_TEXT, messageText);
        startActivity(intent);
    }
}

```

Удалите
эти две
строки.

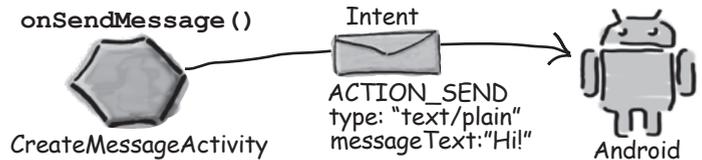


Вместо того, чтобы создавать интент, предназначенный конкретно для `ReceiveMessageActivity`, мы создаем интент с указанием действия отправки.



Что происходит при выполнении кода

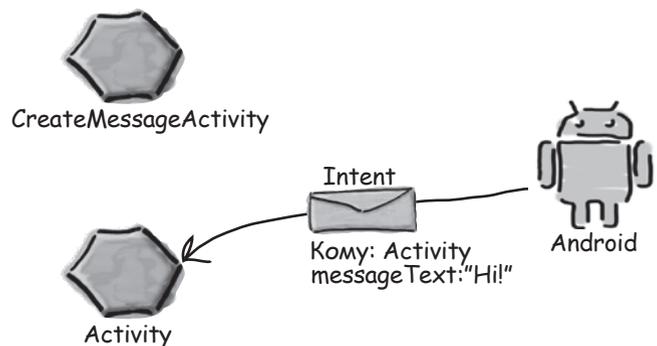
- 1 При вызове метода `onSendMessage()` создается интент. Метод `startActivity()` передает интент Android. Интену назначается действие `ACTION_SEND` и тип MIME `text/plain`.



- 2 Android видит, что интент может передаваться только активностям, способным обрабатывать действие `ACTION_SEND` и данные `text/plain`. Android проверяет все активности и ищет среди них те, которые смогут обработать интент. Если ни одно действие не способно обработать интент, инициируется исключение `ActivityNotFoundException`.



- 3 Если только одна активность способна обработать интент, Android приказывает этой активности запуститься и передает ей интент.



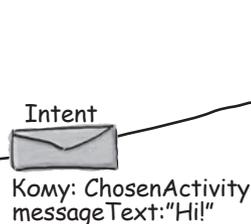
История продолжается...

→  **Определение действия**
Выбор активности

- 4 Если найдется несколько активностей, способных обработать интент, Android открывает диалоговое окно для выбора активности и предлагает пользователю выбрать.



- 5 Когда пользователь выберет активность, которую он хочет использовать, Android приказывает активности запуститься и передает ей интент. Активность выводит дополнительный текст, содержащийся в интенте, в теле нового сообщения.



Чтобы создать диалоговое окно для выбора активности, система Android должна знать, какие активности способны получить интент. На ближайшей паре страниц вы узнаете, как это делается.

Фильтр интен­тов сообщает Android, какие активности могут обработать те или иные действия

При получении интен­та система Android должна определить, какая активность (или активности) может этот интен­т обработать. Этот процесс называется **разрешением интен­та**.

При использовании *явного* интен­та процесс разрешения тривиален: в самом интен­те явно указано, для какого компонента он предназначен, поэтому у Android имеются четкие инструкции, что с ним делать. Например, следующий код явно приказывает Android запустить `ReceiveMessageActivity`:

```
Intent intent = new Intent(this, ReceiveMessageActivity.class);
startActivity(intent);
```

При использовании *неявного* интен­та система Android использует информацию, содержащуюся в интен­те, для определения того, какие компоненты могут его получить. Для этого Android проверяет фильтры интен­тов, содержащиеся в экземплярах `AndroidManifest.xml` всех приложений.

Фильтр интен­тов указывает, какие типы интен­тов могут обрабатываться каждым компонентом. Например, следующая запись относится к активности, способной обрабатывать действие `ACTION_SEND`. Эта активность принимает данные с MIME-типами `text/plain` или `image/*`:

```
<activity android:name="ShareActivity">
  <intent-filter>
    <action android:name="android.intent.action.SEND"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <data android:mimeType="text/plain"/>
    <data android:mimeType="image/*"/>
  </intent-filter>
</activity>
```

Сообщает Android, что активность может обрабатывать `ACTION_SEND`.

Фильтр интен­тов должен включать категорию `DEFAULT`; в противном случае он не сможет получить неявные интен­ты.

Типы данных, которые могут обрабатываться активностью.

Фильтр интен­тов также включает **кате­го­рию**. Категор­ия предоставляет дополнительную информацию об активности: например, может ли она запускаться браузером или является ли она главной точкой входа приложения. Фильтр интен­тов *должен* включать категорию `android.intent.category.DEFAULT`, если он собирается принимать неявные интен­ты. Если активность не имеет фильтра интен­тов или не включает категорию с именем `android.intent.category.DEFAULT`, это означает, что активность не может запускаться неявным интен­том. Она может быть запущена только *явным* интен­том с указанием полного имени компонента.



Как Android использует фильтр интенгов

Получив неявный интенг, Android сравнивает информацию из интенга с информацией, содержащейся в фильтрах интенгов из файла *AndroidManifest.xml* каждого приложения.

Сначала Android рассматривает фильтры интенгов, включающие категорию `android.intent.category.DEFAULT`:

```
<intent-filter>
    <category android:name="android.intent.category.DEFAULT"/>
    ...
</intent-filter>
```

Фильтры интенгов без этой категории пропускаются, так как они не могут получать неявные интенгты.

Затем Android сопоставляет интенгты с фильтрами интенгов, сравнивая действия и тип MIME из интенга с указанными в фильтрах. Допустим, если в интенге указано действие `Intent.ACTION_SEND`:

```
Intent intent = new Intent(Intent.ACTION_SEND);
```

Android будет рассматривать только те активности, для которых указан фильтр интенгов с действием `android.intent.action.SEND`:

```
<intent-filter>
    <action android:name="android.intent.action.SEND"/>
    ...
</intent-filter>
```

Аналогичным образом, если для интенга установлен тип MIME “text/plain”:

```
intent.setType("text/plain");
```

Android будет рассматривать только те активности, которые поддерживают этот тип данных:

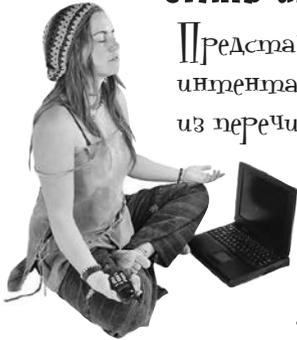
```
<intent-filter>
    <data android:mimeType="text/plain"/>
    ...
</intent-filter>
```

Если тип MIME в интенге не указан, то Android пытается вычислить его на основании данных, содержащихся в интенге.

После того как сравнение интенга с фильтрами интенгов, назначенных компонентам, будет завершено, Android смотрит, сколько совпадений удалось найти. Если найдено только одно совпадение, Android запускает компонент (в нашем случае это активность) и передает ему интенг. Если будет найдено несколько совпадений, Android просит пользователя выбрать один из вариантов.

← Также Android будет проверять категорию фильтра интенгов, если она указана в интенге. Данная возможность используется нечасто, поэтому мы не рассматриваем добавление категорий в интенгты.

СТАНЬ интен­том



Представьте себя на месте
интен­та и скажите, какая
из перечисленных ниже
активностей
совместима с вашим
действием и данными.
В каждом случае
обоснуйте свой ответ.

Это интен­т.

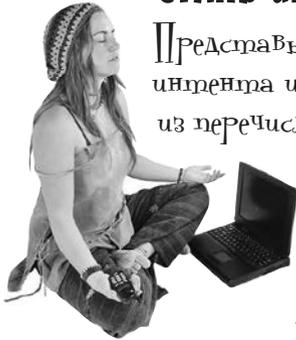
```
Intent intent = new Intent(Intent.ACTION_SEND);
intent.setType("text/plain");
intent.putExtra(Intent.EXTRA_TEXT, "Hello");
```

```
<activity android:name="SendActivity">
  <intent-filter>
    <action android:name="android.intent.action.SEND"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <data android:mimeType="*/"/>
  </intent-filter>
</activity>
```

```
<activity android:name="SendActivity">
  <intent-filter>
    <action android:name="android.intent.action.SEND"/>
    <category android:name="android.intent.category.MAIN"/>
    <data android:mimeType="text/plain"/>
  </intent-filter>
</activity>
```

```
<activity android:name="SendActivity">
  <intent-filter>
    <action android:name="android.intent.action.SENDTO"/>
    <category android:name="android.intent.category.MAIN"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <data android:mimeType="text/plain"/>
  </intent-filter>
</activity>
```

СТАНЬ интентом. Решение



Представьте себя на месте
интента и скажите, какая
из перечисленных ниже
активностей
совместима с вашим
действием и данными.
В каждом случае
обоснуйте свой ответ.

```
Intent intent = new Intent(Intent.ACTION_SEND);  
intent.setType("text/plain");  
intent.putExtra(Intent.EXTRA_TEXT, "Hello");
```

Эта активность принимает ACTION_SEND и может обработать данные любого типа MIME, так что она может отреагировать на интент.

```
<activity android:name="SendActivity">  
  <intent-filter>  
    <action android:name="android.intent.action.SEND"/>  
    <category android:name="android.intent.category.DEFAULT"/>  
    <data android:mimeType="*/*/>  
  </intent-filter>  
</activity>
```



```
<activity android:name="SendActivity">  
  <intent-filter>  
    <action android:name="android.intent.action.SEND"/>  
    <category android:name="android.intent.category.MAIN"/>  
    <data android:mimeType="text/plain"/>  
  </intent-filter>  
</activity>
```



У этой активности отсутствует категория DEFAULT, поэтому она не сможет получить интент.

```
<activity android:name="SendActivity">  
  <intent-filter>  
    <action android:name="android.intent.action.SENDTO"/>  
    <category android:name="android.intent.category.MAIN"/>  
    <category android:name="android.intent.category.DEFAULT"/>  
    <data android:mimeType="text/plain"/>  
  </intent-filter>  
</activity>
```



Эта активность не принимает интенты ACTION_SEND, только ACTION_SENDTO. Действие ACTION_SENDTO позволяет отправить сообщение получателю, указанному в данных интента.



Определение действия
Выбор активности

Запуск приложения на РЕАЛЬНОМ устройстве

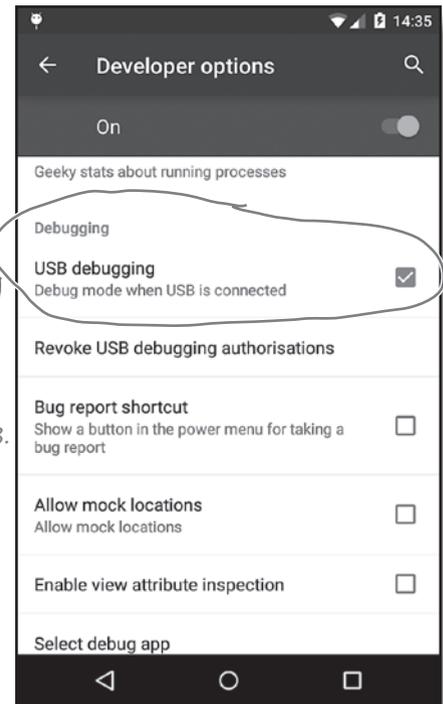
До сих пор мы запускали приложения только под управлением эмулятора. Эмулятор включает крайне ограниченную подборку приложений; может оказаться, что на нем есть всего одно приложение, способное обработать ACTION_SEND. Чтобы нормально протестировать приложение, необходимо запустить его на физическом устройстве, на котором заведомо найдется более одного приложения, поддерживающего нужное действие — например, отправку электронной почты или отправку сообщений. Чтобы протестировать приложение на физическом устройстве, выполните следующие действия:

1. Включите режим отладки через интерфейс USB на устройстве

Да, серьезно.

На устройстве откройте экран “Developer options” (начиная с Android 4.0 по умолчанию этот экран скрыт). Чтобы разрешить его отображение, перейдите в раздел Settings → About Phone и прикоснитесь к номеру сборки семь раз. Когда вы вернетесь к предыдущему экрану, на нем должен появиться раздел “Developer options.” В разделе “Developer options” установите флажок USB debugging.

Необходимо включить отладку через интерфейс USB.



2. Настройте систему для распознавания устройства

Если вы работаете на Mac, пропустите этот шаг.
Если вы работаете в системе Windows, необходимо установить драйвер USB. Самые свежие инструкции можно найти здесь: <http://developer.android.com/tools/extras/oem-usb.html>
Если вы работаете в Ubuntu Linux, создайте файл правил udev. Самые свежие инструкции относительно того, как это делается, находятся здесь: <http://developer.android.com/tools/device.html#setting-up>

3. Подключите свое устройство к компьютеру кабелем USB

Возможно, устройство спросит, хотите ли вы принять ключ RSA, разрешающий отладку через интерфейс USB на вашем компьютере. Если окно с вопросом появится, установите флажок “Always allow from this computer” и щелкните на кнопке ОК, чтобы разрешить отладку.

Это сообщение появится в том случае, если ваше устройство работает под управлением Android 4.2.2 и выше.



Запуск приложения на РЕАЛЬНОМ устройстве (продолжение)

4. Запустите приложение в Android Studio, как обычно

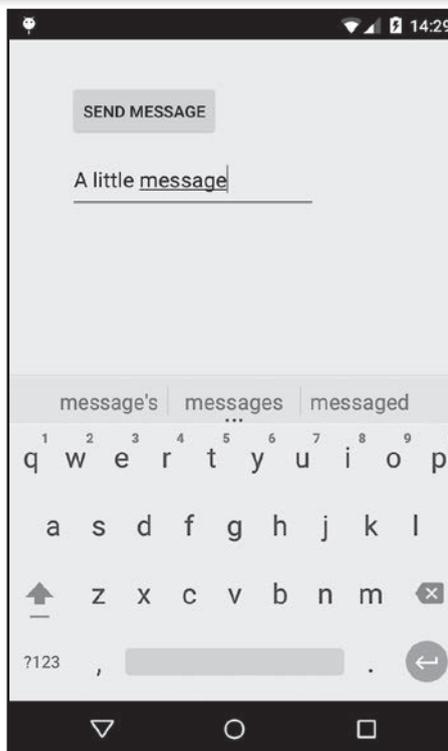
Android Studio устанавливает приложение на устройстве и запускает его. Вам будет предложено выбрать, на каком устройстве следует запустить приложение. Выберите свое устройство в списке и щелкните на кнопке ОК.



Приложение, работающее на физическом устройстве

Приложение на физическом устройстве практически ничем не отличается от приложения, запущенного в эмуляторе. Вероятно, вы заметите, что установка и запуск проходят быстрее.

Теперь, когда вы знаете, как запускать созданные приложения на физическом устройстве, все готово для тестирования новейших изменений в вашем приложении.





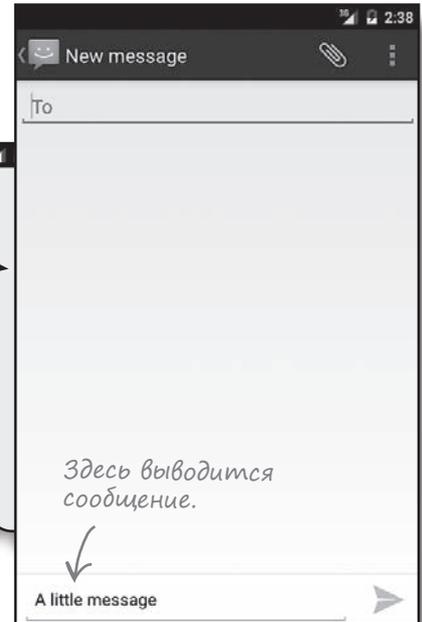
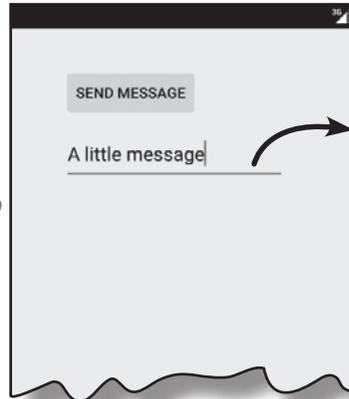
Тест-драйв

Сначала запустите приложение в эмуляторе, а потом на физическом устройстве. Полученные результаты будут зависеть от количества активностей на каждом устройстве, поддерживающих действие Send с текстовыми данными.

Если найдена только одна активность

Щелчок на кнопке Send Message переведет вас прямо к этому приложению.

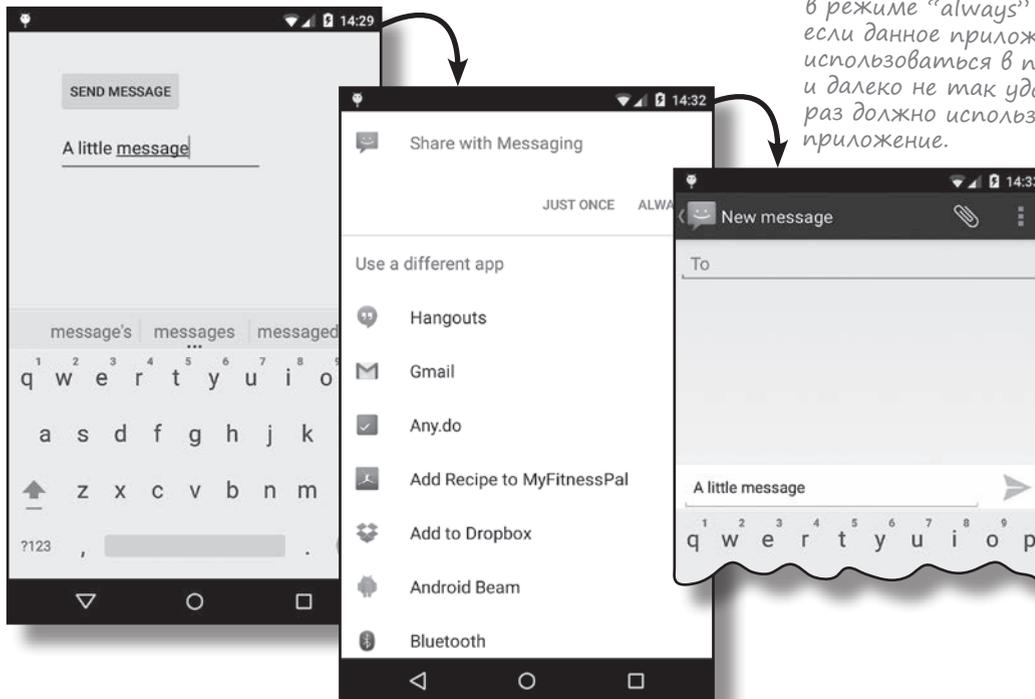
В эмуляторе найдена только одна активность, уменьшая отправлять сообщения с текстовыми данными. При щелчке на кнопке Send Message Android запускает эту активность.



Если найдено несколько активностей

Android выводит окно выбора и предлагает указать, какая из активностей должна использоваться. Также предлагается выбрать, должно ли это действие использоваться только в данном случае или всегда. Если выбрать вариант Always, то в дальнейшем при щелчке на кнопке Send Message та же активность всегда будет использоваться по умолчанию.

На нашем физическом устройстве найдено много подходящих активностей. Мы решили использовать приложение Сообщения (Messaging) в режиме "always" — это удобно, если данное приложение должно всегда использоваться в подобных случаях, и далеко не так удобно, если каждый раз должно использоваться новое приложение.



А если вы хотите, чтобы пользователь ВСЕГДА выбирал активность?

Вы уже видели, что при обнаружении на устройстве нескольких активностей, способных принять интент, Android автоматически предлагает выбрать нужную активность. Пользователь даже может указать, когда должна использоваться эта активность — всегда или только в данном случае.

Однако у этого стандартного поведения есть один недостаток: а что если вы хотите *гарантировать*, что пользователь сможет выбрать активность при каждом щелчке на кнопке Send Message? Например, если он приказал всегда использовать Gmail, то в следующий раз Android уже не предложит ему выбрать Twitter. К счастью, у этой проблемы есть обходное решение. Вы можете создать окно выбора, в котором пользователю будет предложено выбрать активность без каких-либо возможностей всегда использовать именно ее.

`Intent.createChooser()` выводит диалоговое окно выбора

В этом вам поможет метод `Intent.createChooser()`. Он получает уже созданный интент и «упаковывает» его в диалоговое окно выбора. Главная отличительная особенность этого метода — он не предоставляет возможности выбора активности по умолчанию, то есть пользователю придется каждый раз выбирать нужную активность.

Вызвать метод `createChooser()` можно так:

```
Intent chosenIntent = Intent.createChooser(intent, "Send message...");
```

Метод получает два параметра: интент и необязательный заголовок диалогового окна выбора в формате `String`. Параметр `Intent` должен описывать типы активностей, которые должны выводиться в окне выбора. Вы можете использовать интент, созданный ранее, так как он указывает, что для его обработки требуется поддержка `ACTION_SEND` с текстовыми данными.

Метод `createChooser()` возвращает новый объект `Intent`. Он представляет собой новый явный интент, предназначенный для активности, выбранной пользователем. Он содержит всю дополнительную информацию, передававшуюся в исходном интенте, включая весь текст.

Чтобы запустить активность, выбранную пользователем, нужно вызвать:

```
startActivity(chosenIntent);
```

Сейчас вы узнаете, что происходит при вызове метода `createChooser()`.

Метод `createChooser()` позволяет задать заголовок окна выбора и не дает возможности выбрать активность, используемую по умолчанию. Если ни одной подходящей активности не найдено, то пользователь будет оповещен об этом при помощи сообщения.

← Интент, созданный ранее.

↑ Вы можете передать заголовок окна выбора, который будет отображаться у верхнего края экрана.

Что происходит при вызове createChooser()

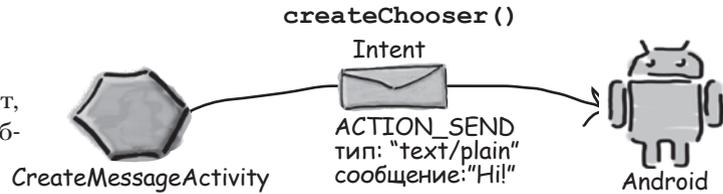


Определение действия
Выбор активности

Давайте разберемся, что происходит при выполнении следующих двух строк кода:

```
Intent chosenIntent = Intent.createChooser(intent, "Send message...");
startActivity(chosenIntent);
```

- 1 **Вызывается метод createChooser().**
При вызове указывается интент, определяющий действие, и необходимый тип MIME.



- 2 **Android проверяет, какие активности могут обработать интент, по их фильтрам интендов.**
Совпадения проверяются по действиям, типам данных и поддерживаемым категориям.



- 3 **Если интент может быть обработан несколькими активностями, Android отображает диалоговое окно для выбора активности.**
В этом окне пользователь указывает, какую активность следует использовать.



На этот раз у пользователя нет возможности выбрать активность по умолчанию, а в заголовке отображается строка "Send message...".
Если Android не находит ни одной подходящей активности, то окно все равно отображается, но пользователь получает сообщение об отсутствии приложений, способных выполнить действие.

История продолжается...

→  Определение действия
Создание окна выбора

- 4 Когда пользователь выберет, какую активность он хочет использовать, Android возвращает новый явный интент с описанием выбранной активности.

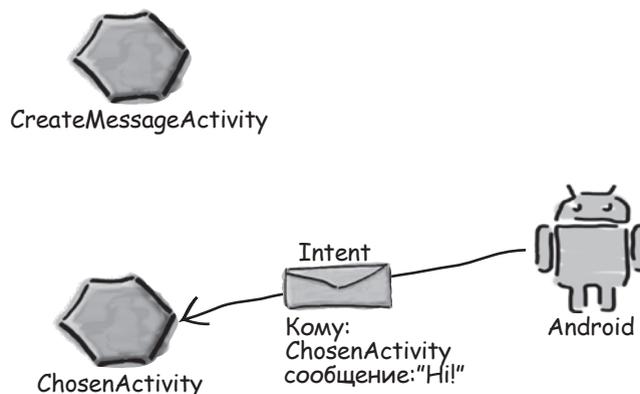
Новый интент содержит всю дополнительную информацию, включенную в исходный интент (в частности, дополнительный текст).



- 5 Исходная активность приказывает Android запустить активность, указанную в интенте.



- 6 Android запускает активность, указанную в интенте, и передает ей интент.



Изменение кода выбора активности

Мы изменим код так, чтобы при каждом щелчке на кнопке Send Message пользователю предлагалось выбрать активность, используемую для отправки сообщения. Мы обновим метод `onSendMessage()` из файла `CreateMessageActivity.java` так, чтобы он вызывал метод `createChooser()`, а также добавим в `strings.xml` строковый ресурс для текста в заголовке окна выбора.

Обновление `strings.xml`...

В заголовке диалогового окна выбора должен отображаться текст "Send message...". Добавьте в `strings.xml` строку с именем "chooser" и присвойте ей значение "Send message..." (не забудьте сохранить изменения):

```
...
<string name="chooser">Send message...</string>
...
```

...и обновление метода `onSendMessage()`

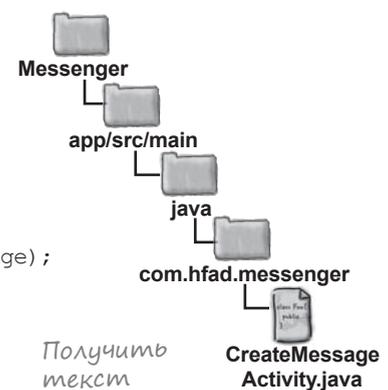
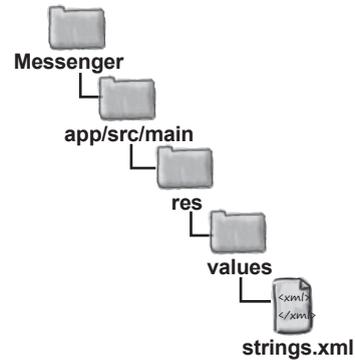
Метод `onSendMessage()` необходимо изменить так, чтобы он получал значение ресурса `chooser` из файла `strings.xml`, вызывал метод `createChooser()`, после чего запускал активность, выбранную пользователем. Приведите код к следующему виду:

```
...
//Вызвать onSendMessage() при щелчке на кнопке
public void onSendMessage(View view) {
    EditText messageView = (EditText)findViewById(R.id.message);
    String messageText = messageView.getText().toString();
    Intent intent = new Intent(Intent.ACTION_SEND);
    intent.setType("text/plain");
    intent.putExtra(Intent.EXTRA_TEXT, messageText);
    String chooserTitle = getString(R.string.chooser);
    Intent chosenIntent = Intent.createChooser(intent, chooserTitle);
    startActivity(intent);
    startActivity(chosenIntent);
}
...
```

← Запустить активность, выбранную пользователем.

← Получить текст заголовка.

← Вывести диалоговое окно выбора.



Метод `getString()` используется для получения значений строковых ресурсов. Он получает один параметр – идентификатор ресурса (в нашем случае `R.string.chooser`):

```
getString(R.string.chooser);
```

← Заглянув в файл `R.java`, вы найдете `chooser` во внутреннем классе с именем `string`.

После того как в приложение будут внесены все необходимые изменения, запустите приложение и посмотрите, как работает окно выбора.



Тест-драйв



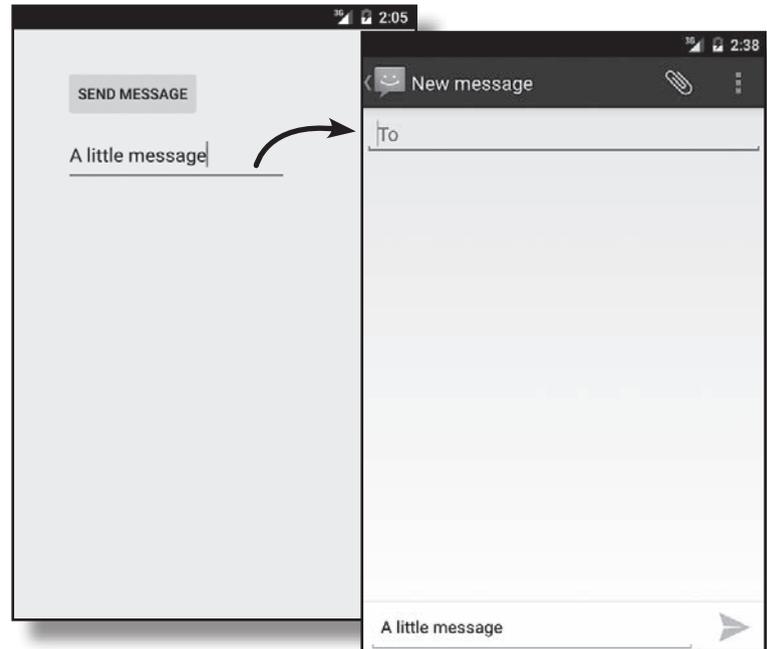
Определение действия
Выбор активности

Сохраните изменения и попробуйте снова запустить приложение.

Если найдена только одна активность

Щелчок на кнопке Send Message приведет вас сразу к приложению, как и прежде.

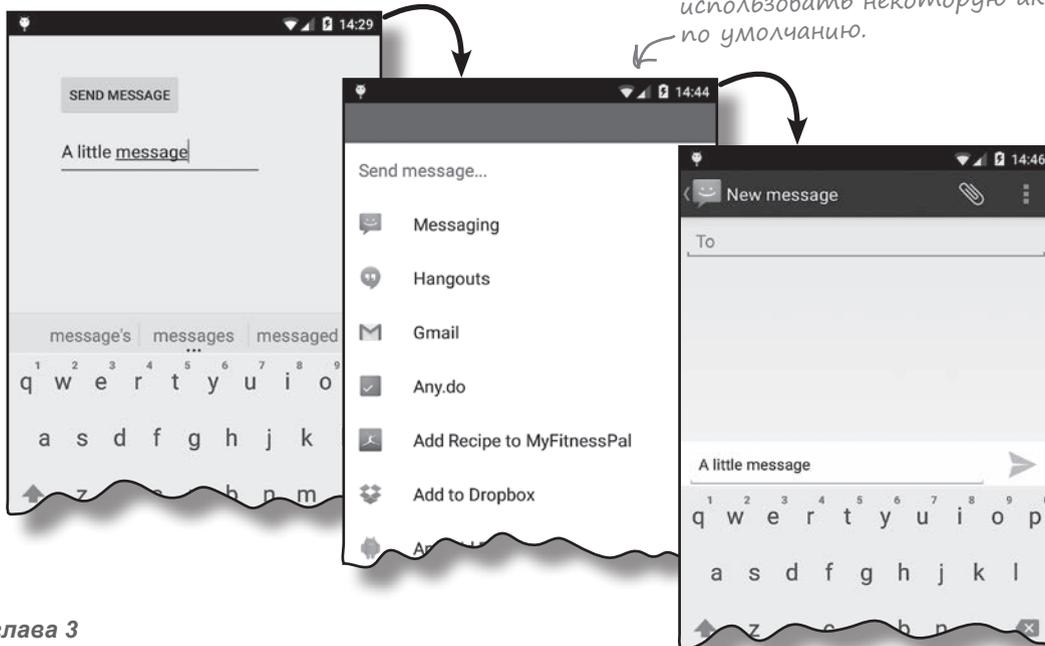
Здесь ничего не изменилось — Android, как и прежде, сразу запускает активность.



Если найдено несколько активностей

Android выводит окно выбора, но на этот раз не спрашивает, нужно ли всегда использовать некоторую активность. Также в заголовке выводится значение строкового ресурса.

Окно выбора, созданное вызовом createChooser(). Оно уже не предлагает использовать некоторую активность по умолчанию.

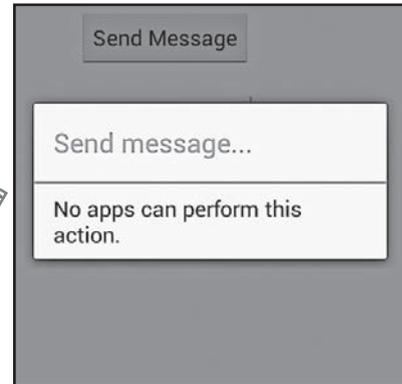


Если подходящих активностей НЕТ

Если на устройстве не обнаружено ни одной активности, способной отправлять сообщения, метод `createChooser()` выводит соответствующее сообщение.

Это еще одно из преимуществ метода `createChooser()`. Метод `createChooser()` корректно справляется с ситуацией, в которой указанное действие не может быть выполнено ни одной активностью.

Если вы захотите воспроизвести это сообщение, попробуйте запустить приложение в эмуляторе с отключением приложения Сообщения.



Часто задаваемые вопросы

В: Итак, я могу запускать приложения в эмуляторе или на физическом устройстве. Что лучше?

О: У каждого варианта есть свои достоинства и недостатки.

Если приложение выполняется на физическом устройстве, оно обычно загружается намного быстрее, чем в эмуляторе. Также запуск на физическом устройстве полезен при написании кода, взаимодействующего с оборудованием устройства.

Эмулятор позволяет протестировать приложение в разных версиях Android, с разными разрешениями экрана и спецификациями устройства. Эмулятор избавляет от необходимости покупать много разных устройств.

Главное — позаботьтесь о том, чтобы ваши приложения были тщательно протестированы как в эмуляторе, так и на физических устройствах, прежде чем публиковать их для широкой аудитории.

В: Какие интенны мне использовать — явные или неявные?

О: Все зависит от того, хотите ли вы, чтобы система Android подобрала активность для выполнения вашего действия. Предположим, вы хотите отправить электронную почту. Если вас не интересует, какое почтовое приложение будет использовано для отправки — лишь бы сообщение было, — используйте неявный интенн. С другой стороны, если вы хотите передать интенн конкретной активности своего приложения, используйте явный интенн. Необходимо явно указать, для какой активности предназначен интенн.

В: Вы упомянули, что в фильтре интеннов активности может быть указано не только действие, но и категория. Чем они различаются?

О: Действие указывает, что может сделать активность, а категория предоставля-

ет более подробную информацию. Мы не будем подробно рассматривать категории, потому что при создании интеннов категории используются относительно редко.

В: Вы говорите, что при отсутствии активностей, способных обработать интенн, метод `createChooser()` выводит сообщение. А что произойдет, если я использую механизм выбора по умолчанию и передам неявный интенн `startActivity()`?

О: Если передать методу `startActivity()` интенн, для которого не найдется подходящей активности, инициируется исключение `ActivityNotFoundException`. Если вы не перехватите его в блоке `try/catch`, это может привести к аварийному завершению приложения.



Ваш инструментарий Android

Глава 3 осталась позади, а ваш инструментарий пополнился приемами работы с несколькими активностями и интентами.

Весь код для этой главы можно загрузить по адресу <https://tinyurl.com/HeadFirstAndroid>.

КЛЮЧЕВЫЕ МОМЕНТЫ



- Задачей называются две и более активности, объединенные в цепочку.
- Элемент `<EditText>` определяет текстовое поле с возможностью редактирования и ввода текста. Класс текстового поля наследует от класса `Android View`.
- Новая активность в `Android Studio` создается командой `File → New... → Activity`.
- Для каждой создаваемой активности в файле `AndroidManifest.xml` должна быть создана запись.
- **ИНТЕНТ** представляет собой разновидность сообщений, используемых для организации взаимодействия между компонентами `Android`.
- Явный интент предназначен для конкретного компонента. Явный интент создается командой

```
Intent intent = new Intent(this, Target.class);
```
- Активности запускаются вызовом `startActivity(intent)`. Если ни одна подходящая активность не найдена, метод инициирует исключение `ActivityNotFoundException`.
- Используйте метод `putExtra()` для включения дополнительной информации в интент.
- Используйте метод `getIntent()` для получения интента, запустившего активность.
- Используйте методы `get*Extra()` для чтения дополнительной информации, связанной с интентом. Метод `getStringExtra()` читает `String`, `getIntExtra()` читает `int`, и т. д.
- Действие описывает стандартную операцию, которую может выполнять активность. Так, для отправки сообщений используется обозначение `Intent.ACTION_SEND`.
- Чтобы создать неявный интент с указанием действия, используйте запись

```
Intent intent = new Intent(action);
```
- Для описания типа данных в интенте используется метод `setType()`.
- `Android` производит разрешение интентов на основании имени компонента, действия, типа данных и категорий, указанных в интенте. Содержимое интента сравнивается с фильтрами интентов из файла `AndroidManifest.xml` каждого приложения. Чтобы активность получала неявные интенты, она должна включать категорию `DEFAULT`.
- Метод `createChooser()` позволяет переопределить стандартное диалоговое окно выбора активности в `Android`. При использовании этого метода можно указать текст заголовка, а у пользователя нет возможности назначить активность по умолчанию. Если метод не находит ни одной активности, способной получить переданный интент, он выводит сообщение. Метод `createChooser()` возвращает объект `Intent`.
- Для чтения значений строковых ресурсов используется синтаксис `getString(R.string.stringname);`

Из жизни активностей

...и тут я говорю, что если он немедленно не `onStop()`, то я его просто `onDestroy()` на месте.



Активности образуют основу любого Android-приложения.

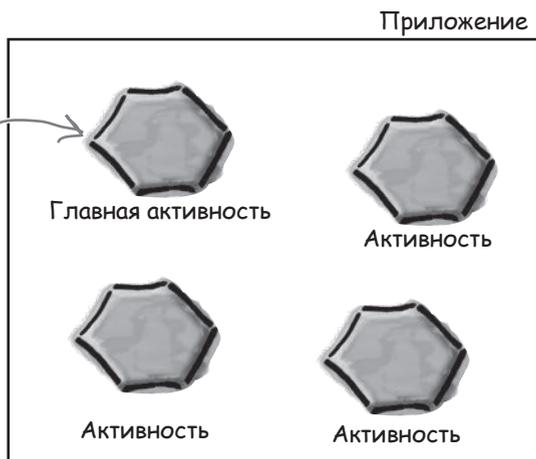
Ранее вы видели, как создавать активности и как организовать запуск одной активности из другой с использованием интента. Но **что при этом происходит, если заглянуть поглубже?** В этой главе более подробно рассматривается **жизненный цикл активностей**. Что происходит при **создании** или **уничтожении** активностей? Какие методы вызываются, когда активность **становится видимой** и **появляется на переднем плане**, и какие методы вызываются, когда активность **теряет фокус** и **скрывается**? И как выполняются операции **сохранения** и **восстановления состояния активности**?

Как на самом деле работают активности?

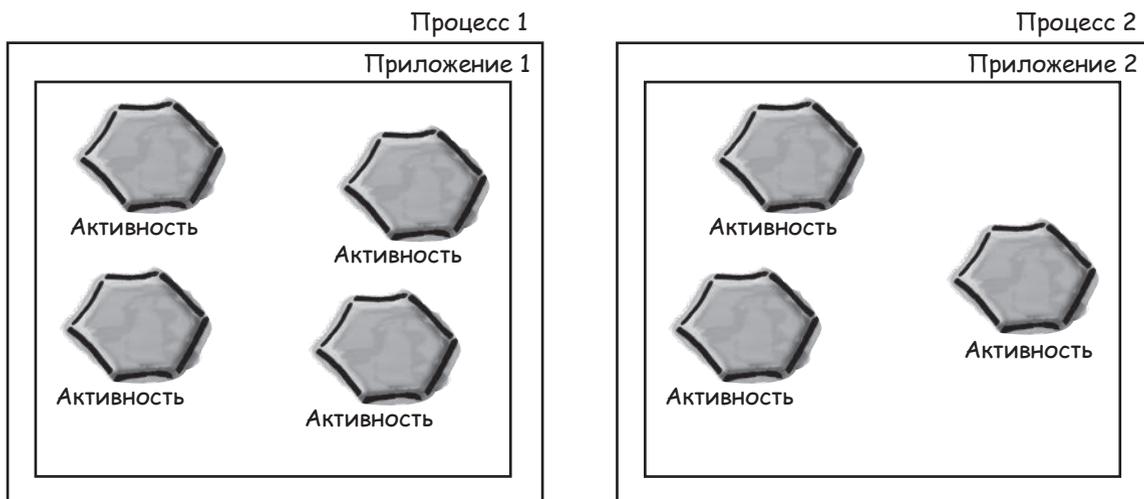
К настоящему моменту вы узнали, как создать приложения, взаимодействующие с пользователем, и приложения, использующие несколько активностей для выполнения задач. Сейчас, когда вы освоили базовые навыки, пришло время глубже разобраться в том, как *на самом деле* работают активности. Ниже приведена краткая сводка того, о чем говорилось ранее, с некоторыми дополнительными подробностями.

- ★ **Приложение состоит из активностей, макетов и других ресурсов.**
Одна из этих активностей является главной активностью приложения.

У каждого приложения имеется главная активность, заданная в файле `AndroidManifest.xml`.

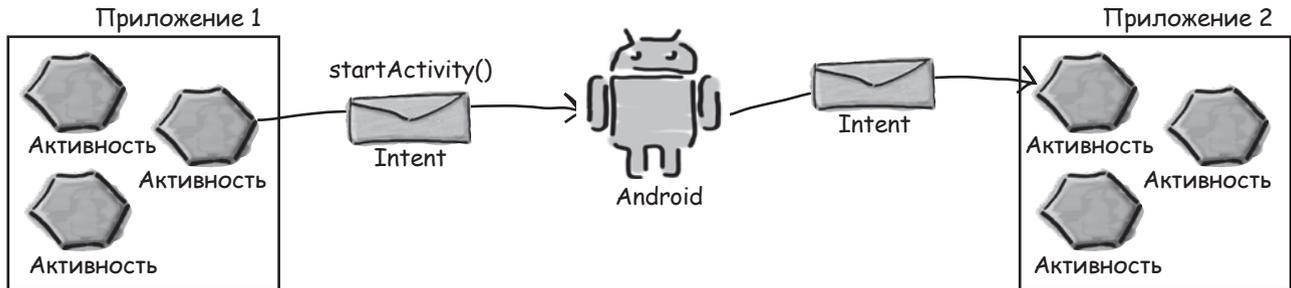


- ★ **По умолчанию каждое приложение выполняется в отдельном процессе.**
Такое разделение обеспечивает безопасность и защиту данных приложений. Дополнительную информацию можно найти в приложении I (в котором рассматривается исполнительная среда TAndroid, или ART) в конце книги.



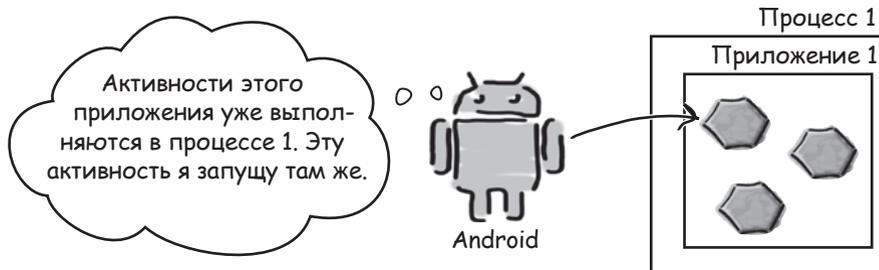
- ★ Вы можете запустить активность в другом приложении, передав интент при вызове `startActivity()`.

Система Android все знает об установленных приложениях и их активностях и использует интент для запуска правильной активности.



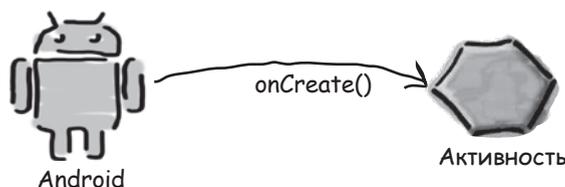
- ★ Непосредственно перед запуском активности Android проверяет, существует ли процесс для этого приложения.

Если процесс существует, то Android запускает активность в этом процессе. Если же процесса нет, то Android создает его.



- ★ При запуске активности Android вызывает ее метод `onCreate()`.

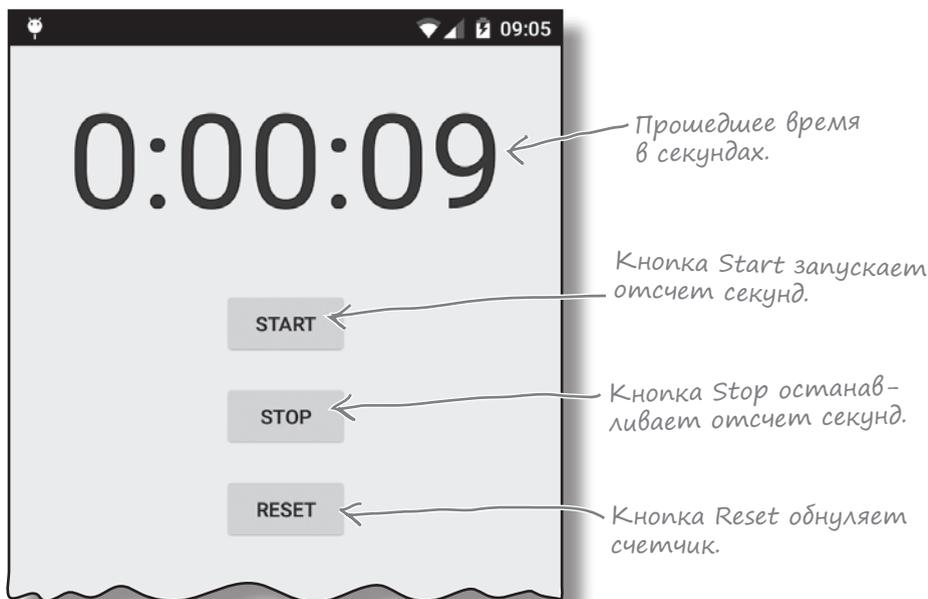
Метод `onCreate()` всегда выполняется при создании активности.



Но мы еще очень многого не знаем о том, как работают активности. Как долго существует активность? Что происходит при исчезновении активности с экрана? Продолжает ли она работать? Остается ли в памяти? И что происходит, когда выполнение приложения прерывается входящим телефонным звонком? Нам хотелось бы управлять поведением наших активностей в *самых разнообразных ситуациях*, но как это сделать?

Приложение Stopwatch

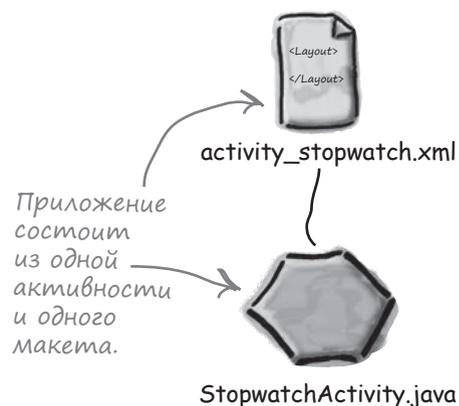
В этой главе мы поближе познакомимся с внутренними механизмами работы активностей, стандартными проблемами в работе приложений и возможностями их решения с помощью методов жизненного цикла активности. Мы будем изучать методы жизненного цикла приложения на примере простого приложения-секундомера Stopwatch. Приложение включает одну активность и один макет. Макет состоит из надписи, в которой выводится прошедшее время, кнопки Start для запуска секундомера, кнопки Stop для его остановки и кнопки Reset для обнуления таймера.



Построение приложения

Вероятно, у вас уже достаточно опыта, чтобы построить приложение без особой помощи с нашей стороны. Мы приведем ровно столько кода, сколько необходимо для его самостоятельного построения; вы сможете сами увидеть, что происходит при попытке запустить его.

Начните с создания нового проекта Android для приложения с именем "Stopwatch" и именем пакета `com.hfad.stopwatch`. Минимальная версия SDK должна быть равна API 15, чтобы приложение работало на большинстве устройств. Приложение должно включать активность с именем "StopwatchActivity" и макет с именем "activity_stopwatch".



Разметка макета приложения Stopwatch

Ниже приведена разметка XML макета. В ней объявляется одна надпись, которая используется для отображения таймера, и три кнопки для управления отсчетом времени. Замените текущую разметку XML из `activity_stopwatch.xml` следующей:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    tools:context=".StopwatchActivity" >
```

<TextView

```
    android:id="@+id/time_view"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="0dp"
    android:text=""
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:textSize="92sp" />
```

Эта надпись используется для вывода прошедшего времени в секундах.

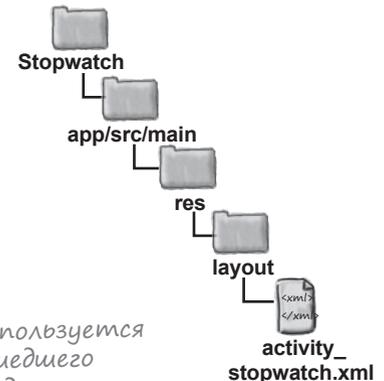
С этими атрибутами время выводится крупными, удобными цифрами.

<Button

```
    android:id="@+id/start_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/time_view"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="20dp"
    android:onClick="onClickStart"
    android:text="@string/start" />
```

Для кнопки Start. При щелчке на этой кнопке вызывается метод с именем `onClickStart()`.

Разметка макета продолжается на следующей странице.



Разметка макета (продолжение)

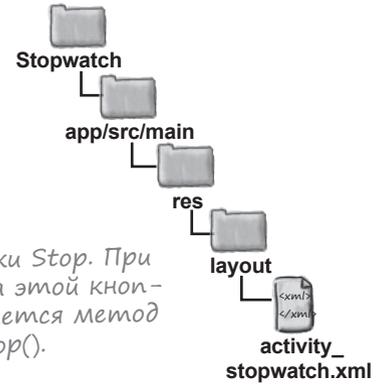
```
<Button
    android:id="@+id/stop_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/start_button"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="10dp"
    android:onClick="onClickStop"
    android:text="@string/stop" />
```

Для кнопки Stop. При щелчке на этой кнопке вызывается метод `onClickStop()`.

```
<Button
    android:id="@+id/reset_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/stop_button"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="10dp"
    android:onClick="onClickReset"
    android:text="@string/reset" />
```

Для кнопки Reset. При щелчке на кнопке вызывается метод `onClickReset()`.

```
</RelativeLayout>
```



Задание!

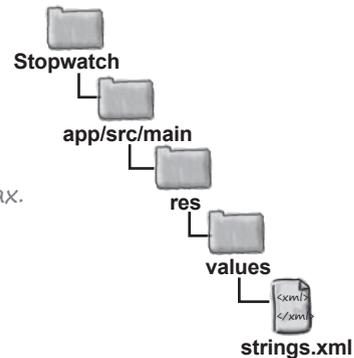
Обязательно обновите макет и файл `strings.xml`, прежде чем продолжить.

Файл `strings.xml` в приложении Stopwatch

Макет использует три строковых значения, по одному для текста на каждой кнопке. Эти значения определяются строковыми ресурсами, поэтому их необходимо включить в файл `strings.xml`. Добавьте строковые значения, приведенные ниже:

```
...
<string name="start">Start</string>
<string name="stop">Stop</string>
<string name="reset">Reset</string>
...
```

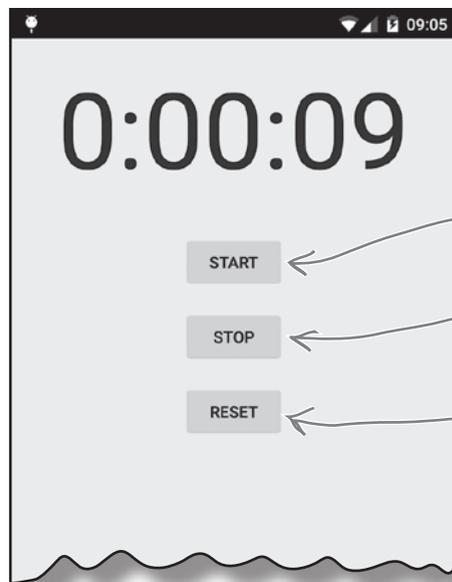
Текст на кнопках.



Макет готов! Теперь перейдем к активности.

Как работает код активности

Макет определяет три кнопки, которые будут использоваться для управления отсчетом времени. Атрибут `onClick` каждой кнопки определяет метод активности, который будет выполняться при щелчке на кнопке: щелчок на кнопке `Start` выполняет метод `onClickStart()`, щелчок на кнопке `Stop` — метод `onClickStop()`, а щелчок на кнопке `Reset` — метод `onClickReset()`. Эти методы будут использоваться для запуска, остановки и сброса секундомера.



При щелчке на кнопке `Start` вызывается метод `onClickStart()`.

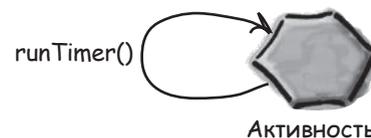
При щелчке на кнопке `Stop` вызывается метод `onClickStop()`.

При щелчке на кнопке `Reset` вызывается метод `onClickReset()`.

Для обновления показаний секундомера будет использоваться метод `runTimer()`, который мы сейчас создадим. Метод `runTimer()` будет каждую секунду проверять, работает ли секундомер, увеличивает число секунд и выводит его в надпись.

Для отслеживания состояния секундомера будут использоваться две приватные переменные. В переменной `seconds` типа `int` хранится количество секунд, прошедших с момента запуска секундомера, а в переменной `running` типа `boolean` хранится признак того, работает ли секундомер в настоящий момент.

Начнем с написания кода кнопок, а затем перейдем к методу `runTimer()`.



Добавление кода кнопок

Когда пользователь щелкает на кнопке Start, переменной `running` присваивается значение `true`, чтобы секундомер начал отсчет. Когда пользователь щелкает на кнопке Stop, переменной `running` присваивается значение `false`, чтобы отсчет времени прекратился. Когда пользователь щелкает на кнопке Reset, переменной `running` присваивается значение `false`, а переменная `seconds` обнуляется, чтобы секундомер обнулится и прекратил отсчет времени.

Замените содержимое `StopwatchActivity.java` следующим кодом:

```
package com.hfad.stopwatch;

import android.os.Bundle;
import android.app.Activity;
import android.view.View;

public class StopwatchActivity extends Activity {
```

```
    private int seconds = 0;
    private boolean running;
```

В переменных `seconds` и `running` хранится количество прошедших секунд и флаг работы секундомера.

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_stopwatch);
    }
```

//Запустить секундомер при щелчке на кнопке Start.

```
    public void onClickStart(View view) {
        running = true;
```

← Запустить секундомер.

← Вызывается при щелчке на кнопке Start.

//Остановить секундомер при щелчке на кнопке Stop.

```
    public void onClickStop(View view) {
        running = false;
```

← Остановить секундомер.

← Вызывается при щелчке на кнопке Stop.

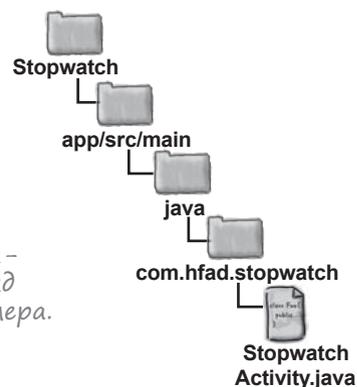
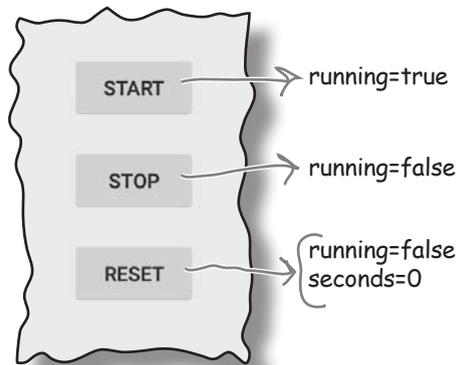
//Обнулить секундомер при щелчке на кнопке Reset.

```
    public void onClickReset(View view) {
        running = false;
        seconds = 0;
```

← Остановить секундомер и присвоить `seconds` значение 0.

← Вызывается при щелчке на кнопке Reset.

```
}
```



Мемог runTimer()

Следующим шагом должно стать создание метода `runTimer()`. Метод `runTimer()` получает ссылку на надпись в макете, форматирует содержимое переменной `seconds` в часы, минуты и секунды, а затем выводит результаты в надписи. Если переменной `running` присвоено значение `true`, то переменная `seconds` увеличивается. Код выглядит так:

```
private void runTimer() {
    final TextView timeView = (TextView)findViewById(R.id.time_view);
    ...
    int hours = seconds/3600;
    int minutes = (seconds%3600)/60;
    int secs = seconds%60;
    String time = String.format("%d:%02d:%02d",
        hours, minutes, secs);
    timeView.setText(time);
    if (running) {
        seconds++;
    }
    ...
}
```

Получить ссылку на надпись.

Часть кода пропущена. Мы вернемся к ней на следующей странице.

Отформатировать значение `seconds` в часы, минуты и секунды. Это самый обычный код Java.

Назначить текст надписи.

Если переменная `running` истинна, увеличить переменную `seconds`.

Код должен выполняться в цикле, чтобы увеличение переменной `seconds` и обновление текстового представления происходили каждую секунду — причем операции должны выполняться без блокирования главного программного потока Android.

В Java-программах, не предназначенных для Android, такие задачи выполняются с использованием фоновых потоков. В мире Android возникает проблема — только главный программный поток Android может обновлять пользовательский интерфейс, а любые попытки такого рода из других потоков порождают исключение `CalledFromWrongThreadException`. Проблема решается при помощи класса `Handler`. Эта тема рассматривается на следующей странице.

Объекты Handler позволяют планировать выполнение кода

Handler — класс Android, который может использоваться для планирования выполнения кода в некоторый момент в будущем. Также класс может использоваться для передачи кода, который должен выполняться в другом программном потоке. В нашем примере Handler будет использоваться для планирования выполнения кода секундомера каждую секунду.

Чтобы использовать класс Handler, упакуйте код, который нужно запланировать, в объект Runnable, после чего используйте методы `post()` и `postDelayed()` класса Handler для определения того, когда должен выполняться код. Давайте поближе познакомимся с этими методами.

Метод `post()`

Метод `post()` передает код, который должен быть выполнен как можно скорее (обычно почти немедленно). Метод `post()` вызывается с одним параметром — объектом типа Runnable. Объект Runnable в мире Android, как и в традиционном языке Java, представляет выполняемое задание. Код, который требуется выполнить, помещается в метод `run()` объекта Runnable, а объект Handler позаботится о том, чтобы код был выполнен как можно быстрее. Вызов метода выглядит так:

```
final Handler handler = new Handler();
handler.post(Runnable);
```

← Выполняемый код передается методу `run()` объекта `Handler`.

Метод `postDelayed()`

Метод `postDelayed()` работает почти так же, как `post()`, но выполнение кода планируется на некоторый момент в будущем. Метод `postDelayed()` получает два параметра: Runnable и long. Объект Runnable содержит выполняемый код в методе `run()`, а long задает задержку выполнения кода в миллисекундах. Код выполняется при первой возможности после истечения задержки. Вызов метода выглядит так:

```
final Handler handler = new Handler();
handler.postDelayed(Runnable, long);
```

← Используйте этот метод для выполнения кода с заданной задержкой в миллисекундах.

На следующей странице мы используем эти методы для ежесекундного обновления секундомера.

Полный код runTimer()

Чтобы обновить секундомер, мы будем многократно планировать выполнение кода с использованием Handler; при этом каждый раз будет назначаться задержка продолжительностью 1000 миллисекунд. Каждое выполнение кода сопровождается увеличением переменной seconds и обновлением надписи. Полный код метода runTimer():

```
private void runTimer() {
    final TextView timeView = (TextView) findViewById(R.id.time_view);
    final Handler handler = new Handler();
    handler.post(new Runnable() {
        @Override
        public void run() {
            int hours = seconds/3600;
            int minutes = (seconds%3600)/60;
            int secs = seconds%60;
            String time = String.format("%d:%02d:%02d",
                hours, minutes, secs);
            timeView.setText(time);
            if (running) {
                seconds++;
            }
            handler.postDelayed(this, 1000);
        }
    });
}
```

← Создать новый объект Handler.

← Вызов метода post() с передачей нового объекта Runnable. Метод post() обеспечивает выполнение без задержки, так что код в Runnable будет выполнен практически немедленно.

← Метод run() объекта Runnable содержит код, который требуется выполнить, — в нашем случае это код обновления надписи.

← Код из объекта Runnable передается на повторное выполнение после истечения задержки в 1000 миллисекунд (1 секунда). Так как эта строка кода включена в метод run() объекта Runnable, код будет вызываться снова и снова.

Такое использование методов post() и postDelayed() означает, что код будет выполнен при первой возможности при истечении необходимой задержки; на практике это значит «почти немедленно». И хотя код будет чуть-чуть запаздывать по времени, для исследований методов жизненного цикла в этой главе такой точности достаточно.

Метод runTimer() должен начинать работу при создании StopwatchActivity, поэтому мы вызываем его в методе onCreate() активности:

```
protected void onCreate(Bundle savedInstanceState) {
    ...
    runTimer();
}
```

Полный код активности приведен на следующей странице.

Полный код StopwatchActivity

Ниже приведен полный код *StopwatchActivity.java*. Внесите в него изменения, представленные ниже.

```

package com.hfad.stopwatch;

import android.os.Bundle;
import android.os.Handler;
import android.app.Activity;
import android.view.View;
import android.widget.TextView;

public class StopwatchActivity extends Activity {
    //Количество секунд на секундомере.
    private int seconds = 0;
    //Секундомер работает?
    private boolean running;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_stopwatch);
        runTimer();
    }

    //Запустить секундомер при щелчке на кнопке Start.
    public void onClickStart(View view) {
        running = true;
    }

    //Запустить секундомер при щелчке на кнопке Stop.
    public void onClickStop(View view) {
        running = false;
    }

```

В переменных seconds и running хранится количество прошедших секунд и флаг работы секундомера.

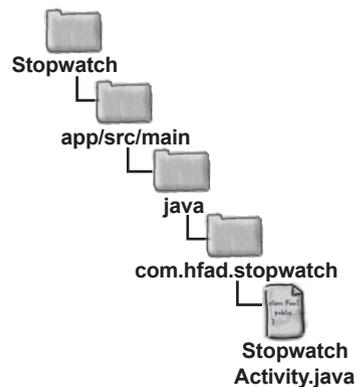
Для обновления секундомера используется отдельный метод. Он запускается при создании активности.

Вызывается при щелчке на кнопке Start.

Запустить секундомер.

Вызывается при щелчке на кнопке Stop.

Остановить секундомер.



Разметка макета (продолжение)

```

//Сбросить секундомер при щелчке на кнопке Reset.
public void onClickReset(View view) {
    running = false;
    seconds = 0;
}

//Обновление показаний таймера.
private void runTimer() {
    final TextView timeView = (TextView)findViewById(R.id.time_view);
    final Handler handler = new Handler();
    handler.post(new Runnable() {
        @Override
        public void run() {
            int hours = seconds/3600;
            int minutes = (seconds%3600)/60;
            int secs = seconds%60;
            String time = String.format("%d:%02d:%02d",
                hours, minutes, secs);
            timeView.setText(time);
            if (running) {
                seconds++;
            }
            handler.postDelayed(this, 1000);
        }
    });
}
}

```

Вызывается при щелчке на кнопке Reset.

Остановить секундомер и обнулить переменную seconds.

Получить ссылку на надпись.

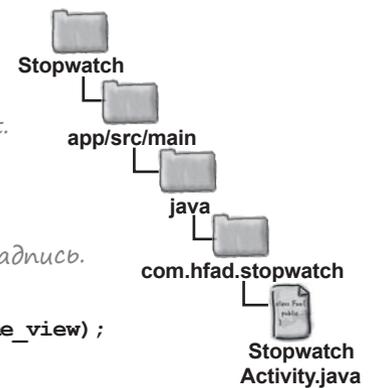
Использовать Handler для передачи кода на выполнение.

Отформатировать seconds в часы, минуты и секунды.

Задать текст надписи.

Если значение running истинно, увеличить переменную seconds.

Запланировать повторное выполнение кода с задержкой в 1 секунду.



Посмотрим, что происходит при выполнении кода.

Задание!

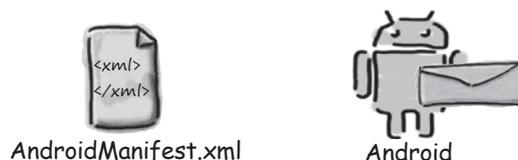
Не забудьте внести эти изменения в код активности.

Что происходит при запуске приложения

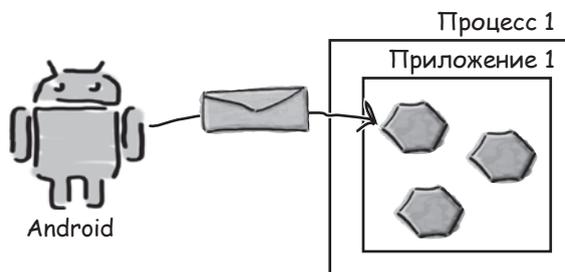
- 1** Пользователь решает, что он хочет выполнить приложение.
Пользователь щелкает на значке приложения на своем устройстве.



- 2** В файле `AndroidManifest.xml` приложения указано, какая активность должна использоваться как стартовая.
Система строит интент для запуска этой активности вызовом `startActivity(intent)`.



- 3** Android проверяет, существует ли процесс для этого приложения, и если не существует — создает новый процесс.
После этого создается новый объект активности — в данном случае `StopwatchActivity`.



История продолжается

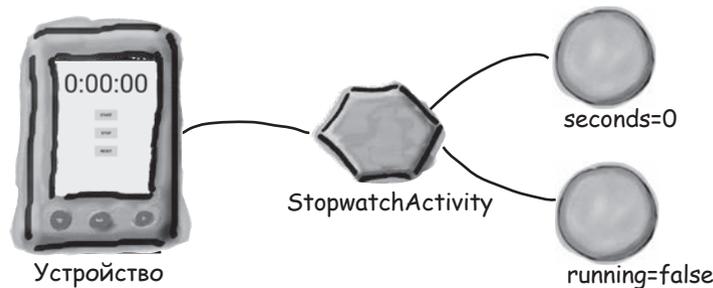
4 Вызывается метод `onCreate()` активности.

Метод включает вызов `setContentView()` с указанием макета. После этого секундомер запускается вызовом `runTimer()`.



5 При завершении работы `onCreate()` макет отображается на устройстве.

Метод `runTimer()` использует переменную `seconds` для получения текста, отображаемого в надписи, а переменную `running` — для принятия решения о том, нужно ли увеличивать число секунд. Так как переменная `running` в исходном состоянии равна `false`, количество секунд не увеличивается.



Часть Задаваемые Вопросы

В: Почему Android выполняет приложение в отдельном процессе?

О: По соображениям безопасности и стабильности: такая изоляция не позволяет одному приложению обратиться к данным другого приложения. Кроме того, если в одном приложении произойдет сбой, он не затронет другие приложения.

В: Для чего нужен метод `onCreate()`? Почему нельзя просто поместить код в конструктор?

О: После конструирования активности система Android должна настроить для нее рабочее окружение. Когда это будет сделано, Android вызывает `onCreate()`. Этим и объясняется то, что код настройки экрана размещается в `onCreate()`, а не в конструкторе.

В: А разве нельзя просто включить в `onCreate()` цикл для обновления таймера?

О: Нет, метод `onCreate()` должен завершиться до того, как экран приложения появится перед пользователем. С бесконечным циклом это никогда не произойдет.

В: Метод `runTimer()` получился каким-то сложным. Это действительно необходимо?

О: Код действительно непростой, но зато в любой ситуации, когда вам потребуется запланировать выполнение кода, решение получается похожим на `runTimer()`.



Тест-драйв

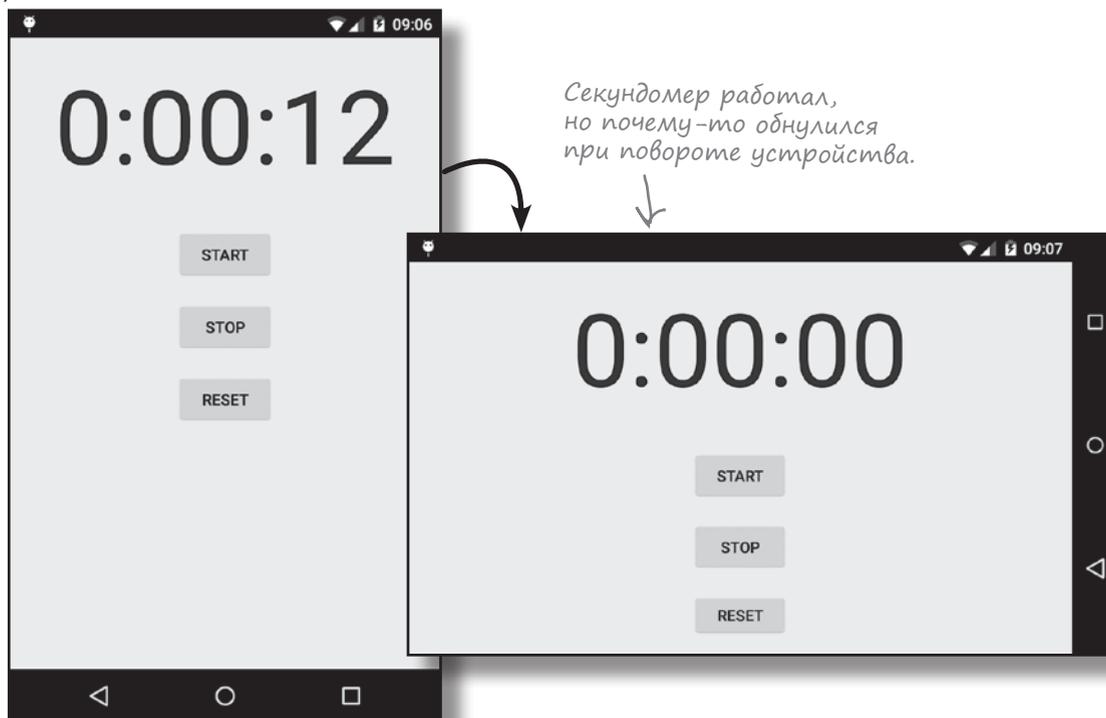
Когда мы запустили приложение в эмуляторе, оно прекрасно работало. Секундомер запускался, останавливался и обнулялся без малейших проблем – приложение работало в точности так, как и ожидалось.

Кнопки работают так, как ожидалось. Кнопка Start запускает секундомер, кнопка Stop его останавливает, а кнопка Reset обнуляет показания.



Но есть одна проблема...

При запуске на физическом устройстве приложение работало нормально... пока устройство оставалось в исходной ориентации. Когда устройство было повернуто, секундомер обнулится.



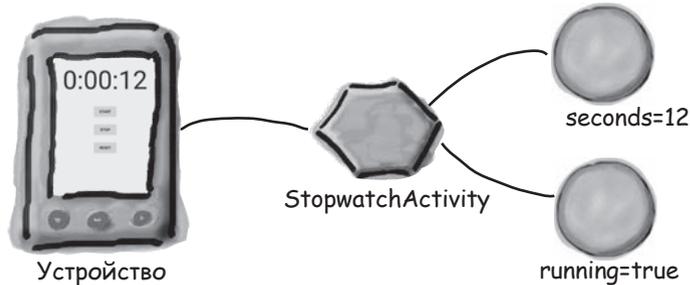
В мире Android приложения на удивление часто «ломаются» при повороте устройства. Прежде чем решать проблему, стоит лучше разобраться с ее причинами.

Что произошло?

Почему же поворот экрана нарушил работу приложения? Давайте внимательнее присмотримся к тому, что произошло на самом деле.

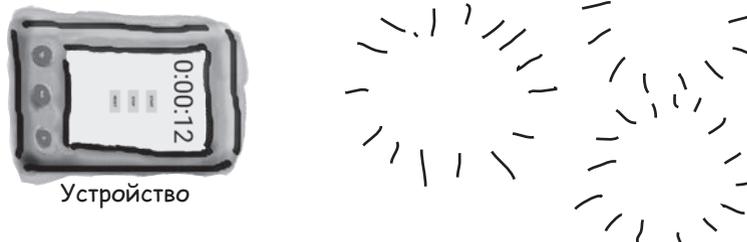
- 1 **Пользователь запускает приложение и щелкает на кнопке Start, чтобы секундомер заработал.**

Метод `runTimer()` начинает увеличивать число секунд, выводимое в надписи `time_view`, с использованием переменных `seconds` и `running`.



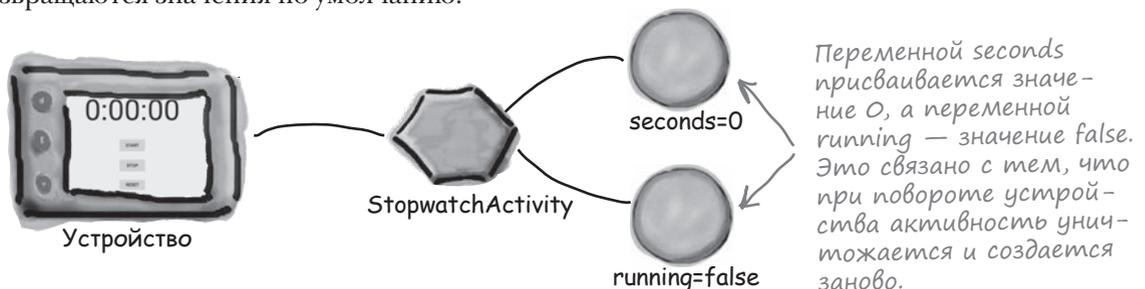
- 2 **Пользователь поворачивает устройство.**

Android видит, что ориентация и размер экрана изменились, и уничтожает активность вместе с переменными, которые использовались методом `runTimer()`.



- 3 **StopwatchActivity создается заново.**

Метод `onCreate()` выполняется заново, и вызывается метод `runTimer()`. Так как активность была создана заново, переменным `seconds` и `running` возвращаются значения по умолчанию.

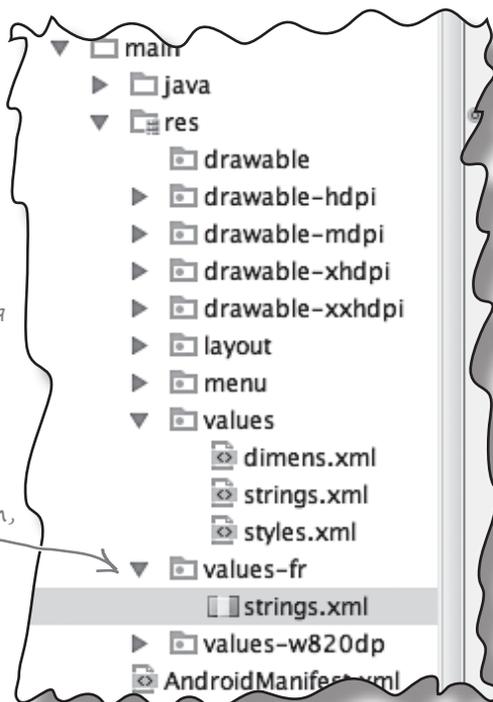


Поворот экрана изменяет конфигурацию устройства

При запуске активности в начале работы приложения Android принимает во внимание **конфигурацию устройства**. Под этим термином понимается как конфигурация физического устройства (размер экрана, ориентация экрана, наличие клавиатуры), так и параметры конфигурации, заданные пользователем (например, локальный контекст).

Система Android должна знать конфигурацию устройства при запуске активности, потому что эта информация может повлиять на ресурсы, необходимые приложению. Например, в горизонтальной ориентации может использоваться другой макет, а при выборе французского локального контекста может потребоваться другой набор строковых ресурсов.

Android-приложения могут содержать разные файлы ресурсов в папке `app/src/main/res`. Так, если на устройстве выбран французский локальный контекст, Android использует файл `strings.xml` из папки `values-fr`.



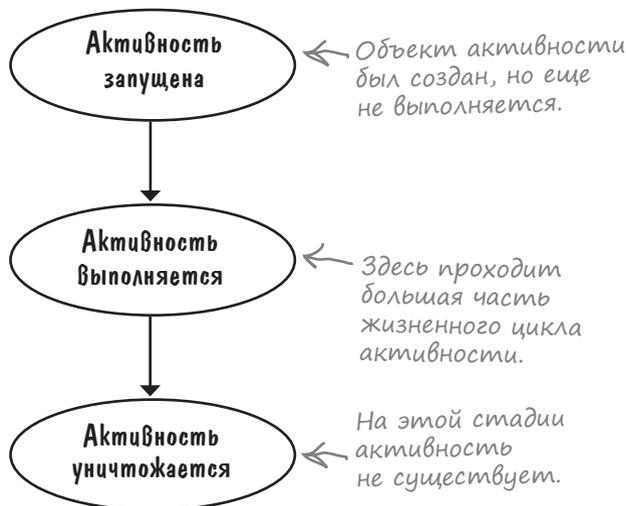
Конфигурация устройства включает как параметры, заданные пользователем (например, локальный контекст), так и параметры, относящиеся к физическому устройству (например, ориентация и размер экрана). При изменении любого из этих параметров активность уничтожается и создается заново.

При изменении конфигурации устройства все компоненты приложения, отображающие пользовательский интерфейс, должны быть обновлены в соответствии с новой конфигурацией. Если повернуть устройство, Android замечает, что ориентация и размеры экрана изменились, и интерпретирует этот факт как изменение конфигурации устройства. Текущая активность уничтожается и создается заново, чтобы приложение могло выбрать ресурсы, соответствующие новой конфигурации.

От рождения до смерти: состояния активности

При создании и уничтожении активность переходит между несколькими состояниями.

Главным состоянием активности является состояние **выполнения** (или **активное** состояние). Активность в состоянии выполнения находится на переднем плане экрана, обладает фокусом и доступна для взаимодействия с пользователем. Большую часть своего срока жизни активность пребывает в этом состоянии. Активность переходит в состояние выполнения от момента запуска и до того момента, когда она **уничтожается**.



На пути активности от запуска к уничтожению срабатывают ключевые методы жизненного цикла активности: `onCreate()` и `onDestroy()`. Ваша активность наследует эти методы жизненного цикла и может переопределить их при необходимости. Метод `onCreate()` вызывается сразу же после запуска активности. В этом методе выполняется вся обычная подготовка активности, например вызов `setContentView()`. Всегда переопределяйте этот метод. Если вы *не* переопределите его, то не сможете сообщить Android, какой макет должна использовать ваша активность. Метод `onDestroy()` вызывается непосредственно перед уничтожением активности. Существует немало ситуаций, в которых активность может уничтожаться, — например, если она получила приказ завершиться, если она создается заново из-за изменений в конфигурации устройства или если Android решает уничтожить активность для экономии памяти.

Сейчас мы подробнее рассмотрим, какое место эти методы занимают в состояниях активности.

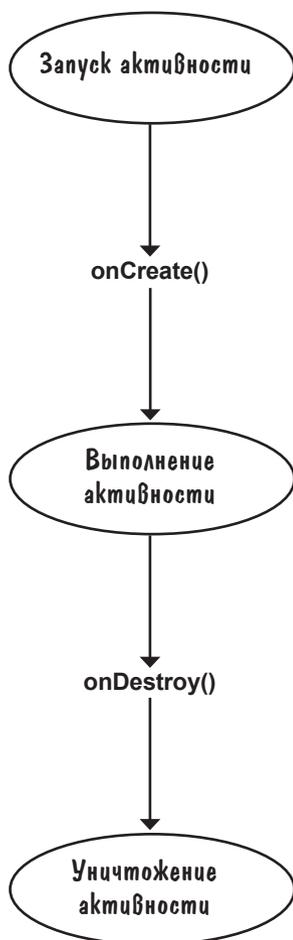
Активность выполняется, когда она находится на переднем плане на экране.

Метод `onCreate()` вызывается при создании активности; именно здесь происходит основная настройка активности.

Метод `onDestroy()` вызывается непосредственно перед уничтожением активности.

Жизненный цикл активности: от создания до уничтожения

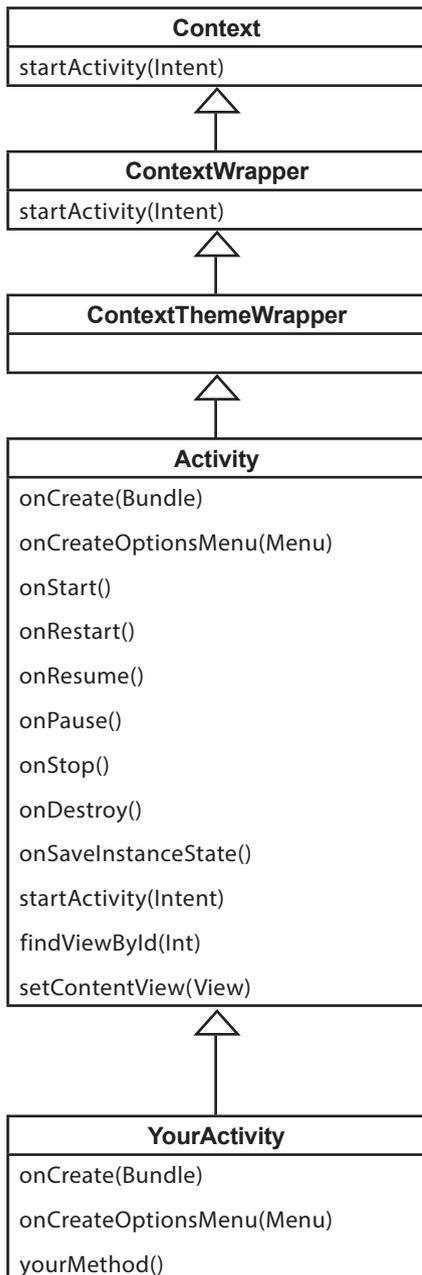
Ниже приведен обзор жизненного цикла активности от запуска до уничтожения. Как вы увидите позднее в этой главе, некоторые подробности были опущены, но сейчас нас интересуют только методы `onCreate()` и `onDestroy()`.



- 1 Запуск активности.**
Android создает объект активности и выполняет его конструктор.
- 2 Метод `onCreate()` выполняется непосредственно после запуска активности.**
В методе `onCreate()` должен размещаться весь код инициализации, так как этот метод всегда вызывается после запуска активности и до начала ее выполнения.
- 3 Во время выполнения активность находится на переднем плане, а пользователь может взаимодействовать с ней.**
В этой фазе проходит большая часть жизненного цикла активности.
- 4 Метод `onDestroy()` вызывается непосредственно перед уничтожением активности.**
Метод `onDestroy()` позволяет выполнить любые необходимые завершающие действия — например, освободить ресурсы.
- 5 После выполнения метода `onDestroy()` активность уничтожается.**
Активность перестает существовать.

Активность наследует свои методы жизненного цикла

Как было показано ранее, ваша активность расширяет класс `android.app.Activity`. Именно этот класс предоставляет активности доступ к методам жизненного цикла Android:



Абстрактный класс Context

(`android.content.Context`)

Интерфейс к глобальной информации об окружении приложения; открывает доступ к ресурсам приложения, классам и операциям уровня приложения.

Класс ContextWrapper

(`android.content.ContextWrapper`)

Промежуточная реализация для Context.

Класс ContextThemeWrapper

(`android.view.ContextThemeWrapper`)

ContextThemeWrapper позволяет изменить тему оформления того, что содержится в ContextWrapper.

Класс Activity

(`android.app.Activity`)

Класс Activity реализует версии по умолчанию для методов жизненного цикла. Он также определяет такие методы, как `findViewById(Int)` и `setContentView(View)`.

Класс YourActivity

(`com.hfad.foo`)

Большая часть поведения вашей активности обслуживается методами суперкласса. Остается лишь переопределить те методы, которые нужны вам.

Что делать при изменении конфигурации?

Как вы видели, при повороте экрана у нашего приложения начались проблемы. Активность была уничтожена и создана заново, а это означает, что локальные переменные активности были потеряны. Что же делать в таких случаях? Как справиться с такими изменениями конфигурации устройства, как изменение ориентации экрана?

Есть два варианта: либо запретить Android перезапуск активности, либо сохранить текущее состояние, чтобы активность могла воссоздать себя в том же состоянии. Рассмотрим эти два варианта более подробно.

Запретить повторное создание активности

Первый вариант — запретить Android перезапуск активности при изменениях в конфигурации. Хотя мы покажем, как это делается, учтите, что это почти всегда не лучший вариант. Дело в том, что при повторном создании активности Android выбирает правильные ресурсы, соответствующие новой конфигурации. Если вы пропустите этот шаг, возможно, вам придется написать немалый объем кода, чтобы обработать изменение конфигурации вручную. Чтобы запретить Android заново создавать активность из-за изменений конфигурации, добавьте строку в элемент `activity` в файле `AndroidManifest.xml`:

```
android:configChanges="изменение_конфигурации"
```

где `изменение_конфигурации` — вид изменения конфигурации. В нашем случае система Android должна обойти изменение ориентации и размера экрана, поэтому в файл `AndroidManifest.xml` нужно будет добавить следующую строку:

```
<activity
    android:name="com.hfad.stopwatch.StopwatchActivity"
    android:label="@string/app_name"
    android:configChanges="orientation|screenSize" >
```

Обнаружив такое изменение конфигурации, Android вызывает метод `onConfigurationChanged(Configuration)` вместо того, чтобы создавать активность заново:

```
public void onConfigurationChanged(Configuration config) {
}
```

При необходимости вы можете реализовать этот метод, чтобы он реагировал на изменение конфигурации.



Будьте
осторожны!

Такой подход
к изменениям
конфигурации
следует при-
менять только

в крайнем случае.

При этом игнорируется встроенное поведение Android, а это может создать проблемы.



Знак | означает, что игнорируются оба вида изменения конфигурации. Дело в том, что на большинстве устройств экран имеет прямоугольную форму, поэтому поворот устройства изменяет как ориентацию, так и размер экрана.

Или сохранить текущее состояние...

Более правильный способ обработки изменений конфигурации, который вы будете применять чаще всего, — сохранение текущего состояния активности и ее последующее воссоздание в методе `onCreate()`.

Чтобы сохранить текущее состояние активности, необходимо реализовать метод `onSaveInstanceState()`. Метод `onSaveInstanceState()` вызывается перед уничтожением активности; это означает, что вам представится возможность сохранить все значения, которые нужно сохранить, прежде чем они будут безвозвратно потеряны.

Метод `onSaveInstanceState()` получает один параметр типа `Bundle`. Тип `Bundle` позволяет объединить разные типы данных в один объект:

```
public void onSaveInstanceState(Bundle savedInstanceState) {
}
```

Метод `onCreate()` получает параметр `Bundle`. Таким образом, если вы добавите значения переменных `running` и `seconds` в `Bundle`, метод `onCreate()` сможет восстановить их при повторном создании активности. Для включения пар «имя/значение» в `Bundle` используются методы `Bundle`, которые имеют следующую форму:

```
bundle.put*("name", value)
```

где `bundle` — имя объекта `Bundle`, `*` — тип сохраняемого значения, а `name` и `value` — имя и значение в сохраняемых данных. Например, для включения в `Bundle` значения `seconds` типа `int` используется следующая команда:

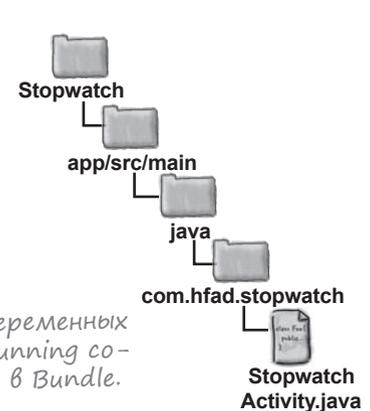
```
bundle.putInt("seconds", seconds);
```

В `Bundle` можно сохранить сколько угодно пар данных «имя/значение». Вот как выглядит полная реализация метода `onSaveInstanceState()` в нашем примере:

```
@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
    savedInstanceState.putInt("seconds", seconds);
    savedInstanceState.putBoolean("running", running);
}
```

После того как значения переменных были сохранены в `Bundle`, их можно будет использовать в методе `onCreate()`.

Метод `onSaveInstanceState()` вызывается перед `onDestroy()`. Он предоставляет возможность сохранить состояние активности перед ее уничтожением.



Значения переменных `seconds` и `running` сохраняются в `Bundle`.

...а затем восстановить состояние в onCreate()

Как упоминалось ранее, метод onCreate() получает один параметр Bundle. Если активность создается с нуля, то этот параметр содержит null. Но если активность создается заново, а созданию предшествовал вызов onSaveInstanceState(), активности передается объект Bundle, использованный в onSaveInstanceState():

```
protected void onCreate(Bundle savedInstanceState) {
    ...
}
```

Значения из Bundle читаются методами вида

```
bundle.get*("name");
```

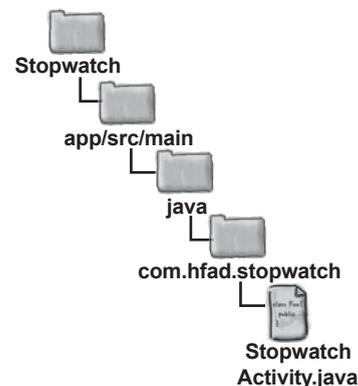
где bundle — имя объекта Bundle, * — тип значения, которое вы хотите прочитать, а name — имя из пары «имя/значение», заданной на предыдущей странице. Например, для получения из Bundle значения seconds типа int используется команда:

```
int seconds = bundle.getInt("seconds");
```

Если собрать воедино все сказанное, метод onCreate() принимает следующий вид:

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_stopwatch);
    if (savedInstanceState != null) {
        seconds = savedInstanceState.getInt("seconds");
        running = savedInstanceState.getBoolean("running");
    }
    runTimer();
}
```

Как же этот способ работает на практике?



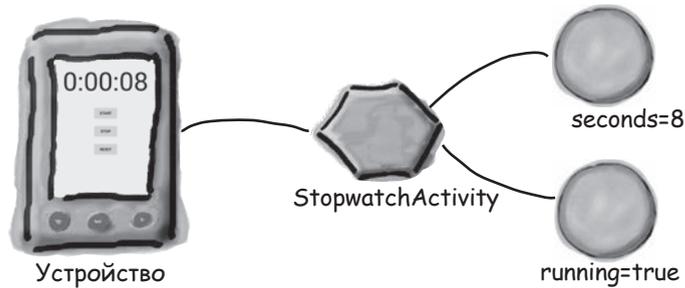
Получить значения переменных seconds и running из Bundle.

Задание!

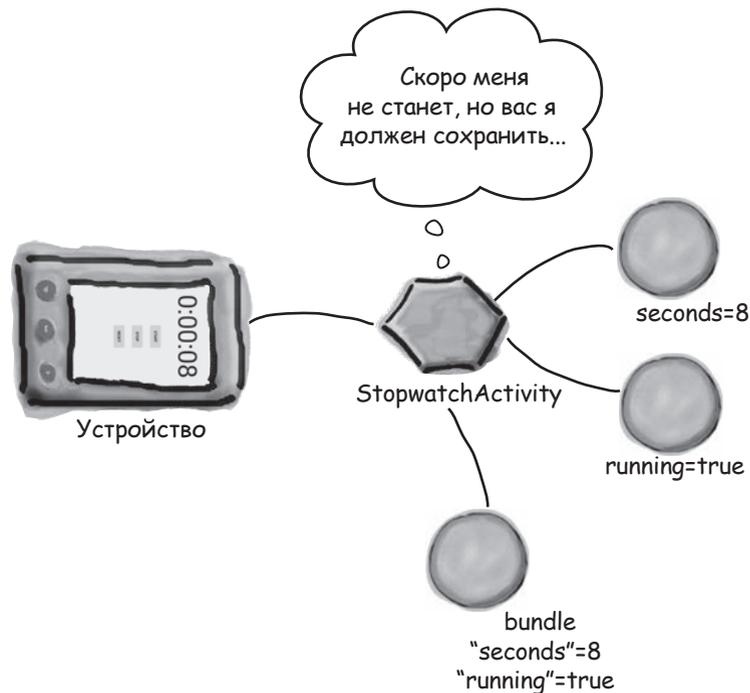
Не забудьте обновить метод onCreate() и добавить метод onSaveInstanceState().

Что происходит при запуске приложения

- 1** Пользователь запускает приложение и щелкает на кнопке Start, чтобы запустить отсчет времени.
Метод `runTimer()` начинает увеличивать число секунд, выводимое в надписи `time_view`.

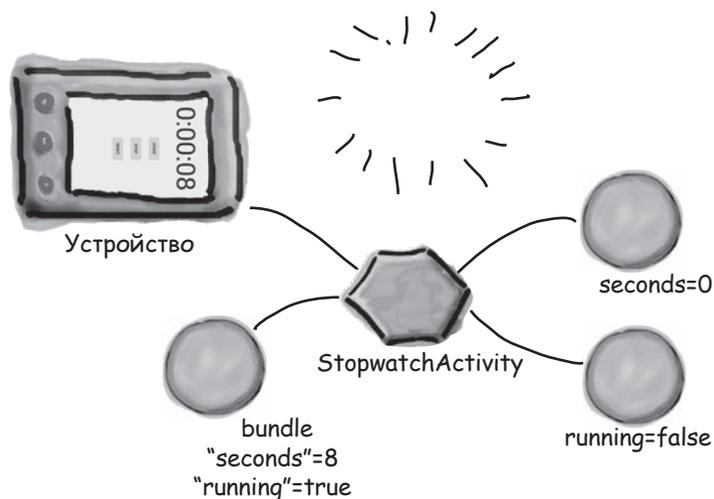


- 2** Пользователь поворачивает устройство.
Android рассматривает это событие как изменение конфигурации и готовится к уничтожению активности. Перед уничтожением активности вызывается метод `onSaveInstanceState()`. Метод `onSaveInstanceState()` сохраняет значения переменных `seconds` и `running` в `Bundle`.

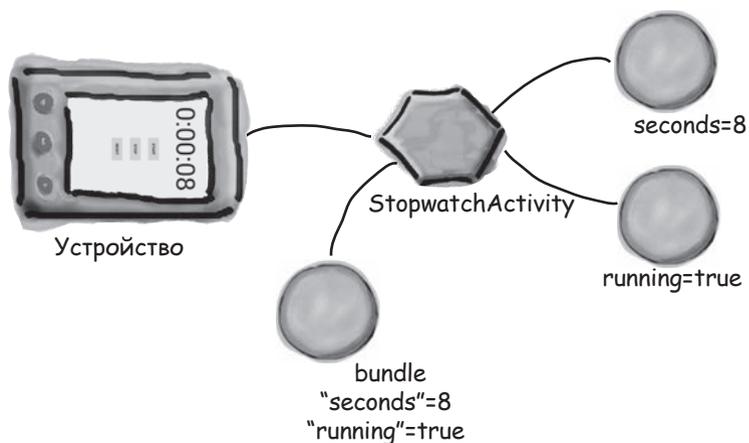


История продолжается

- 3 **Android уничтожает активность, после чего создает ее заново.**
Вызывается метод `onCreate()`, и ему передается объект `Bundle`.



- 4 **Объект Bundle содержит значения переменных `seconds` и `running` на момент уничтожения активности.**
Код метода `onCreate()` присваивает новым переменным значения из `Bundle`.



- 5 **Вызывается метод `runTimer()`, и таймер продолжает работать с того момента, на котором он остановился.**
Показания секундомера отображаются на устройстве.





Тест-драйв

Внесите изменения в код активности и запустите приложение. Щелкните на кнопке Start — секундомер запускается и продолжает отсчет времени при повороте устройства.



Часть Задаваемые Вопросы

В: Почему Android создает заново активность при простом повороте экрана?

О: Метод `onCreate()` обычно используется для настройки экрана. Если код `onCreate()` зависит от конфигурации экрана (например, при использовании разных макетов для горизонтальной и вертикальной ориентации), метод `onCreate()` должен вызываться при каждом изменении конфигурации. Кроме того, если пользователь сменит локальный контекст, пользовательский интерфейс необходимо создать заново на выбранном языке.

В: Почему Android не сохраняет все переменные экземпляра автоматически? Почему мне нужно писать весь этот код самостоятельно?

О: Далеко не всегда нужно сохранять все переменные экземпляра. Например, в вашей программе может использоваться переменная для хранения текущей ширины экрана. Значение такой переменной должно быть вычислено заново при следующем вызове `onCreate()`.

В: Класс `Bundle` — разновидность ассоциативных массивов из Java?

О: Нет, но проектировщики намеренно сделали его похожим на `java.util.Map`. Класс `Bundle` предоставляет дополнительные возможности — например, объекты `Bundle` могут передаваться между процессами. И это чрезвычайно полезно, потому что ОС Android остается в курсе состояния активности.

Жизнь активности не ограничивается созданием и уничтожением

До настоящего момента мы рассмотрели фазы создания и уничтожения в жизненном цикле; также вы узнали, как обрабатываются изменения конфигурации — например, изменение ориентации экрана. Однако в жизни приложения существуют и другие события, обработка которых поможет вам добиться того, чтобы приложение работало так, как вам нужно.

Например, предположим, что во время работы секундомера поступил телефонный звонок. Хотя секундомер и не виден на экране, он будет продолжать работу. Но что, если вы хотите, чтобы секундомер на это время останавливался и запускался снова, когда приложение снова становится видимым?

← Даже если вы не хотите, чтобы ваш секундомер вел себя подобным образом, это отличный повод рассмотреть другие методы жизненного цикла.

Старт, остановка и перезапуск

К счастью, использование правильных методов жизненного цикла позволяет легко обрабатывать действия, связанные с видимостью приложения. Кроме методов `onCreate()` и `onDestroy()`, связанных с началом и завершением всего жизненного цикла активности, также существуют другие методы жизненного цикла, связанные с видимостью активности. Есть три ключевых метода жизненного цикла, связанных с переходами активности в видимое или невидимое состояние. Это методы `onStart()`, `onStop()` и `onRestart()`. Как и методы `onCreate()` и `onDestroy()`, они наследуются от класса `Android Activity`.

Метод `onStart()` вызывается, когда активность становится видимой для пользователя.

Метод `onStop()` вызывается, когда активность перестает быть видимой для пользователя. Это может произойти из-за того, что она полностью закрывается другой активностью, отображаемой поверх нее, или из-за того, что активность готовится к уничтожению. Если метод `onStop()` вызывается из-за того, что активность готовится к уничтожению, перед `onStop()` вызывается метод `onSaveInstanceState()`.

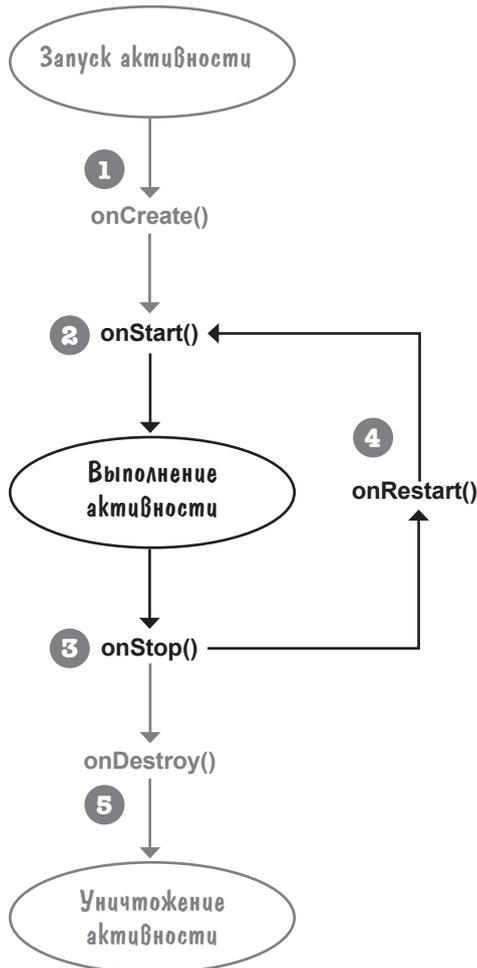
Метод `onRestart()` вызывается перед тем, как активность, ставшая невидимой, снова должна появиться на экране.

На следующей странице разберемся, какое отношение эти методы имеют к методам `onCreate()` и `onDestroy()`.

Активность находится в состоянии остановки, если она полностью закрыта другой активностью и невидима для пользователя. При этом активность продолжает существовать на заднем плане и сохраняет всю информацию состояния.

Жизненный цикл активности: видимость

Ниже диаграмма жизненного цикла активности, приводившаяся ранее в этой главе, дополняется методами `onStart()`, `onStop()` и `onRestart()` (аспекты, которые сейчас представляют для нас интерес, выделены жирным шрифтом):



- 1** Активность запускается, выполняется ее метод `onCreate()`.

Выполняется код инициализации активности из метода `onCreate()`. В этой точке активность еще не видна, так как метод `onStart()` еще не вызывался.

- 2** Метод `onStart()` выполняется после метода `onCreate()`. Он вызывается, когда активность собирается стать видимой.

После выполнения метода `onStart()` пользователь видит активность на экране.

- 3** Метод `onStop()` выполняется, когда активность перестает быть видимой пользователю.

После выполнения метода `onStop()` активность не видна на экране.

- 4** Если активность снова становится видимой пользователю, вызывается метод `onRestart()`, за которым следует вызов `onStart()`.

Активность может проходить через этот цикл многократно, если она несколько раз скрывается и снова становится видимой.

- 5** Наконец, активность уничтожается.

Метод `onStop()` обычно вызывается перед `onDestroy()`, но он может быть пропущен при серьезной нехватке памяти в системе.



Будьте осторожны!

Если на устройстве серьезно не хватает памяти, метод `onStop()` может не вызываться перед уничтожением активности.

Необходимо реализовать еще два метода жизненного цикла

Чтобы обновить приложение Stopwatch, необходимо сделать две вещи. Во-первых, необходимо реализовать метод `onStop()` активности, чтобы отсчет времени останавливался, если приложение стало невидимым. Когда это будет сделано, необходимо реализовать метод `onStart()`, чтобы отсчет времени возобновлялся, когда приложение становится видимым. Начнем с метода `onStop()`.

Реализация остановки секундомера в `onStop()`

Переопределите метод `onStop()` в классе Android Activity, добавив в активность следующий метод:

```
@Override
protected void onStop() {
    super.onStop();
}
```

Каждый раз, когда вы переопределяете один из методов жизненного цикла Android, важно начать с вызова версии метода из суперкласса:

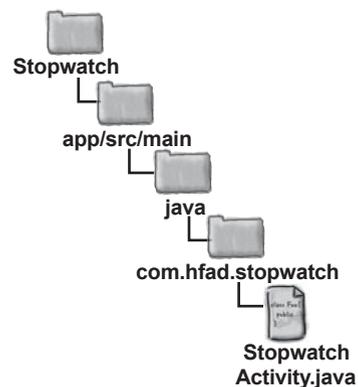
```
super.onStop();
```

Для этого есть пара причин. Во-первых, нужно позаботиться о том, чтобы активность выполнила все действия метода жизненного цикла суперкласса. Во-вторых, Android не простит попытки обойти этот шаг — будет сгенерировано исключение.

Отсчет времени должен останавливаться при вызове метода `onStop()`. Для этого булевой переменной `running` следует присвоить значение `false`. Полный метод выглядит так:

```
@Override
protected void onStop() {
    super.onStop();
    running = false;
}
```

Итак, теперь секундомер останавливается, когда активность становится невидимой. Теперь нужно сделать следующий шаг — снова запустить отсчет времени, когда активность станет видимой.



В переопределениях методов жизненного цикла активности должен вызываться метод суперкласса. Если этого не сделать, произойдет исключение.

Возьми в руку карандаш



Теперь ваша очередь. Измените код активности так, чтобы если секундомер работал перед вызовом `onStop()`, он снова запустился после получения фокуса активностью.

```
public class StopwatchActivity extends Activity {
    private int seconds = 0;
    private boolean running;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_stopwatch);
        if (savedInstanceState != null) {
            seconds = savedInstanceState.getInt("seconds");
            running = savedInstanceState.getBoolean("running");
        }
        runTimer();
    }

    @Override
    public void onSaveInstanceState(Bundle savedInstanceState) {
        savedInstanceState.putInt("seconds", seconds);
        savedInstanceState.putBoolean("running", running);
        savedInstanceState.putBoolean("wasRunning", wasRunning);
    }

    @Override
    protected void onStop() {
        super.onStop();
        running = false;
    }
}
```

Первая часть кода активности. Вам также придется реализовать метод `onStart()` и внести небольшие изменения в другие методы.

Возьми в руку карандаш



Решение

Теперь ваша очередь. Измените код активности так, чтобы если секундомер работал перед вызовом `onStop()`, он снова запускался после получения фокуса активностью.

```
public class StopwatchActivity extends Activity {
```

```
    private int seconds = 0;
    private boolean running;
```

```
    private boolean wasRunning;
```

```
    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_stopwatch);
```

```
        if (savedInstanceState != null) {
```

```
            seconds = savedInstanceState.getInt("seconds");
```

```
            running = savedInstanceState.getBoolean("running");
```

```
            wasRunning = savedInstanceState.getBoolean("wasRunning");
```

```
        }
```

```
        runTimer();
```

```
    }
```

```
    @Override
```

```
    public void onSaveInstanceState(Bundle savedInstanceState) {
```

```
        savedInstanceState.putInt("seconds", seconds);
```

```
        savedInstanceState.putBoolean("running", running);
```

```
        savedInstanceState.putBoolean("wasRunning", wasRunning);
```

```
    }
```

```
    @Override
```

```
    protected void onStop() {
```

```
        super.onStop();
```

```
        wasRunning = running;
```

```
        running = false;
```

```
    }
```

```
    @Override
```

```
    protected void onStart() {
```

```
        super.onStart();
```

```
        if (wasRunning) {
```

```
            running = true;
```

```
        }
```

```
    }
```

Мы добавили новую переменную `wasRunning` для хранения информации о том, работал ли секундомер перед вызовом метода `onStop()`. В зависимости от состояния переменной мы решаем, нужно ли снова запускать отсчет времени, когда активность снова становится видимой.

↑
Восстанавливаем состояние переменной `wasRunning`, если активность создается заново.

←
Сохранить состояние переменной `wasRunning`.

←
Сохранить информацию о том, работал ли секундомер на момент вызова метода `onStop()`.

←
Реализация метода `onStart()`. Если секундомер работал, то отсчет времени возобновляется.

Обновленный `kog StopwatchActivity`

Мы обновили код активности: если секундомер работал до потери фокуса, он должен возобновить отсчет времени при возвращении фокуса. Внесите следующие изменения в ваш код:

```
public class StopwatchActivity extends Activity {
    private int seconds = 0;
    private boolean running;
    private boolean wasRunning;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_stopwatch);
        if (savedInstanceState != null) {
            seconds = savedInstanceState.getInt("seconds");
            running = savedInstanceState.getBoolean("running");
            wasRunning = savedInstanceState.getBoolean("wasRunning");
        }
        runTimer();
    }

    @Override
    public void onSaveInstanceState(Bundle savedInstanceState) {
        savedInstanceState.putInt("seconds", seconds);
        savedInstanceState.putBoolean("running", running);
        savedInstanceState.putBoolean("wasRunning", wasRunning);
    }

    @Override
    protected void onStop() {
        super.onStop();
        wasRunning = running;
        running = false;
    }

    @Override
    protected void onStart() {
        super.onStart();
        if (wasRunning) {
            running = true;
        }
    }
}
```

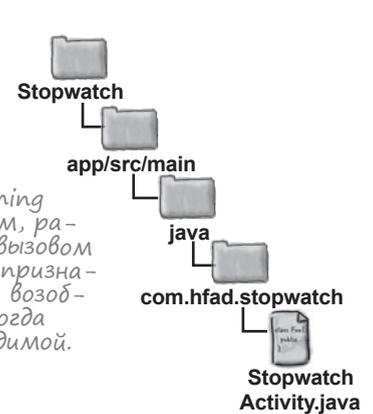
В новой переменной `wasRunning` хранится информация о том, работал ли секундомер перед вызовом метода `onStop()`. По этому признаку мы определяем, нужно ли возобновлять отсчет времени, когда активность становится видимой.

Восстановит состояние переменной `wasRunning`, если активность создается заново.

Сохранить состояние переменной `wasRunning`.

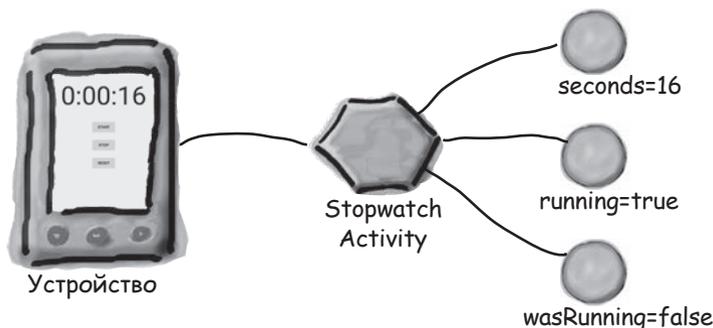
Сохранить информацию о том, работал ли секундомер на момент вызова метода `onStop()`.

Реализация метода `onStart()`. Если секундомер работал, то отсчет времени возобновляется.

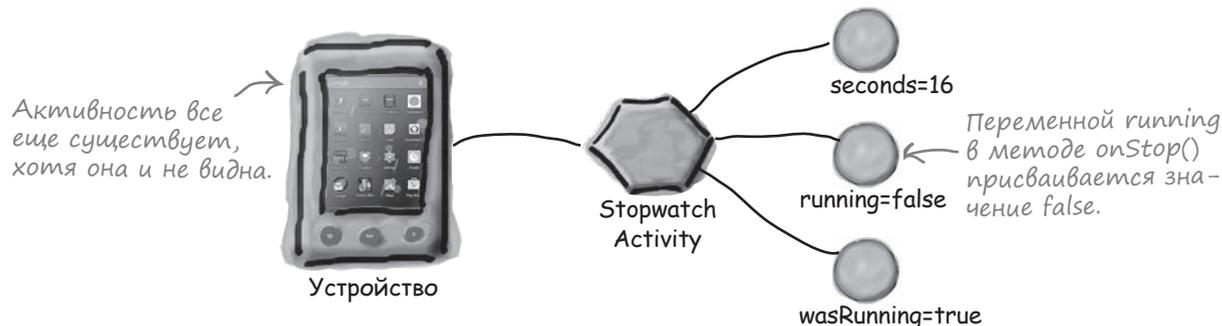


Что происходит при запуске приложения

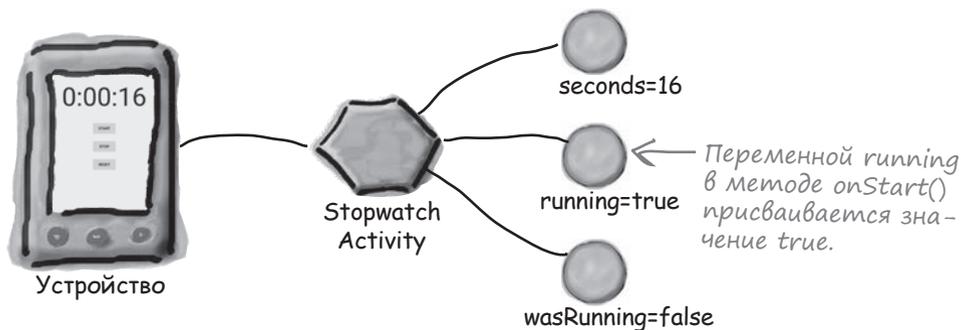
- 1** Пользователь запускает приложение и щелкает на кнопке **Start**, чтобы запустить отсчет времени.
Метод `runTimer()` начинает увеличивать число секунд, выводимое в надписи `time_view`.



- 2** Пользователь переходит к домашнему экрану устройства, так что приложение **Stopwatch** перестает быть видимым.
Вызывается метод `onStop()`, переменной `wasRunning` присваивается значение `true`, переменной `running` присваивается `false`, а отсчет времени останавливается.



- 3** Пользователь снова переходит к приложению **Stopwatch**.
Вызывается метод `onStart()`, переменной `running` присваивается значение `true`, а отсчет времени возобновляется.





Тест-драйв

Сохраните изменения в коде активности и запустите приложение. Как и прежде, щелчок на кнопке Start запускает секундомер. Отсчет времени останавливается, когда приложение становится невидимым, и возобновляется, когда приложение снова появляется на экране.



Часто задаваемые вопросы

В: А нельзя ли было воспользоваться методом `onRestart()`?

О: Метод `onRestart()` используется в тех случаях, когда приложение появляется после того, как оно ранее становилось невидимым. Он не выполняется тогда, когда активность становится видимой впервые. Мы хотели, чтобы приложение продолжало работать при повороте устройства.

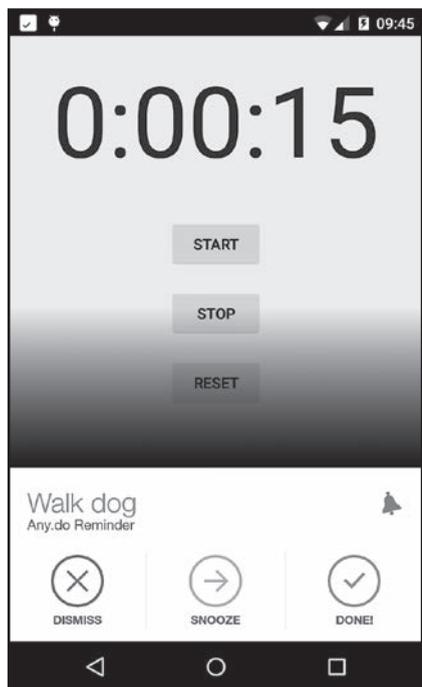
В: А что бы при этом изменилось?

О: При повороте устройства активность уничтожается, а на ее месте создается новая активность. Если бы код был включен в метод `onRestart()`, то он не выполнялся бы при повторном создании активности. С другой стороны, метод `onStart()` выполняется в обеих ситуациях.

А если приложение видимо только частично?

Итак, теперь вы знаете, что происходит при создании и уничтожении активности, а также когда активность скрывается и снова становится видимой. Однако существует еще одна важная ситуация: когда активность видима, но не обладает фокусом.

Если активность видима, но не имеет фокуса, она приостанавливается (paused). Это может произойти, если поверх нее отображается другая активность, которая занимает лишь часть экрана или частично прозрачна. Верхняя активность имеет фокус, но нижняя остается видимой — а следовательно, приостанавливается.



← Активность секундомера остается видимой, но она частично скрыта и не имеет фокуса.

← Активность из другого приложения, которое отображается поверх секундомера.

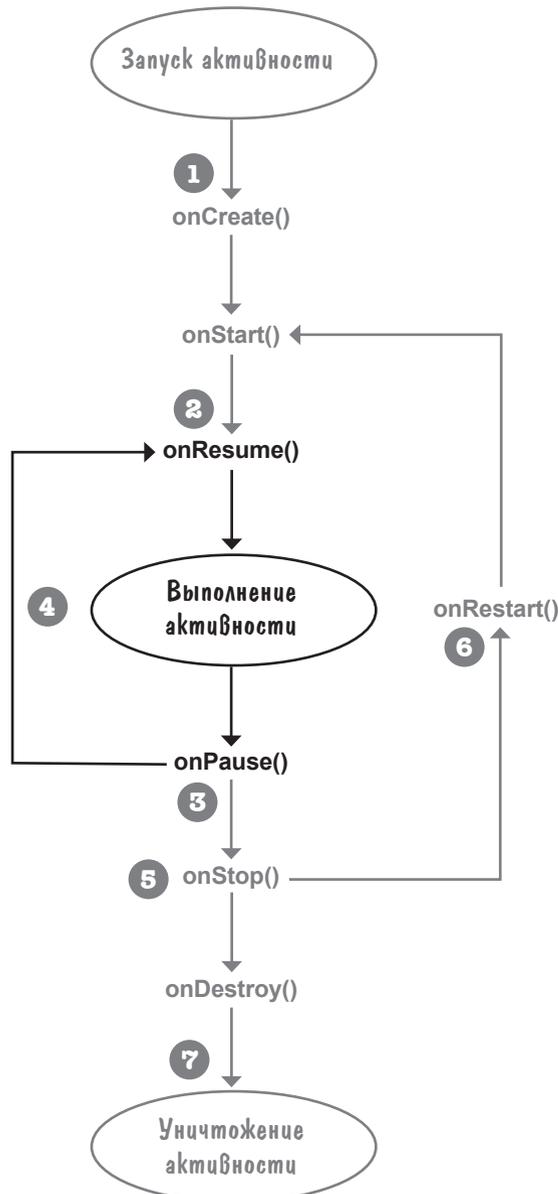
Активность находится в приостановленном состоянии, если она потеряла фокус, но остается видимой для пользователя. Такая активность продолжает существовать и сохраняет всю свою информацию состояния.

С приостановкой активности и ее последующим переходом в активное состояние связаны два метода жизненного цикла: `onPause()` и `onResume()`. Метод `onPause()` вызывается тогда, когда ваша активность видима, но фокус принадлежит другой активности. Метод `onResume()` вызывается непосредственно перед тем, как ваша активность начинает взаимодействовать с пользователем. Если ваше приложение должно как-либо реагировать на приостановку активности, вы должны реализовать эти методы.

На следующей странице показано, какое место эти методы занимают в схеме уже знакомых вам методов жизненного цикла.

Жизненный цикл активности: видимость

Дополним диаграмму жизненного цикла, приводившуюся ранее в этой главе, методами `onResume()` и `onPause()` (новые фрагменты выделены жирным шрифтом):



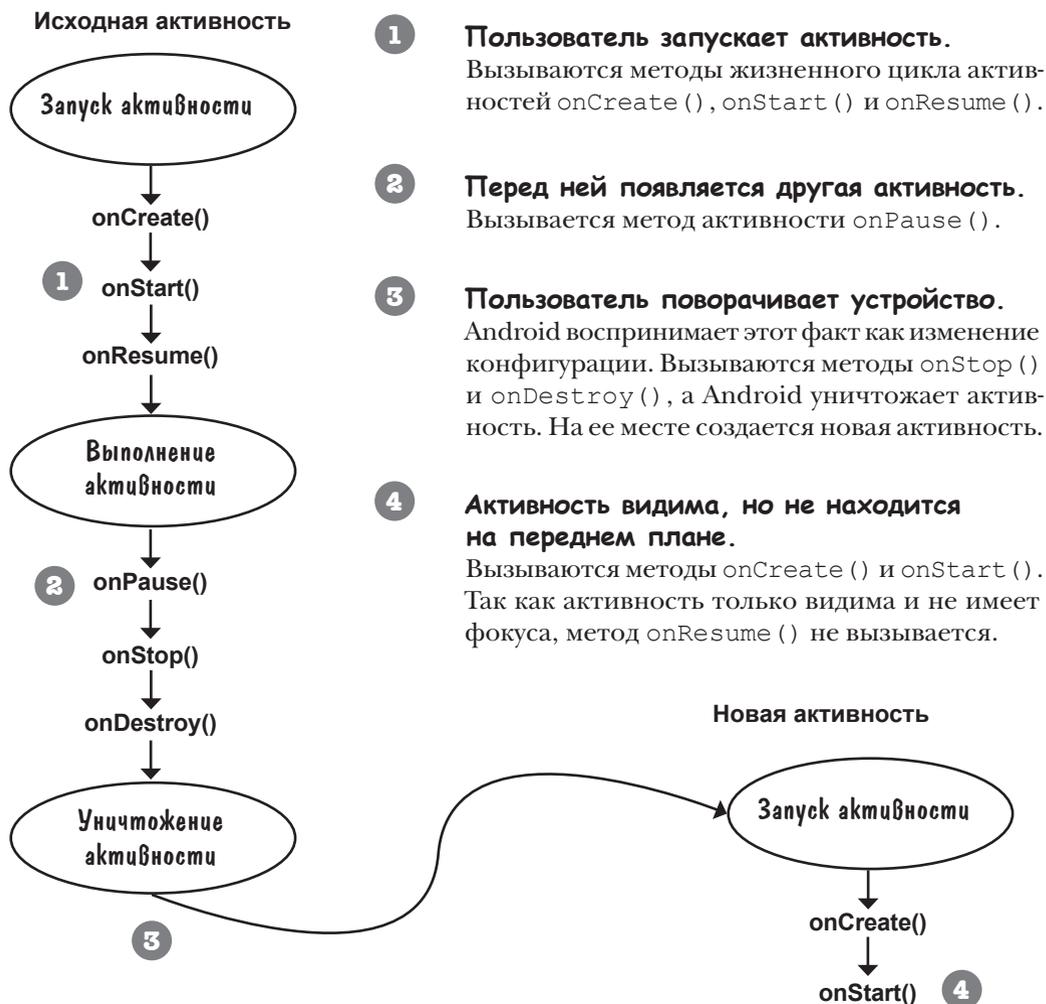
- 1 **Активность запускается, выполняются ее методы `onCreate()` и `onStart()`.**
В этой точке активность видна, но еще не обладает фокусом.
- 2 **Метод `onResume()` вызывается после метода `onStart()`. Он выполняется тогда, когда активность собирается перейти на передний план.**
После выполнения метода `onResume()` активность обладает фокусом, а пользователь может взаимодействовать с ней.
- 3 **Метод `onPause()` выполняется тогда, когда активность перестает находиться на переднем плане.**
После выполнения метода `onPause()` активность остается видимой, но не обладает фокусом.
- 4 **Если активность снова возвращается на передний план, вызывается метод `onResume()`.**
Активность, которая многократно теряет и получает фокус, может проходить этот цикл несколько раз.
- 5 **Если активность перестает быть видимой пользователю, вызывается метод `onStop()`.**
После выполнения метода `onStop()` активность не видна пользователю.
- 6 **Если активность снова станет видимой, вызывается метод `onRestart()`, за которым следуют `onStart()` и `onResume()`.**
Активность может проходить через этот цикл многократно.
- 7 **Наконец, активность уничтожается.**
При переходе от выполнения к уничтожению метод `onPause()` вызывается до того, как активность будет уничтожена. Также обычно при этом вызывается метод `onStop()`.



Ранее в этой главе вы говорили, что при повороте устройства пользователем активность уничтожается, и вместо нее создается новая активность. А что произойдет, если на момент поворота активность приостановлена? Пройдет ли она через те же методы жизненного цикла?

Хороший вопрос! Давайте подробнее разберемся в этом, прежде чем возвращаться к приложению Stopwatch.

Исходная активность проходит все свои методы жизненного цикла, от `onCreate()` до `onDestroy()`. Новая активность создается при уничтожении исходной. Так как новая активность не находится на переднем плане, вызываются только методы жизненного цикла `onCreate()` и `onStart()`:





Понятно, новая активность не переходит в состояние «выполнения», потому что она не находится на переднем плане. Но что, если пользователь полностью уйдет от активности, так что она даже не будет видна? Если активность останавливается, вызываются ли методы `onResume()` и `onPause()` перед `onStop()`?

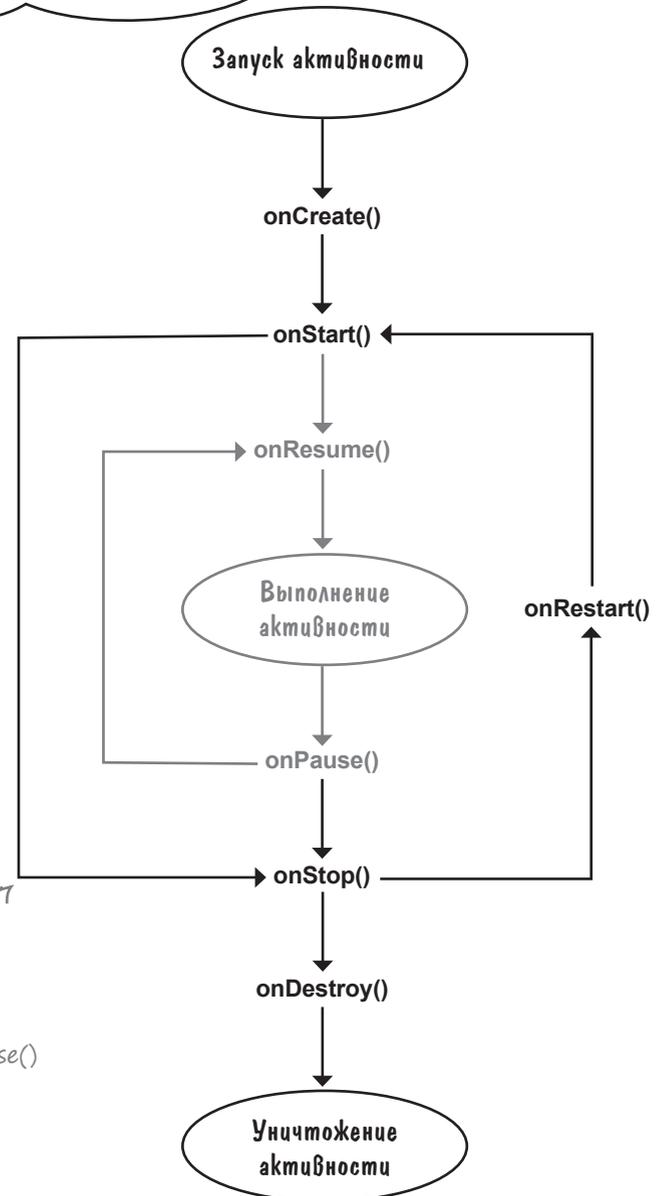
Активности могут переходить прямо от `onStart()` к `onStop()`, обходя вызовы `onPause()` и `onResume()`.

Если ваша активность видима, но никогда не находится на переднем плане и никогда не получает фокус, методы `onPause()` и `onResume()` **никогда не вызываются**.

Метод `onResume()` вызывается тогда, когда активность появляется на переднем плане и обладает фокусом. Если активность видима только за другими активностями, метод `onResume()` не вызывается.

Аналогичным образом метод `onPause()` вызывается тогда, когда активность уходит с переднего плана. Если активность никогда не находится на переднем плане, то и метод вызываться не будет.

Если активность останавливается или уничтожается до того, как она окажется на переднем плане, то за методом `onStart()` следует метод `onStop()`. Методы `onResume()` и `onPause()` не вызываются.



Прекращение отсчета времени при приостановке активности

Вернемся к приложению Stopwatch.

Пока что наш секундомер останавливается, если приложение Stopwatch становится невидимым, и снова запускается, когда приложение снова оказывается на экране. Давайте в дополнение к этому сделаем так, чтобы отсчет времени прекращался при приостановке активности и запускался снова при продолжении ее работы. Какие же методы жизненного цикла для этого следует реализовать?

Вообще говоря, следует использовать методы `onPause()` и `onResume()`, но мы пойдем еще дальше. **Эти методы будут использованы для замены уже реализованных нами вызовов `onStop()` и `onStart()`**. Если вы снова взглянете на диаграмму жизненного цикла, вызовы `onPause()` и `onResume()` производятся в дополнение к `onStop()` и `onStart()` при каждой остановке и старте активности. Мы используем одни и те же методы в обеих ситуациях, так как приложение должно вести себя одинаково.

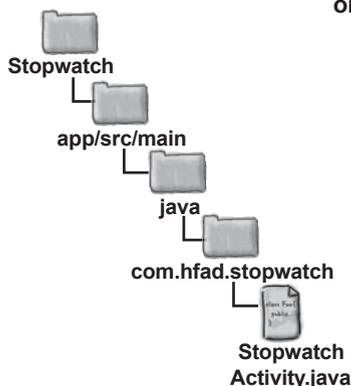
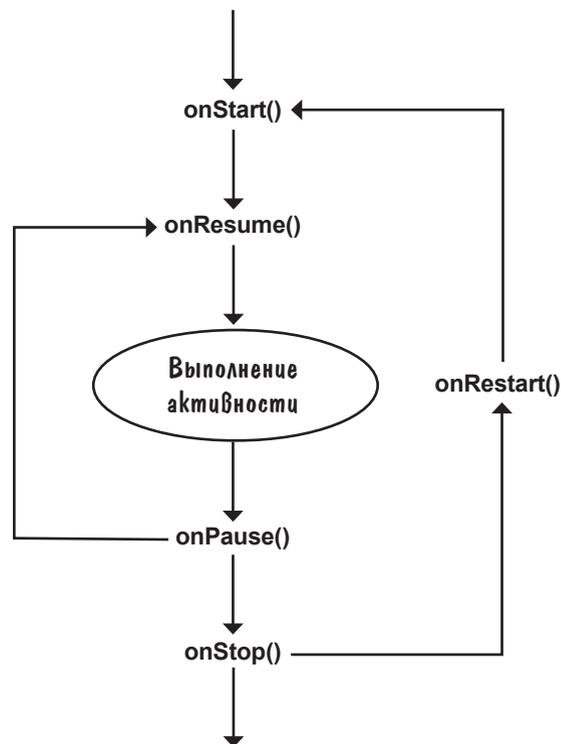
Ниже приведена наша версия метода `onPause()`:

```
@Override
protected void onPause() {
    super.onPause();
    wasRunning = running;
    running = false;
}
```

А вот как выглядит метод `onResume()`:

```
@Override
protected void onResume() {
    super.onResume();
    if (wasRunning) {
        running = true;
    }
}
```

Посмотрим, что произойдет при запуске приложения.

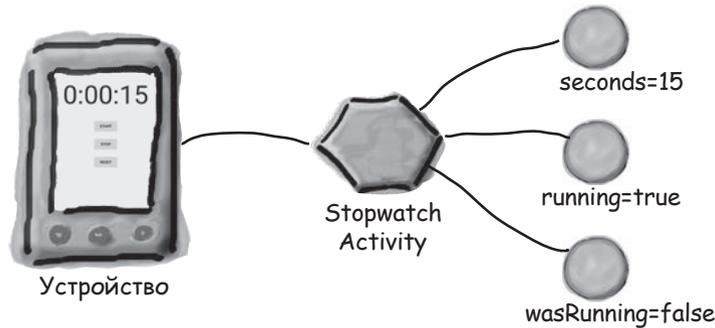


Задание!

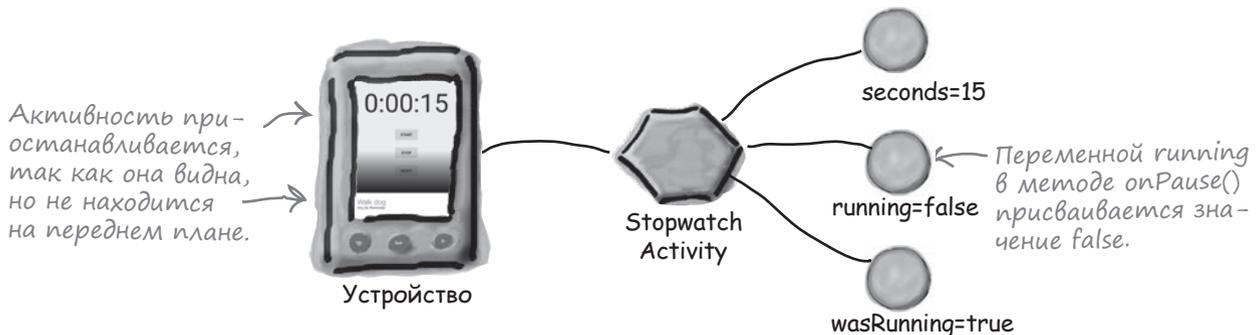
Замените методы `onStop()` и `onStart()` в своем коде методами `onPause()` и `onResume()`, приведенными на этой странице.

Что происходит при запуске приложения

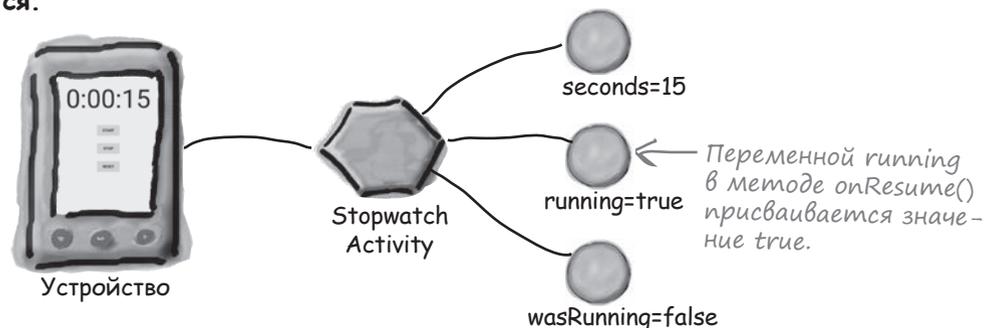
- 1 Пользователь запускает приложение и щелкает на кнопке **Start**, чтобы запустить отсчет времени. Метод `runTimer()` начинает увеличивать число секунд, выводимое в надписи `time_view`.



- 2 На переднем плане появляется другая активность, частично скрывающая **StopwatchActivity**. Вызывается метод `onPause()`, переменной `wasRunning` присваивается значение `true`, переменной `running` присваивается `false`, а отсчет времени прекращается.



- 3 Когда **StopwatchActivity** возвращается на передний план, вызывается метод `onResume()`, переменной `running` присваивается значение `true`, а отсчет времени возобновляется.





Тест-драйв

Сохраните изменения в коде активности и запустите приложение. Щелчок на кнопке Start запускает секундомер. Отсчет времени останавливается, когда приложение частично закрывается другой активностью, и возобновляется, когда приложение возвращается на передний план.



Часто задаваемые вопросы

В: Значит, вызов некоторых методов жизненного цикла не гарантирован? Похоже, это может обернуться ненадежным поведением приложений. Верно?

О: В некоторых обстоятельствах Android может пропустить вызовы таких методов, как `onStop()` и `onPause()`. Такие методы обычно содержат код, относящийся к завершающим действиям.

Методы `onCreate()` и `onStart()` всегда вызываются в правильные моменты; это означает, что ваше приложение может удостовериться в правильности своего состояния в начале работы — это намного важнее.

Очень важно, чтобы вы действительно хорошо поняли, какие методы жизненного цикла вызываются в тех или иных обстоятельствах.

Полный код активности

Ниже приведен полный код активности `StopwatchActivity.java` завершенного приложения:

```
package com.hfad.stopwatch;

import android.os.Bundle;
import android.os.Handler;
import android.app.Activity;
import android.view.View;
import android.widget.TextView;

public class StopwatchActivity extends Activity {
    //Количество секунд на секундомере.
    private int seconds = 0;
    //Секундомер работает?
    private boolean running;
    private boolean wasRunning;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_stopwatch);
        if (savedInstanceState != null) {
            seconds = savedInstanceState.getInt("seconds");
            running = savedInstanceState.getBoolean("running");
            wasRunning = savedInstanceState.getBoolean("wasRunning");
        }
        runTimer();
    }

    @Override
    protected void onPause() {
        super.onPause();
        wasRunning = running;
        running = false;
    }

    @Override
    protected void onResume() {
        super.onResume();
        if (wasRunning) {
            running = true;
        }
    }
}
```

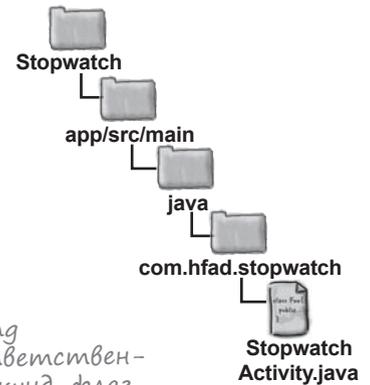
В переменных `seconds`, `running` и `wasRunning` хранятся соответственно количество прошедших секунд, флаг отсчета времени и флаг отсчета времени до приостановки активности.

Получить предыдущее состояние секундомера, если активность была уничтожена и создана заново.

Если активность приостановлена, то отсчет времени прекращается.

Если активность возобновляет работу, снова запустить отсчет времени, если он происходил до этого.

Код продолжается на следующей странице.



Код активности (продолжение)

```

@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
    savedInstanceState.putInt("seconds", seconds);
    savedInstanceState.putBoolean("running", running);
    savedInstanceState.putBoolean("wasRunning", wasRunning);
}

//Запустить секундомер при щелчке на кнопке Start.
public void onClickStart(View view) {
    running = true;
}

//Остановить секундомер при щелчке на кнопке Stop.
public void onClickStop(View view) {
    running = false;
}

//Обнулить секундомер при щелчке на кнопке Reset.
public void onClickReset(View view) {
    running = false;
    seconds = 0;
}

//Обновление показаний таймера.
private void runTimer() {
    final TextView timeView = (TextView)findViewById(R.id.time_view);
    final Handler handler = new Handler();
    handler.post(new Runnable() {
        @Override
        public void run() {
            int hours = seconds/3600;
            int minutes = (seconds%3600)/60;
            int secs = seconds%60;
            String time = String.format("%d:%02d:%02d",
                hours, minutes, secs);
            timeView.setText(time);
            if (running) {
                seconds++;
            }
            handler.postDelayed(this, 1000);
        }
    });
}
}

```

Сохранить состояние секундомера, если он готовится к уничтожению.

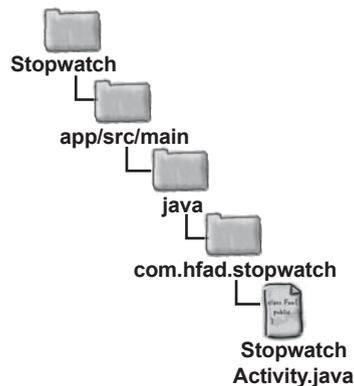


Вызывается при щелчке на кнопке Start.

Вызывается при щелчке на кнопке Stop.

Вызывается при щелчке на кнопке Reset.

Метод runTimer() использует объект Handler для увеличения числа секунд и обновления надписи.



СТАНЬ активностью



Справа приведен код активности. Ваша задача — представить себя на месте активности и определить, какие сегменты кода будут выполняться в каждой из ситуаций, описанных ниже. Мы поместили сегменты кода буквами и выполнили первое задание, чтобы вам было проще взяться за работу.

Пользователь запускает активность и начинает работать с ней.

Сегменты А, С, D. Активность создается, затем становится видимой, после чего получает фокус.

Пользователь запускает активность, начинает работать с ней, после чего переключается на другое приложение.

Здесь придется потрудиться.

Пользователь запускает активность, начинает работать с ней, поворачивает устройство, переключается на другое приложение, а затем возвращается к активности.

```
...
class MyActivity extends Activity{

    protected void onCreate(
        Bundle savedInstanceState) {
        A //Выполняется код А
        ...
    }

    protected void onPause() {
        B //Выполняется код В
        ...
    }

    protected void onRestart() {
        C //Выполняется код С
        ...
    }

    protected void onResume() {
        D //Выполняется код D
        ...
    }

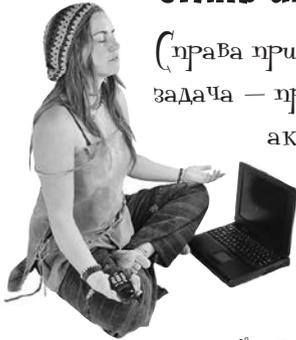
    protected void onStop() {
        E //Выполняется код E
        ...
    }

    protected void onRecreate() {
        F //Выполняется код F
        ...
    }

    protected void onStart() {
        G //Выполняется код G
        ...
    }

    protected void onDestroy() {
        H //Выполняется код H
        ...
    }
}
```

СТАНЬ активностью. Решение



Справа приведен код активности. Ваша задача — представить себя на месте активности и определить, какие сегменты кода будут выполняться в каждой из ситуаций, описанных ниже. Мы пометили сегменты кода буквами и выполнили первое задание, чтобы вам было проще взяться за работу.

Пользователь запускает активность и начинает работать с ней.

Сегменты A, C, D. Активность создается, затем становится видимой, после чего получает фокус.

Пользователь запускает активность, начинает работать с ней, после чего переключается на другое приложение.

Сегменты A, C, D, B, E. Активность создается, становится видимой и получает фокус. Когда пользователь переключается на другое приложение, активность теряет фокус и перестает быть видимой.

Пользователь запускает активность, начинает работать с ней, поворачивает устройство, переключается на другое приложение, а затем возвращается к активности.

Сегменты A, C, D, B, E, H, A, C, D, B, E, C, G, D. Сначала активность создается, становится видимой и получает фокус. При повороте устройства активность теряет фокус, перестает быть видимой и уничтожается. Затем она создается снова, становится видимой и получает фокус. Когда пользователь переключается на другое приложение и обратно, активность теряет фокус, теряет видимость, снова становится видимой и снова получает фокус.

```

...
class MyActivity extends Activity{

    protected void onCreate(
        Bundle savedInstanceState) {
        A //Выполняется код A
        ...
    }

    protected void onPause() {
        B //Выполняется код B
        ...
    }

    protected void onRestart() {
        C //Выполняется код C
        ...
    }

    protected void onResume() {
        D //Выполняется код D
        ...
    }

    protected void onStop() {
        E //Выполняется код E
        ...
    }

    protected void onRecreate() {
        F //Выполняется код F
        ...
    }

    protected void onStart() {
        G //Выполняется код G
        ...
    }

    protected void onDestroy() {
        H //Выполняется код H
        ...
    }
}
    
```

Не существует метода жизненного цикла с именем onRecreate().

Краткое руководство по методам жизненного цикла

Метод	Когда вызывается	Следующий метод
onCreate()	При создании активности. Используется для обычной статической инициализации — например, создания представлений. Также предоставляет объект <code>Bundle</code> с сохраненным предыдущим состоянием активности.	<code>onStart()</code>
onRestart()	Если активность ранее была остановлена — перед ее повторным стартом.	<code>onStart()</code>
onStart()	Когда активность становится видимой. Сопровождается вызовом <code>onResume()</code> , если активность выходит на передний план, или вызовом <code>onStop()</code> , если активность скрывается.	<code>onResume()</code> или <code>onStop()</code>
onResume()	При выходе активности на передний план.	<code>onPause()</code>
onPause()	При уходе активности с переднего плана из-за продолжения работы другой активности. Следующая активность сможет продолжить работу только после завершения этого метода, поэтому его код должен выполняться быстро. Сопровождается вызовом <code>onResume()</code> , если активность возвращается на передний план, или вызовом <code>onStop()</code> , если активность скрывается.	<code>onResume()</code> или <code>onStop()</code>
onStop()	Когда активность перестает быть видимой из-за того, что другая активность накрывает ее или активность уничтожается. Сопровождается вызовом <code>onRestart()</code> , если активность снова становится видимой, или вызовом <code>onDestroy()</code> , если активность уничтожается.	<code>onRestart()</code> или <code>onDestroy()</code>
onDestroy()	Перед уничтожением активности или ее завершением.	---



Ваш инструментарий Android

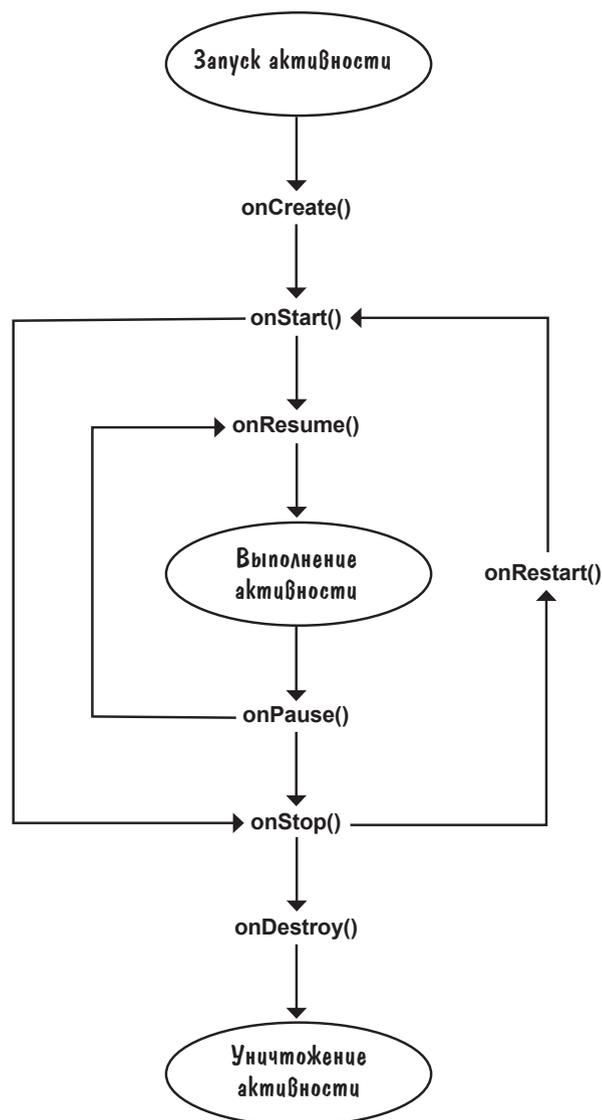
Глава 4 осталась позади, а ваш инструментарий пополнился методами жизненного цикла приложения.

Весь код для этой главы можно загрузить по адресу <https://tinyurl.com/HeadFirstAndroid>.

КЛЮЧЕВЫЕ МОМЕНТЫ



- Каждое приложение по умолчанию выполняется в отдельном процессе.
- Только главный программный поток может обновлять пользовательский интерфейс.
- Объекты `Handler` используются для планирования выполнения или передачи кода другому потоку.
- При изменении конфигурации устройства активность уничтожается и создается заново.
- Активность наследует методы жизненного цикла от класса `Android Activity`. Если вы переопределяете какие-либо из этих методов, обязательно вызывайте в своей реализации метод суперкласса.
- Метод `onSaveInstanceState(Bundle)` позволяет вашей активности сохранить свое состояние перед ее уничтожением. Затем объект `Bundle` используется для восстановления состояния в `onCreate()`.
- Для добавления значений в `Bundle` используются методы `bundle.put*("name", value)`. Чтение значений из объекта `Bundle` осуществляется методами `bundle.get*("name")`.
- Методы `onCreate()` и `onDestroy()` связаны с созданием и уничтожением активности.
- Методы `onRestart()`, `onStart()` и `onStop()` связаны с изменениями видимости активности.
- Методы `onResume()` и `onPause()` связаны с получением и потерей фокуса активностью.



Представление начинается



Давайте честно признаем: создавать хорошие макеты нужно уметь. Если вы строите приложения, которые должны использоваться людьми, необходимо позаботиться о том, чтобы эти макеты выглядели в точности так, **как вам нужно.** До настоящего момента мы едва затронули тему создания макетов; пришло время **разобраться поглубже.** Мы познакомим вас с другими **типами макетов**, которые могут использоваться в программах, после чего будет представлен **обзор основных компонентов графического интерфейса и способов их использования.** К концу главы вы увидите, что несмотря на внешние различия, у всех макетов и компонентов графического интерфейса *больше общего, чем кажется на первый взгляд.*

Пользовательский интерфейс состоит из макетов и компонентов графического интерфейса

Как вам уже известно, макет определяет внешний вид экрана, а для описания используется формат разметки XML. Макеты обычно содержат компоненты графического интерфейса — кнопки, текстовые поля и т. д. Пользователь взаимодействует с ними, чтобы приложение выполняло нужные операции.

Во всех приложениях, встречавшихся в книге, использовались относительные макеты, но существуют и другие типы макетов. С их помощью вы сможете добиться того, чтобы ваше приложение выглядело именно так, как требуется.



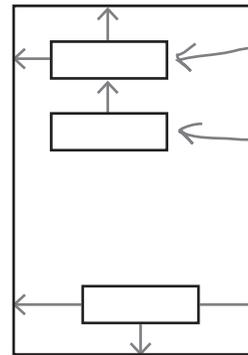
В этой главе представлены некоторые разновидности макетов, которые пригодятся вам при построении приложений. Также будут описаны компоненты графического интерфейса, закладывающие основу взаимодействий пользователя с приложением. Начнем с макетов.

Три ключевых макета: относительный, линейный и табличный

Макеты делятся на несколько разновидностей, и каждая из них следует своим правилам для принятия решений о позиционировании содержащихся в нем представлений (views). Ниже приведена краткая сводка трех важнейших разновидностей макетов. Пока не беспокойтесь о подробностях — на нескольких ближайших страницах мы подробно рассмотрим каждую из них.

Относительный макет (RelativeLayout)

В **относительном макете** входящие в него представления размещаются в относительных позициях. Позиция каждого представления определяется относительно других представлений в макете или относительно его родительского макета. Например, надпись можно разместить относительно верхнего края родительского макета, раскрывающийся список разместить под текстовым представлением, а кнопку — относительно нижнего края родительского макета.



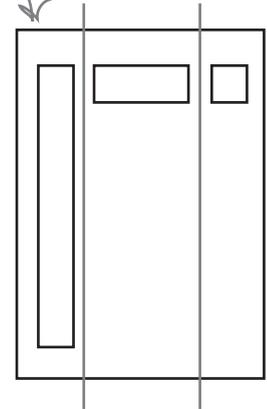
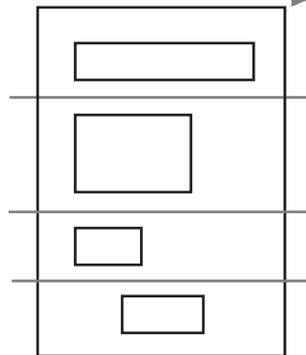
Представления могут размещаться относительно родительского макета...

...или относительно других представлений.

Представления размещаются рядом друг с другом, по вертикали или горизонтали.

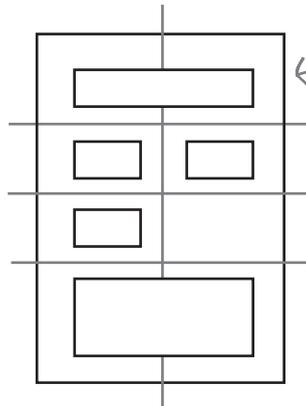
Линейный макет (LinearLayout)

В **линейном макете** представления размещаются рядом друг с другом по вертикали или горизонтали. Если используется вертикальное размещение, представления отображаются в один столбец. В варианте с горизонтальным размещением представления выводятся в одну строку.



Табличный макет (GridLayout)

В **табличном макете** экран делится на строки и столбцы, на пересечении которых находятся ячейки. Вы указываете, сколько столбцов должно входить в макет, где должны отображаться представления, и сколько строк или столбцов они должны занимать.



Экран делится на строки и столбцы; вы указываете, в какой ячейке (или ячейках) должно отображаться каждое представление.

В относительном макете представления отображаются в относительных позициях



RelativeLayout
LinearLayout
GridLayout

Как вы уже знаете, в относительном макете представления позиционируются относительно родительского макета или относительно других представлений в макете.

Относительный макет определяется элементом `<RelativeLayout>`:

Сообщаем \rightarrow `<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"`
 Android, что `android:layout_width="match_parent"`
 вы используете `android:layout_height="match_parent"` } Атрибуты `layout_width` и `layout_height` задают размер макета.
 относительный макет. `...>` \leftarrow Здесь также могут быть другие атрибуты.
`...</RelativeLayout>`

Атрибут `xmlns:android` используется для определения пространства имен Android; ему всегда присваивается значение `"http://schemas.android.com/apk/res/android"`.

ОБЯЗАТЕЛЬНО задайте ширину и высоту макета

Атрибуты `android:layout_width` и `android:layout_height` определяют ширину и высоту макета. Эти атрибуты обязательны для всех типов макетов и представлений.

Атрибутам `android:layout_width` и `android:layout_height` можно задать как обобщенные значения `"match_parent"`, `"wrap_content"`, так и конкретные размеры, например `10dp` (10 аппаратно-независимых пикселей). Значение `"wrap_content"` означает, что размеры макета должны быть минимально достаточными для того, чтобы разместить все представления, а `"match_parent"` означает, что размеры макета выбираются по размерам родителя — в данном случае это размер экрана за вычетом отступов. Чаще всего ширине и высоте макета задается значение `"match_parent"`.

Иногда в программах можно встретить атрибуты `android:layout_width` и `android:layout_height`, которым присвоено значение `"fill_parent"`. Это значение использовалось в старых версиях Android, сейчас оно заменено `"match_parent"`. В настоящее время значение `"fill_parent"` считается устаревшим.



Для любознательных

Что такое «аппаратно-независимые пиксели»?

Некоторые устройства создают очень четкие изображения за счет использования очень маленьких пикселей. Другие устройства обходятся дешевле в производстве, потому что они используют меньшее количество более крупных пикселей. Чтобы ваши интерфейсы не были слишком мелкими на одних устройствах и слишком крупными на других, используйте аппаратно-независимые пиксели (dp). Размеры, выраженные в аппаратно-независимых пикселях, приблизительно одинаковы на всех устройствах.

Отступы

Если вы хотите, чтобы макет окружало некоторое пустое пространство, воспользуйтесь атрибутами `padding`. Эти атрибуты сообщают Android, какое расстояние должно отделять стороны макета от родителя. Следующий фрагмент приказывает Android добавить ко всем сторонам макета отступы величиной 16dp:

```
<RelativeLayout ...
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp">
    ...
</RelativeLayout>
```

Добавить отступы величиной 16dp.

Атрибуты `android:padding*` не являются обязательными, но они могут использоваться с **любым макетом или представлением**.

В приведенном выше примере используются отступы с жестко запрограммированной величиной 16dp. Также можно задать величину отступов в ресурсном файле размеров. Использование ресурсов упрощает управление отступами всех макетов вашего приложения. Чтобы использовать ресурсный файл размеров, укажите в атрибутах отступов из макета имена ресурсов размеров:

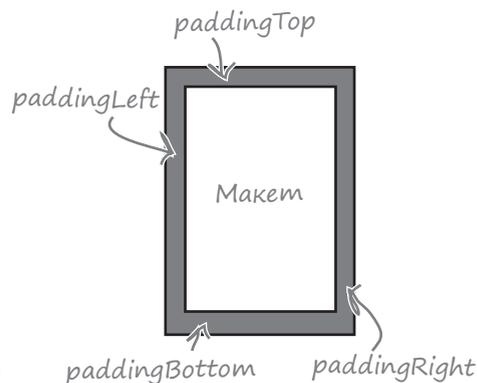
```
<RelativeLayout ...
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin">
```

Далее Android на стадии выполнения ищет значения атрибутов в ресурсном файле размеров. Этот файл находится в папке `app/src/main/res/values`; обычно ему присваивается имя `dimens.xml`:

```
<resources>
    <dimen name="activity_horizontal_margin">16dp</dimen>
    <dimen name="activity_vertical_margin">16dp</dimen>
</resources>
```

При создании нового проекта Android Studio и добавления в него активности среда разработки обычно создает этот файл за вас.

пользовательский интерфейс



Атрибутам `paddingLeft` и `paddingRight` присваивается ссылка `@dimen/activity_horizontal_margin`.

Атрибутам `paddingTop` и `paddingBottom` присваивается ссылка `@dimen/activity_vertical_margin`.

Макет берет величину отступов из этих ресурсов `dimen`.

Позиционирование представлений относительно родительского макета



При использовании относительного макета необходимо сообщить Android, где должны располагаться представления по отношению к другим представлениям в макете, или его родителю. Родителем представления является макет, содержащий это представление. Если вы хотите, чтобы представление всегда отображалось в определенной позиции экрана независимо от его размеров и ориентации, позиционируйте представление относительно его *родителя*. Например, чтобы кнопка всегда располагалась в правом верхнем углу макета, используйте следующую разметку:

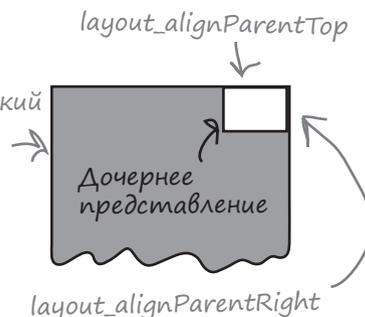
```

<RelativeLayout ... >
  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/click_me"
    android:layout_alignParentTop="true"
    android:layout_alignParentRight="true" />
</RelativeLayout>

```

Макет содержит кнопку, поэтому макет является родителем кнопки.

Родительский макет



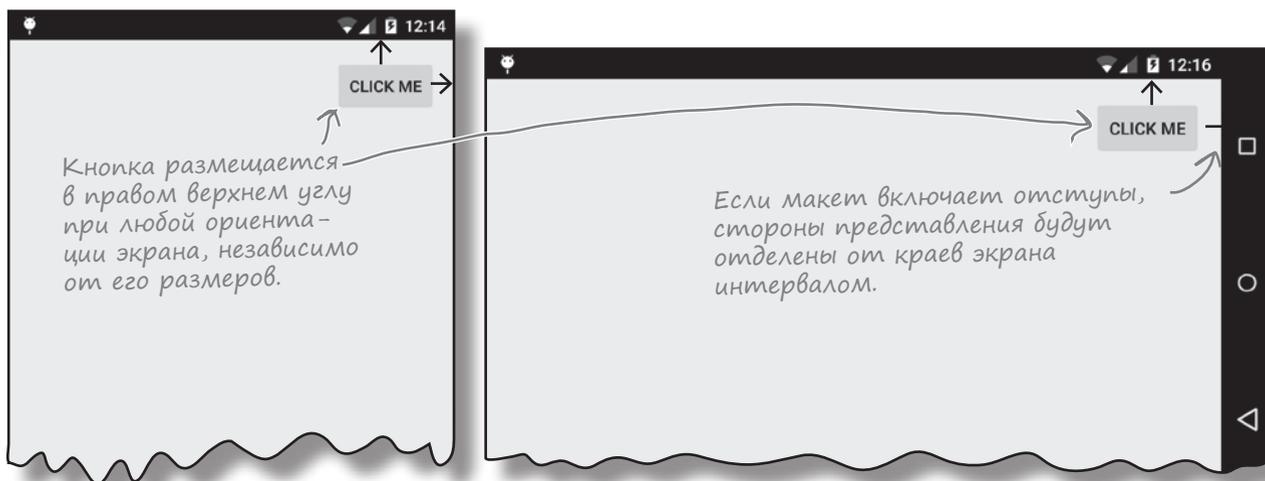
Строки

```

android:layout_alignParentTop="true"
android:layout_alignParentRight="true"

```

означают, что верхняя сторона кнопки выравнивается по верхнему краю макета, а правая сторона кнопки выравнивается по правому краю макета. Такое размещение будет применяться независимо от размера экрана или ориентации устройства:



Атрибуты для позиционирования представлений относительно родительского макета

Перед вами сводка атрибутов, наиболее часто используемых при позиционировании представлений относительно родительского макета. Включите нужный атрибут в представление, положение которого требуется определить, и присвойте ему значение "true":

```
android:attribute="true"
```

Атрибут

Что делает

**android:
layout_alignParentBottom**

Нижний край представления выравнивается по нижнему краю родителя.

**android:
layout_alignParentLeft**

Левый край представления выравнивается по левому краю родителя.

**android:
layout_alignParentRight**

Правый край представления выравнивается по правому краю родителя.

**android:
layout_alignParentTop**

Верхний край представления выравнивается по верхнему краю родителя.

**android:
layout_centerInParent**

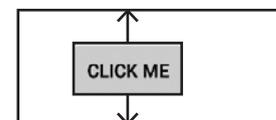
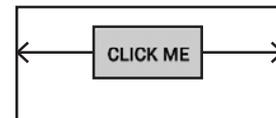
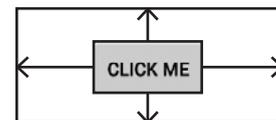
Выравнивается по центру внутри родителя (по горизонтали и вертикали).

**android:
layout_centerHorizontal**

Выравнивается по центру внутри родителя (по горизонтали).

**android:
layout_centerVertical**

Выравнивается по центру внутри родителя (по вертикали).



Позиционирование представлений относительно других представлений



Кроме позиционирования относительно родительского макета, вы также можете размещать представления *относительно других представлений*. Эта возможность применяется в тех случаях, если представления должны сохранять выравнивание независимо от размера или ориентации экрана.

Чтобы определить позицию представления относительно другого представления, следует назначить идентификатор тому представлению, которое используется в качестве якоря. Для этого используется атрибут `android:id`:

```
android:id="@+id/button_click_me"
```

Синтаксис `"@+id"` приказывает Android включить идентификатор в виде ресурса в файл ресурсов `R.java`. Если пропустить "+", Android не добавит идентификатор как ресурс, и при выполнении кода возникнут ошибки.

В следующем примере создается макет с двумя кнопками: первая кнопка расположена по центру макета, а вторая кнопка размещается под первой:

```
<RelativeLayout ... >
```

```
<Button
```

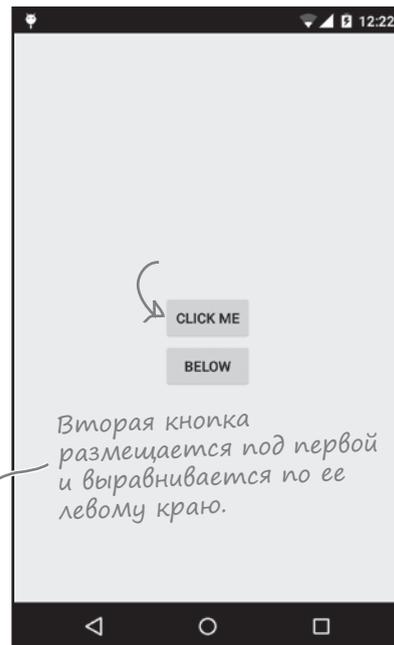
```
    android:id="@+id/button_click_me"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerInParent="true"
    android:text="@string/click_me" />
```

```
<Button
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/button_click_me"
    android:layout_below="@+id/button_click_me"
    android:text="@string/new_button_text" />
```

```
</RelativeLayout>
```

Эта кнопка используется в качестве якоря для второй, поэтому ей необходимо присвоить идентификатор.



Вторая кнопка размещается под первой и выравнивается по ее левому краю.

Строки

```
android:layout_alignLeft="@+id/button_click_me"
android:layout_below="@+id/button_click_me"
```

гарантируют, что левый край второй кнопки будет выравниваться по левому краю первой и вторая кнопка всегда будет размещаться под первой.

Атрибуты для позиционирования представлений относительно других представлений

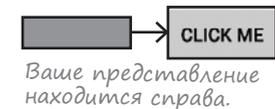
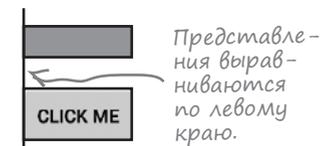
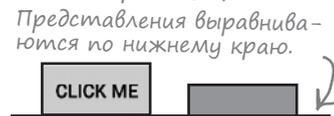
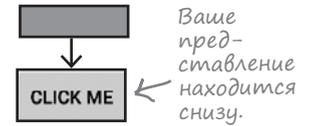
Ниже приведена сводка атрибутов, которые могут использоваться при позиционировании представлений относительно других представлений. Включите нужный атрибут в представление, положение которого требуется определить, и задайте ему в качестве значения то представление, относительно которого оно позиционируется:

```
android:attribute="@+id/view_id"
```



Атрибут	Что делает
android:layout_above	Представление размещается над якорным представлением.
android:layout_below	Представление размещается под якорным представлением.
android:layout_alignTop	Верхний край представления выравнивается по верхнему краю якорного представления.
android:layout_alignBottom	Нижний край представления выравнивается по нижнему краю якорного представления.
android:layout_alignLeft	Левый край представления выравнивается по левому краю якорного представления.
android:layout_alignRight	Правый край представления выравнивается по правому краю якорного представления.
android:layout_toLeftOf	Правый край представления располагается у левого края якорного представления.
android:layout_toRightOf	Левый край представления располагается у правого края якорного представления.

Ваше представление находится сверху.



Создание интервалов между представлениями

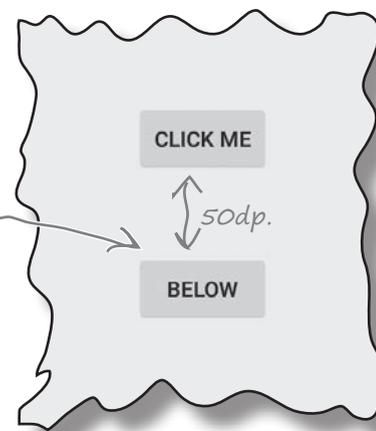
Когда вы применяете атрибуты для размещения, представления располагаются вплотную друг к другу. Чтобы представления разделялись промежутками, добавьте к представлениям **интервалы**.

Допустим, вы хотите, чтобы одно представление размещалось под другим, но при этом они разделялись дополнительным промежутком величиной 50dp. Для этого к верхнему краю нижнего представления добавляется интервал величиной 50dp:

```
<RelativeLayout ... >
  <Button
    android:id="@+id/button_click_me"
    ... />

  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/button_click_me"
    android:layout_below="@+id/button_click_me"
    android:layout_marginTop="50dp"
    android:text="@string/button_below" />
</RelativeLayout>
```

При добавлении интервала к верхнему краю нижней кнопки два представления разделяются дополнительным промежутком.



Ниже перечислены интервалы, которые могут использоваться для создания дополнительных интервалов между представлениями. Добавьте атрибут в представление и присвойте ему значение — величину интервала:

```
android:attribute="10dp"
```

Атрибут

android:layout_marginTop

Что делает

Добавляет дополнительный интервал у верхнего края представления.



android:layout_marginBottom

Добавляет дополнительный интервал у нижнего края представления.

android:layout_marginLeft

Добавляет дополнительный интервал у левого края представления.



android:layout_marginRight

Добавляет дополнительный интервал у правого края представления.





RelativeLayout
 LinearLayout
 GridLayout

Относительный макет: итоги

Прежде чем переходить к следующей разновидности макетов, мы приведем краткую сводку создания относительных макетов.

Определение относительного макета

Относительный макет определяется элементом `<RelativeLayout>`. Атрибуты ширины и высоты обязательны, а отступы задаются по желанию:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"...>
    ...
</RelativeLayout>
```

Представления могут позиционироваться относительно другого представления

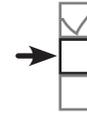
Чтобы указать, в какой позиции должно находиться представление, добавьте к нему атрибуты `layout*`. Эти атрибуты определяют позицию представления относительно родительского макета (в правом нижнем углу, в центре и т. д.). Атрибуты также могут использоваться для позиционирования представлений относительно других представлений; при определении связи используется идентификатор, назначаемый якорному представлению.

Для представлений можно определять интервалы, чтобы они отделялись от соседей свободными промежутками.

При использовании атрибутов `layout` представления размещаются вплотную друг к другу. Чтобы создать между ними свободные промежутки, определите для представления один или несколько интервалов:

```
android:layout_marginTop="5dp"
android:layout_marginBottom="5dp"
android:layout_marginLeft="5dp"
android:layout_marginRight="5dp"
```

До сих пор мы работали с относительными макетами, но существует и другая, не менее распространенная разновидность макетов: **линейные макеты**. Давайте познакомимся с ними поближе.



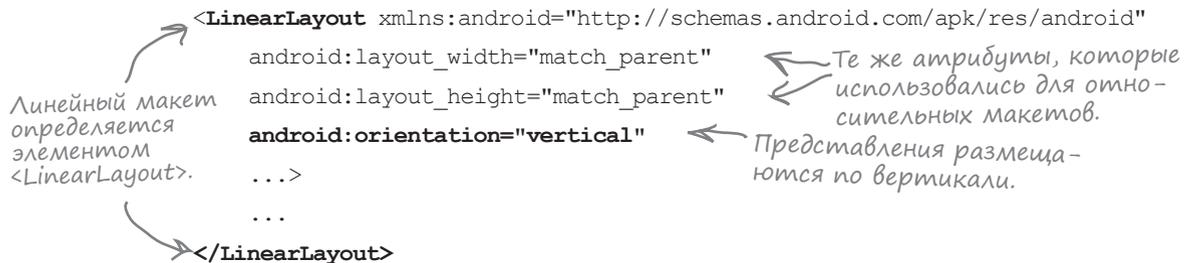
RelativeLayout
 LinearLayout
 GridLayout

В линейных макетах представления выводятся в одну строку или столбец

В линейном макете представления размещаются рядом друг с другом, по вертикали или горизонтали. Если представления размещаются по вертикали, они выводятся в один столбец. При горизонтальном размещении представления выводятся в одну строку.

Как определяется линейный макет

Линейный макет определяется при помощи элемента `<LinearLayout>`:



Атрибуты `android:layout_width`, `android:layout_height` и `android:orientation` являются обязательными. Атрибуты `android:layout_width` и `android:layout_height` определяют ширину и высоту макета, как и для относительных макетов. Атрибут `android:orientation` задает направление размещения представлений.

Чтобы разместить представления по вертикали, используйте атрибут:

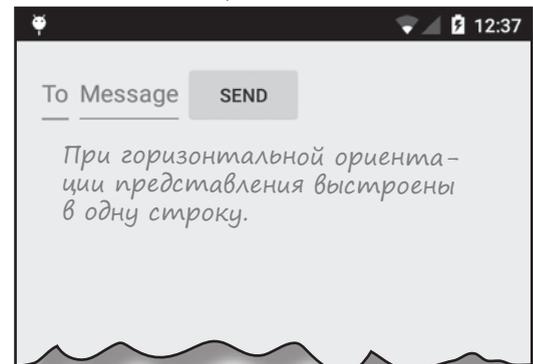
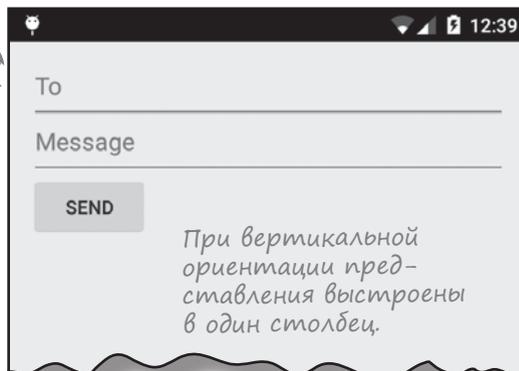
```
android:orientation="vertical"
```

Для горизонтального размещения представлений используется следующий атрибут:

```
android:orientation="horizontal"
```

Линейный макет с горизонтальной ориентацией.

Линейный макет с вертикальной ориентацией.



В линейном макете представления отображаются в порядке их следования в разметке XML

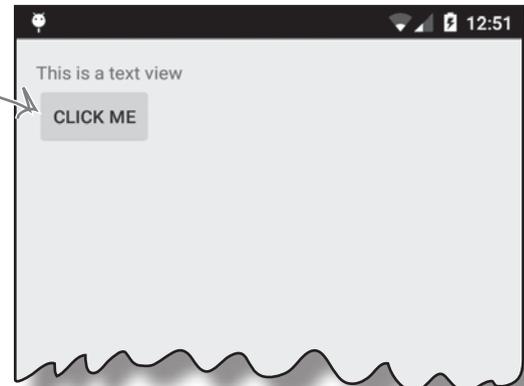


При определении линейного макета представления включаются в макет в том порядке, в котором они должны следовать на экране. Следовательно, если вы хотите, чтобы надпись размещалась над кнопкой, надпись *должна* определяться первой в разметке:

```
<LinearLayout ... >
  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/textView1" />

  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/click_me" />
</LinearLayout>
```

Если надпись определяется в XML до кнопки, то надпись будет размещаться над кнопкой на экране.



В линейном макете идентификаторы представлений понадобятся только в том случае, если вы собираетесь явно обращаться к ним из кода активности. Дело в том, что в линейном макете позиция каждого представления определяется его порядком в разметке XML. Чтобы указать, где должно размещаться представление, разработчику не нужно обращаться к другим представлениям.

Как и с относительными макетами, ширина и высота представлений задаются атрибутами `android:layout_width` и `android:layout_height`. Атрибут:

```
android:layout_width="wrap_content"
```

означает, что ширина представления должна быть минимально необходимой для того, чтобы в нем поместилось все содержимое. Как, например, при выводе текста на кнопке или в надписи. Атрибут:

```
android:layout_width="match_parent"
```

означает, что ширина представления определяется шириной родительского макета.

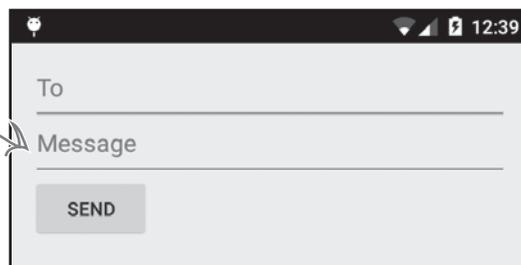
Атрибуты `android:layout_width` и `android:layout_height` обязательны для всех представлений независимо от используемого макета.

Атрибуту присваиваются значения `wrap_content`, `match_parent` или конкретный размер (например, `16dp`).

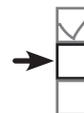
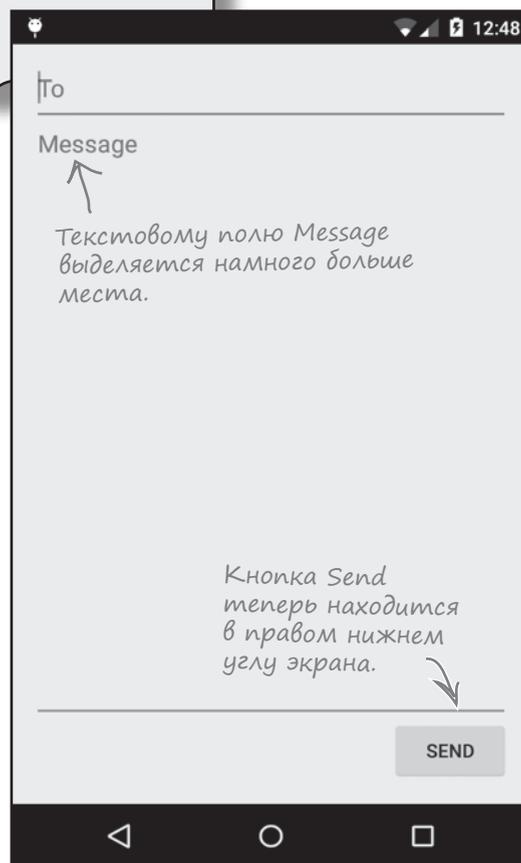
Изменение базового линейного макета

На первый взгляд линейный макет кажется примитивным и негибким — в конце концов, он всего лишь выстраивает представления в заданном порядке. Впрочем, вы все же можете немного повлиять на внешний вид макета при помощи атрибутов. Чтобы вы лучше поняли, как это делается, мы рассмотрим пример настройки простейшего линейного макета. Макет состоит из двух текстовых полей и кнопки. В исходном варианте текстовые поля размещаются на экране друг над другом:

Каждое представление занимает минимально возможное вертикальное пространство.



Мы изменим макет так, чтобы кнопка находилась в правом нижнем углу макета, а одно из текстовых полей занимало все оставшееся пространство.



RelativeLayout
LinearLayout
GridLayout

Начало настройки линейного макета

Линейный макет содержит два текстовых поля и кнопку. На кнопке выводится текст “Send”, а в двух текстовых полях выводятся подсказки “To” и “Message”.

Подсказка представляет собой временный текст, который выводится в пустом текстовом поле. Этот текст дает пользователю представление о том, какие данные следует вводить в этом поле. Текст определяется при помощи атрибута `android:hint`:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:orientation="vertical"
    tools:context=".MainActivity" >
```

```
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="@string/to" />
```

Ширина текстовых полей совпадает с шириной родительского макета.

Атрибут `android:hint` выводит в текстовом поле подсказку, поясняющую, какие данные следует в нем вводить.

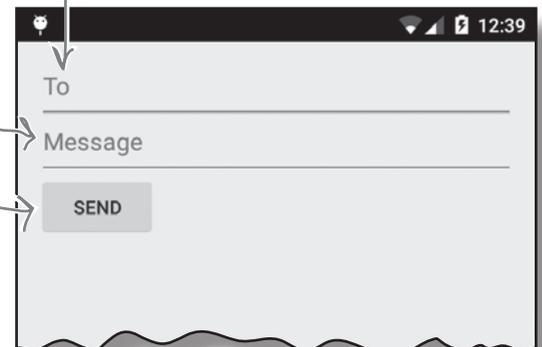
Строковые значения, как обычно, определяются в файле `Strings.xml`.

```
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="@string/message" />
```

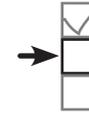
```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/send" />
```

```
</LinearLayout>
```

Каждое из этих представлений занимает столько места по вертикали, сколько необходимо для их содержимого. Как же увеличить высоту текстового поля Message?



Добавление весов



RelativeLayout
LinearLayout
GridLayout

Все представления в базовом макете занимают столько вертикального пространства, сколько необходимо для их содержимого. Но мы хотим, чтобы текстовое поле Message растягивалось по вертикали и занимало в макете все вертикальное пространство, не используемое другими представлениями.

Текстовое поле Message должно растягиваться по вертикали, занимая все свободное пространство в макете.



Для этого нужно назначить текстовому полю Message **весовой коэффициент**, или **вес**. Назначение весов — способ приказать представлению занять дополнительное пространство в макете. Для назначения веса представлению используется атрибут

```
android:layout_weight="число"
```

где число — некоторое положительное значение.

Если представлению назначен вес, макет прежде всего выделяет каждому представлению место, достаточное для вывода его содержимого: каждой кнопке хватает места для вывода ее текста, каждому текстовому полю — для вывода подсказок, и т. д. После этого все оставшееся пространство пропорционально распределяется между представлениями с весом 1 и более.

Назначение веса одному представлению

Текстовое поле Message должно занимать все свободное место в макете. Для этого мы присвоим его атрибуту `layout_weight` значение 1. Так как это единственное представление в макете, которому назначен вес, текстовое поле растягивается по вертикали, занимая всю оставшуюся часть экрана. Разметка выглядит так:



```
<LinearLayout ... >
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/to" />
```

У элементов `<EditText>` и `<Button>` атрибут `layout_weight` не задан. Они занимают столько места, сколько необходимо для их содержания, но не более.

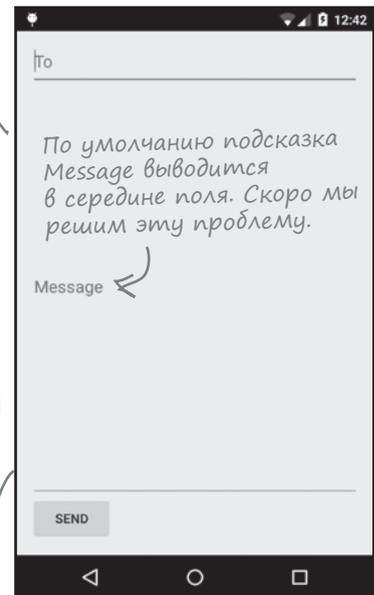
Единственное представление в макете, которому назначен вес. Представление заполняет все пространство, не занятое другими представлениями.

```
    <EditText
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:hint="@string/message" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/send" />
</LinearLayout>
```

Высота представления в линейном макете определяется по значению `layout_weight`. Присваивание `layout_height` значения `0dp` более эффективно, чем присваивание значения `"wrap_content"`.

Присваивание текстовому полю веса 1 означает, что оно займет все свободное пространство, не занятое другими представлениями в макете. Причина заключается в том, что двум другим представлениям веса в разметке XML макета не назначены.

Представлению Message присвоен вес 1. Так как это единственное представление с заданным атрибутом веса, представление растягивается и занимает всё вертикальное пространство в макете.



Назначение весов нескольким представлениям

В рассмотренном примере атрибут веса был назначен только одному представлению. Но что, если таких представлений будет *несколько*? Предположим, текстовому полю «То» назначен вес 1, а текстовому полю Message назначен вес 2:

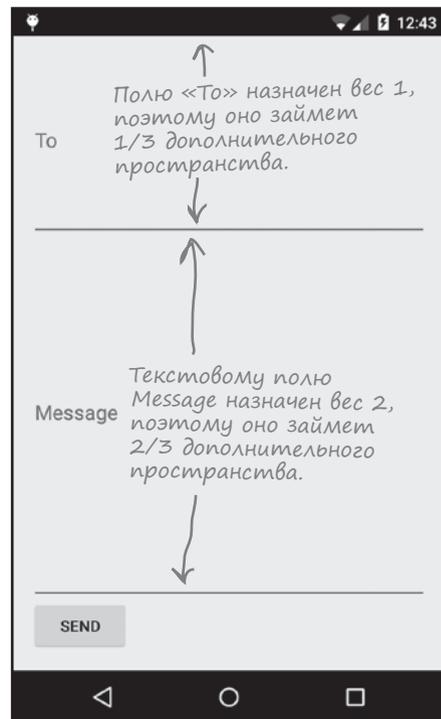
```
<LinearLayout ... >
...
<EditText
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="1"
    android:hint="@string/to" />

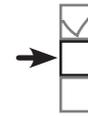
<EditText
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="2"
    android:hint="@string/message" />
...
</LinearLayout>
```

Чтобы вычислить, сколько свободного пространства займет каждое представление, начнем со сложения атрибутов `layout_weight` всех представлений. В нашем примере получится $1+2=3$. Доля свободного пространства, занятого каждым представлением, будет равна весу представления, разделенному на сумму весов. Текстовому полю «То» назначен вес 1; следовательно, оно будет занимать $1/3$ свободного пространства в макете. Текстовому полю Message назначен вес 2, поэтому оно займет $2/3$ свободного пространства.



RelativeLayout
LinearLayout
GridLayout





RelativeLayout
LinearLayout
GridLayout

Атрибут `gravity` и управление выравниванием текста в представлении

Наша следующая задача — переместить текст подсказки в текстовом поле `Message`. На данный момент он выводится в середине поля по вертикали. Нужно изменить разметку так, чтобы текст отображался у верхнего края. Эта задача решается с помощью атрибута `android:gravity`.

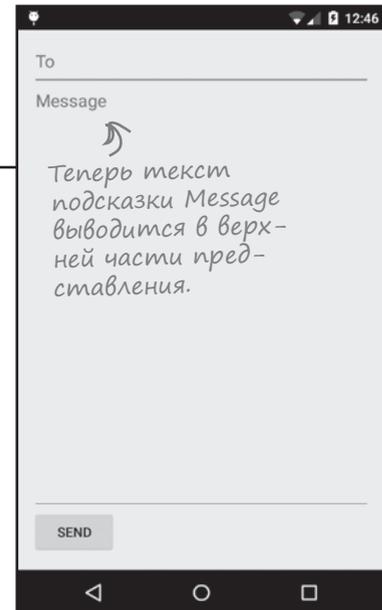
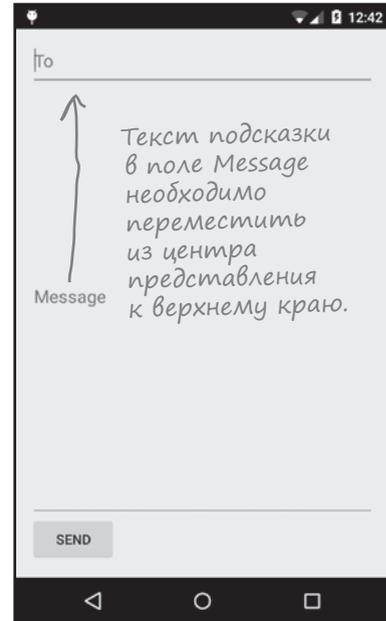
Атрибут `android:gravity` позволяет указать, как содержимое должно размещаться внутри представления. Например, как текст должен позиционироваться в текстовом поле. Если вам нужно, чтобы текст выводился у верхнего края, следующий фрагмент кода обеспечит нужный эффект:

```
android:gravity="top"
```

Чтобы текст подсказки сместился в верхнюю часть текстового поля, следует включить в разметку поля атрибут `android:gravity`:

```
<LinearLayout ... >
...
<EditText
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="1"
    android:gravity="top"
    android:hint="@string/message" />
...
</LinearLayout>
```

← Внутренний текст должен выводиться у верхнего края текстового поля.



Тест-драйв

С добавлением атрибута `android:gravity` к текстовому полю `Message` текст подсказки, как и требовалось, смещается к верхнему краю представления.

Список других возможных значений атрибута `android:gravity` приведен на следующей странице.

Атрибут `android:gravity`: список значений



RelativeLayout
LinearLayout
GridLayout

Ниже перечислены другие значения, которые могут использоваться с атрибутом `android:gravity`. Добавьте атрибут в представление и присвойте ему значение из следующего списка:

```
android:gravity="значение"
```

Значение	Что делает
top	Содержимое размещается у верхнего края представления.
bottom	Содержимое размещается у нижнего края представления.
left	Содержимое размещается у левого края представления.
right	Содержимое размещается у правого края представления.
center_vertical	Содержимое выравнивается по центру представления (по вертикали).
center_horizontal	Содержимое выравнивается по центру представления (по горизонтали).
center	Содержимое выравнивается по центру представления (по вертикали и горизонтали).
fill_vertical	Содержимое заполняет представление по вертикали.
fill_horizontal	Содержимое заполняет представление по горизонтали.
fill	Содержимое заполняет представление.

Атрибут `android:gravity` управляет размещением содержимого внутри представления.

Перемещение кнопки вправо с использованием атрибута `layout_gravity`

Осталось внести в макет последнее изменение. В текущей версии кнопка Send отображается в левом нижнем углу. Нужно сместить ее вправо, чтобы она заняла место в правом нижнем углу. Для этого мы воспользуемся атрибутом `android:layout_gravity`.

Атрибут `android:layout_gravity` позволяет указать, в какой части внешнего пространства должно находиться представление в линейном макете. Например, атрибут может использоваться для смещения представления вправо или для горизонтального выравнивания по центру. Для смещения кнопки вправо в ее разметку включается следующий атрибут:

```
android:layout_gravity="right"
```

Но зачем применять `layout_gravity` для смещения кнопки? Ранее мы уже рассматривали атрибут `layout_alignRight` attribute — не проще ли воспользоваться им?

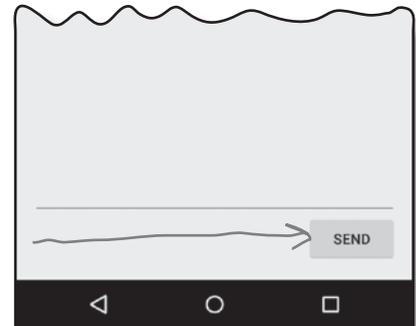
Атрибут `android:layout_alignRight` действует только в относительных макетах.

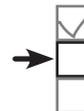
У макетов есть некоторые общие атрибуты — например, `android:layout_width` и `android:layout_height`. Однако многие атрибуты применимы только для одной конкретной разновидности макетов.

Многие атрибуты, рассмотренные ранее для относительных макетов, неприменимы к линейным макетам. Вместо них в линейных макетах применяется концепция притяжения (`gravity`), поэтому для смещения кнопки вправо приходится использовать атрибут

```
android:layout_gravity="right"
```

Некоторые допустимые значения атрибута `android:layout_gravity` приведены на следующей странице.





RelativeLayout
LinearLayout
GridLayout

Другие допустимые значения атрибута `android:layout_gravity`

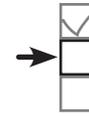
Ниже приведена сводка некоторых возможных значений атрибута `android:layout_gravity`. Добавьте атрибут в свое представление и задайте ему одно из значений, перечисленных ниже:

```
android:layout_gravity="значение"
```

Значение	Что делает
top, bottom, left, right	Размещает представление у верхнего, нижнего, левого или правого края контейнера.
start, end	Размещает представление в начале или в конце контейнера.
center_vertical, center_horizontal	Выравнивает представление по вертикали или по горизонтали внутри контейнера.
center	Выравнивает представление по вертикали и по горизонтали внутри контейнера.
fill_vertical, fill_horizontal	Масштабирует представление так, чтобы оно заполняло контейнер в вертикальном или горизонтальном направлении.
fill	Масштабирует представление так, чтобы оно заполняло контейнер по вертикали и по горизонтали.

Атрибут `android:layout_gravity` позволяет указать, в какой части доступного пространства должно выводиться представление.

Атрибут `android:layout_gravity` определяет размещение самого представления, тогда как атрибут `android:gravity` определяет размещение содержимого представления.



RelativeLayout
LinearLayout
GridLayout

Полная разметка линейного макета

Ниже приведена полная разметка линейного макета:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:orientation="vertical"
    tools:context=".MainActivity" >

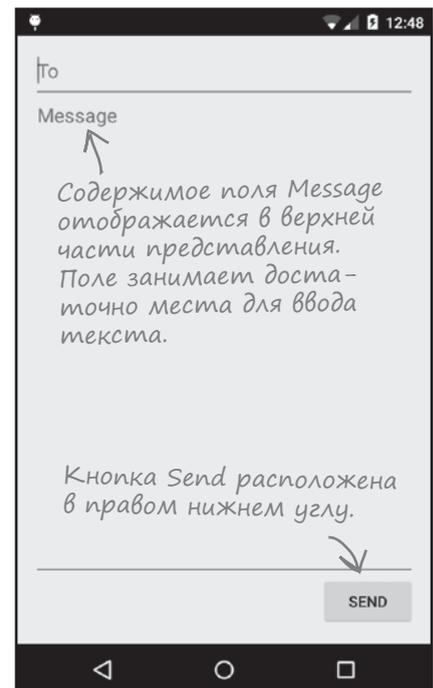
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/to" />

    <EditText
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:gravity="top"
        android:hint="@string/message" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="right"
        android:text="@string/send" />

</LinearLayout>
```

Не путайте `android:gravity` с `android:layout_gravity`. Атрибут `android:gravity` относится к содержанию представления, а `android:layout_gravity` относится к самому представлению.



Линейный макет: итоги



RelativeLayout
LinearLayout
GridLayout

Ниже приведена краткая сводка создания линейных макетов.

Определение линейного макета

Линейный макет определяется элементом `<LinearLayout>`. Атрибуты ширины, высоты и ориентации макета обязательны, а отступы задаются по желанию:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    ...>
    ...
</LinearLayout>
```

Представления отображаются в порядке их определения

При определении линейного макета представления добавляются в том порядке, в котором они должны отображаться на экране.

Для растяжения представлений используются веса

По умолчанию каждое представление занимает столько места, сколько необходимо для его содержимого. Если вы хотите, чтобы одно или несколько представлений занимали больше места, назначьте им атрибут веса:

```
android:layout_weight="1"
```

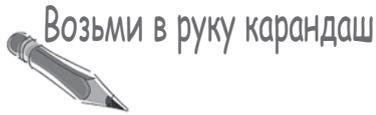
Используйте атрибут `gravity` для управления размещением содержимого внутри представления

Атрибут `android:gravity` указывает, где должно размещаться содержимое внутри представления — например, в какой части текстового поля должен размещаться текст.

Используйте атрибут `layout_gravity` для управления размещением представления внутри контейнера

Атрибут `android:layout_gravity` управляет размещением представления линейного макета в его внешнем пространстве. Например, с его помощью можно сместить представление вправо или выровнять его по центру, по горизонтали.

Вот и все, что необходимо сказать о линейных макетах. Осталось рассмотреть последнюю группу макетов: **табличные макеты**.



Ниже приведена разметка XML для приложения Beer Adviser, которое мы создали в главе 2. Преобразуйте ее в линейный макет, изображенный в нижней части страницы.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    tools:context=".FindBeerActivity" >

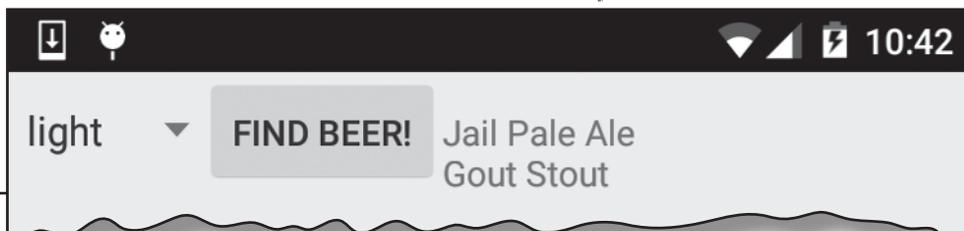
    <Spinner
        android:id="@+id/color"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="37dp"
        android:entries="@array/beer_colors" />

    <Button
        android:id="@+id/find_beer"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/color"
        android:layout_below="@+id/color"
        android:text="@string/find_beer"
        android:onClick="onClickFindBeer" />

    <TextView
        android:id="@+id/brands"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/find_beer"
        android:layout_below="@+id/find_beer"
        android:layout_marginTop="18dp"
        android:text="@string/brands" />

</RelativeLayout>
```

На премию за “самый стильный дизайн” рассчитывать не стоит, и все же попробуйте внести изменения в разметку XML и сформировать такой макет.





Возьми в руку карандаш

Решение

Ниже приведена разметка XML для приложения Beer Adviser, которое мы создали в главе 2. Преобразуйте ее в линейный макет, изображенный в нижней части страницы.

Перейти на линейный макет.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:orientation="horizontal"
    tools:context=".FindBeerActivity" >
```

Линейные макеты используют атрибут `android:orientation`. В режиме "horizontal" представления отображаются рядом друг с другом по горизонтали.

```
<Spinner
    android:id="@+id/color"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="37dp"
    android:entries="@array/beer_colors" />
```

Эти строки не нужны.

```
<Button
    android:id="@+id/find_beer"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/color"
    android:layout_below="@+id/color"
    android:text="@string/find_beer"
    android:onClick="onClickFindBeer" />
```

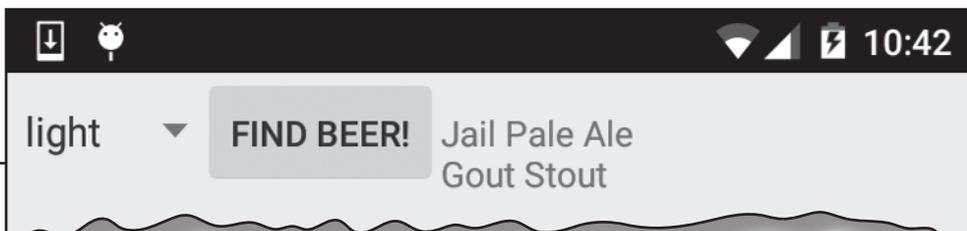
Эти строки не нужны.

```
<TextView
    android:id="@+id/brands"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/find_beer"
    android:layout_below="@+id/find_beer"
    android:layout_marginTop="18dp"
    android:text="@string/brands" />
```

Эти строки не нужны.

Заменить на линейный макет.

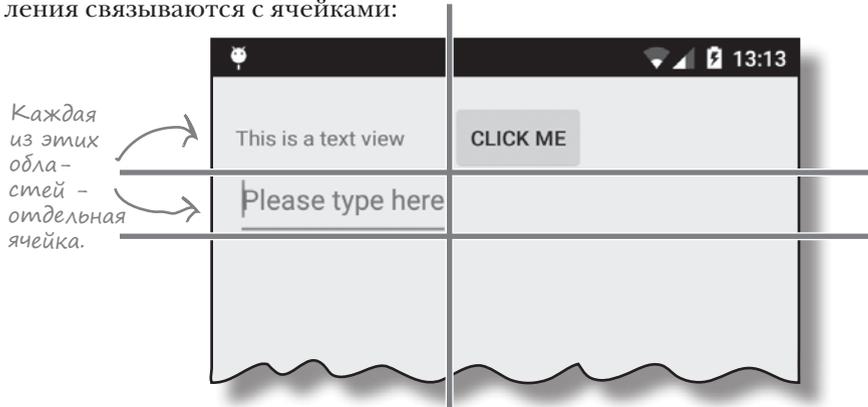
```
</RelativeLayout>
```



В табличных макетах (GridLayout) представления размещаются по строкам и столбцам



В табличном макете экран разбивается на строки и столбцы, а представления связываются с ячейками:



Будьте осторожны!

GridLayout требует API уровня 14 и выше.

Если вы собираетесь использовать табличный макет, убедитесь в том, что приложение использует SDK с уровнем API 14 и выше.

Определение табличного макета

Определение табличного макета очень похоже на определение других типов макетов, только в этом случае используется элемент `<GridLayout>`:

```

<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:columnCount="2"
    ... >
</GridLayout>
    
```

Здесь используется элемент `<GridLayout>`.

Те же атрибуты, которые использовались для других макетов.

Количество столбцов в макете (в данном случае 2).

Количество столбцов в табличном макете задается следующим атрибутом:

```
android:columnCount="число"
```

где число — количество столбцов. Также можно задать максимальное число строк с использованием атрибута:

```
android:rowCount="число"
```

но на практике обычно лучше доверить вычисление количества строк Android в зависимости от количества представлений в макете. Android создает столько строк, сколько потребуется для отображения всех представлений.

Добавление представлений в табличный макет



RelativeLayout
LinearLayout
GridLayout

В табличный макет представления добавляются примерно так же, как и в линейный:

```
<GridLayout ... >

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/textview" />

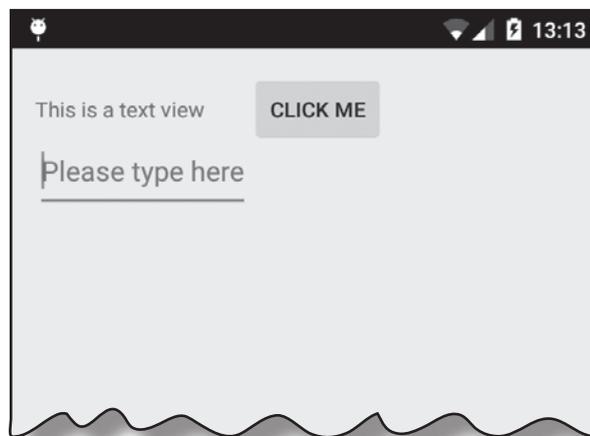
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/click_me" />

<EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:hint="@string/edit" />

</GridLayout>
```

Как и при использовании линейного макета, нет необходимости назначать представлениям идентификаторы, если только вы не собираетесь явно ссылаться на них в коде активности. Представлениям не нужно обращаться друг к другу в макете, поэтому для этой цели идентификаторы не понадобятся.

По умолчанию табличный макет размещает представления в порядке их следования в XML. Если создать табличный макет с двумя столбцами, табличный макет поместит первое представление в первой позиции, второе представление во второй позиции и т. д. У такого решения есть один недостаток: исключение одного из представлений из макета может привести к серьезному изменению внешнего вида макета. Чтобы избежать подобных проблем, вы указываете, где должно находиться каждое представление и сколько столбцов оно должно занимать.





RelativeLayout

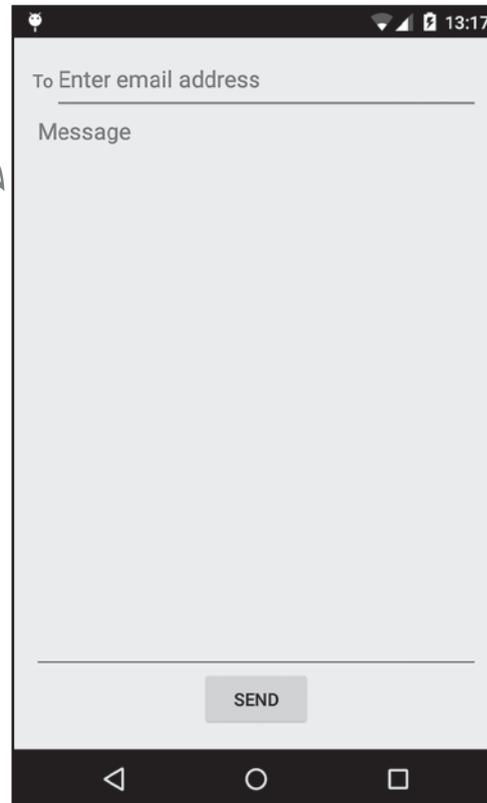
LinearLayout

GridLayout

Создание табличного макета

Чтобы показать, как работают табличные макеты, мы создадим макет и укажем, где должны находиться представления, и сколько столбцов они должны занимать. Макет состоит из надписи, содержащей текст “To”, текстового поля с подсказкой “Enter email address”, текстового поля с подсказкой “Message” и кнопки с текстом “Send”:

*Похоже на пример, использо-
ванный с линейным макетом,
только здесь наверху появилась
надпись To, а кнопка Send вы-
равнивается по центру в нижней
части макета.*



Что мы собираемся сделать

- 1 **Нарисовать эскиз пользовательского интерфейса и разбить его на строки и столбцы.**
При наличии эскиза нам будет проще представить, как должен строиться макет.
- 2 **Построить макет строку за строкой.**

Начнем с построения эскиза



Создание нового макета начинается с построения эскиза. Это поможет нам понять, сколько строк и столбцов потребуется, где должно находиться каждое представление и сколько столбцов оно должно занимать.



Табличный макет должен состоять из двух столбцов

Итак, нужное расположение представлений достигается с табличным макетом, состоящим из двух столбцов:

```
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:columnCount="2"
    tools:context=".MainActivity" >
</GridLayout>
```

После определения основы табличного макета можно переходить к добавлению представлений.

Строка 0: добавление представлений в конкретные строки и столбцы

Первая строка табличного макета состоит из надписи (в первом столбце) и текстового поля (во втором столбце). Все начинается с добавления представлений в макет:

```
<GridLayout...>
  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/to" />

  <EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="fill_horizontal"
    android:hint="@string/to_hint" />
</GridLayout>
```

Атрибуты `android:layout_row` и `android:layout_column` используются для обозначения строк и столбцов, в которых должны располагаться представления. Индексы строк и столбцов начинаются с 0; следовательно, чтобы представление располагалось в первом столбце и первой строке, нужно задать следующие значения:

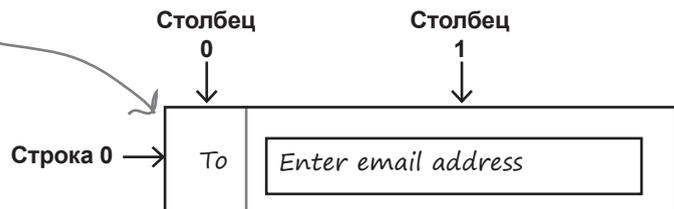
```
android:layout_row="0"
android:layout_column="0"
```

Индексы столбцов и строк начинаются с 0, поэтому эти атрибуты обозначают первую строку и первый столбец.

Применим эти обозначения к разметке макета: разместим надпись в столбце 0, а текстовое поле — в столбце 1:

```
<GridLayout...>
  <TextView
    ...
    android:layout_row="0"
    android:layout_column="0"
    android:text="@string/to" />

  <EditText
    ...
    android:layout_row="0"
    android:layout_column="1"
    android:hint="@string/to_hint" />
</GridLayout>
```



Атрибуты `android:gravity` и `android:layout_gravity` могут использоваться с табличными макетами.

Атрибут `layout_gravity` также может использоваться с табличными макетами. Мы используем режим `fill_horizontal`, потому что текстовое поле должно заполнять все оставшееся горизонтальное пространство.

Индексы строк и столбцов начинаются с 0. Атрибут `layout_column="n"` обозначает столбец `n+1` в макете.

Строка 1: представление занимает несколько столбцов



Вторая строка табличного макета состоит из текстового поля, которое начинается в первом столбце и распространяется на второй столбец. Представление занимает все свободное пространство.

Чтобы представление занимало несколько столбцов, необходимо сначала указать, с какого столбца и строки должно начинаться представление. Наше текстовое поле должно начинаться в первом столбце второй строки, поэтому атрибуты выглядят так:

```
android:layout_row="1"
android:layout_column="0"
```

Представление в нашем эскизе занимает два столбца. Чтобы добиться этого, можно воспользоваться атрибутом `android:layout_columnSpan` следующего вида:

```
android:layout_columnSpan="число"
```

где число — количество столбцов, которые должно занимать представление. В нашем примере атрибут должен выглядеть так:

```
android:layout_columnSpan="2"
```

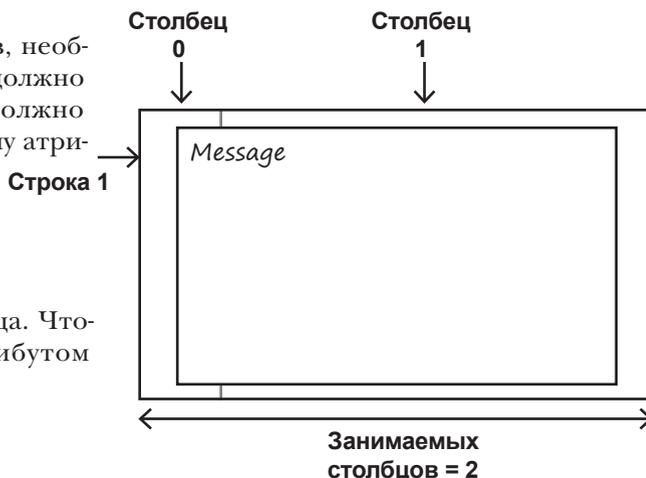
Объединяя все сказанное, мы приходим к следующей разметке поля `Message`:

```
<GridLayout...>
  <TextView... />
  <EditText.../>
  <EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="fill"
    android:gravity="top"
    android:layout_row="1"
    android:layout_column="0"
    android:layout_columnSpan="2"
    android:hint="@string/message" />
</GridLayout>
```

Представления, добавленные на предыдущей странице для строки 0.

Текстовое поле занимает все свободное пространство, а текст отображается в верхней части.

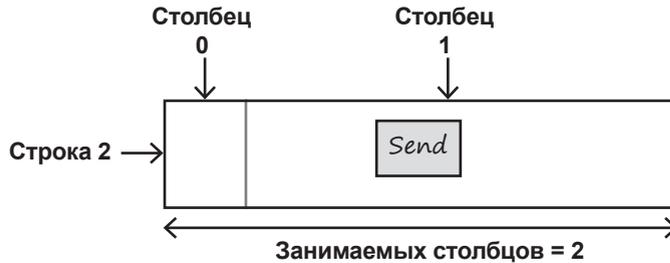
Текстовое поле начинается в столбце 0 и занимает два столбца.



После добавления представлений для первых двух строк остается только добавить кнопку.

Строка 2: представление занимает несколько столбцов

Кнопка должна быть выровнена по горизонтали в центре области, состоящей из двух столбцов:



Развлечения с магнитами

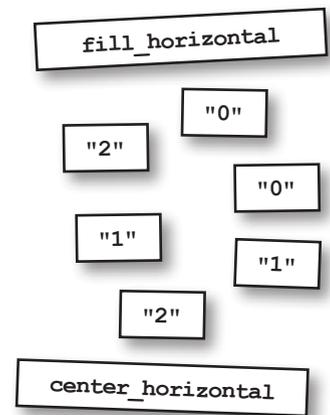
Мы написали разметку для выравнивания кнопки Send по центру третьей строки табличного макета, но от порыва ветра часть магнитов упала на пол. Удается ли вам заполнить образовавшиеся пропуски магнитами?

```
<GridLayout...>
  <TextView... />
  <EditText... />
  <EditText... />
```

Представления, добавленные ранее.

```
<Button
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"

  android:layout_row= .....
  android:layout_column= .....
  android:layout_gravity= .....
  android:layout_columnSpan= .....
  android:text="@string/send" />
```



```
</GridLayout>
```

→ Ошибки на с. 260

Полный код табличного макета



RelativeLayout
LinearLayout
GridLayout

```

<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:columnCount="2"
    tools:context=".MainActivity" >

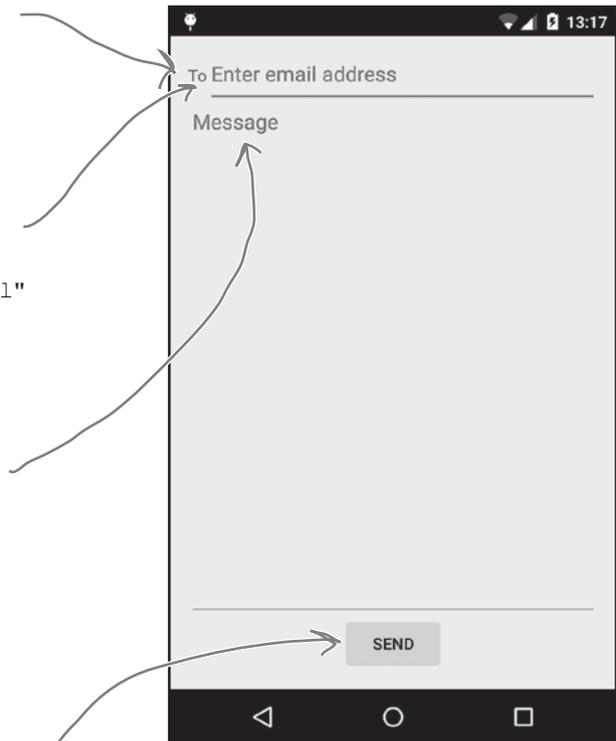
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_row="0"
        android:layout_column="0"
        android:text="@string/to" />

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="fill_horizontal"
        android:layout_row="0"
        android:layout_column="1"
        android:hint="@string/to_hint" />

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="fill"
        android:gravity="top"
        android:layout_row="1"
        android:layout_column="0"
        android:layout_columnSpan="2"
        android:hint="@string/message" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_row="2"
        android:layout_column="0"
        android:layout_gravity="center_horizontal"
        android:layout_columnSpan="2"
        android:text="@string/send" />
</GridLayout>

```



Кнопка занимает область из двух столбцов, начиная со столбца 1 строки 2. Кнопка выравнивается по центру области.

Табличный макет: итоги

Ниже приведена краткая сводка создания линейных макетов.



RelativeLayout
LinearLayout
GridLayout

Определение табличного макета

Табличный макет определяется элементом `<GridLayout>`. Количество столбцов в макете задается атрибутом `android:columnCount`, а количество строк – атрибутом `android:rowCount`:

```
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:columnCount="2"
    ... >
    ...
</GridLayout>
```

Определение начальной ячейки каждого представления

Атрибуты `android:layout_row` и `android:layout_column` определяют строку и столбец, на пересечении которых должно выводиться представление. Индексы строк и столбцов начинаются с 0; чтобы представление отображалось в первом столбце первой строки, атрибуты должны выглядеть так:

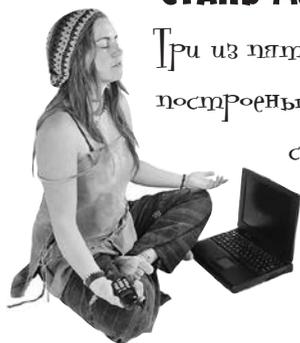
```
android:layout_row="0"
android:layout_column="0"
```

Определение количества столбцов, занимаемых каждым представлением

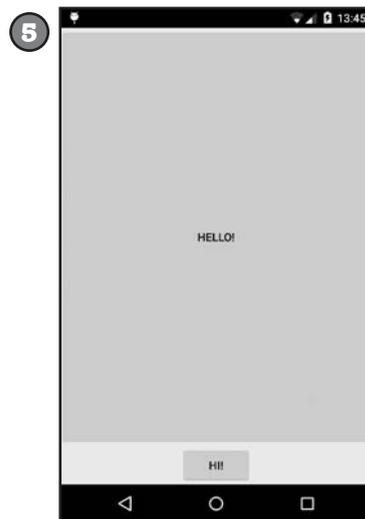
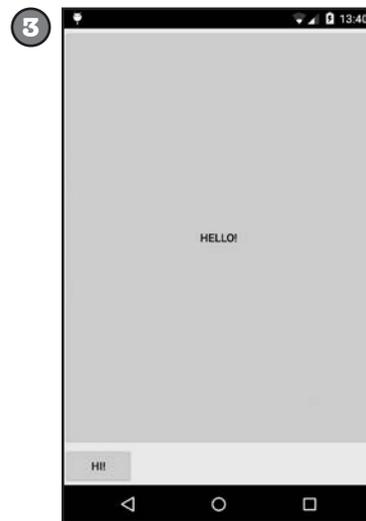
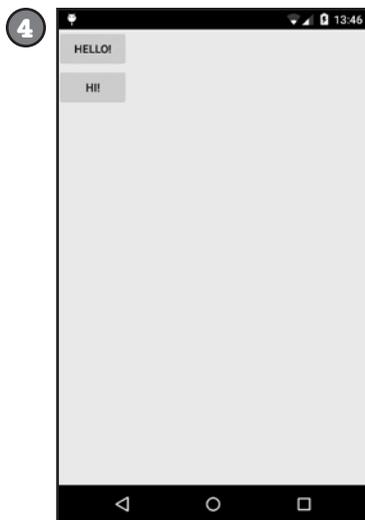
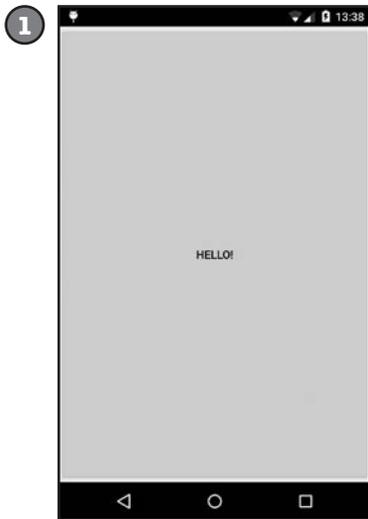
Атрибут `android:layout_columnSpan` определяет количество столбцов, занимаемых представлением. Например, если вы хотите, чтобы представление занимало два столбца, атрибут должен выглядеть так:

```
android:layout_columnSpan="2"
```

СТАНЬ макетом



Три из пяти экранов, изображенных ниже, были построены на основе макетов на развороте страницы. Ваша задача – соединить каждый из трех макетов с экраном, который ему соответствует.



```

A <GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:tools="http://schemas.android.com/tools"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:columnCount="3"
      tools:context=".MainActivity" >
    <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_gravity="fill"
      android:layout_columnSpan="3"
      android:text="@string/hello" />
  </GridLayout>

```

```

B <GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:tools="http://schemas.android.com/tools"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:columnCount="2"
      tools:context=".MainActivity" >
    <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_gravity="fill"
      android:layout_columnSpan="2"
      android:text="@string/hello" />
    <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/hi" />
  </GridLayout>

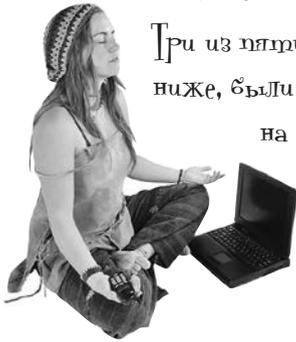
```

```

C <GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:tools="http://schemas.android.com/tools"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:columnCount="2"
      tools:context=".MainActivity" >
    <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_row="0"
      android:layout_column="0"
      android:layout_columnSpan="2"
      android:text="@string/hello" />
    <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_row="1"
      android:layout_column="0"
      android:text="@string/hi" />
  </GridLayout>

```

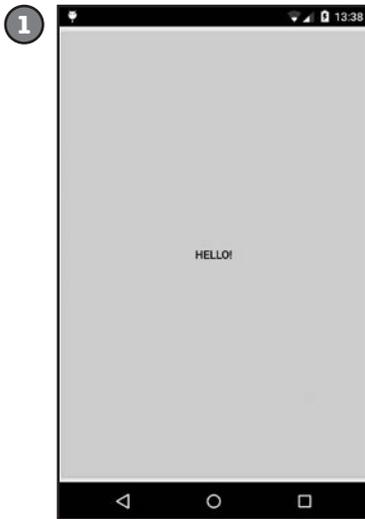
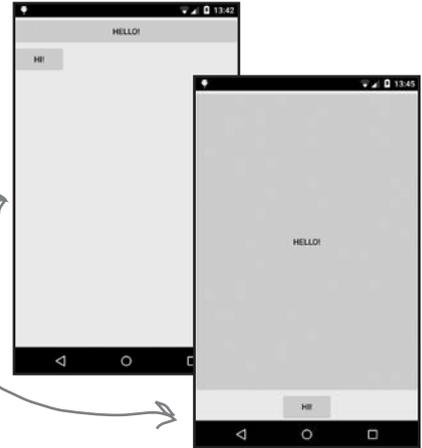
СТАНЬ макетом. Решение



Три из пяти экранов, изображенных ниже, были построены на основе макетов на развороте страницы.

Ваша задача — соединить каждый из трех макетов с экраном, который ему соответствует.

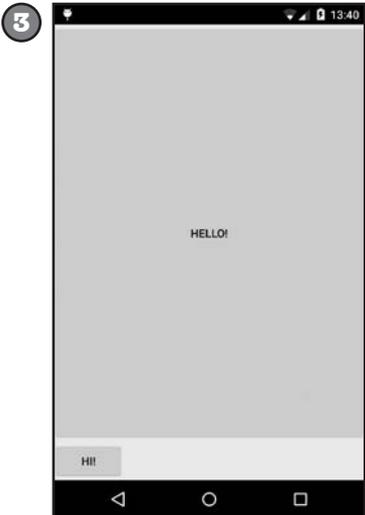
Эти экраны не соответствуют ни одному макету.



A

```
<GridLayout xmlns:android=
    "http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:columnCount="3"
    tools:context=".MainActivity" >
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="fill"
        android:layout_columnSpan="3"
        android:text="@string/hello" />
</GridLayout>
```

Одна кнопка, заполняющая весь экран.



B

```
<GridLayout xmlns:android=
    "http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:columnCount="2"
    tools:context=".MainActivity" >
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="fill"
        android:layout_columnSpan="2"
        android:text="@string/hello" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hi" />
</GridLayout>
```

Эта кнопка заполняет экран, оставляя место для другой кнопки, расположенной под ней.



```

  4  <GridLayout xmlns:android=
      "http://schemas.android.com/apk/res/android"
      xmlns:tools="http://schemas.android.com/tools"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:columnCount="2"
      tools:context=".MainActivity" >
      <Button
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:layout_row="0"
          android:layout_column="0"
          android:layout_columnSpan="2"
          android:text="@string/hello" />
      <Button
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:layout_row="1"
          android:layout_column="0"
          android:text="@string/hi" />
    </GridLayout>
  
```

И хотя эта кнопка занимает два столбца, мы не приказывали ей заполнять экран по горизонтали.

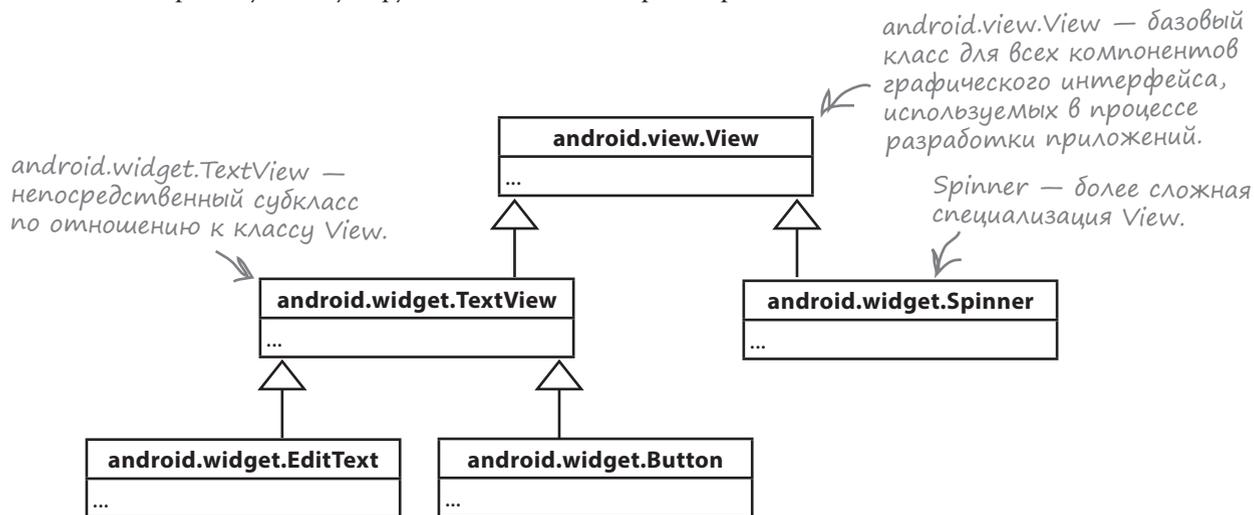
У макетов и компонентов графического интерфейса много общего

Вероятно, вы заметили, что у всех видов макетов имеются общие атрибуты. Какой бы макет вы ни выбрали, ширина и высота обязательно должны задаваться атрибутами `android:layout_width` и `android:layout_height`. Впрочем, их применение не ограничивается макетами — атрибуты `android:layout_width` и `android:layout_height` обязательны также и для компонентов графического интерфейса.

Это объясняется тем, что **все макеты и компоненты графического интерфейса являются subclasses класса `Android View`**. Давайте познакомимся с этим классом поближе.

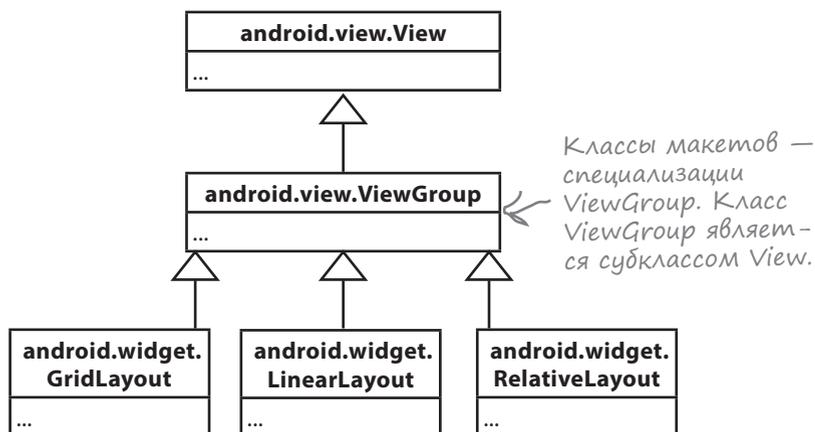
Компоненты графического интерфейса являются специализациями View

Вы уже знаете, что все компоненты графического интерфейса являются специализациями представлений — во внутренней реализации все они являются subclasses класса `android.view.View`. Это означает, что все компоненты, используемые в интерфейсе приложения, обладают общими атрибутами и поведением. Например, все они могут отображаться на экране, а также могут сообщать информацию о своей ширине и высоте. Каждый компонент графического интерфейса, используемый в интерфейсе приложения, берет эту базовую функциональность и расширяет ее.



Макеты являются специализациями ViewGroup

Не только компоненты графического интерфейса являются специализациями класса представления View. В иерархии Android макет является специализацией **группы представлений**, представленной классом `android.view.ViewGroup`. Группа представлений — особая разновидность представлений, способных содержать другие представления.



Компонент графического интерфейса — специализация представления; объект, занимающий место на экране.

Макет — специализация группы представлений; особая разновидность представлений, способных содержать другие представления.

Что дает наследование от View

Объект View занимает прямоугольную область экрана. Он включает функциональность, необходимую всем представлениям для нормального существования в мире Android. Ниже перечислены некоторые аспекты этой функциональности — самые важные, на наш взгляд:

Чтение и запись свойств

Каждое представление представлено объектом Java; это означает, что вы можете задавать и читать его свойства в коде активности — например, получить значение, выбранное в раскрывающемся списке, или изменить текст надписи. Конкретный набор свойств и методов, которые могут использоваться в коде, зависит от типа представления. Каждому представлению можно назначить идентификатор, по которому к нему можно обращаться из кода.

Размер и позиция

По значениям ширины и высоты, заданным в программе, Android определяет необходимые размеры представления. Также можно указать, нужно ли снабдить представление отступами.

После того как представление появится на экране, вы сможете получить данные о его позиции, а также определить фактические размеры.

Обработка фокуса

Android управляет передачей фокуса в зависимости от действий пользователя. В частности, при передаче фокуса учитываются события сокрытия, удаления или появления представлений.

Обработка событий и слушатели

Каждое из представлений может реагировать на события. Также разработчик может создавать слушателей (listeners) для реакции на события, происходящие в представлении. Например, все представления способны реагировать на получение или потерю фокуса, а кнопка (а также все ее subclasses) может реагировать на щелчки.

Так как группа представлений также является специализацией представления, это означает, что все макеты и компоненты графического интерфейса тоже поддерживают общую функциональность.

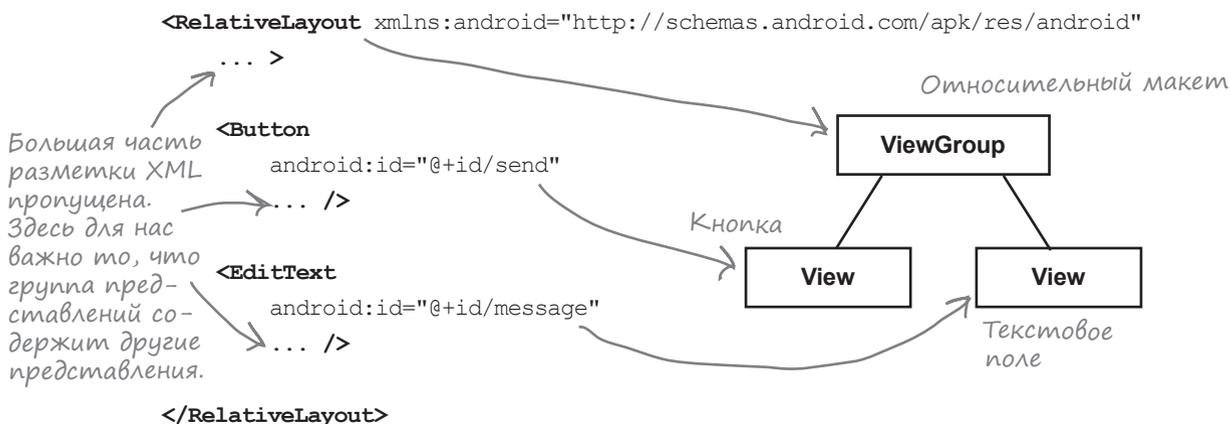
Некоторые методы View, которые могут использоваться в коде активности. Так как эти методы принадлежат базовому классу View, они являются общими для всех представлений и групп представлений.



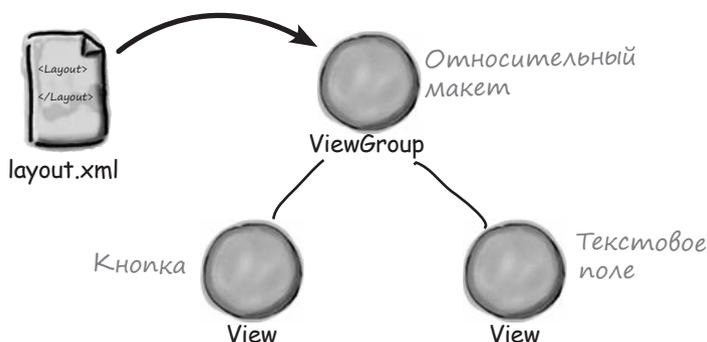
android.view.View
getId()
getHeight()
getWidth()
setVisibility(int)
findViewById(int)
isClickable()
isFocused()
requestFocus()
...

Макет в действительности является иерархией представлений

Макет, определяемый в разметке XML, формирует *иерархическое дерево представлений и групп представлений*. Например, представьте относительный макет с кнопкой и текстовым полем. Относительный макет является группой представлений, а кнопка и текстовое поле — представлениями. Группа представлений является родителем текстового поля, а представления являются потомками по отношению к группе:



При построении приложения XML-разметка макета автоматически преобразуется в объект `ViewGroup`, содержащий дерево элементов `View`. В приведенном выше примере кнопка преобразуется в объект `Button`, а надпись преобразуется в объект `TextView`. И `Button`, и `TextView` являются subclasses `View`.



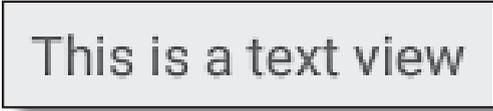
Все это и позволяет работать с представлениями из макета в коде Java. Каждое представление незаметно для разработчика преобразуется в объект `Java View`.

Знакомство с представлениями

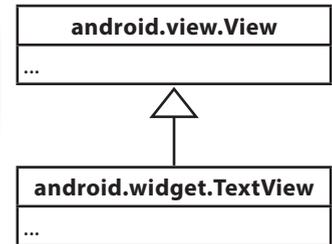
В этом разделе представлены самые популярные компоненты графического интерфейса. Некоторые из них уже упоминались выше, но мы все равно рассмотрим их. Мы не будем приводить весь API для каждого компонента — только самые важные моменты.

Надпись

Используется для вывода текста.



This is a text view



Определение в XML

Надпись определяется в макете элементом `<TextView>`. Атрибут `android:text` указывает, какой текст должен выводиться — обычно в форме строкового ресурса:

```

<TextView
    android:id="@+id/text_view"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/text" />
  
```

В API `TextView` входят многочисленные атрибуты для управления внешним видом текста, например размером текста. Для изменения размера используется атрибут `android:textSize`:

```

    android:textSize="14sp"
  
```

Размер текста задается в аппаратно-независимых пикселах (sp). Аппаратно-независимые пиксели учитывают факт включения крупных шрифтов на устройствах пользователей. Текст с размером 14 sp на устройстве, настроенном на использование крупных шрифтов, будет физически больше такого же шрифта на устройстве с мелкими шрифтами.

Использование надписи в коде активности

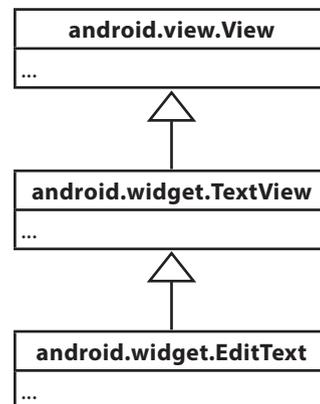
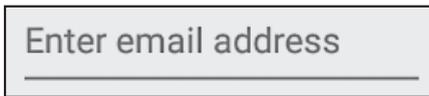
Для изменения текста, выводимого в надписи, используется программный код следующего вида:

```

TextView textView = (TextView) findViewById(R.id.text_view);
textView.setText("Some other string");
  
```

Текстовое поле

Аналог надписи, но с возможностью редактирования.



Определение в XML

Текстовое поле в XML определяется элементом `<EditText>`. Атрибут `android:hint` задает текст подсказки, которая объясняет пользователю, какую информацию следует вводить в поле.

```
<EditText
    android:id="@+id/edit_text"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:hint="@string/edit_text" />
```

Атрибут `android:inputType` определяет тип данных, которые должны вводиться в поле. Эта информация позволяет Android помочь пользователю в процессе ввода. Например, если поле предназначено для ввода числовых данных, используйте атрибут

```
android:inputType="number"
```

для выбора цифровой клавиатуры. На наш взгляд, наиболее полезны следующие типы:

Полный список типов приведен в электронной документации разработчика Android.

Значение	Что делает
phone	Предоставляет клавиатуру для ввода телефонных номеров.
textPassword	Предоставляет клавиатуру для ввода текста, вводимые данные маскируются.
textCapSentences	Первое слово в предложении начинается с прописной буквы.
textAutoCorrect	Автоматически исправляет вводимый текст.

В значении атрибута можно перечислить несколько типов, разделенных символом `|`. Например, чтобы первое слово предложения начиналось с прописной буквы, а во вводимом тексте автоматически исправлялись опечатки, используйте атрибут:

```
android:inputType="textCapSentences|textAutoCorrect"
```

Использование в коде активности

Для получения текста, содержащегося в текстовом поле, используется программный код следующего вида:

```
EditText editText = (EditText) findViewById(R.id.edit_text);
String text = editText.getText().toString();
```

Кнопка

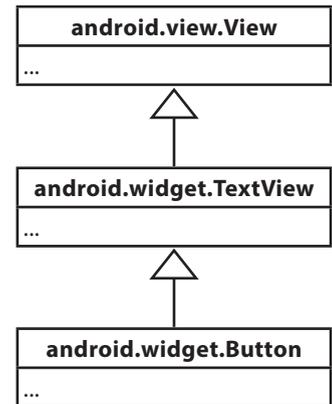
Обычно используется для того, чтобы приложение выполняло какие-либо действия при щелчке на кнопке.



Определение в XML

Кнопка в XML определяется элементом `<Button>`. Атрибут `android:text` указывает, какой текст должен отображаться на кнопке:

```
<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_text" />
```



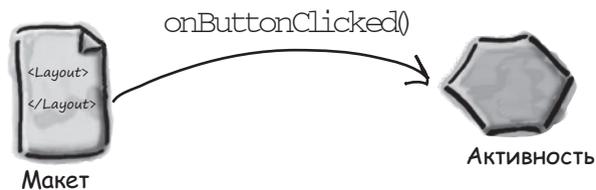
Использование в коде активности

Чтобы кнопка реагировала на щелчки, включите в XML макета атрибут `android:onClick` и присвойте ему имя вызываемого метода из кода активности:

```
android:onClick="onButtonClicked"
```

Затем в активности определяется метод следующего вида:

```
/** Вызывается при щелчке на кнопке */
public void onButtonClicked(View view) {
    // Сделать что-то по щелчку на кнопке
}
```



Двухпозиционная кнопка

Щелкая на двухпозиционной кнопке, пользователь выбирает одно из двух состояний.

Так выглядит двухпозиционная кнопка в выключенном состоянии.



А так она выглядит во включенном состоянии.

Определение в XML

Двухпозиционная кнопка определяется в XML элементом `<ToggleButton>`. Атрибуты `android:textOn` и `android:textOff` определяют текст, который должен выводиться на двухпозиционной кнопке в зависимости от ее состояния:

```
<ToggleButton
    android:id="@+id/toggle_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textOn="@string/on"
    android:textOff="@string/off" />
```

Использование в коде активности

Чтобы двухпозиционная кнопка реагировала на щелчки, включите атрибут `android:onClick` в XML макета. Присвойте ему имя вызываемого метода из кода активности:

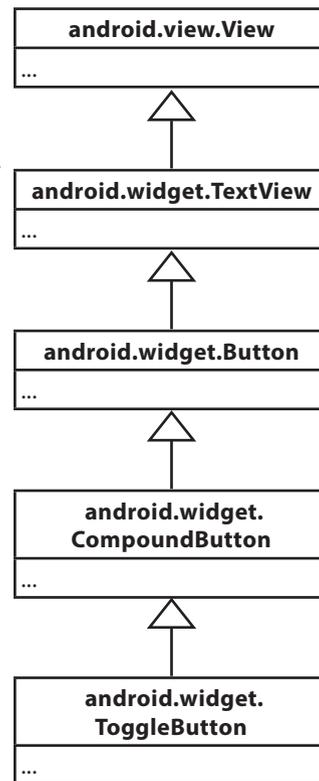
```
android:onClick="onToggleButtonClicked"
```

Точно так же, как при вызове метода по щелчку на обычной кнопке.

Затем в активности определяется метод следующего вида:

```
/** Вызывается при щелчке на двухпозиционной кнопке */
public void onToggleClicked(View view) {
    // Получить состояние двухпозиционной кнопки.
    boolean on = ((ToggleButton) view).isChecked();
    if (on) {
        // Вкл
    } else {
        // Выкл
    }
}
```

Возвращает true, если двухпозиционная кнопка находится во включенном состоянии, или false, если она находится в выключенном состоянии.



Выключатель

Выключатель представляет собой рычажок, который работает по тому же принципу, что и двухпозиционная кнопка.

Выключатель в выключенном состоянии.



А так он выглядит во включенном состоянии.

Определение в XML

Выключатель определяется в XML элементом `<Switch>`. Атрибуты `android:textOn` и `android:textOff` указывают, какой текст должен отображаться в зависимости от состояния выключателя:

```
<Switch
    android:id="@+id/switch_view"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textOn="@string/on"
    android:textOff="@string/off" />
```

Использование в коде активности

Чтобы выключатель реагировал на щелчки, включите атрибут `android:onClick` в XML макета. Присвойте ему имя вызываемого метода из кода активности:

```
android:onClick="onSwitchClicked"
```

Затем в активности определяется метод следующего вида:

```
/** Вызывается при щелчке на выключателе. */
public void onSwitchClicked(View view) {
    // Включенное состояние?
    boolean on = ((Switch) view).isChecked();

    if (on) {
        // Вкл
    } else {
        // Выкл
    }
}
```

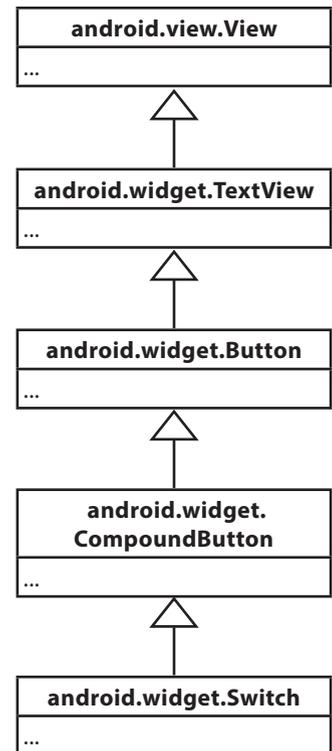
Код очень похож на тот, который использовался с двухпозиционной кнопкой.



Будьте осторожны!

Выключатель требует API уровня 14 и выше.

Если вы собираетесь использовать выключатель, убедитесь в том, что приложение использует SDK с уровнем API 14 и выше.



Флажки

Флажки (check boxes) предоставляют пользователю набор независимых вариантов. Пользователь может выбрать любые варианты по своему усмотрению. Каждый флажок может устанавливаться или сниматься независимо от всех остальных флажков.



← Два флажка. Пользователь может установить один флажок, другой, оба сразу или ни один из них.

Определение в XML

Флажок определяется в XML элементом `<CheckBox>`. Атрибут `android:text` используется для определения текста, выводимого рядом с флажком:

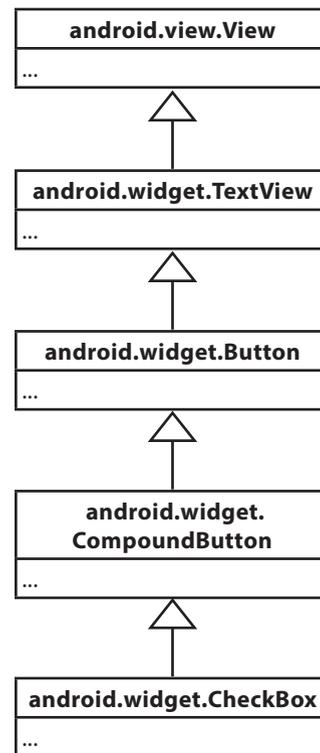
```
<CheckBox android:id="@+id/checkbox_milk"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="@string/milk" />
```

```
<CheckBox android:id="@+id/checkbox_sugar"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="@string/sugar" />
```

Использование в коде активности

Чтобы проверить, установлен ли некоторый флажок, используйте метод `isChecked()`. Если этот метод возвращает `true`, значит, флажок установлен:

```
CheckBox checkbox = (CheckBox) findViewById(R.id.checkbox_milk);
boolean checked = checkbox.isChecked();
if (checked) {
    //Действия для установленного флажка
}
```



Флажки (продолжение)

Чтобы обрабатывать щелчки на флажках (по аналогии со щелчками на кнопках), включите атрибут `android:onClick` в XML макета и присвойте ему имя вызываемого метода из кода активности:

```
<CheckBox android:id="@+id/checkbox_milk"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/milk"
    android:onClick="onCheckboxClicked"/>

<CheckBox android:id="@+id/checkbox_sugar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/sugar"
    android:onClick="onCheckboxClicked"/>
```

В этом случае метод `onCheckboxClicked()` будет вызван независимо от того, на каком флажке щелкнул пользователь. Также при желании можно было бы указать разные методы для всех флажков.

Затем в активности определяется метод следующего вида:

```
public void onCheckboxClicked(View view) {
    // Был ли установлен флажок, на котором щелкнул пользователь?
    boolean checked = ((CheckBox) view).isChecked();

    // Определить, на каком флажке был сделан щелчок
    switch(view.getId()) {
        case R.id.checkbox_milk:
            if (checked)
                // Кофе с молоком
            else
                // Черный кофе
            break;
        case R.id.checkbox_sugar:
            if (checked)
                // С сахаром
            else
                // Без сахара
            break;
    }
}
```

Переключатели

Переключатели (radio buttons) предоставляют набор вариантов, из которого пользователь может выбрать ровно один вариант:



← Группа переключателей ограничивает выбор пользователя ровно одним вариантом.

Определение в XML

Начните с определения группы переключателей — особой разновидности группы представлений — элементом `<RadioGroup>`. Внутри этого элемента отдельные переключатели определяются элементами `<RadioButton>`:

```
<RadioGroup android:id="@+id/radio_group"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:orientation="vertical">
```

← Вы можете выбрать между отображением переключателей в горизонтальном или вертикальном списке.

```
<RadioButton android:id="@+id/radio_cavemen"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="@string/cavemen" />
```

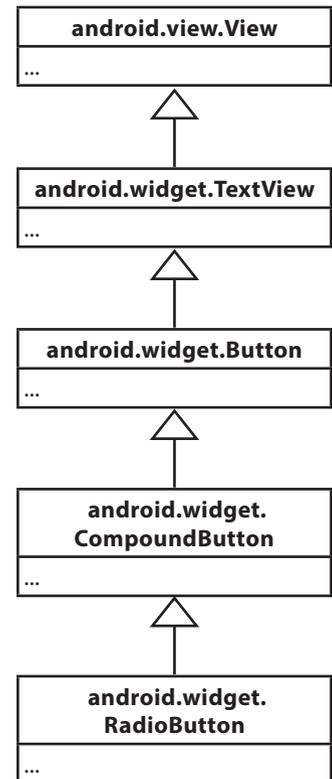
```
<RadioButton android:id="@+id/radio_astronauts"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="@string/astronauts" />
```

```
</RadioGroup>
```

Использование в коде активности

Чтобы определить, какой переключатель в группе установлен, используйте метод `getCheckedRadioButtonId()`:

```
RadioGroup radioGroup = findViewById(R.id.radioGroup);
int id = radioGroup.getCheckedRadioButtonId();
if (id == -1){
    //ни один переключатель не установлен
}
else{
    RadioButton radioButton = findViewById(id);
}
```



Группа переключателей, содержащая переключатели, является субклассом `LinearLayout`. Для групп переключателей можно использовать те же атрибуты, что и для линейных макетов.

Переключатели (продолжение)

Чтобы обрабатывать щелчки на переключателях, включите атрибут `android:onClick` в XML макета и присвойте ему имя вызываемого метода из кода активности:

```
<RadioGroup android:id="@+id/radio_group"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">

    <RadioButton android:id="@+id/radio_cavemen"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/cavemen"
        android:onClick="onRadioButtonClicked" />

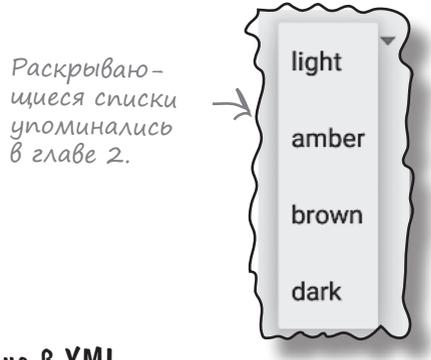
    <RadioButton android:id="@+id/radio_astronauts"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/astronauts"
        android:onClick="onRadioButtonClicked" />
</RadioGroup>
```

Затем в активности определяется метод следующего вида:

```
public void onRadioButtonClicked(View view) {
    RadioGroup radioGroup = findViewById(R.id.radioGroup);
    int id = radioGroup.getCheckedRadioButtonId();
    switch(id) {
        case R.id.radio_cavemen:
            // Установлен переключатель Cavemen
            break;
        case R.id.radio_astronauts:
            // Установлен переключатель Astronauts
            break;
    }
}
```

Раскрывающийся список

Как вы уже видели, раскрывающийся список содержит набор значений, из которых пользователь может выбрать только одно.

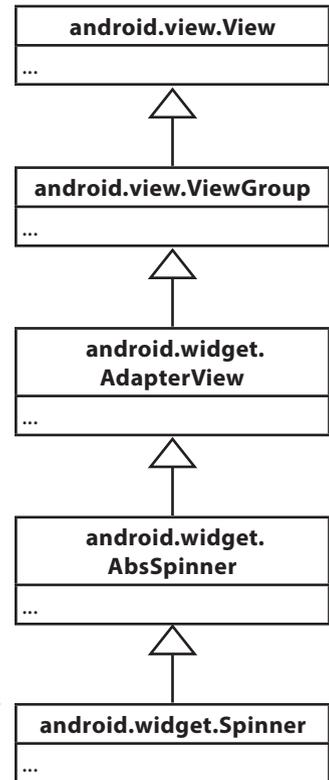


Определение в XML

Раскрывающийся список определяется в XML элементе `<Spinner>`. Чтобы добавить в раскрывающийся список статический массив элементов, используйте атрибут `android:entries` и присвойте ему массив строк.

```
<Spinner
    android:id="@+id/spinner"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:entries="@array/spinner_values" />
```

Существуют и другие способы заполнения раскрывающихся списков; они будут описаны позднее.



Массив строк добавляется в файл `strings.xml` следующим образом:

```
<string-array name="spinner_values">
    <item>light</item>
    <item>amber</item>
    <item>brown</item>
    <item>dark</item>
</string-array>
```

Использование в коде активности

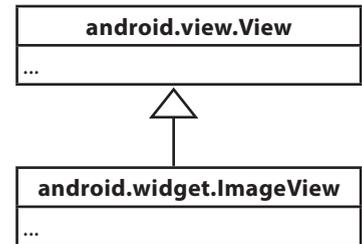
Чтобы получить значение текущего выбранного варианта, используйте метод `getSelectedItem()` и преобразуйте результат к типу `String`:

```
Spinner spinner = (Spinner) findViewById(R.id.spinner);
String string = String.valueOf(spinner.getSelectedItem());
```

Графическое представление

Графическое представление используется для вывода изображений:

Графическое представление используется для вывода изображений.



Класс `ImageView` является непосредственным subclasses `View`.

Добавление изображений в проект

Для начала включите файл изображения в проект как ресурс. Открыв папку `app/src/main/res` в своем проекте, вы увидите, что в ней находится папка с именем `drawable`. Она используется по умолчанию для хранения ресурсов изображений. Чтобы добавить файл с изображением, просто перетащите его в папку.

При желании вы можете использовать разные файлы изображений в зависимости от плотности экрана устройства. На экранах с высокой плотностью пикселей будут использоваться изображения с более высоким разрешением, а на экранах с низкой плотностью пикселей — изображения с пониженным разрешением. Для этого создайте в `app/src/main/res` вложенные папки `drawable` для разных вариантов плотности пикселей. Имя папки соответствует плотности пикселей устройства:

Чтобы создать новую папку, переключитесь в режим Project структуры папок, выделите папку `res` и выберите команду `File, New...`, `Android resource directory`.

<code>android-ldpi</code>	Экраны низкой плотности (около 120 dpi).
<code>android-mdpi</code>	Экраны средней плотности (около 160 dpi).
<code>android-hdpi</code>	Экраны высокой плотности (около 240 dpi).
<code>android-xhdpi</code>	Экраны сверхвысокой плотности (около 320 dpi).
<code>android-xxhdpi</code>	Экраны сверх-сверхвысокой плотности (около 480 dpi).
<code>android-xxxhdpi</code>	Экраны сверх-сверх-сверхвысокой плотности (около 640 dpi).

В зависимости от того, какую версию `Android Studio` вы используете, среда разработки может автоматически создать некоторые из этих папок за вас.

Затем разместите изображения с разными разрешениями в папках `drawable*`; проследите за тем, чтобы файлам с разными разрешениями присваивались совпадающие имена. `Android` выбирает используемое изображение на стадии выполнения, в зависимости от плотности устройства, на котором работает программа. Например, если устройство оснащено экраном сверхвысокой плотности, система будет использовать графику из папки `drawable-xhdpi`. Если изображение добавлено только в одну из папок, то `Android` использует один файл на всех устройствах. Обычно для этой цели используется папка `drawable`.

Определение в XML макета

Графическое представление определяется в XML элементом `<ImageView>`. Атрибут `android:src` указывает, какое изображение должно выводиться. Атрибут `android:contentDescription` позволяет добавить текстовое описание изображения, чтобы сделать приложение более доступным:

```
<ImageView
    android:layout_width="200dp"
    android:layout_height="100dp"
    android:src="@drawable/starbuzz_logo"
    android:contentDescription="@string/starbuzz_logo" />
```

Значение атрибута `android:src` задается в форме `"@drawable/имя_изображения"`, где `имя_изображения` — имя файла изображения (без расширения). Ресурсы изображений снабжаются префиксом `@drawable`. Префикс `@drawable` сообщает Android, что ресурс изображения хранится в одной или нескольких папках *drawable*.

Использование в коде активности

Исходное изображение и его текстовое описание задаются в коде активности методами `setImageResource()` и `setContentDescription()`:

```
ImageView photo = (ImageView) findViewById(R.id.photo);
int image = R.drawable.starbuzz_logo;
String description = "This is the logo";
photo.setImageResource(image);
photo.setContentDescription(description);
```

Этот фрагмент кода ищет ресурс изображения с именем `starbuzz_logo` в папках *drawable** и назначает его источником данных для графического представления с идентификатором `photo`. Для ссылок на ресурс изображения в коде активности используется синтаксис `R.drawable.имя_изображения`, где `имя_изображения` — имя файла изображения (без расширения).

Вывод изображений на кнопках

Кроме вывода изображений в графических представлениях, также можно выводить изображения на кнопках.

Вывод текста и изображения на кнопке

Чтобы вывести на кнопке текст, справа от которого находится графическое изображение, используйте атрибут `android:drawableRight` и укажите нужное изображение:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:drawableRight="@drawable/android"
    android:text="@string/click_me" />
```

Вывести графический ресурс android в правой части кнопки.



Чтобы изображение располагалось слева от текста, воспользуйтесь атрибутом `android:drawableLeft`:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:drawableLeft="@drawable/android"
    android:text="@string/click_me" />
```



Чтобы изображение располагалось под текстом, воспользуйтесь атрибутом `android:drawableBottom`:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:drawableBottom="@drawable/android"
    android:text="@string/click_me" />
```



При установке атрибута `android:drawableBottom` изображение выводится над текстом:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:drawableTop="@drawable/android"
    android:text="@string/click_me" />
```



Графическая кнопка

Графическая кнопка почти не отличается от обычной — просто на ней выводится только изображение, без текста.



Определение в XML

Графическая кнопка определяется в XML макета элементом `<ImageButton>`. Атрибут `android:src` определяет изображение, которое должно выводиться на кнопке:

```
<ImageButton
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/button_icon" />
```

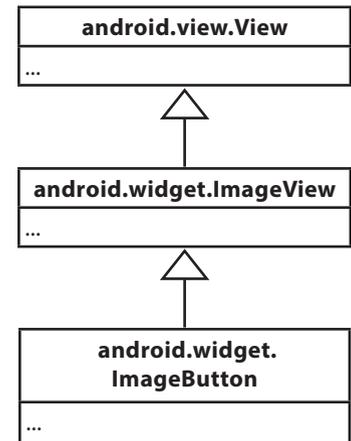
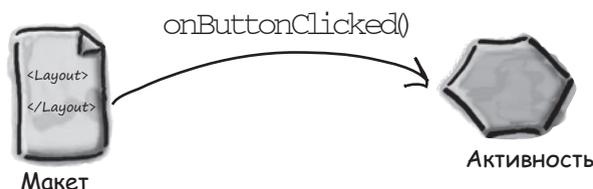
Использование в коде активности

Чтобы графическая кнопка реагировала на щелчки, включите в XML макета атрибут `android:onClick` и присвойте ему имя вызываемого метода из кода активности:

```
android:onClick="onButtonClicked"
```

Затем в активности определяется метод следующего вида:

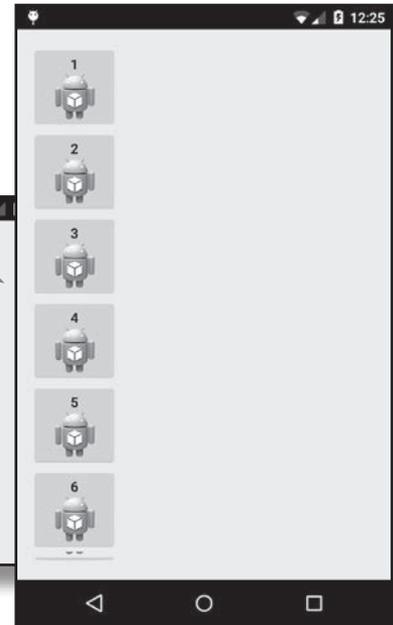
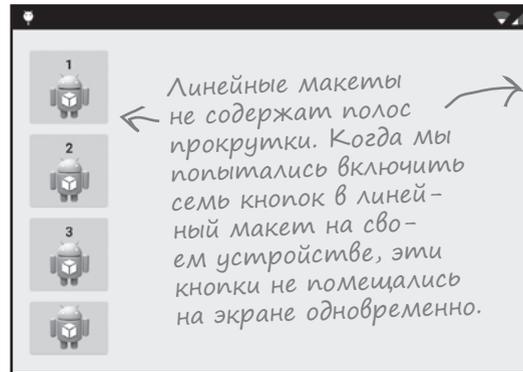
```
/** Вызывается при щелчке на кнопке */
public void onButtonClicked(View view) {
    // Сделать что-то по щелчку на кнопке
}
```



Класс `ImageButton` расширяет класс `ImageView`, а не класс `Button`. Вас это удивляет?

Прокручиваемые представления

Если в макет добавляется большое количество представлений, на устройствах с маленькими экранами могут возникнуть проблемы — во многих макетах нет полос прокрутки, которые бы позволяли прокручивать страницу. Например, если добавить семь больших кнопок в линейный макет, пользователь не сможет видеть их одновременно.

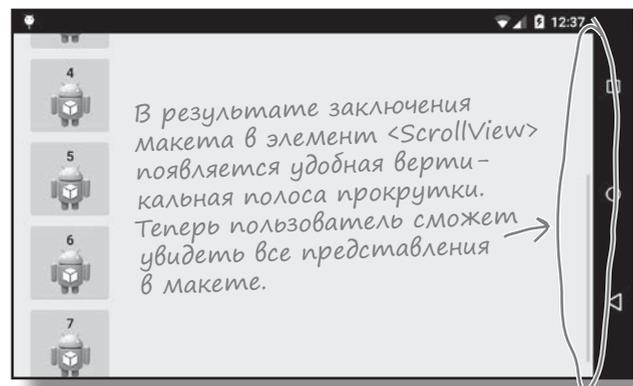


Чтобы добавить вертикальную полосу прокрутки, заключите существующий макет в элемент **<ScrollView>**:

```
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context=".MainActivity" >
```

Переместите эти атрибуты из исходного макета в элемент **<ScrollView>**, так как элемент **<ScrollView>** теперь является корневым.

```
<LinearLayout
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingBottom="16dp"
  android:paddingLeft="16dp"
  android:paddingRight="16dp"
  android:paddingTop="16dp"
  android:orientation="vertical" >
  ...
</LinearLayout>
</ScrollView>
```



Чтобы добавить в макет горизонтальную полосу прокрутки, заключите существующий макет в элемент **<HorizontalScrollView>**.

Уведомления

Остался еще один виджет, который мы хотим представить в этой главе: уведомление (toast). Это простое всплывающее сообщение, которое появляется на экране.

Уведомления выполняют чисто информационные функции, пользователь не может с ними взаимодействовать. Пока уведомление находится на экране, активность остается видимой и доступной для взаимодействия с пользователем. Уведомление автоматически закрывается по истечении тайм-аута.

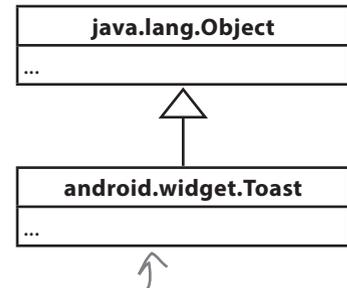
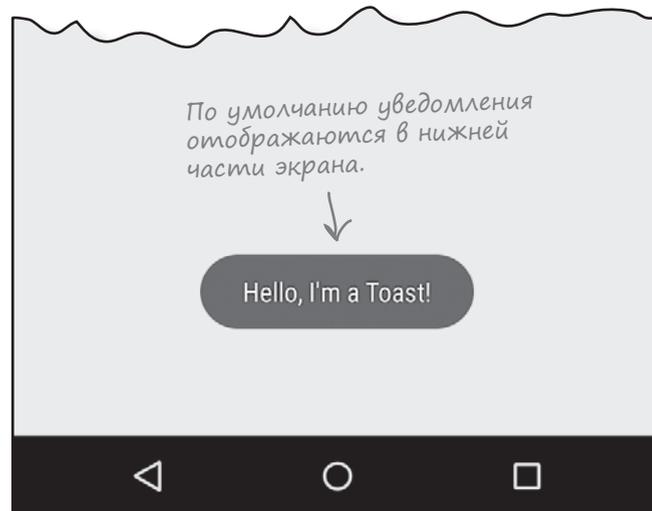
Использование в коде активности

Уведомления создаются только в коде активности; определить их в макете невозможно.

Чтобы создать уведомление, вызовите метод `Toast.makeText()` и передайте ему три параметра: `Context` (обычно для текущей активности), `CharSequence` (выводимое сообщение) и `int` (продолжительность). После того как объект уведомления будет создан, его можно вывести на экран вызовом метода `show()`. Пример кода с созданием уведомления, ненадолго появляющегося на экране:

```
CharSequence text = "Hello, I'm a Toast!";
int duration = Toast.LENGTH_SHORT;

Toast toast = Toast.makeText(this, text, duration);
toast.show();
```



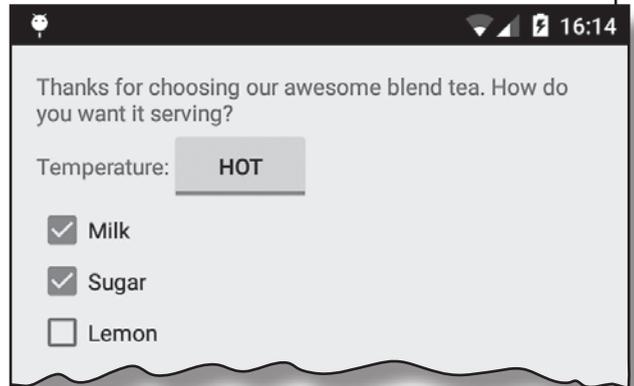
Тип Toast не является специализацией View. Тем не менее объекты этого типа хорошо подходят для вывода коротких сообщений, поэтому мы протаскили их в эту главу.



Упражнение

Пора применить на практике некоторые представления, описанные в этой главе. Создайте макет для следующего экрана:

Вы, вероятно, не хотите писать код здесь, но почему бы не поэкспериментировать в IDE?





Упражнение
Решение

Это всего лишь один из многочисленных способов создания макета. Не огорчайтесь, если ваш код выглядит иначе — существует много разных решений.

```
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingBottom="16dp"
  android:paddingLeft="16dp"
  android:paddingRight="16dp"
  android:paddingTop="16dp"
  android:columnCount="2"
  tools:context=".MainActivity" >
```

Мы использовали табличный макет, но с таким же успехом можно было выбрать относительный макет.

← Макет состоит из двух столбцов.

```
<TextView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_row="0"
  android:layout_column="0"
  android:layout_columnSpan="2"
  android:text="@string/message" />
```

← Надпись в верхней строке занимает оба столбца.

← Строки, необходимые для всех представлений, добавляются в файл strings.xml.

```
<TextView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_row="1"
  android:layout_column="0"
  android:text="@string/temp" />
```

← Надпись размещается в первом столбце следующей строки.

Для выбора температуры напиток используется двухпозиционная кнопка. Мы разместили ее в столбце 1 строки 1.

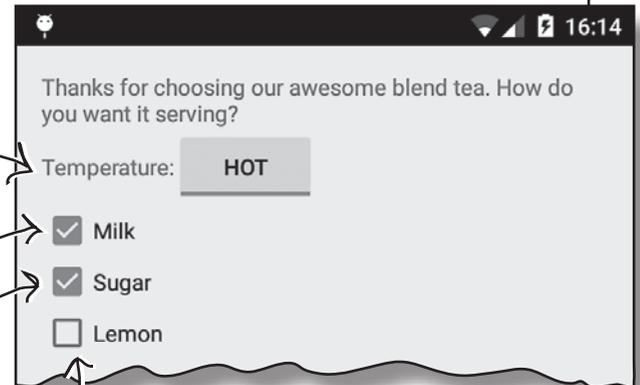
```
<ToggleButton
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_row="1"
  android:layout_column="1"
  android:textOn="@string/hot"
  android:textOff="@string/cold" />
```

```
<CheckBox android:id="@+id/checkbox_milk"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_row="2"
  android:layout_column="0"
  android:text="@string/milk" />
```

```
<CheckBox android:id="@+id/checkbox_sugar"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_row="3"
  android:layout_column="0"
  android:text="@string/sugar" />
```

```
<CheckBox android:id="@+id/checkbox_lemon"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_row="4"
  android:layout_column="0"
  android:text="@string/lemon" />
```

```
</GridLayout>
```



Каждый из вариантов (Milk, Sugar и Lemon) представлен флажком, который выводится в отдельной строке.



Развлечения с магнитами. Решение

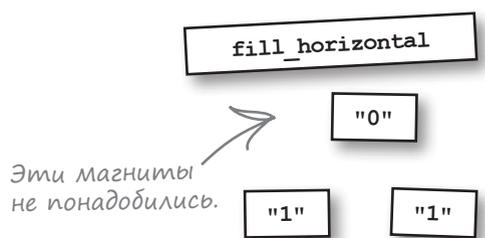
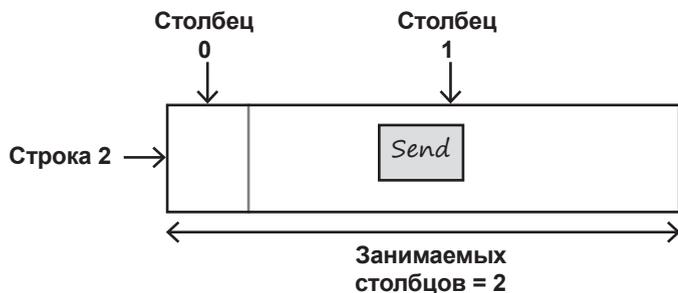
Мы написали разметку для выравнивания кнопки Send по центру третьей строки табличного макета, но от порыва ветра часть магнитов упала на пол. Удастся ли вам заполнить образовавшиеся пропуски магнитами?

```
<GridLayout...>  
  <TextView... />  
  <EditText.../>  
  <EditText.../>
```

Уже добавленные представления.

```
<Button  
  android:layout_width="wrap_content"  
  android:layout_height="wrap_content"  
  
  android:layout_row="2" Кнопка начинается в столбце 0 строки 2.  
  android:layout_column="0" Горизонтальное выравнивание по центру.  
  android:layout_gravity="center_horizontal"  
  android:layout_columnSpan="2" Занимает два столбца.  
  android:text="@string/send" />
```

```
</GridLayout>
```





Ваш инструментарий Android

Глава 5 осталась позади, а ваш инструментарий пополнился представлениями и группами представлений.

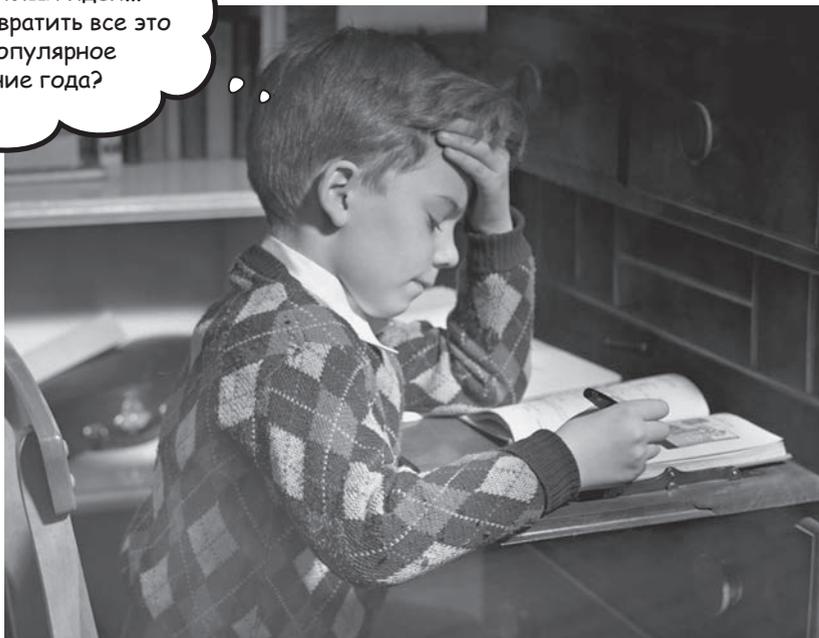
КЛЮЧЕВЫЕ МОМЕНТЫ



- Все компоненты графического интерфейса являются специализациями обобщенного представления. Все они представлены subclasses класса `android.view.View`.
- Все макеты являются subclasses класса `android.view.ViewGroup`. Группа представлений — разновидность представления, которая может содержать несколько представлений.
- Файл с разметкой XML макета преобразуется в объект `ViewGroup`, содержащий иерархическое дерево представлений.
- В относительном макете представления размещаются относительно других представлений или относительно родительского макета.
- В линейном макете представления размещаются либо по горизонтали, либо по вертикали. Направление задается атрибутом `android:orientation`.
- В табличном макете экран делится на ячейки, а разработчик указывает, какую ячейку (или ячейки) занимает каждое представление. Количество столбцов задается атрибутом `android:columnCount`. Атрибуты `android:layout_row` и `android:layout_column` определяют, в какой ячейке должно находиться каждое представление, а атрибут `android:layout_columnSpan` — сколько столбцов оно должно занимать.
- Атрибуты `android:padding*` определяют величину отступов вокруг представления.
- Используйте атрибут `android:layout_weight` в линейном представлении, если вы хотите, чтобы представление занимало дополнительное место в макете.
- Атрибут `android:layout_gravity` указывает, в какой части доступного пространства должно находиться представление.
- Атрибут `android:gravity` указывает, в какой части представления должно отображаться его содержимое.
- Элемент `<ToggleButton>` определяет двухпозиционную кнопку. Щелкая на кнопке, пользователь выбирает одно из двух состояний.
- Элемент `<Switch>` определяет выключатель, который работает по тому же принципу, что и двухпозиционная кнопка. Для использования выключателя необходим API уровня 14 и выше.
- Элемент `<CheckBox>` определяет флажок.
- Чтобы определить группу переключателей, сначала определите группу переключателей элементом `<RadioGroup>`. Отдельные переключатели в группе определяются элементом `<RadioButton>`.
- Элемент `<ImageView>` предназначен для вывода графики.
- Элемент `<ImageButton>` определяет кнопку, которая не содержит текста — только изображение.
- Для добавления полос прокрутки используются элементы `<ScrollView>` и `<HorizontalScrollView>`.
- Объект `Toast` представляет временное уведомление.

Обо всем по порядку

Столько светлых идей...
И как мне превратить все это
в самое популярное
приложение года?

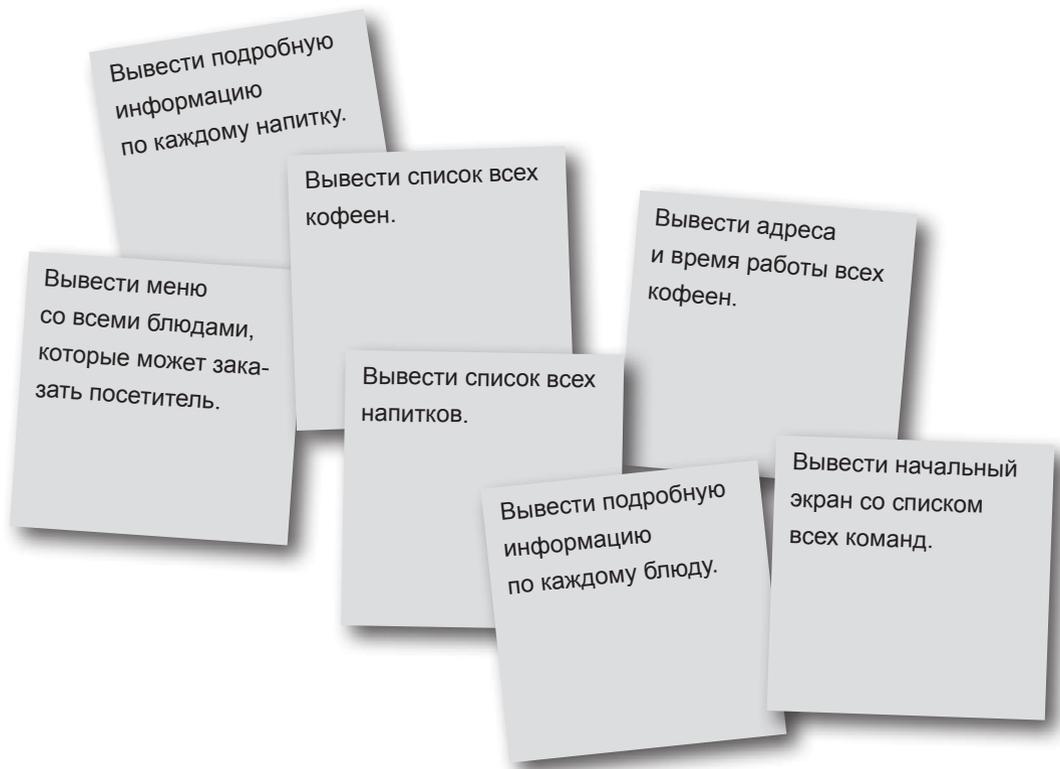


Хотите знать, как лучше организовать Android-приложение?

Мы рассмотрели основные структурные элементы, используемые при построении приложений; теперь пора **привести знания в порядок**. В этой главе мы покажем, как взять разрозненные идеи и **превратить их в классное приложение**. Мы покажем, как **списки данных** могут стать основой структуры вашего приложения и что **связывание списков** позволяет создавать **мощные и удобные приложения**. Попутно вы в общих чертах узнаете, как при помощи **слушателей событий** и **адаптеров** сделать ваше приложение более динамичным.

Каждое приложение начинается с идей

Когда разработчик задумывает новое приложение, у него обычно уже есть масса идей относительно того, что должно содержать это приложение. Допустим, руководство сети кофеен Starbuzz хочет создать новое приложение, которое привлечет в их заведения больше народа. Вот лишь некоторые возможности, которые, как они считают, должны быть реализованы в новом приложении:



Безусловно, все эти идеи будут полезны для пользователя. Но как построить из них интуитивно понятное, хорошо организованное приложение?

Проведите классификацию идей: Верхний уровень, категории, детализация/редактирование

Полезный способ упорядочения таких идей заключается в их классификации на три типа активностей: активности **верхнего уровня**, активности **категорий** и активности **детализации/редактирования**.

Активности верхнего уровня

Активности верхнего уровня представляют операции, наиболее важные для пользователя, и предоставляют простые средства для навигации к ним. В большинстве приложений первая активность, которую видит пользователь, является активностью верхнего уровня.

Активности категорий

Активности категорий выводят данные, принадлежащие конкретной категории, — часто в виде списка. Такие активности часто помогают пользователю перейти к активностям детализации/редактирования. Пример активности категории — вывод списка всех напитков, имеющих в Starbuzz.

Активности детализации/редактирования

Активности детализации/редактирования выводят подробную информацию по конкретной записи, предоставляют пользователю возможность редактирования существующих записей или ввода новых записей. Пример активности детализации/редактирования — активность, которая выводит подробную информацию о конкретном напитке.

После того как активности будут разделены на категории, классификация используется для построения иерархии, описывающей переходы между активностями.

Вывести начальный экран со списком всех команд.

Вывести список всех кофеен.

Вывести меню со всеми блюдами, которые может заказать посетитель.

Вывести список всех напитков.

Вывести подробную информацию по каждому напитку.

Вывести адреса и время работы всех кофеен.

Вывести подробную информацию по каждому блюду.



Упражнение

Представьте какое-нибудь приложение, которое вам хотелось бы создать. Какие активности оно должно содержать? Разделите их на активности верхнего уровня, активности категорий и активности детализации/редактирования.

Навигация по активностям

После того как вы разделите свои идеи на активности верхнего уровня, категорий и детализации/редактирования, эта классификация может использоваться для планирования навигации по приложению. Как правило, переход от активностей верхнего уровня к активностям детализации/редактирования должен осуществляться через активности категорий.

Активности верхнего уровня во главе иерархии

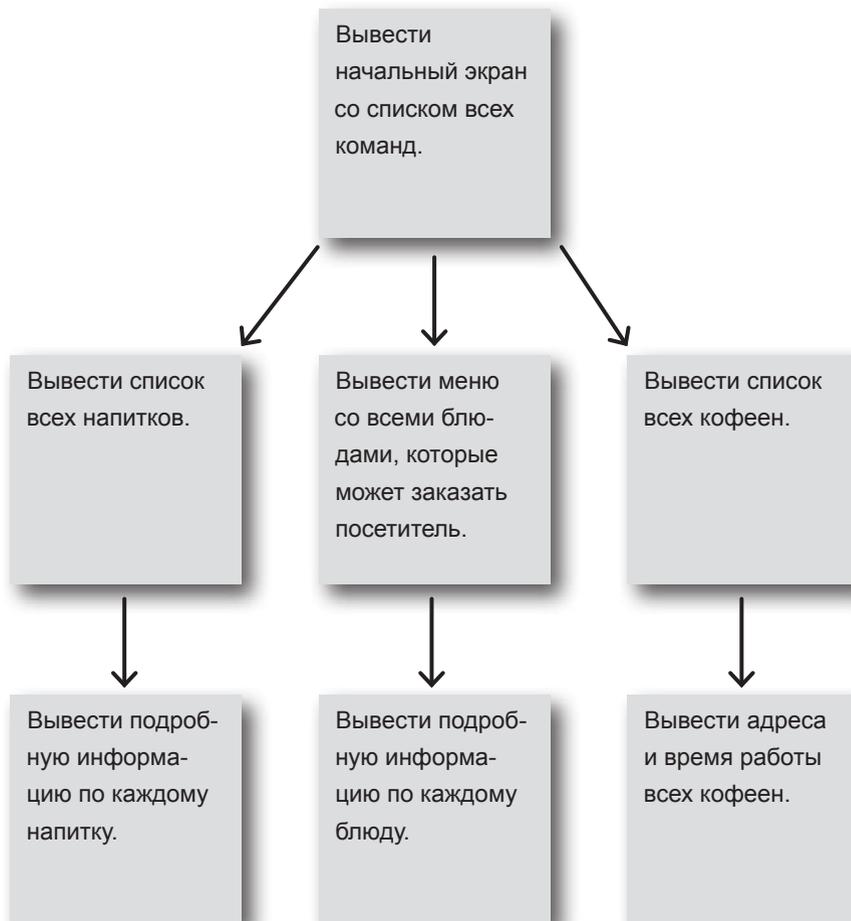
Пользователь начинает работу с приложением с этих активностей, поэтому они размещаются в начале иерархии.

Активности категорий занимают место между активностями верхнего уровня и активностями детализации/редактирования

Пользователи будут переходить от активностей верхнего уровня к активностям категорий. В сложных приложениях иерархия может включать несколько уровней категорий и подкатегорий.

Активности детализации/редактирования

Эти активности образуют нижний уровень иерархии активностей. Пользователи будут переходить к ним от активностей категорий.



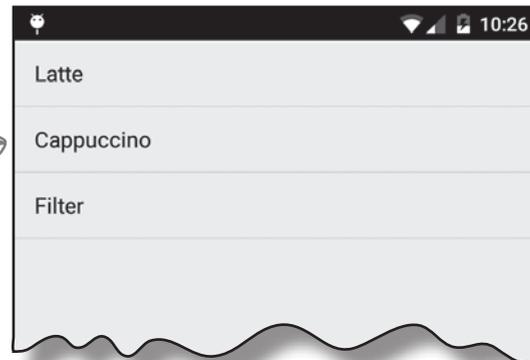
Для примера представьте, что пользователь хочет просмотреть подробную информацию об одном из напитков, подаваемых в Starbuzz. Для этого он запускает приложение и видит начальный экран активности верхнего уровня со списком команд. Пользователь выбирает команду вывода списка напитков. Чтобы увидеть подробную информацию о конкретном напитке, пользователь выбирает его в списке.

Навигация с использованием списковых представлений

В приложениях с такой структурой необходимо организовать навигацию, то есть переходы между активностями. В таких ситуациях чаще всего применяются **списковые представления**. Списковое представление отображает перечень объектов данных, который затем используется для навигации по приложению.

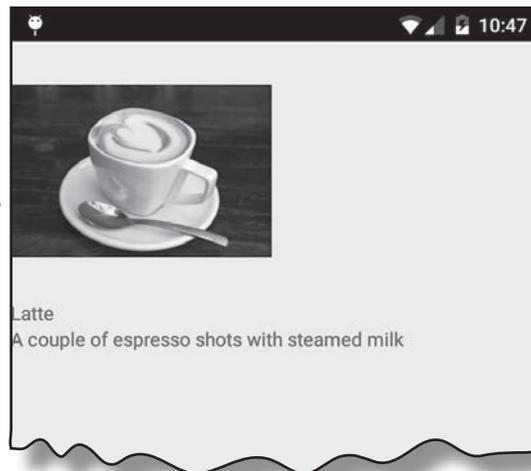
Например, на предыдущей странице было указано, что нам понадобится активность категории для вывода списка напитков, продаваемых в кофейнях Starbuzz. Эта активность может выглядеть так:

Компонент ListView со списком напитков. →



Активность использует списковое представление для вывода всех напитков, продаваемых в кофейнях Starbuzz. Чтобы перейти к конкретному напитку, пользователь щелкает на соответствующей строке, и на экране появляется подробное описание напитка.

Если щелкнуть в строке Latte списка ListView, вы увидите подробное описание этого напитка. →



В оставшейся части этой главы мы покажем на примере приложения Starbuzz, как использовать списковые представления для реализации этого механизма.

а вернее, его часть

Построим приложение Starbuzz



Мы не будем строить все активности категорий и детализации/редактирования, необходимые для всего приложения Starbuzz, а **ограничимся только напитками**. Мы построим активность верхнего уровня, которую будет видеть пользователь при запуске приложения; активность категории, которая выводит список напитков; и активность детализации/редактирования, которая выводит подробную информацию об одном напитке.

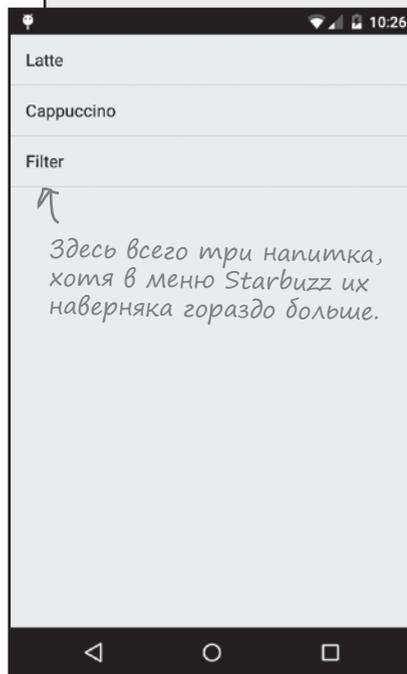
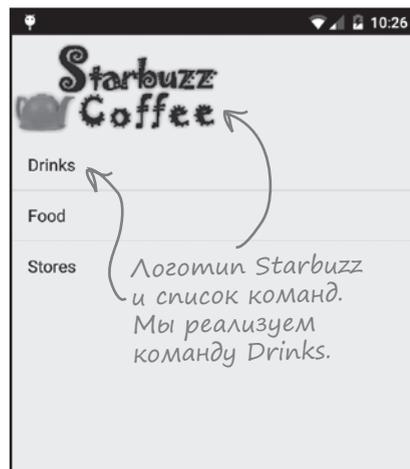
Активность верхнего уровня

При запуске приложения пользователь видит активность верхнего уровня — главную точку входа приложения. Эта активность включает изображение логотипа Starbuzz и навигационный список с командами для получения информации о напитках, еде и кофейнях.

Когда пользователь щелкает на одном из пунктов списка, приложение использует его выбор для перехода к другой активности. Например, если пользователь щелкнул на команде Drinks, приложение запускает активность категорий со списком напитков.

Активность категории для вывода списка напитков

Эта активность открывается при выборе пользователем команды Drinks в навигационном списке активности верхнего уровня. Активность выводит список всех напитков, продаваемых в кофейнях Starbuzz. Пользователь выбирает один из напитков, чтобы получить более подробную информацию о нем.



Активность детализации с информацией о напитке

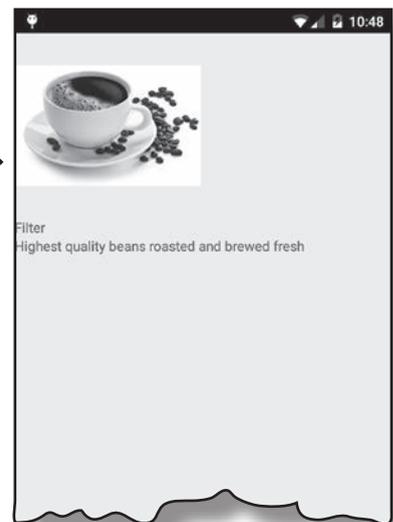
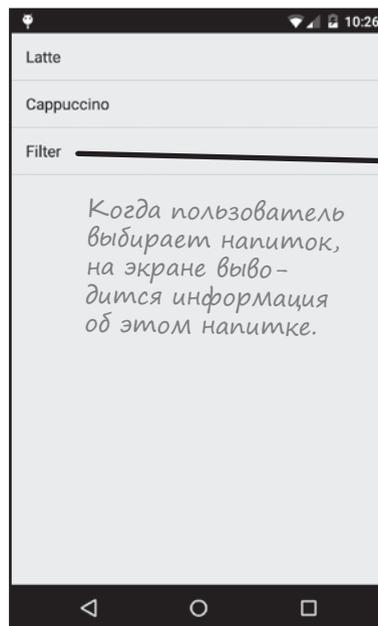
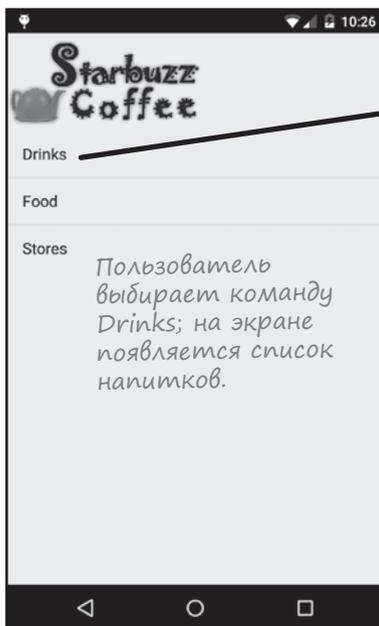
Активность для вывода подробной информации о напитке запускается тогда, когда пользователь щелкает на одном из напитков, перечисленных в активности категории напитков.

Активность выводит подробную информацию о напитке, выбранном пользователем: имя, фотография и описание.



Навигация пользователя в приложении

Пользователь переходит от активности верхнего уровня к активности с подробной информацией о напитке, щелкая на команде “Drinks” в активности верхнего уровня. После этого он выбирает конкретный напиток в открывшемся списке.



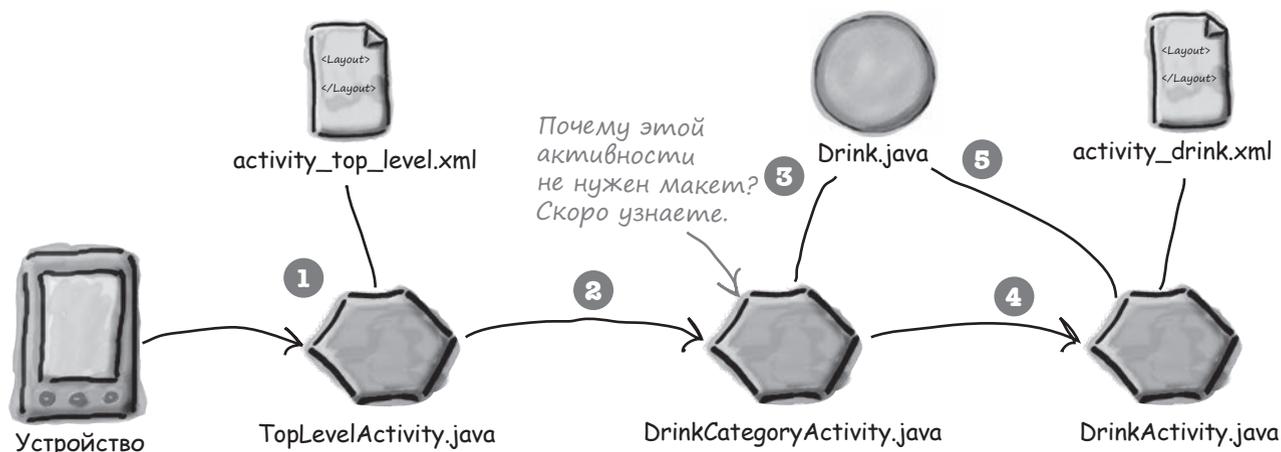
Структура приложения Starbuzz

Приложение состоит из трех активностей. `TopLevelActivity` — активность верхнего уровня — обеспечивает основную навигацию по разделам приложения. `DrinkCategoryActivity` — активность категории со списком напитков. Третья активность, `DrinkActivity`, содержит подробную информацию о конкретном напитке.

В этой версии данные напитков будут храниться в классе Java. В одной из следующих глав информация будет перенесена в базу данных, но пока мы хотим сосредоточиться на построении приложения, не отвлекаясь на поддержку баз данных.

- 1 **При запуске приложения открывается активность `TopLevelActivity`.**
Активность использует макет `activity_top_level.xml`. Активность выводит список команд для перехода к разделам напитков, блюд и кофеен.
- 2 **Пользователь выбирает команду `Drinks` в `TopLevelActivity`.**
Команда открывает активность `DrinkCategoryActivity`, которая отображает список напитков.
- 3 **Информация о напитках хранится в файле класса `Drink.java`.**
`DrinkCategoryActivity` получает данные напитков из этого класса.
- 4 **Пользователь выбирает напиток в `DrinkCategoryActivity`.**
При выборе запускается активность `DrinkActivity`, использующая макет `activity_drink.xml`.
- 5 **`DrinkActivity` получает подробную информацию о напитке из файла класса `Drink.java`.**

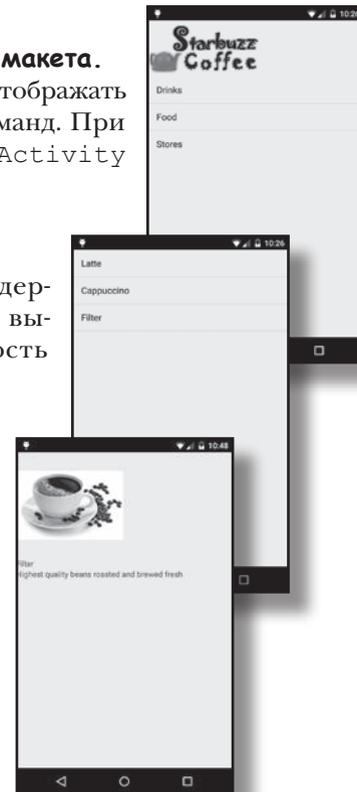
← Для активности `DrinkCategoryActivity` создавать макет не нужно — вскоре вы увидите, почему.



Последовательность действий

Ниже перечислены основные этапы построения приложения:

- 1** **Добавление класса Drink и ресурсов изображений.**
 Класс содержит подробную информацию о напитках; также в приложении используются ресурсы изображений напитков и логотипа Starbuzz.
- 2** **Создание активности TopLevelActivity и ее макета.**
 «Точка входа» приложения: активность должна отображать логотип Starbuzz и навигационный список команд. При выборе команды Drink активность TopLevelActivity должна открывать DrinkCategoryActivity.
- 3** **Создание DrinkCategoryActivity.**
 Активность DrinkCategoryActivity содержит список всех имеющихся напитков. При выборе напитка должна открываться активность DrinkCategory.
- 4** **Создание активности DrinkActivity и ее макета.**
 Активность DrinkActivity выводит информацию о напитке, выбранном пользователем в списке DrinkCategoryActivity.



Создание проекта

Проект приложения создается точно так же, как это делалось в предыдущих главах.

Создайте новый проект Android для приложения с именем “Starbuzz” и именем пакета `com.hfad.starbuzz`. Выберите минимальный уровень SDK равным API 15. Приложение должно содержать активность с именем “TopLevelActivity” и макет с именем “activity_top_level”.



Добавление ресурсов
 TopLevelActivity
 DrinkCategoryActivity
 DrinkActivity

Класс Drink

Для начала добавим в приложение класс Drink. *Drink.java* — обычный файл класса Java, из которого активности будут получать данные напитков. Класс определяет массив из трех объектов, представляющих напитки; каждый объект состоит из названия, описания и идентификатора ресурса изображения. Добавьте класс в пакет *com.hfad.starbuzz* в папке *app/src/main/java* вашего проекта, присвоив ему имя Drink. Сохраните изменения.



Добавление ресурсов
 TopLevelActivity
 DrinkCategoryActivity
 DrinkActivity

```

package com.hfad.starbuzz;

public class Drink {
    private String name;
    private String description;
    private int imageResourceId;

    //drinks — массив из трех объектов Drink
    public static final Drink[] drinks = {
        new Drink("Latte", "A couple of espresso shots with steamed milk",
            R.drawable.latte),
        new Drink("Cappuccino", "Espresso, hot milk, and a steamed milk foam",
            R.drawable.cappuccino),
        new Drink("Filter", "Highest quality beans roasted and brewed fresh",
            R.drawable.filter)
    };

    //Для каждого напитка хранится имя, описание и ресурс изображения
    private Drink(String name, String description, int imageResourceId) {
        this.name = name;
        this.description = description;
        this.imageResourceId = imageResourceId;
    }

    public String getDescription() {
        return description;
    }

    public String getName() {
        return name;
    }

    public int getImageResourceId() {
        return imageResourceId;
    }

    public String toString() {
        return this.name;
    }
}

```

Каждый объект Drink состоит из полей имени, описания и идентификатора ресурса изображения. Идентификаторы ресурсов принадлежат изображениям напитков, которые будут добавлены в проект на следующей странице.

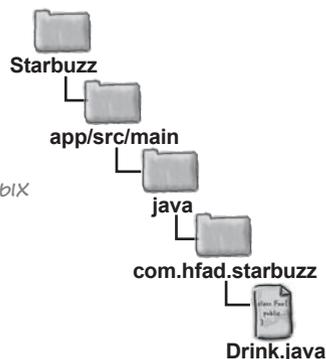
drinks — массив из трех объектов Drink.

Изображения напитков. Мы добавим их на следующем этапе.

Конструктор Drink.

Get-методы для приватных переменных.

В качестве строкового представления Drink используется название напитка.

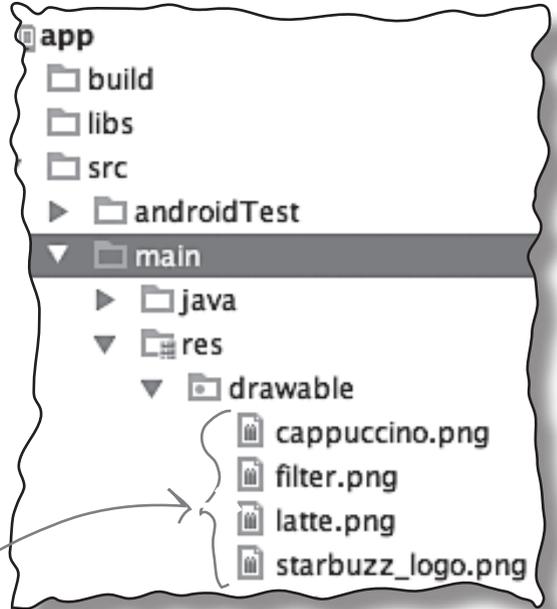


Файлы изображений

Код `Drink` включает три ресурса изображений напитков с идентификаторами `R.drawable.latte`, `R.drawable.cappuccino` и `R.drawable.filter`. Они нужны для того, чтобы пользователь мог увидеть фотографию напитка. `R.drawable.latte` ссылается на файл изображения с именем `latte`, `R.drawable.cappuccino` — на файл изображения с именем `cappuccino`, а `R.drawable.filter` — на файл изображения с именем `filter`.

Эти файлы изображений необходимо добавить в проект вместе с изображением логотипа `Starbuzz`, чтобы его можно было использовать в активности верхнего уровня. Для этого загрузите файлы `starbuzz-logo.png`, `cappuccino.png`, `filter.png` и `latte.png` по адресу <https://tinyurl.com/HeadFirstAndroid> и перетащите их в папку `app/src/main/res/drawable` в проекте `Starbuzz`. При добавлении изображений в приложение необходимо решить, собираетесь ли вы использовать разные изображения для экранов с разной плотностью пикселей. В нашем примере одно изображение будет использоваться для всех экранов, поэтому достаточно поместить один экземпляр изображения в одну папку. Если вы в своем приложении решите использовать разные версии графики для разных экранов, разместите разные варианты изображений в папках `drawable*`, как описано в главе 5.

Четыре файла изображений. Добавьте их в Android Studio перетаскиванием в папку `drawable`.



При сохранении изображений в проекте Android назначает им идентификаторы в формате `R.drawable.имя_файла`. Например, изображению из файла `latte.png` присваивается идентификатор `R.drawable.latte`, соответствующий значению ресурса изображения `latte` из класса `Drink`.



Drink

name: "Latte"
 description: "A couple of
 espresso shots with steamed
 milk"
 imageResourceId: R.drawable.latte

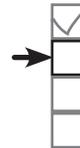


R.drawable.latte

Изображению `latte.png` присваивается идентификатор `R.drawable.latte`.

После добавления класса `Drink` и ресурсов изображений в проект можно переходить к активностям. Начнем с активности верхнего уровня.

Макет верхнего уровня содержит изображение и список

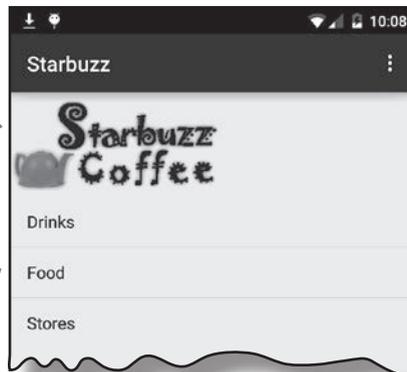


Добавление ресурсов
 TopLevelActivity
 DrinkCategoryActivity
 DrinkActivity

При создании проекта активности по умолчанию было присвоено имя `TopLevelActivity.java`, а ее макету — имя `activity_top_level.xml`. Макет необходимо изменить так, чтобы в нем выводились изображение и список.

Логотип Starbuzz. Изображение было добавлено в проект на предыдущей странице.

Статический список вариантов.



В главе 5 было показано, как включить изображение в макет с использованием графического представления. В нашем примере понадобится графическое представление для логотипа Starbuzz, поэтому мы создадим представление, использующее `starbuzz_logo.png` в качестве источника. Следующая разметка определяет графическое представление в макете:

```
<ImageView
    android:layout_width="200dp"
    android:layout_height="100dp"
    android:src="@drawable/starbuzz_logo"
    android:contentDescription="@string/starbuzz_logo" />
```

Размеры, которыми должно обладать изображение.

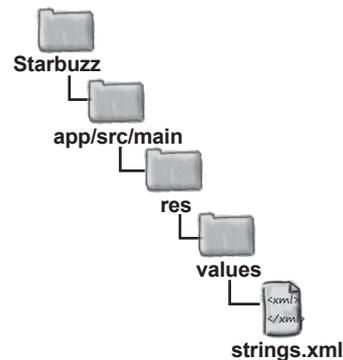
Источником графических данных является файл `starbuzz_logo.png`, который мы добавили в приложение.

Добавление описания улучшает доступность приложения.

При использовании графического представления в приложении атрибут `android:contentDescription` применяется для добавления описания; это улучшает доступность приложения для пользователей с ограниченными возможностями. В нашем примере используется строка `@string/starbuzz_logo`. Добавьте ее в файл `strings.xml`:

```
<resources>
    ...
    <string name="starbuzz_logo">Starbuzz logo</string>
</resources>
```

Вот и все, что необходимо для включения изображения в макет. Теперь можно переходить к списку.



Использование спискового представления для вывода списка

Списковое представление позволяет вывести вертикальный список объектов данных, который в дальнейшем может использоваться для навигации по приложению. Добавим в макет списковое представление для набора команд, которые в дальнейшем будут открывать другие активности.

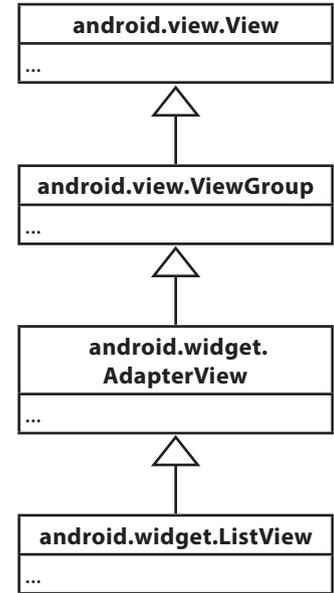
Определение спискового представления в XML

Для добавления спискового представления в макет используется элемент `<ListView>`. Чтобы заполнить списковое представление данными, используйте атрибут `android:entries` и присвойте ему массив строк. Строки из массива будут отображаться в списковом представлении в виде набора надписей `TextView`.

Пример добавления в макет спискового представления, которое получает значения из массива строк `options`:

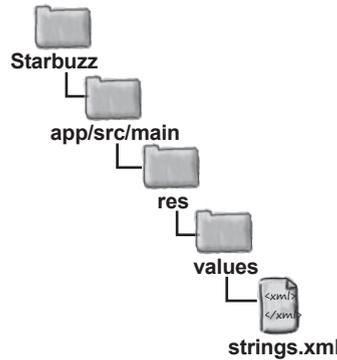
```
<ListView ← Определяет списковое представление.
    android:id="@+id/list_options"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:entries="@array/options" />
```

Значения, выводимые в списковом представлении, определяются массивом `options`.



Массив определяется точно так же, как это уже делалось ранее, — данные включаются в массив `strings.xml`:

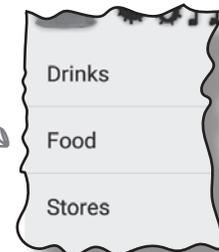
```
<resources>
...
<string-array name="options">
    <item>Drinks</item>
    <item>Food</item>
    <item>Stores</item>
</string-array>
</resources>
```



Списковое представление заполняется тремя значениями: `Drinks`, `Food` и `Stores`.



Атрибут `entries` заполняет компонент `ListView` значениями из массива `options`. Каждый пункт списка `ListView` представляет собой компонент `TextView`.



Полный код макета верхнего уровня

Так выглядит полная разметка нашего макета (не забудьте внести изменения, выделенные жирным шрифтом, в свой макет):

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".TopLevelActivity" >

    <ImageView
        android:layout_width="200dp"
        android:layout_height="100dp"
        android:src="@drawable/starbuzz_logo"
        android:contentDescription="@string/starbuzz_logo" />

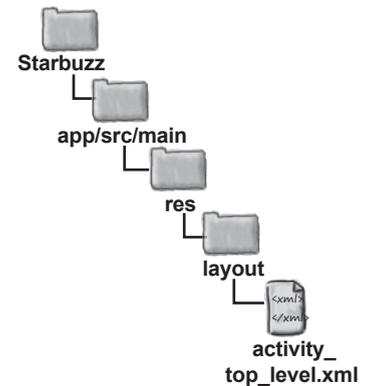
    <ListView
        android:id="@+id/list_options"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:entries="@array/options" />

</LinearLayout>
```

Используется линейный макет с вертикальной ориентацией. В этом случае списковое представление отображается прямо под логотипом Starbuzz.



Добавление ресурсов
TopLevelActivity
DrinkCategoryActivity
DrinkActivity



Тест-драйв

Внесите все необходимые изменения в файл `activity_top_level.xml`, а также обновите `strings.xml`. При запуске приложения на экране должен появиться логотип Starbuzz, под которым находится списковое представление с тремя значениями из массива `options`.

Если щелкнуть на любой из команд списка, ничего не произойдет, так как мы еще не объяснили списковому представлению, как следует реагировать на щелчки. На следующем шаге вы увидите, как научить списковое представление реагировать на щелчки и как открыть вторую активность.

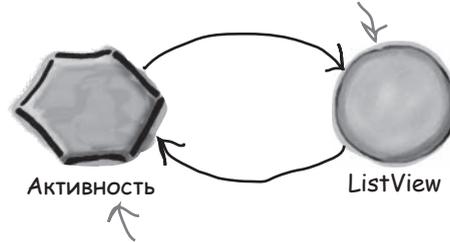


Обработка щелчков компонентом ListView

Чтобы пункты списка реагировали на щелчки, следует реализовать **слушателя событий**.

Слушатель событий отслеживает события, происходящие в приложении, — например, щелчки на представлениях, потерю или получение ими фокуса или нажатие физической клавиши на устройстве. Реализация слушателя событий позволит вам обнаруживать конкретные действия пользователя — скажем, щелчки на вариантах списка — и реагировать на них.

Компонент `ListView` должен знать, что происходящие с ним события представляют интерес для активности.



`ListView` сообщает активности, что на пункте списка был сделан щелчок, чтобы активность могла среагировать на событие.

`OnItemClickListener` отслеживает щелчки на вариантах списка

Если вы хотите, чтобы варианты списка реагировали на щелчки, создайте объект `OnItemClickListener` и реализуйте его метод `onItemClick()`. Слушатель `OnItemClickListener` отслеживает щелчки на вариантах списка, а метод `onItemClick()` определяет реакцию активности на щелчок. По параметрам, передаваемым методу `onItemClick()`, можно получить дополнительную информацию о событии — например, получить ссылку на вариант из списка, узнать его позицию в списке представлении (начиная с 0) и идентификатор записи используемого набора данных.

В нашем примере при щелчке на первом варианте спискового представления — варианте в позиции 0 — должна запускаться активность `DrinkCategoryActivity`. Если щелчок сделан на варианте в позиции 0, необходимо создать интент для запуска `DrinkCategoryActivity`. Код создания слушателя выглядит так:

`OnItemClickListener` — вложенный класс по отношению к классу `AdapterView`. `ListView` — subclass `AdapterView`.

```

AdapterView.OnItemClickListener itemClickListener = new AdapterView.OnItemClickListener() {
    public void onItemClick(AdapterView<?> listView,
        Drinks — первый вариант в списковом представлении находится в позиции 0.
        View itemView,
        int position,
        long id) {
            if (position == 0) {
                Intent intent = new Intent(TopLevelActivity.this, DrinkCategoryActivity.class);
                startActivity(intent);
            }
        }
};
    
```

Представление, на котором был сделан щелчок (списковое представление в данном случае).

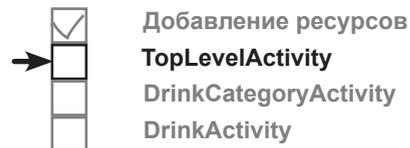
Дополнительная информация о варианте спискового представления — например, представление и его позиция.

Интент выдается `TopLevelActivity`.

Должен запускать `DrinkCategoryActivity`.

После того как слушатель будет создан, его необходимо добавить к `ListView`.

Назначение слушателя для спискового представления



После того как объект `OnItemClickListener` будет создан, его необходимо связать со списковым представлением. Эта задача решается при помощи метода `setOnItemClickListener()` класса `ListView`. Метод получает один аргумент – самого слушателя:

```

AdapterView.OnItemClickListener itemClickListener = new AdapterView.OnItemClickListener() {
    public void onItemClick(AdapterView<?> listView,
        ...
    }
};

ListView listView = (ListView) findViewById(R.id.list_options);
listView.setOnItemClickListener(itemClickListener);
    
```

← Только что созданный слушатель.

Добавление слушателя к списковому представлению крайне важно – именно эта операция обеспечивает получение слушателем оповещений о том, что пользователь щелкает на списковом представлении. Если этого не сделать, варианты из спискового представления не будут реагировать на щелчки.

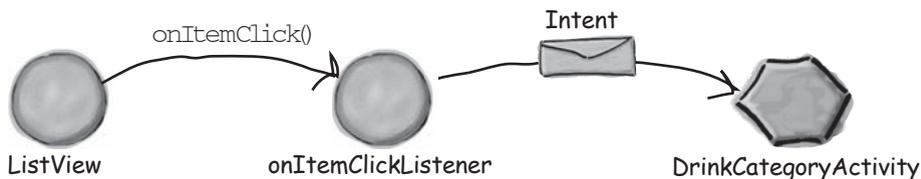
Итак, вы знаете все необходимое для того, чтобы научить списковое представление `TopLevelActivity` реагировать на щелчки.

Что происходит при выполнении кода

- 1 **Метод `onCreate()` из `TopLevelActivity` создает объект `onItemClickListener` и связывает его с объектом `ListView`.**



- 2 **Когда пользователь щелкает на варианте из спискового представления, вызывается метод `onItemClick()` слушателя `onItemClickListener`. Если щелчок был сделан на команде `Drinks`, `onItemClickListener` создает интент для запуска активности `DrinkCategoryActivity`.**



Полный код TopLevelActivity

Перед вами полный код TopLevelActivity.java. Замените код, сгенерированный мастером, тем, что приведен ниже, и сохраните изменения:

```

package com.hfad.starbuzz;

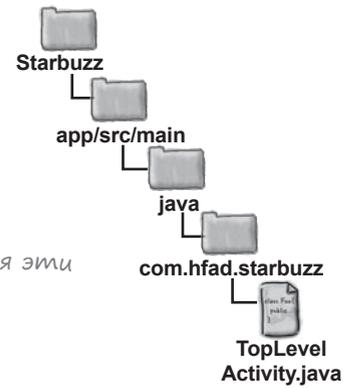
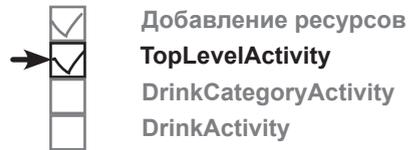
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ListView;
import android.view.View;

public class TopLevelActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_top_level);
        //Создать OnItemClickListener
        AdapterView.OnItemClickListener itemClickListener =
            new AdapterView.OnItemClickListener() {
                public void onItemClick(AdapterView<?> listView,
                    View v,
                    int position,
                    long id) {
                    if (position == 0) {
                        Intent intent = new Intent(TopLevelActivity.this,
                            DrinkCategoryActivity.class);
                        startActivity(intent);
                    }
                }
            };
        //Добавить слушателя к списковому представлению
        ListView listView = (ListView) findViewById(R.id.list_options);
        listView.setOnItemClickListener(itemClickListener);
    }
}

```

В коде используются эти внешние классы.



Создание слушателя.

Реализация его метода onItemClick().

Запустить DrinkCategoryActivity, если пользователь щелкнул на варианте Drinks. Даже если Android Studio скажет, что активность не существует, не беспокойтесь — сейчас мы ее создадим.

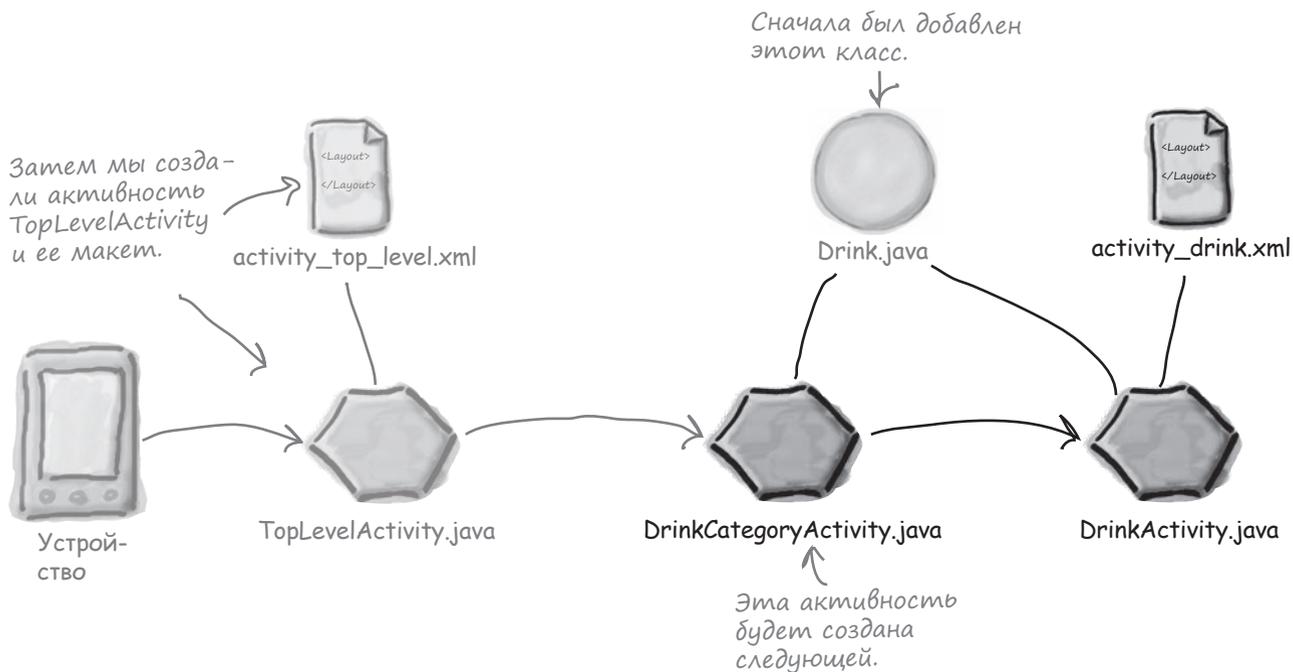
Добавление слушателя к списковому представлению.

Что было сделано

Итак, мы добавили в приложение класс `Drink.java`, создали активность `TopLevelActivity` и ее макет.



Добавление ресурсов
`TopLevelActivity`
`DrinkCategoryActivity`
`DrinkActivity`



Следующее, что нужно сделать, — создать активность `DrinkCategoryActivity`, которая должна запускаться щелчком на команде `Drinks` в активности `TopLevelActivity`.

Часто задаваемые вопросы

В: Зачем создавать слушателя событий, чтобы варианты из `ListView` реагировали на щелчки? Почему бы не воспользоваться атрибутом `android:onClick` в разметке?

О: Атрибут `android:onClick` в макетах может использоваться только для кнопок или любых представлений, являющихся subclasses `Button`, например `CheckBox` и `RadioButton`.

Класс `ListView` не является subclassом `Button`, поэтому решение с атрибутом `android:onClick` не работает. Именно поэтому приходится создавать свою реализацию слушателя.



Упражнение

Перед вами код активности из другого проекта. Когда пользователь щелкает на варианте в списке представлении, код должен выводить текст этого варианта в надписи. Будет ли этот код работать как положено? Если нет, то почему? Надписи назначен идентификатор `text_view`, а списковому представлению — идентификатор `list_view`.

```
package com.hfad.ch06_ex;

import android.app.Activity;
import android.os.Bundle;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ListView;
import android.widget.TextView;
import android.view.View;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        final TextView textView = (TextView) findViewById(R.id.text_view);
        AdapterView.OnItemClickListener itemClickListener =
            new AdapterView.OnItemClickListener() {
                public void onItemClick(AdapterView<?> listView,
                    View v,
                    int position,
                    long id) {
                    TextView item = (TextView) v;
                    textView.setText(item.getText());
                }
            };
        ListView listView = (ListView) findViewById(R.id.list_view);
    }
}
```



Упражнение
Решение

Перед вами код активности из другого проекта. Когда пользователь щелкает на варианте в списке представлении, код должен выводить текст этого варианта в надписи. Будет ли этот код работать как положено? Если нет, то почему? Надписи назначен идентификатор `text_view`, а списковому представлению — идентификатор `list_view`.

```
package com.hfad.ch06_ex;

import android.app.Activity;
import android.os.Bundle;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.AdapterView.OnItemSelectedListener;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.AdapterView.OnItemSelectedListener;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.AdapterView.OnItemSelectedListener;
import android.view.View;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        final TextView textView = (TextView) findViewById(R.id.text_view);
        AdapterView.OnItemClickListener itemClickListener =
            new AdapterView.OnItemClickListener() {
                public void onItemClick(AdapterView<?> listView,
                    View v,
                    int position,
                    long id) {
                    TextView item = (TextView) v;
                    textView.setText(item.getText());
                }
            };
        ListView listView = (ListView) findViewById(R.id.list_view);
    }
}
```

Вариант спискового представления, на котором щелкнул пользователь. Возвращается объект `TextView`, и для получения текста следует использовать метод `getText()`.

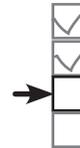
→ `TextView item = (TextView) v;`
 → `textView.setText(item.getText());`

← Код не будет работать так, как задумано, потому что в конце метода отсутствует строка

`listView.setOnItemClickListener(itemClickListener);`

В остальном все нормально.

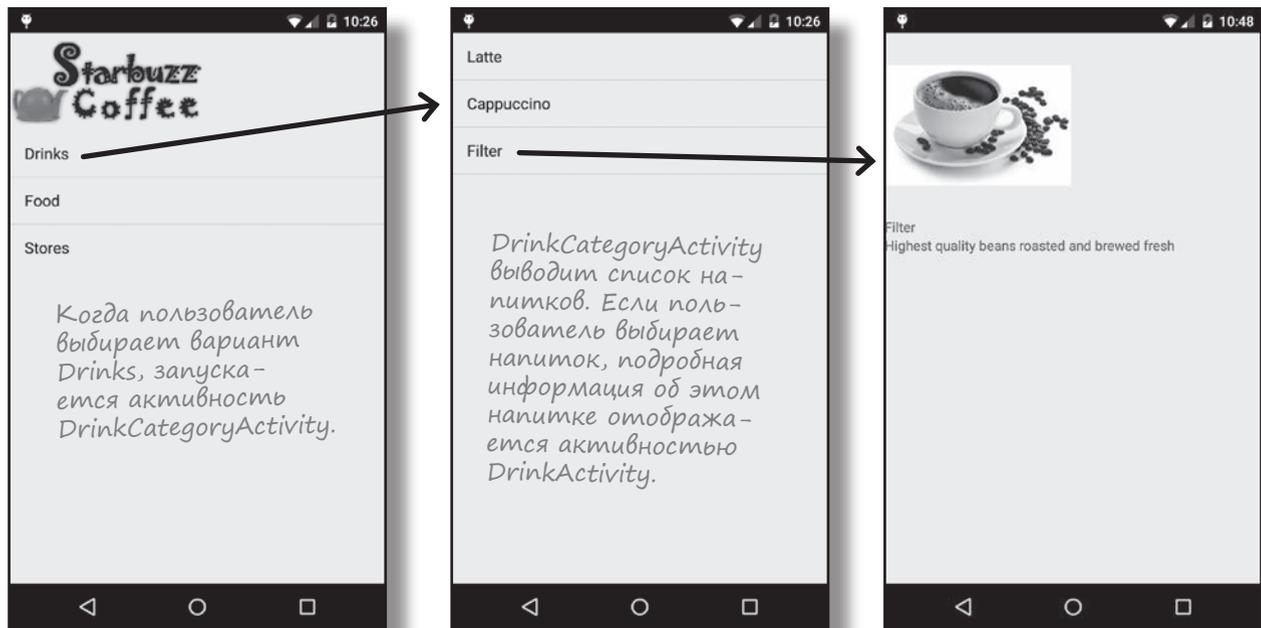
Активность категории выводит данные, относящиеся к одной категории



Добавление ресурсов
 TopLevelActivity
 DrinkCategoryActivity
 DrinkActivity

Как упоминалось ранее, `DrinkCategoryActivity` является примером активности категории. Такие активности предназначены для вывода данных, относящихся к определенной категории или разделу, — часто в виде списка. Затем активность используется для перехода к подробным описаниям отдельных вариантов.

В нашем приложении активность `DrinkCategoryActivity` используется для вывода списка напитков. Когда пользователь выбирает один из напитков в списке, на экране появляется подробная информация об этом напитке.

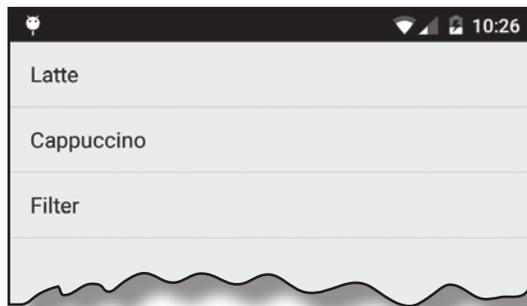


Для этого мы создадим активность, которая состоит из единственного спискового представления для вывода полного перечня напитков. Так как активность содержит только один список без каких-либо других компонентов графического интерфейса, мы создадим особую разновидность активностей: так называемую **списковую активность**. Что же это такое — списковая активность?

Списковая активность (ListActivity) содержит только список данных

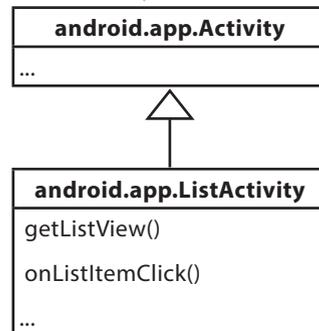
Списковая активность специализируется на работе со списком. Она автоматически связывается со списковым представлением, поэтому вам не придется создавать такое представление самостоятельно. Списковая активность выглядит примерно так:

Списковая активность содержит собственное списковое представление, так что вам не придется добавлять его самостоятельно. Впрочем, встроенный список все равно нужно будет заполнить данными; вскоре вы увидите, как это делается.



Добавление ресурсов
TopLevelActivity
DrinkCategoryActivity
DrinkActivity

ListActivity является subclassом Activity.



Использование списковой активности для вывода категорий данных обладает парой преимуществ:



Вам не придется строить макет самостоятельно.

Списковые активности определяют свой макет на программном уровне, поэтому вам не придется создавать или заниматься сопровождением разметки XML. Макет, генерируемый списковой активностью, содержит одно списковое представление. Для обращения к списковому представлению из кода активности используется метод `getListView()` списковой активности. Такое обращение необходимо для того, чтобы вы могли задать данные, которые должны выводиться в списковом представлении.



Вам не нужно реализовать собственного слушателя.

Класс `ListActivity` уже реализует слушателя событий, который обнаруживает щелчки на вариантах спискового представления. Вместо того, чтобы создавать собственного слушателя и привязывать его к списковому представлению, разработчику достаточно реализовать метод `onListItemClick()` списковой активности. При таком подходе вам будет проще организовать реакцию активности на выбор вариантов спискового представления. Обработка щелчков в действии будет продемонстрирована позднее, когда мы используем метод `onListItemClick()` для запуска другой активности.

Типичный способ применения активностей категорий — вывод одного спискового представления для перехода к подробным описаниям, так что списковые активности хорошо подходят для такой ситуации. Как же выглядит код использования списковой активности?

ListActivity — специализация Activity для работы с ListView. Имеет макет по умолчанию, в который входит компонент ListView.

Создание списковой активности

Ниже приведен базовый код создания списковой активности. Как видите, он очень похож на код создания обычной активности. Воспользуйтесь мастером New Activity для создания в проекте новой активности с именем `DrinkCategoryActivity`, после чего замените содержимое `DrinkCategoryActivity.java` приведенным ниже кодом:

```
package com.hfad.starbuzz;

import android.app.ListActivity;
import android.os.Bundle;

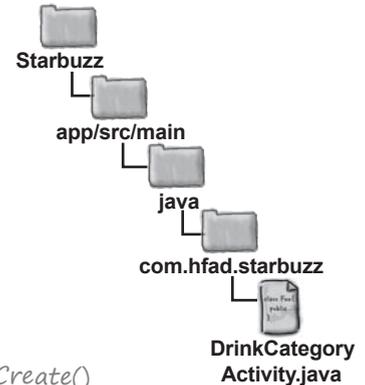
public class DrinkCategoryActivity extends ListActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }
}
```

Активность должна расширять класс `ListActivity` вместо `Activity`.

`ListActivity` наследует метод `onCreate()` от класса `Activity`. Вскоре мы напишем код этого метода.

Android Studio может автоматически сгенерировать файл макета. Мы не будем его использовать, потому что списковые активности определяют собственный макет.



Приведенный выше код создает базовую списковую активность с именем `DrinkCategoryActivity`. Так как класс представляет именно списковую активность, он должен расширять класс `ListActivity` вместо класса `Activity`.

Другое отличие заключается в том, что вам не нужно назначать макет, используемый списковой активностью, вызовом `setContentView()`. Дело в том, что списковые активности определяют свои макеты самостоятельно, поэтому вам это делать не придется — списковая активность сделает все за вас. Списковые активности, как и обычные, должны быть зарегистрированы в файле `AndroidManifest.xml`. Это необходимо для того, чтобы они могли использоваться в приложении. При создании активности Android Studio делает это за вас.

```
<application
... >
<activity
    android:name=".TopLevelActivity"
    android:label="@string/app_name"
...
</activity>
<activity
    android:name=".DrinkCategoryActivity"
    android:label="@string/title_activity_drink_category" >
</activity>
</application>
```

Первая активность, созданная нами.

А это новая активность. Каждая активность приложения должна быть представлена элементом в файле `AndroidManifest.xml`.



После создания списковой активности ее необходимо заполнить данными. Давайте посмотрим, как это делается.

android:entries подходит для статических массивов, хранящихся в strings.xml



Добавление ресурсов
 TopLevelActivity
 DrinkCategoryActivity
 DrinkActivity

При создании первой активности `TopLevelActivity` мы могли связать данные со списковым представлением при помощи атрибута `android:entries` в XML макета. Такое решение работало, потому что данные хранились в виде ресурса статического массива строк. Массив был описан в файле `strings.xml`, что позволяло легко сослаться на него с использованием синтаксиса

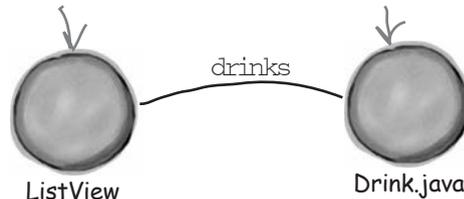
```
android:entries="@array/options"
```

где `options` — имя массива строк.

Атрибут `android:entries` подходит только для данных, представленных статическим массивом в `strings.xml`. А если в приложении используется другой способ хранения? Что, если данные хранятся в массиве, созданном на программном уровне в коде Java, или в базе данных? В этом случае атрибут `android:entries` не работает.

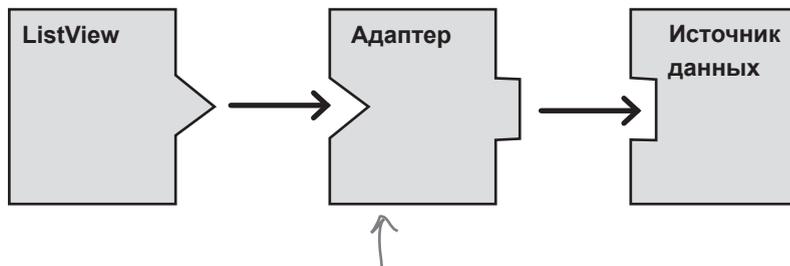
Если списковое представление необходимо связать с данными, хранящимися в чем-то отличном от ресурса массива строк, придется действовать иначе: необходимо написать код активности для привязки данных. В нашем примере списковое представление требуется связать с массивом `drinks` из класса `Drink`.

Списковое представление должно заполняться данными напитков. Данные напитков должны быть взяты из массива `drinks` в классе `Drink`.



Для нестатических данных используйте адаптер

Если данные спискового представления должны поступать из нестатического источника (например, из массива Java или базы данных), необходимо использовать **адаптер**. Адаптер играет роль моста между источником данных и списковым представлением:



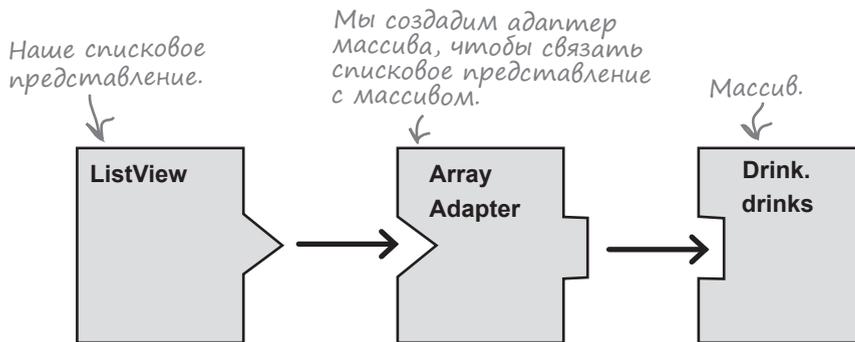
В нашем примере источником данных будет массив, но с таким же успехом можно использовать базу данных или веб-службу.

Адаптер заполняет пробел между списковым представлением и источником данных. Адаптеры позволяют списковым представлениям выводить информацию из самых разнообразных источников.

Существуют разные типы адаптеров. Сейчас мы займемся **адаптерами массивов**.

Связывание списковых представлений с массивами при помощи адаптера массива

Адаптер массива — разновидность адаптеров для связывания массивов с представлениями. Адаптер массива может использоваться с любым субклассом класса `AdapterView`; это означает, что он будет работать как со списковым представлением, так и с раскрывающимся списком. В нашем примере адаптер массива будет использоваться для вывода данных из массива `Drink.drinks` в списковом представлении.



Адаптер соединяет представление с источником данных. ArrayAdapter — разновидность адаптеров, предназначенная для работы с массивами.

Чтобы использовать адаптер массива, следует инициализировать его и присоединить к списковому представлению.

При инициализации адаптера массива вы сначала указываете тип данных массива, который вы хотите связать со списковым представлением. Затем адаптеру передаются три параметра: `Context` (обычно текущая активность), ресурс макета, который определяет, как должен отображаться каждый элемент из массива, и сам массив.

Приведенный ниже код создает адаптер массива для отображения данных из массива `Drink.drinks`:

```
ArrayAdapter<Drink> listAdapter = new ArrayAdapter<Drink>(
    this,
    android.R.layout.simple_list_item_1,
    Drink.drinks);
```

Текущая активность. Класс `Activity` является субклассом `Context`.

Массив содержит объекты `Drink`.

Встроенный ресурс макета. Он приказывает адаптеру массива отображать каждый элемент массива в виде надписи.

Затем адаптер массива связывается со списковым представлением при помощи метода `setAdapter()` класса `ListView`:

```
ListView listView = getListView();
listView.setAdapter(listAdapter);
```

Во внутренней реализации адаптер массива берет каждый элемент массива, преобразует его в `String` методом `toString()` и помещает каждый результат в надпись. Затем каждая надпись выводится в отдельной строке спискового представления.

Добавление адаптера массива в DrinkCategoryActivity

Мы изменим код *DrinkCategoryActivity.java* так, чтобы списковое представление использовало адаптер массива для получения данных напитков из класса *Drink*. Код будет включен в метод *onCreate()*, чтобы списковое представление заполнялось при создании активности.

Ниже приведен полный код активности (внесите изменения в свою версию кода и сохраните изменения):

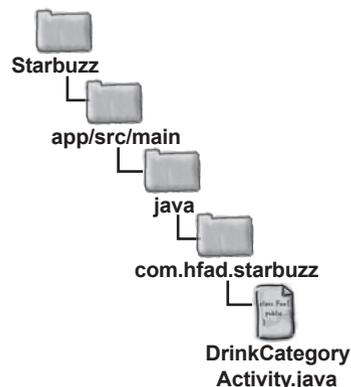
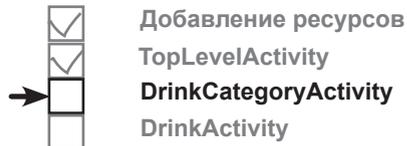
```
package com.hfad.starbuzz;

import android.app.ListActivity;
import android.os.Bundle;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.ListView;

public class DrinkCategoryActivity extends ListActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ListView listDrinks = getListView();
        ArrayAdapter<Drink> listAdapter = new ArrayAdapter<Drink>(
            this,
            android.R.layout.simple_list_item_1,
            Drink.drinks);
        listDrinks.setAdapter(listAdapter);
    }
}
```

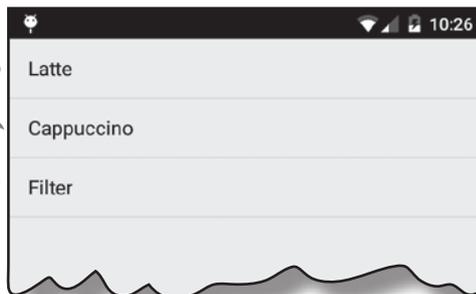
В коде используются эти внешние классы.



Списковое представление заполняется данными из массива drinks.

Вот и все, что необходимо сделать для того, чтобы в списковом представлении выводились напитки из класса *Drink*.

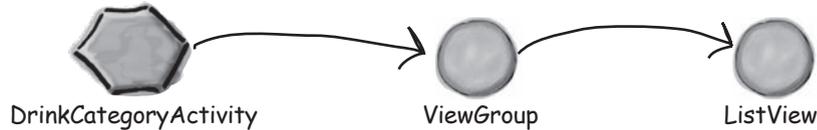
Напитки из массива Drink.drinks.



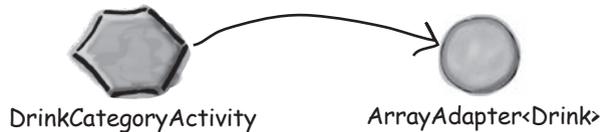
Что происходит при выполнении кода

- 1 Когда пользователь выбирает команду `Drinks`, открывается активность `DrinkCategoryActivity`.

Так как `DrinkCategoryActivity` является списковой активностью, она имеет макет по умолчанию с одним объектом `ListView`. Этот макет незаметно генерируется в коде Java, и он не определяется в разметке XML.



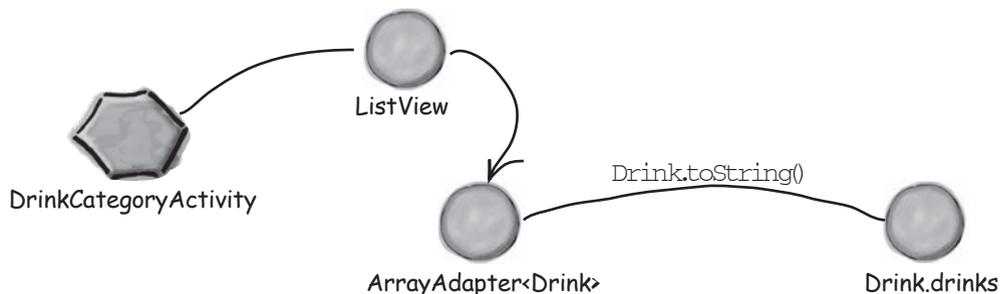
- 2 `DrinkCategoryActivity` создает `ArrayAdapter<Drink>` — адаптер массива для массивов, содержащих объекты `Drink`.



- 3 Источником данных адаптера массива является массив `drinks` из класса `Drink`. Для получения названия каждого напитка используется метод `Drink.toString()`.



- 4 `DrinkCategoryActivity` приказывает `ListView` использовать адаптер массива, вызывая метод `setAdapter()`. Списковое представление использует адаптер для вывода списка названий напитков.





Тест-драйв

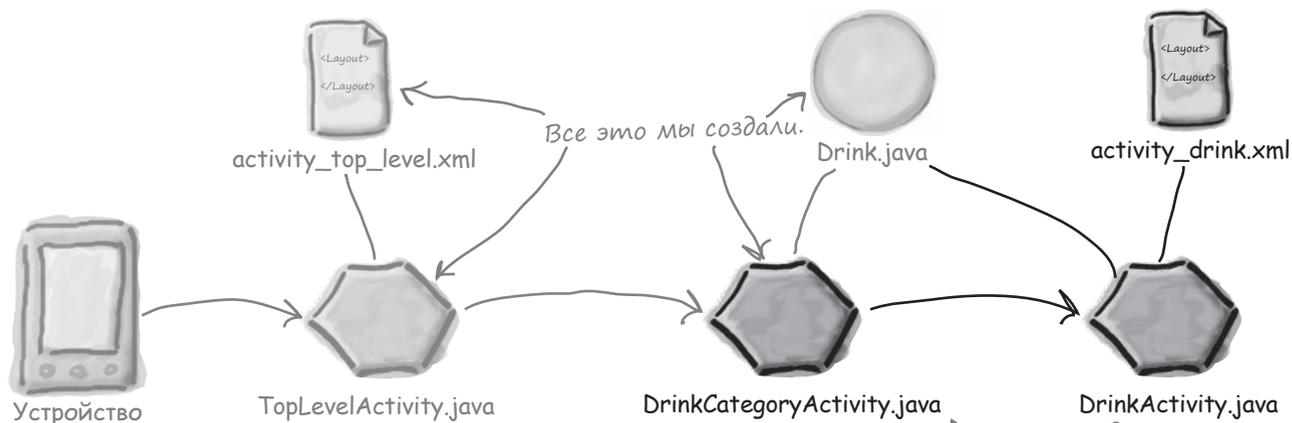
При запуске приложения, как и прежде, отображается активность `TopLevelActivity`. При выборе варианта `Drinks` открывается активность `DrinkCategoryActivity`. Она выводит названия всех напитков из класса `Java Drink`.

Щелкните в строке `Drinks`, чтобы увидеть список напитков.



Что было сделано

К настоящему моменту мы добавили в приложение класс `Drink.java`, а также создали активности `TopLevelActivity` и `DrinkCategoryActivity`.



Теперь нужно сделать так, чтобы активность `DrinkCategoryActivity` открывала `DrinkActivity` и передавала ей информацию о том, на каком напитке был сделан щелчок.

Следующий шаг — сделать так, чтобы активность `DrinkCategoryActivity` вызывала `DrinkActivity`.

У бассейна



Ваша **задача** — создать активность для привязки массива Java с названиями цветов к раскрывающемуся списку. Выловите фрагменты кода из бассейна и расставьте их в пропусках в коде активности. Каждый фрагмент может использоваться **только один** раз; использовать все фрагменты не обязательно.

...

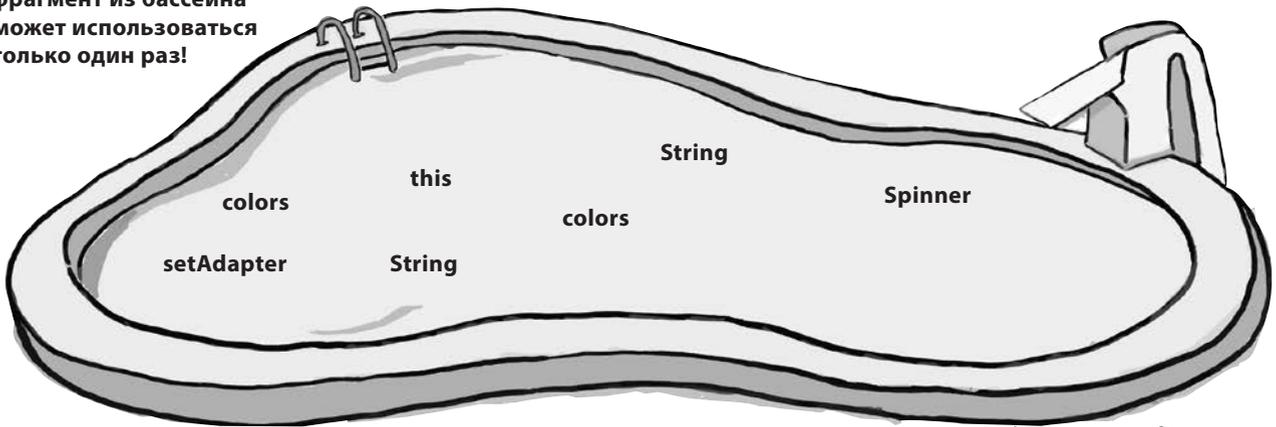
```
public class MainActivity extends Activity {

    String[] colors = new String[] {"Red", "Orange", "Yellow", "Green", "Blue"};

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Spinner spinner = (.....) findViewById(R.id.spinner);
        ArrayAdapter<.....> adapter = new ArrayAdapter<.....>(
            .....,
            android.R.layout.simple_spinner_item,
            colors);
        spinner.....(adapter);
    }
}
```

↑
Каждое значение из массива выводится в отдельной строке раскрывающегося списка.

Внимание: каждый фрагмент из бассейна может использоваться только один раз!



→ Ответы на с. 299.

Как мы обрабатывали щелчку в TopLevelActivity



Добавление ресурсов
 TopLevelActivity
 DrinkCategoryActivity
 DrinkActivity

Ранее в этой главе мы уже заставляли TopLevelActivity реагировать на щелчки пользователя на списковом представлении. Для этого мы создавали объект onItemClickListener, реализовывали его метод onItemClick() и назначали его списковому представлению:

```

AdapterView.OnItemClickListener onItemClickListener = new AdapterView.OnItemClickListener() {
    public void onItemClick(AdapterView<?> listView, ← Списковое представление.
        View itemView,
        int position,
        long id) {
        // Действия при щелчке на варианте
    }
};
ListView listView = (ListView) findViewById(R.id.list_options);
listView.setOnItemClickListener(itemClickListener); ← Назначить слушателя
                                                    для спискового представления.
    
```

Представление варианта, на котором был сделан щелчок, его позиция в списке и идентификатор записи в используемых данных.

Нам пришлось назначать слушателя подобным образом, потому что списковые представления изначально не запрограммированы на обработку щелчков (в отличие, скажем, от кнопок).

Как же заставить DrinkCategoryActivity обрабатывать щелчки?

ListActivity содержит реализацию слушателя щелчков по умолчанию

Между TopLevelActivity и DrinkCategoryActivity существует серьезное отличие. TopLevelActivity является обычным объектом Activity, тогда как DrinkCategoryActivity представляет собой ListActivity — особую разновидность активности, предназначенную для работы со списковыми представлениями.

Это обстоятельство играет важную роль в обработке щелчков. Принципиальное отличие между Activity и ListActivity заключается в том, что класс ListActivity уже реализует слушателя событий щелчков. Вместо того, чтобы создавать собственного слушателя событий, при использовании списковой активности достаточно реализовать метод onListItemClick().

```

public void onListItemClick(ListView listView, ← Те же аргументы, что у приведенного
    View itemView,                               выше метода onItemClick(): списковое
    int position,                                представление; представление вари-
    long id) {                                   анта, на котором был сделан щелчок;
    //Что-то происходит                         его позиция в списке; идентификатор
    }
    
```

Передача данных активности с использованием метода onItemClick() класса ListActivity

При использовании списковой активности для вывода категорий метод onItemClick() обычно используется для запуска другой активности, которая выводит подробное описание варианта, выбранного пользователем. Для этого разработчик создает интент, открывающий вторую активность. Идентификатор варианта, выбранного пользователем, включается в дополнительную информацию, чтобы вторая активность могла использовать его при запуске.

В нашем случае нужно запустить активность DrinkActivity и передать ей идентификатор выбранного напитка. DrinkActivity использует эту информацию для вывода подробного описания напитка. Код выглядит так:

```
public void onItemClick(ListView listView,
                        View itemView,
                        int position,
                        long id) {
    Intent intent = new Intent(DrinkCategoryActivity.this, DrinkActivity.class);
    intent.putExtra(DrinkActivity.EXTRA_DRINKNO, (int) id);
    startActivity(intent);
}
```



← Вызывается при щелчке на одном из вариантов в списке.

Активность DrinkCategoryActivity должна запускать DrinkActivity.

← Добавить идентификатор варианта, на котором был сделан щелчок, в интент. Он определяет индекс напитка в массиве drinks.

Имя дополнительной информации в интенте обозначается константой, чтобы DrinkCategoryActivity и DrinkActivity заведомо использовали одну строку. Константа добавляется в DrinkActivity при создании активности.

Передача идентификатора варианта, на котором был сделан щелчок, — практика весьма распространенная, так как передаваемое значение одновременно является идентификатором в используемом наборе данных. Если набор данных хранится в массиве, то идентификатор совпадает с индексом элемента массива. Если информация хранится в базе данных, то идентификатор является индексом записи в таблице. При подобном способе передачи идентификатора второй активности будет проще получить подробную информацию о данных, а затем вывести ее.

Вот и все, что необходимо сделать для того, чтобы активность DrinkCategoryActivity запускала активность DrinkActivity и сообщала ей, какой напиток был выбран. Полный код активности приведен на следующей странице.

Полный код DrinkCategoryActivity

Ниже приведен полный код `DrinkCategoryActivity.java` (добавьте новый метод в свой код и сохраните изменения):

```
package com.hfad.starbuzz;

import android.app.ListActivity;
import android.os.Bundle;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.view.View;
import android.content.Intent;

public class DrinkCategoryActivity extends ListActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ListView listDrinks = getListView();
        ArrayAdapter<Drink> listAdapter = new ArrayAdapter<Drink>(
            this,
            android.R.layout.simple_list_item_1,
            Drink.drinks);
        listDrinks.setAdapter(listAdapter);
    }

    @Override
    public void onItemClick(ListView listView,
                            View itemView,
                            int position,
                            long id) {
        Intent intent = new Intent(DrinkCategoryActivity.this, DrinkActivity.class);
        intent.putExtra(DrinkActivity.EXTRA_DRINKNO, (int) id);
        startActivity(intent);
    }
}
```

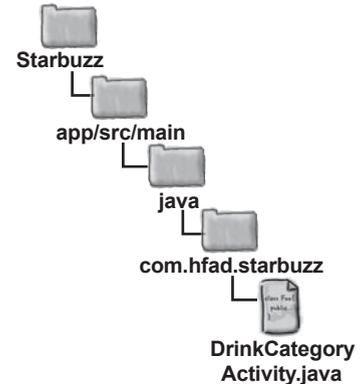
В коде используются эти внешние классы.

Метод `onItemClickListener()` реализуется так, чтобы при щелчке на варианте списка запускалась активность `DrinkActivity`.

Даже если Android Studio сообщит, что активность `DrinkActivity` не существует, не беспокойтесь — сейчас мы ее создадим.



Добавление ресурсов
 TopLevelActivity
 DrinkCategoryActivity
 DrinkActivity



Активность детализации выводит данные из одной записи



Добавление ресурсов
 TopLevelActivity
 DrinkCategoryActivity
 DrinkActivity

Как упоминалось ранее, `DrinkActivity` является активностью детализации. Такие активности выводят подробную информацию о конкретной записи, и обычно переход к ним осуществляется из активностей категорий.

Мы воспользуемся `DrinkActivity` для вывода подробной информации о напитке, выбранном пользователем. Класс `Drink` включает название напитка, его описание и идентификатор ресурса изображения; все эти данные будут выводиться в макете. Для изображения напитка будет создано графическое представление, а для названия и описания — надписи. Ниже приведена разметка макета. Добавьте в проект новую активность с именем `DrinkActivity` и макет с именем `activity_drink`, после чего замените содержимое `activity_drink.xml` следующей разметкой:

← Не забудьте создать новую активность.

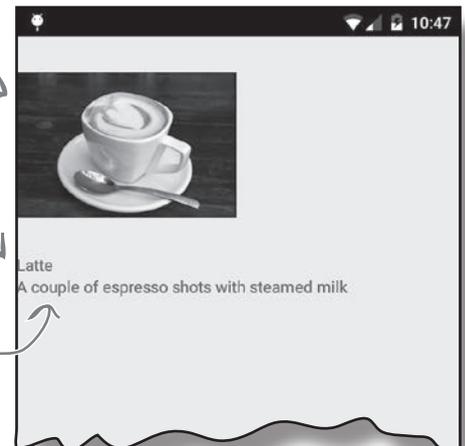
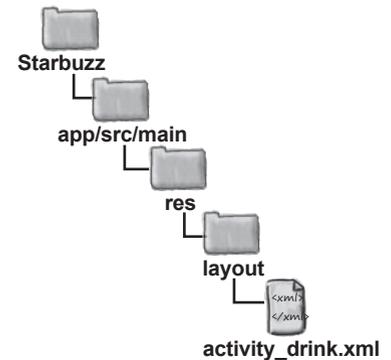
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.hfad.starbuzz.DrinkActivity" >

    <ImageView
        android:id="@+id/photo"
        android:layout_width="190dp"
        android:layout_height="190dp" />

    <TextView
        android:id="@+id/name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

    <TextView
        android:id="@+id/description"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

</LinearLayout>
```



После того как макет активности детализации будет создан, можно переходить к заполнению его представлений.



Добавление ресурсов
 TopLevelActivity
 DrinkCategoryActivity
 DrinkActivity

Чтение данных из интента

Как было показано ранее, когда активность категории используется для запуска активности детализации, варианты активности категории реагируют на щелчки. При выборе варианта создается интент для запуска активности детализации. Идентификатор варианта, выбранного пользователем, передается в составе дополнительной информации интента. При запуске активность детализации читает из интента дополнительную информацию и использует ее для заполнения своих представлений. В нашем примере данные из интента, запустившего `DrinkActivity`, используются для получения подробной информации о напитке, выбранном пользователем.

Создавая `DrinkCategoryActivity`, мы включили идентификатор напитка, выбранного пользователем, как дополнительную информацию в интент. Ему присваивается метка `DrinkActivity.EXTRA_DRINKNO`, которую необходимо определить как константу в `DrinkActivity`:

```
public static final String EXTRA_DRINKNO = "drinkNo";
```

В главе 3 было показано, как получить интент, запустивший активность, методом `getIntent()`. Если интент содержит дополнительную информацию, для чтения такой информации используются методы `get*()` интента. Следующий код читает значение `EXTRA_DRINKNO` из интента, запустившего `DrinkActivity`:

```
int drinkNo = (Integer) getIntent().getExtras().get(EXTRA_DRINKNO);
```

Информация из интента используется для получения данных, которые должны выводиться в подробном описании.

В нашем примере значение `drinkNo` используется для получения подробной информации о напитке, выбранном пользователем. `drinkNo` содержит идентификатор напитка, то есть его индекс в массиве `drinks`. Таким образом, для получения напитка, на котором щелкнул пользователь, можно воспользоваться следующей командой:

```
Drink drink = Drink.drinks[drinkNo];
```

Полученный объект `Drink` содержит всю информацию, необходимую для обновления атрибутов представлений в активности:



drink

```
name="Latte"  

description="A couple of espresso shots with steamed milk"  

imageResourceId=R.drawable.latte
```

Обновление представлений

При обновлении представлений в активности детализации необходимо позаботиться о том, чтобы отображаемые в них значения соответствовали данным, полученным из интента.

Наша активность детализации содержит две надписи и графическое представление. Нужно занести в каждое из этих представлений информацию, соответствующую данным напитка.



name
description
imageResourceId

drink



Развлечения с Магнитами

Удастся ли вам расставить магниты так, чтобы представления активности DrinkActivity заполнялись правильными данными?

...

```
//Получить напиток из данных интента
int drinkNo = (Integer) getIntent().getExtras().get(EXTRA_DRINKNO);
Drink drink = Drink.drinks[drinkNo];
```

```
//Заполнение изображения напитка
ImageView photo = (ImageView) findViewById(R.id.photo);
```

```
photo. ....(drink.getImageResourceId());
```

```
photo. ....(drink.getName());
```

```
//Заполнение названия напитка
TextView name = (TextView) findViewById(R.id.name);
```

```
name. ....(drink.getName());
```

```
//Заполнение описания напитка
TextView description = (TextView) findViewById(R.id.description);
```

```
description. ....(drink.getDescription());
```

...





Развлечения с магнитами. Решение

Удастся ли вам расставить магниты так, чтобы представления активности DrinkActivity заполнялись правильными данными?

```
...  
//Получить напиток из данных интента  
int drinkNo = (Integer) getIntent().getExtras().get(EXTRA_DRINKNO);  
Drink drink = Drink.drinks[drinkNo];
```

```
//Заполнение изображения напитка  
ImageView photo = (ImageView) findViewById(R.id.photo);  
photo.setImageResource..(drink.getImageResourceId());
```

Источник данных графического поля назначается вызовом setImageResource().

```
photo.setContentDescription..(drink.getName());
```

Необходимо для того, чтобы повысить уровень доступности приложения.

```
//Заполнение названия напитка  
TextView name = (TextView) findViewById(R.id.name);
```

```
name.setText..(drink.getName());
```

Метод setText() используется для назначения текста надписи.

```
//Заполнение описания напитка  
TextView description = (TextView) findViewById(R.id.description);
```

```
description.setText..(drink.getDescription());  
...
```

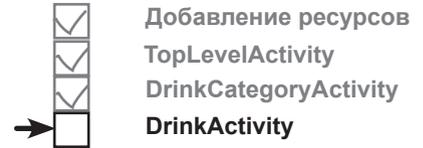
Эти магниты не понадобились.

setContent

setImageResourceId

Kog DrinkActivity

Перед вами код *DrinkActivity.java* (замените код, сгенерированный мастером, и сохраните изменения):



```
package com.hfad.starbuzz;

import android.app.Activity;
import android.os.Bundle;
import android.widget.ImageView;
import android.widget.TextView;

public class DrinkActivity extends Activity {
```

```
    public static final String EXTRA_DRINKNO = "drinkNo";
```

*↑
Добавить константу EXTRA_DRINKNO.*

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_drink);
```

//Получить напиток из данных интента

```
    int drinkNo = (Integer) getIntent().getExtras().get(EXTRA_DRINKNO);
    Drink drink = Drink.drinks[drinkNo];
```

← Использовать drinkNo для получения подробной информации о напитке, выбранном пользователем.

//Заполнение изображения напитка

```
    ImageView photo = (ImageView) findViewById(R.id.photo);
    photo.setImageResource(drink.getImageResourceId());
    photo.setContentDescription(drink.getName());
```

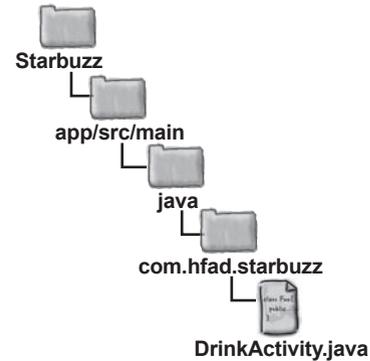
//Заполнение названия напитка

```
    TextView name = (TextView) findViewById(R.id.name);
    name.setText(drink.getName());
```

← Заполнить представления данными напитков.

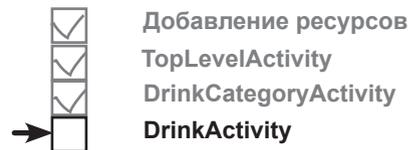
//Заполнение описания напитка

```
    TextView description = (TextView) findViewById(R.id.description);
    description.setText(drink.getDescription());
```

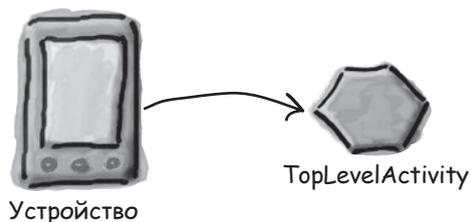


```
    }
}
```

Что происходит при запуске приложения



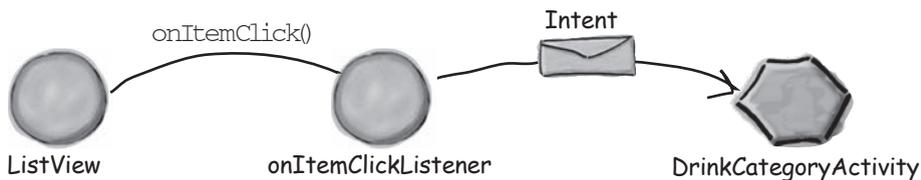
- 1 При запуске приложения открывается активность `TopLevelActivity`.



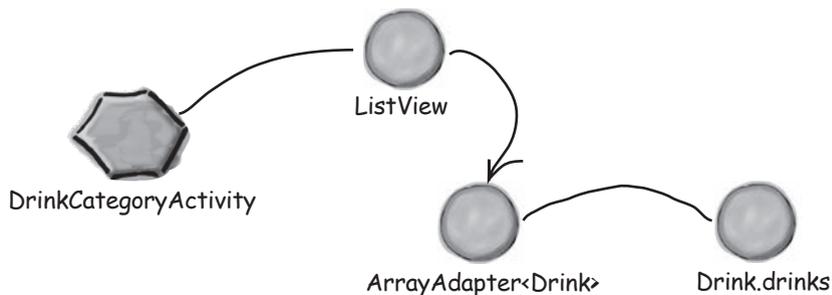
- 2 Метод `onCreate()` активности `TopLevelActivity` создает объект `onItemClickListener` и связывает его с компонентом `ListView` активности.



- 3 Когда пользователь щелкает на варианте спискового представления, вызывается метод `onItemClick()` объекта `onItemClickListener`. Если щелчок был сделан на варианте `Drinks`, `onItemClickListener` создает интент для запуска `DrinkCategoryActivity`.

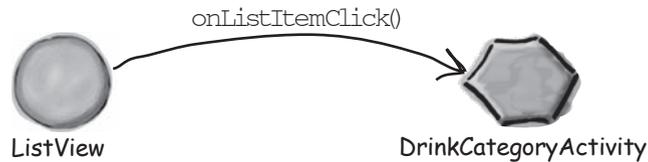


- 4 `DrinkCategoryActivity` наследует от `ListActivity`. Списковое представление `DrinkCategoryActivity` использует `ArrayAdapter<Drink>` для вывода списка названий напитков.

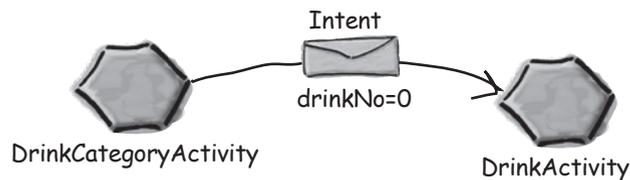


История продолжается

- 5 Когда пользователь выбирает напиток в `ListView`, вызывается метод `onListItemClick()`.

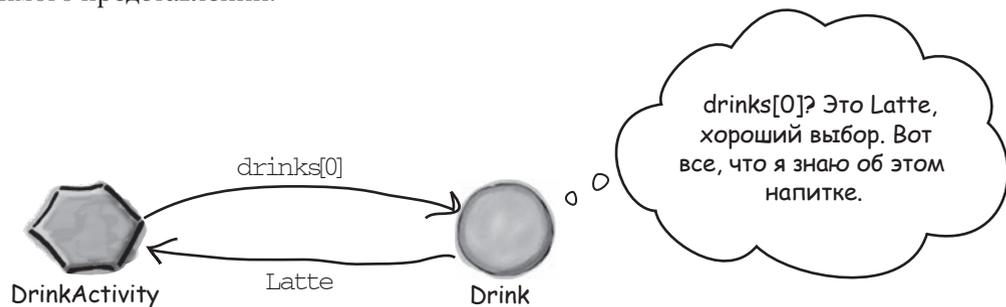


- 6 Метод `onListItemClick()` класса `DrinkCategoryActivity` создает интент для запуска `DrinkActivity`, передавая номер напитка в дополнительной информации.



- 7 Запускается активность `DrinkActivity`.

Активность читает номер напитка из интента и получает подробную информацию о правильном напитке из класса `Drink`. Информация используется для обновления содержимого представлений.





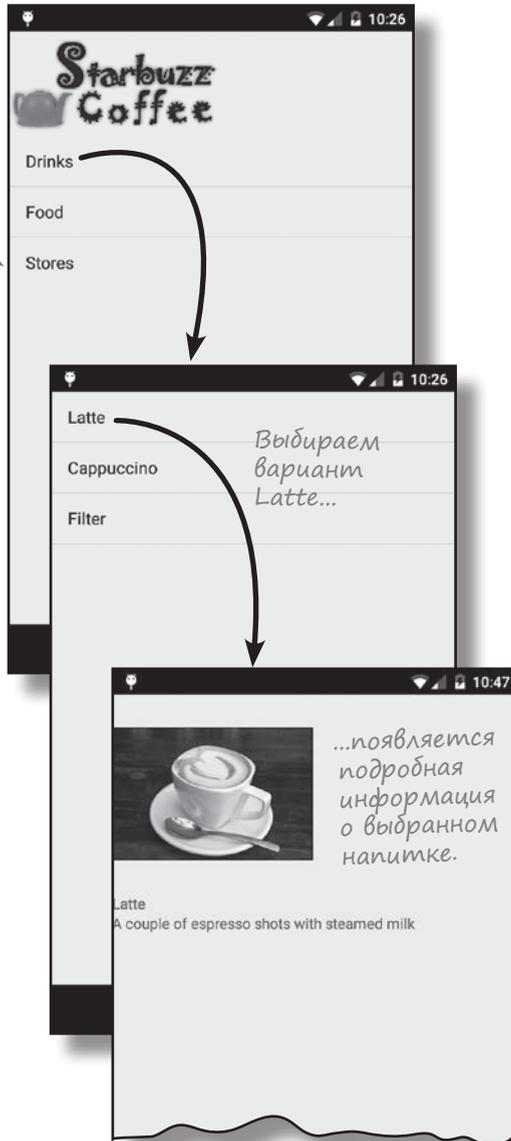
Тест-драйв

При запуске приложения, как и прежде, отображается активность `TopLevelActivity`.

Мы реализовали часть приложения, относящуюся к напиткам. При выборе других команд ничего не происходит.

Если выбрать команду `Drinks`, запускается активность `DrinkCategoryActivity`. Она выводит перечень всех напитков из класса `Java Drink`.

Если выбрать один из напитков в списке, запускается активность `DrinkActivity`, и на экране выводится подробная информация о выбранном напитке.



Пример этих трех активностей показывает, как разделить приложение на активности верхнего уровня, активности категорий и активности детализации/редактирования. Позднее мы еще вернемся к приложению Starbuzz и покажем, как организовать чтение информации из базы данных.

У бассейна. Решение



Ваша **задача** — создать активность для привязки массива Java с названиями цветов к раскрываемому списку. Выловите фрагменты кода из бассейна и расставьте их в пропусках в коде активности. Каждый фрагмент может использоваться **только один** раз; использовать все фрагменты не обязательно.

...

```
public class MainActivity extends Activity {

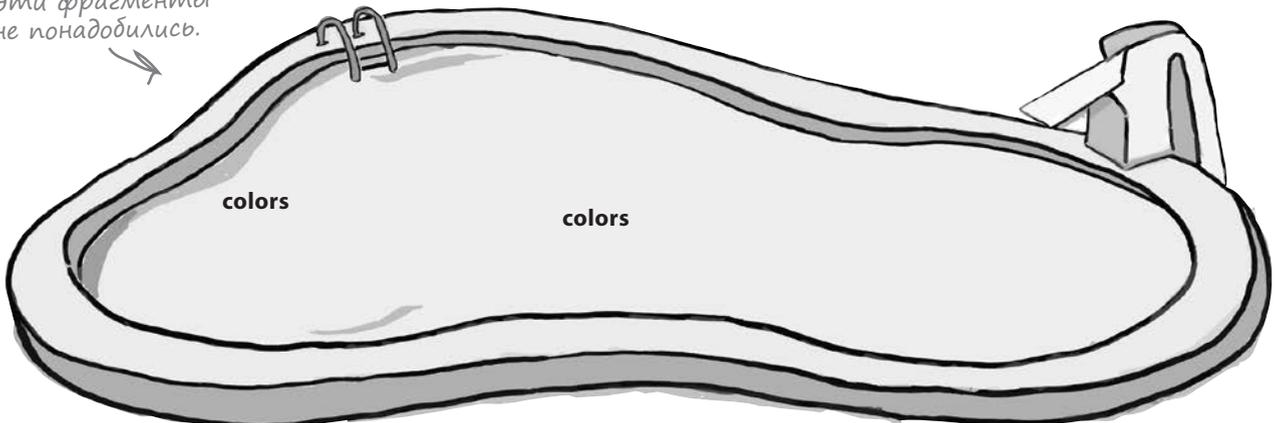
    String[] colors = new String[] {"Red", "Orange", "Yellow", "Green", "Blue"};

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Spinner spinner = ( Spinner findViewById(R.id.spinner);
        ArrayAdapter< String > adapter = new ArrayAdapter< String >
            this,
            android.R.layout.simple_spinner_item,
            colors);
        spinner. setAdapter(adapter);
    }
}
```

Используется массив с элементами String.

↑
Вызов `setAdapter()` заставляет раскрывающийся список использовать адаптер массива.

Эти фрагменты не понадобились.





Ваш инструментарий Android

Глава 6 осталась позади, а ваш инструментарий пополнился списковыми представлениями и планированием структуры приложения.

Весь код для этой главы можно загрузить по адресу <https://tinyurl.com/HeadFirstAndroid>.

КЛЮЧЕВЫЕ МОМЕНТЫ



- Разделите свои идеи по поводу активностей приложения на активности верхнего уровня, активности категорий и активности детализации/редактирования. Используйте активности категорий для перехода от активностей верхнего уровня к активностям детализации/редактирования.
- Ресурсы изображений размещаются в одной или нескольких папках *drawable**. Для обращения к ним в макете используется синтаксис вида `@drawable/имя_изображения`. В коде активности используется синтаксис `R.drawable.имя_изображения`.
- Компонент `ImageView` содержит графическое изображение. Для добавления его в макет используется элемент `<ImageView>`. Источник данных задается атрибутом `android:src`, а атрибут `android:contentDescription` задает текстовое описание, которое улучшает доступность приложения. Эквивалентные методы в коде Java — `setImageResource()` и `setContentDescription()`.
- Компонент `ListView` выводит набор вариантов в виде списка. Добавляется в макет элементом `<ListView>`.
- Атрибут `android:entries` в макете используется для заполнения спискового представления данными из массива, определенного в файле `strings.xml`.
- Класс `ListActivity` представляет активность, содержащую встроенное списковое представление `ListView`. Для получения ссылки на `ListView` используется метод `getListView()`.
- Активность `ListActivity` использует собственный макет по умолчанию, но вы можете заменить его своим макетом.
- Адаптер связывает `AdapterView` с источником данных. И списковые представления, и раскрывающиеся списки являются специализациями `AdapterView`.
- `ArrayAdapter` — адаптер для работы с массивами.
- Для обработки событий щелчков на кнопках в разметке макета используется атрибут `android:onClick`.
- Для обработки событий щелчков на вариантах спискового представления в `ListActivity` реализуется метод `onListItemClick()`.
- Для обработки событий щелчков на любых других компонентах необходимо создать слушателя и написать реализацию события щелчка.

7 фрагменты

Модульная структура

Одна и та же работа —
только в разных местах...
Получается, я — фрагмент?



Вы уже умеете создавать приложения, которые работают одинаково независимо от устройства, на котором они запускаются. Но что, если ваше приложение должно выглядеть и вести себя по-разному в зависимости от того, где оно запущено — на телефоне или планшете? В этой главе мы покажем, как в приложении выбрать наиболее подходящий макет по размерам экрана устройства. Также вы познакомитесь с фрагментами — механизмом создания модульных программных компонентов, которые могут повторно использоваться разными активностями.

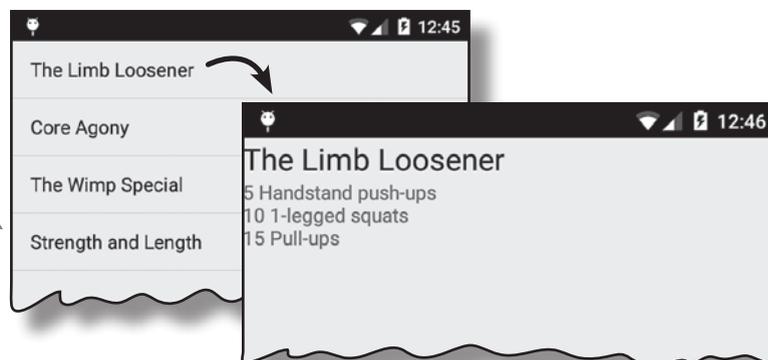
Ваше приложение должно хорошо смотреться на всех устройствах

Одна из самых замечательных особенностей программирования для Android — то, что одно и то же приложение может запускаться на устройствах с разными экранами и процессорами и будет работать на них одинаково. Но это вовсе не означает, что оно будет на них одинаково *выглядеть*.

На телефоне:

Взгляните на это приложение для телефона. Оно выводит список комплексов физических упражнений; если щелкнуть на одном из комплексов, на экране появляется подробное описание.

Если щелкнуть на одном из вариантов списка, запускается вторая активность.



На планшете:

На устройствах с большим экраном (например, на планшетах) свободного места гораздо больше. В такой ситуации будет лучше, если вся информация будет отображаться на одном экране. На планшете список занимает только часть экрана, а если щелкнуть на одном из вариантов, подробно отображаются справа.



На планшете свободного места намного больше, и это место можно использовать более эффективно.

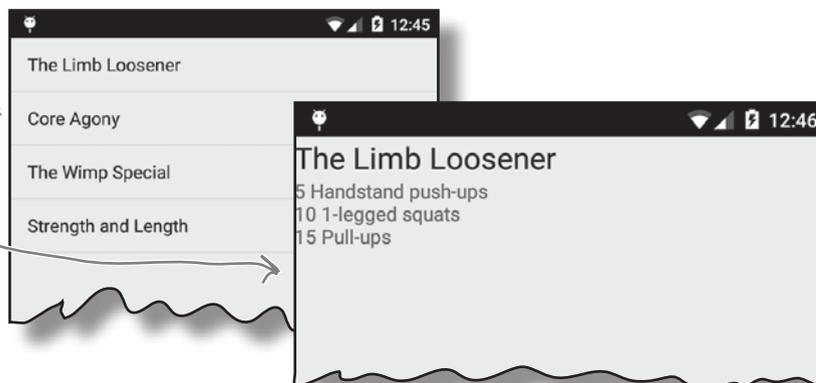
Чтобы интерфейсы приложения на телефоне и планшете отличались друг от друга, можно определить разные макеты для больших и малых устройств.

Поведение приложения тоже может зависеть от устройства

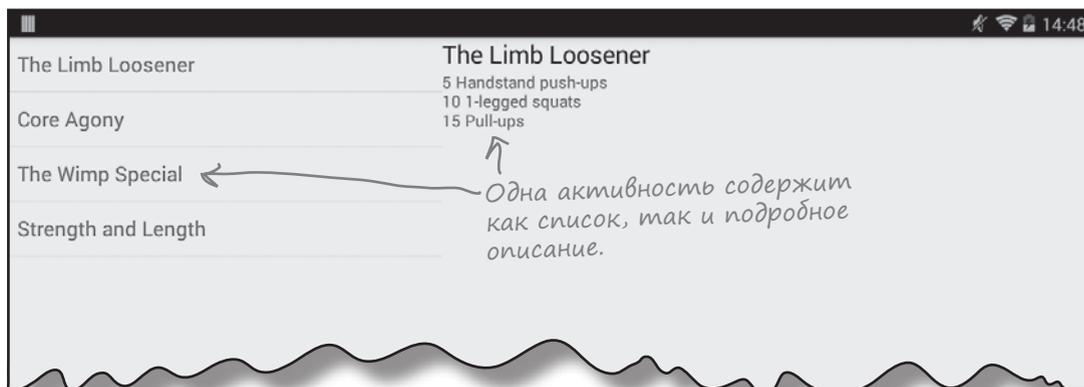
Однако определить разные макеты для разных устройств недостаточно. Чтобы приложение работало по-разному в зависимости от устройства, наряду с разными макетами должен выполняться *разный код Java*. Например, в этом приложении необходимо предоставить **одну активность для планшетов** и **две активности для телефонов**.

На телефоне:

Здесь используются две активности: для списка и для детализации.



На планшете:



Одна активность содержит как список, так и подробное описание.

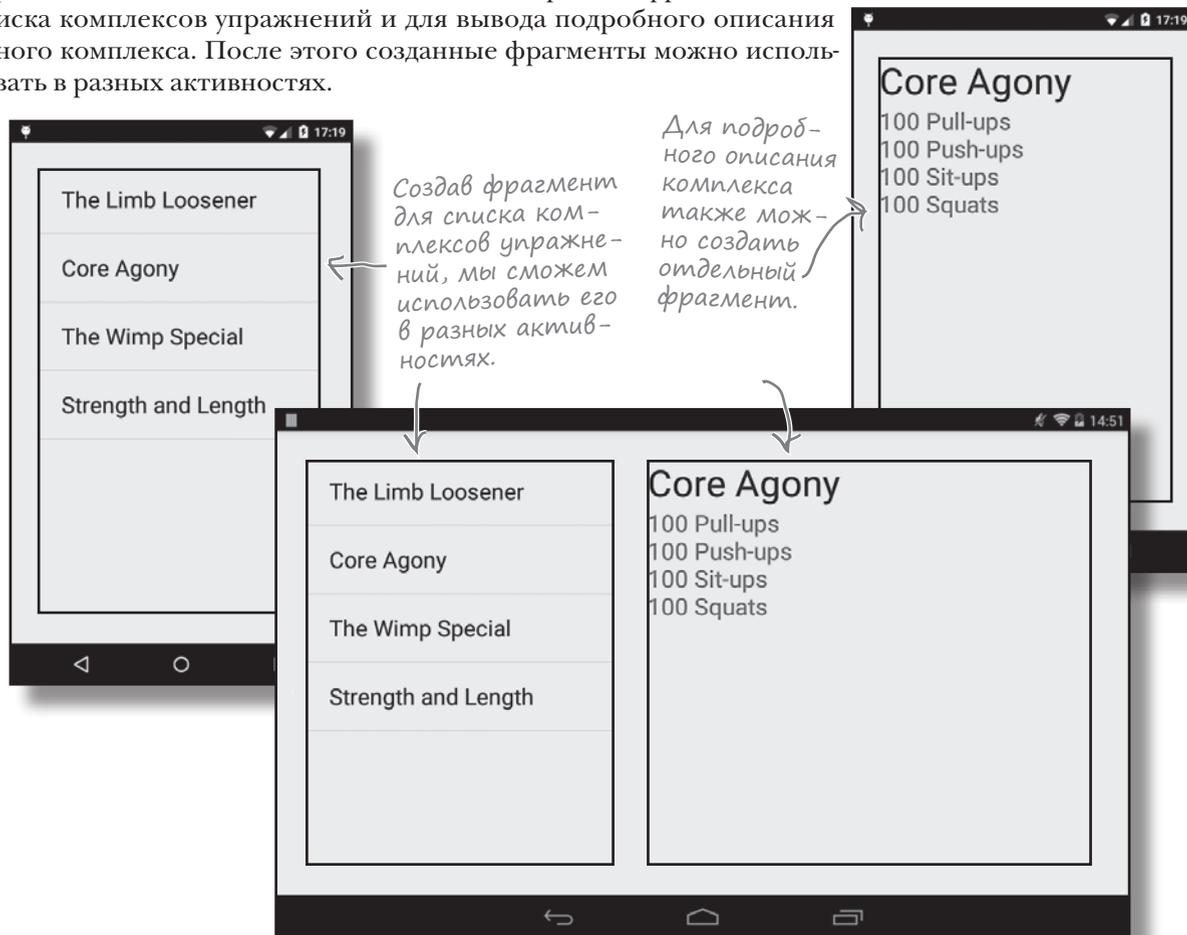
Но это может привести к дублированию кода

Второй активности, которая работает только на телефонах, потребуется вставить подробное описание в макет. Однако этот же код также должен присутствовать и в основной активности при выполнении приложения на планшете. *Один код должен выполняться в нескольких активностях.*

Вместо того, чтобы дублировать код в двух активностях, следует использовать **фрагменты** (fragments). Что же собой представляет фрагмент?

Фрагменты дают возможность повторно использовать код

Фрагменты — нечто вроде компонентов, предназначенных для повторного использования, или вторичных активностей. Фрагмент управляет частью экранного пространства и может использоваться на разных экранах. Это означает, что мы можем создать разные фрагменты для списка комплексов упражнений и для вывода подробного описания одного комплекса. После этого созданные фрагменты можно использовать в разных активностях.



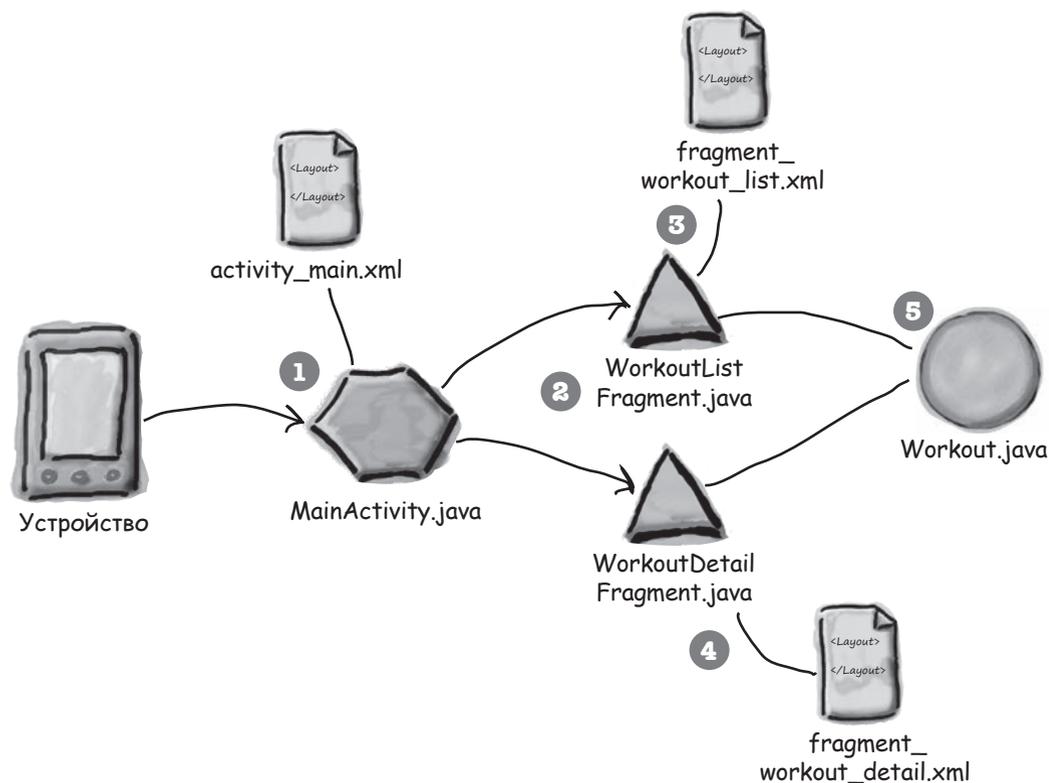
У фрагмента есть макет

Фрагмент, как и активность, связывается с макетом. Если внимательно подойти к его проектированию, для управления всеми аспектами интерфейса может использоваться код Java. Если код фрагмента содержит все необходимое для управления его макетом, вероятность того, что фрагмент можно будет повторно использовать в других частях приложения, значительно возрастает. Процесс создания и использования фрагментов будет продемонстрирован на примере приложения Workout.

Структура приложения Workout

В основном в этой главе мы займемся построением версии приложения, которая отображает два фрагмента рядом друг с другом в одной активности. Ниже кратко разобрана структура приложения и функциональность его частей.

- 1 При запуске приложение открывает активность **MainActivity**. Активность использует макет *activity_main.xml*.
- 2 Активность использует два фрагмента, **WorkoutListFragment** и **WorkoutDetailFragment**.
- 3 Фрагмент **WorkoutListFragment** отображает список комплексов упражнений. Он использует макет *fragment_workout_list.xml*.
- 4 Фрагмент **WorkoutDetailFragment** отображает подробное описание одного комплекса. Он использует макет *fragment_workout_detail.xml*.
- 5 Оба фрагмента получают свои данные из **Workout.java**. *Workout.java* содержит массив с объектами **Workout**.



Последовательность действий

Процесс построения приложения состоит из нескольких шагов:

1

Создание фрагментов.

Мы создадим два фрагмента: `WorkoutListFragment` используется для вывода списка комплексов упражнений, а `WorkoutDetailFragment` — для вывода подробного описания конкретного комплекса. Оба фрагмента будут отображаться в одной активности. Кроме того, мы добавим класс Java `Workout`, из которого фрагменты будут получать свои данные.



2

Связывание фрагментов.

Когда пользователь выбирает комплекс упражнений в `WorkoutListFragment`, подробное описание этого комплекса должно появиться в `WorkoutDetailFragment`.



3

Создание макетов для устройств.

Наконец, мы изменим приложение так, чтобы оно по-разному выглядело и работало в зависимости от типа устройства, на котором оно выполняется. Если приложение запущено на устройстве с большим экраном, то фрагменты будут размещаться рядом друг с другом. На устройствах с малыми экранами фрагменты будут находиться в разных активностях.



Создание проекта

Проект приложения создается точно так же, как это делалось в предыдущих главах.

Создайте новый проект Android с пустой активностью для приложения с именем “Workout” и именем пакета `com.hfad.workout`. Минимальный уровень SDK должен быть *не ниже* API 17, так как в следующей главе приложение будет использоваться для демонстрации возможностей, требующих API 17 и выше. Чтобы ваш код не отличался от нашего, присвойте активности имя “`MainActivity`”, а макету — имя “`activity_main`”.



Создание фрагментов
Связывание фрагментов
Макеты для устройств

Класс Workout



Создание фрагментов
Связывание фрагментов
Макеты для устройств

Начнем с добавления класса `Workout` в приложение. `Workout.java` — обычный файл класса Java, из которого приложение получает информацию о комплексах упражнений. Класс определяет массив из четырех элементов; каждый элемент содержит название и описание комплекса. Добавьте класс в пакет `com.hfad.workout` в папке `app/src/main/java` вашего проекта, присвойте ему имя `Workout` и сохраните изменения.

```
package com.hfad.workout;

public class Workout {
    private String name;
    private String description;

    public static final Workout[] workouts = {
        new Workout("The Limb Loosener",
            "5 Handstand push-ups\n10 1-legged squats\n15 Pull-ups"),
        new Workout("Core Agony",
            "100 Pull-ups\n100 Push-ups\n100 Sit-ups\n100 Squats"),
        new Workout("The Wimp Special",
            "5 Pull-ups\n10 Push-ups\n15 Squats"),
        new Workout("Strength and Length",
            "500 meter run\n21 x 1.5 pood kettleball swing\n21 x pull-ups")
    };

    // В объекте Workout хранится имя и описание
    private Workout(String name, String description) {
        this.name = name;
        this.description = description;
    }

    public String getDescription() {
        return description;
    }

    public String getName() {
        return name;
    }

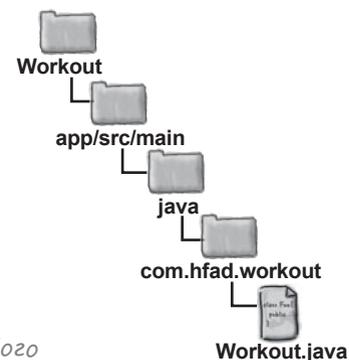
    public String toString() {
        return this.name;
    }
}
```

Каждый объект `Workout` содержит поля названия (`name`) и описания (`description`) приложения.

`workouts` — массив из четырех объектов `Workouts`.

Get-методы для чтения приватных переменных.

В качестве строкового представления объекта `Workout` используется его имя.



Данные будут использоваться фрагментом `WorkoutDetailFragment`. Сейчас мы создадим этот фрагмент.

Добавление фрагмента в проект

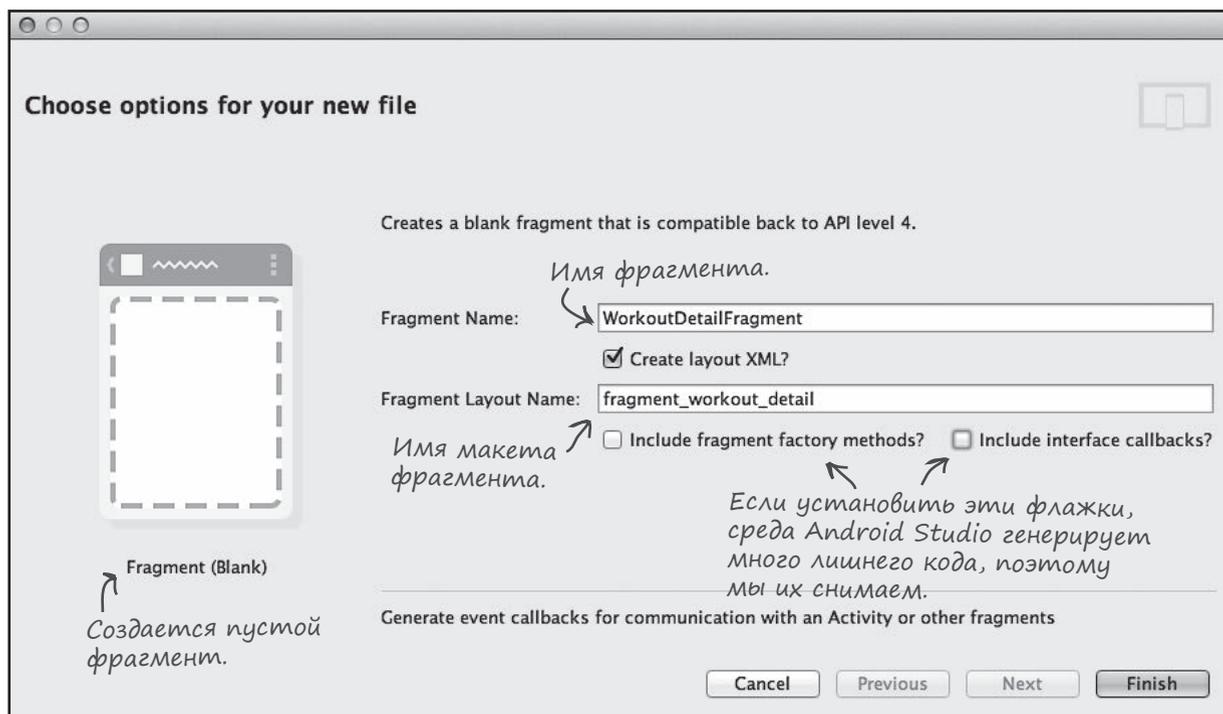
Сейчас мы добавим в проект новый фрагмент с именем `WorkoutDetailFragment`, предназначенный для вывода подробной информации об одном комплексе упражнений. Новые фрагменты добавляются примерно так же, как и новые активности: в Android Studio выберите команду `File→New...→Fragment→Fragment (Blank)`.

Вам будет предложено задать параметры нового фрагмента. Присвойте фрагменту имя `WorkoutDetailFragment`, установите флажок создания XML разметки и присвойте макету фрагмента имя `fragment_workout_detail`. Снимите флажки включения фабричных методов фрагмента и интерфейсных методов обратного вызова; они генерируют дополнительный код, который нам сейчас не нужен. Когда это будет сделано, щелкните на кнопке `Finish`.



Создание фрагментов
Связывание фрагментов
Макеты для устройств

Попробуйте посмотреть дополнительный код, генерируемый Android Studio, когда вы дочитаете книгу. Возможно, вы найдете в нем что-то полезное — это зависит от специфики ваших задач.



Если нажать кнопку `Finish`, Android Studio создает новый файл фрагмента с именем `WorkoutDetailFragment.java` в папке `app/src/main/java` и новый файл макета с именем `fragment_workout_detail.xml` в папке `app/src/res/layout`.

Разметка макета фрагмента не отличается от разметки макета активности

Начнем с обновления разметки макета фрагмента. Откройте файл `fragment_workout_detail.xml` из папки `app/src/res/layout` и замените его содержимое следующей разметкой:

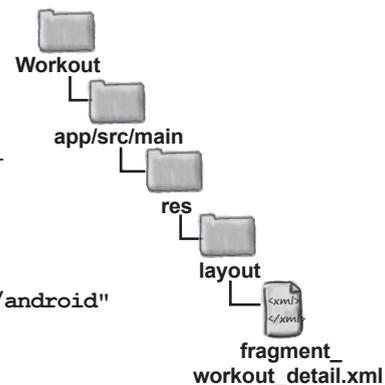
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_height="match_parent"
    android:layout_width="match_parent"
    android:orientation="vertical">
```

```
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text=""
        android:id="@+id/textTitle" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text=""
        android:id="@+id/textDescription" />
</LinearLayout>
```

Название и описание выводятся в двух разных компонентах TextView.

Название.

Описание комплекса упражнений.



Core Agony

100 Pull-ups
100 Push-ups
100 Sit-ups
100 Squats

Теперь мы можем использовать этот фрагмент в своих активностях.

Как видите, разметка макета фрагмента почти не отличается от разметки макета активности. Макет очень прост и состоит всего из двух надписей: в одной (с крупным текстом) выводится название комплекса упражнений, а в другой (с мелким текстом) выводится описание. Создавая макеты фрагментов для своих приложений, вы можете использовать в них все представления и макеты, которые уже использовались нами для построения макетов активностей.

Итак, мы создали макет, который будет использоваться фрагментом; перейдем к коду самого фрагмента.

Код фрагмента

Код фрагмента хранится в файле `WorkoutDetailFragment.java` в папке `app/src/main/java`. Откройте этот файл.

Как и следовало ожидать, среда Android Studio сгенерировала код Java за вас. Замените код, сгенерированный Android Studio, следующим кодом:

```
package com.hfad.workout;

import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class WorkoutDetailFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment_workout_detail, container, false);
    }
}
```

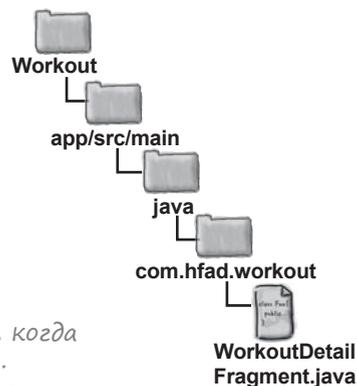
Класс расширяет класс `Fragment` системы Android.

Метод `onCreateView()` вызывается тогда, когда Android потребуется макет фрагмента.

Сообщает Android, какой макет используется фрагментом (в данном случае `fragment_workout_detail`).



Создание фрагментов
Связывание фрагментов
Макеты для устройств



Приведенный выше код создает простейший фрагмент. Как видно из листинга, этот класс расширяет `android.app.Fragment`. Все классы фрагментов должны расширять класс `Fragment`.

Наш фрагмент также реализует метод `onCreateView()`. Этот метод вызывается каждый раз, когда Android потребуется макет фрагмента; в этом методе вы сообщаете, какой макет должен использоваться данным фрагментом. Строго говоря, метод не является обязательным, но он должен быть реализован при создании каждого фрагмента, обладающего макетом, — то есть почти для любого фрагмента.

Макет фрагмента назначается вызовом

```
inflater.inflate(R.layout.fragment_workout_detail,
    container, false);
```

Этот метод является аналогом метода `setContentView()` активностей в мире фрагментов. Как и `setContentView()`, он сообщает, какой макет должен использоваться фрагментом. Аргумент `container` передается активностью, использующей фрагмент. В нем содержится объект `ViewGroup` активности, в который должен быть вставлен макет фрагмента.



Будьте осторожны!

У каждого фрагмента должен быть определен открытый конструктор без аргументов.

Дело в том, что Android использует его для повторного создания экземпляра в случае необходимости, и при отсутствии такого конструктора выдается исключение времени выполнения.

На практике добавлять такой конструктор в код фрагмента нужно лишь в том случае, если вы включаете другой конструктор с одним или несколькими аргументами. Дело в том, что если класс Java не содержит конструкторов, компилятор Java автоматически сгенерирует открытый конструктор без аргументов.

Добавление фрагмента в макет активности

Когда мы делали проект, среда Android Studio создала для нас активность с именем *MainActivity.java* и макет с именем *activity_main.xml*. Сейчас мы изменим макет так, чтобы он содержал только что созданный нами фрагмент.

Откройте файл *activity_main.xml* из папки *app/src/main/res/layout* и измените код, сгенерированный Android Studio, следующим кодом:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

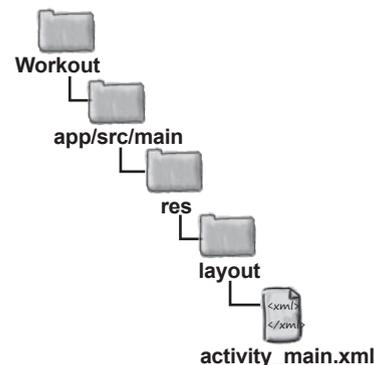
    <fragment
        class="com.hfad.workout.WorkoutDetailFragment"
        android:id="@+id/detail_frag"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</LinearLayout>
```

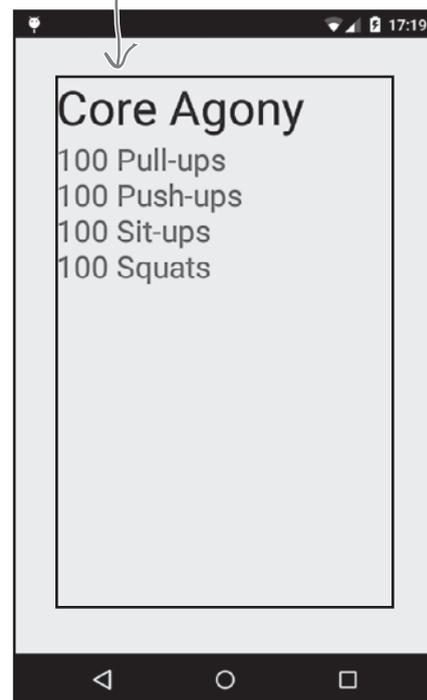
Как видите, макет состоит из единственного элемента `<fragment>`. Элемент `<fragment>` используется для добавления фрагмента в макет активности. Чтобы указать, какой именно фрагмент должен использоваться, вы присваиваете атрибуту `class` полное имя фрагмента. В нашем примере создается фрагмент `WorkoutDetailFragment` из пакета `com.hfad.workout` package, поэтому мы используем запись

```
class="com.hfad.workout.WorkoutDetailFragment"
```

Итак, мы создали фрагмент и добились того, чтобы активность отображала его в своем макете. Впрочем, пока что фрагмент еще ничего не делает. Теперь нужно сделать следующий шаг: активность сообщает фрагменту, какой комплекс упражнений в нем должен отображаться, а фрагмент заполняет свои представления подробным описанием этого комплекса.



← Добавляет фрагмент `WorkoutDetailFragment` в макет активности.



Передача идентификатора фрагменту

Активность, использующая фрагмент, обычно должна как-то «общаться» с ним. Например, если фрагмент предназначен для вывода детализаций, активность должна сообщить фрагменту, данные какой записи в нем должны выводиться.

В нашем примере во фрагменте `WorkoutDetailFragment` должна выводиться подробная информация о комплексе упражнений. Для этого мы добавим во фрагмент простой метод, который будет задавать значение идентификатора. Активность будет использовать этот метод для передачи идентификатора фрагменту. Позднее в зависимости от идентификатора будут заполняться представления фрагмента.

Ниже приведен обновленный код `WorkoutDetailFragment` (внесите изменения в свою версию):

```
package com.hfad.workout;

import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class WorkoutDetailFragment extends Fragment {
    private long workoutId;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment_workout_detail, container, false);
    }

    public void setWorkout(long id) {
        this.workoutId = id;
    }
}
```

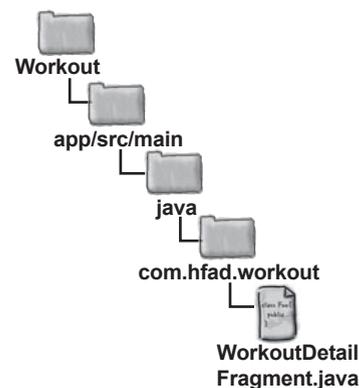
← Идентификатор комплекса упражнений, выбранного пользователем. Позднее, при выводе подробной информации, он будет использован для заполнения представлений фрагмента.

← Метод для присваивания идентификатора. Метод используется активностью для передачи значения идентификатора фрагменту.

Активность должна вызвать метод `setWorkout()` фрагмента и передать ему идентификатор нужного комплекса. Давайте посмотрим, как это делается.



Создание фрагментов
Связывание фрагментов
Макеты для устройств



Присваивание идентификатора

Прежде чем активность сможет взаимодействовать с фрагментом, она должна сначала получить ссылку на него. Для получения ссылки на фрагмент следует сначала получить ссылку на **диспетчера фрагментов** активности при помощи метода `getFragmentManager()`. Затем метод `findFragmentById()` используется для получения ссылки на фрагмент:

```
getFragmentManager().findFragmentById(R.id.fragment_id)
```

Идентификатор фрагмента в макете активности.

Диспетчер фрагментов управляет всеми фрагментами, используемыми активностью. Он используется для получения ссылок на фрагменты и выполнения операций с фрагментами. Мы еще вернемся к этой теме далее в этой главе. Ниже приведен полный код активности (замените существующий код из `MainActivity.java` этим кодом):

Метод `findFragmentById()` отчасти напоминает `findViewById()`, но используется для получения ссылки на фрагмент.

```
package com.hfad.workout;

import android.app.Activity;
import android.os.Bundle;

public class MainActivity extends Activity {
```

```
    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_main);
```

```
        WorkoutDetailFragment frag = (WorkoutDetailFragment)
```

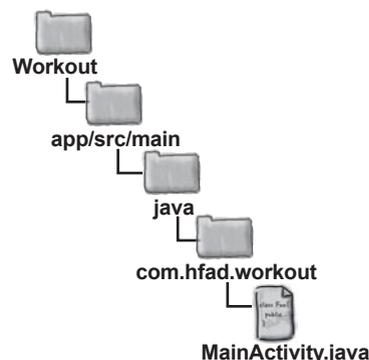
```
            getFragmentManager().findFragmentById(R.id.detail_frag);
```

```
        frag.setWorkout(1);
```

```
    }
```

```
}
```

Приказываем `WorkoutDetailFragment` вывести подробную информацию о произвольно выбранном комплексе, чтобы убедиться в том, что все работает.



Возвращает ссылку на фрагмент `WorkoutDetailFragment`. В макете активности этому фрагменту присвоен идентификатор `detail_frag`.

Как видите, получение ссылки на фрагмент происходит после вызова `setContentView()`. Это очень важно, потому что до этого момента фрагмент еще не был создан.

Вызов `frag.setWorkout(1)` сообщает фрагменту, о каком фрагменте нужно вывести подробную информацию. Пока мы просто задаем конкретный идентификатор в методе `onCreate()` активности, чтобы увидеть на экране хоть какие-нибудь данные. Позднее метод будет изменен, чтобы пользователь мог выбрать интересующий его комплекс упражнений.

Следующее, что нужно сделать, — заставить фрагмент обновить свои представления при выводе на экран. Но прежде чем браться за решение этой задачи, необходимо поближе познакомиться с жизненным циклом фрагментов.

Снова о состояниях активностей



Создание фрагментов
Связывание фрагментов
Макеты для устройств

У фрагментов, как и у активностей, имеются ключевые методы жизненного цикла, вызываемые в определенные моменты. Чтобы ваши фрагменты работали именно так, как вам нужно, важно знать, что это за методы и когда они вызываются.

Фрагменты содержатся в активностях и находятся под их управлением, поэтому жизненный цикл фрагмента тесно связан с жизненным циклом активности. Ниже приведена краткая сводка состояний, через которые проходит активность, а на следующей странице показано, как эти состояния связаны с фрагментом.



Жизненный цикл фрагмента

Жизненный цикл фрагмента очень похож на жизненный цикл активности, но он содержит несколько дополнительных стадий. Это объясняется тем, что фрагмент должен взаимодействовать с жизненным циклом содержащей его активности. Ниже перечислены методы жизненного цикла фрагментов и их место в различных состояниях активности.

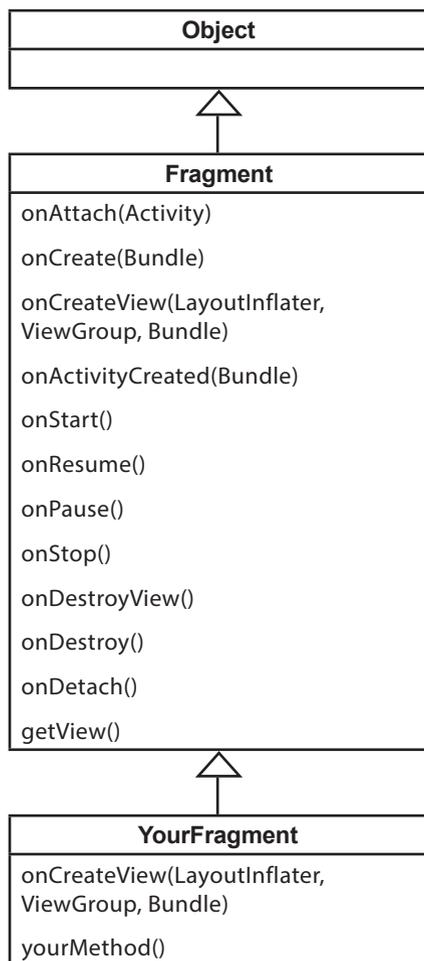
СОСТОЯНИЯ АКТИВНОСТИ	МЕТОДЫ ФРАГМЕНТА
Создание активности	onAttach(Activity) Вызывается при связывании фрагмента с активностью.
	onCreate(Bundle) Метод очень похож на метод onCreate() активности; используется для выполнения инициализации фрагмента.
	onCreateView(LayoutInflater, ViewGroup, Bundle) Фрагменты используют объект LayoutInflater для создания своего представления в этой точке.
	onActivityCreated(Bundle) Метод вызывается при завершении метода onCreate() активности.
	onStart() Метод onStart() вызывается перед тем, как фрагмент становится видимым.
	onResume() Вызывается, когда фрагмент виден и активно работает.
Продолжение активности	onPause() Вызывается, когда фрагмент перестает взаимодействовать с пользователем.
Приостановка активности	onStop() Вызывается, когда фрагмент перестает быть видимым.
Остановка активности	onDestroyView() Дает фрагменту возможность освободить любые ресурсы, связанные с его представлением.
Уничтожение активности	onDestroy() В этом методе фрагмент может освободить любые другие ресурсы, созданные им.
	onDetach() Вызывается при окончательном разрыве связи между фрагментом и активностью.

Ваш фрагмент наследует методы жизненного цикла

Как упоминалось ранее, ваш класс фрагмента расширяет класс `Android Fragment`. Этот класс предоставляет фрагменту доступ к методам жизненного цикла фрагмента.



Создание фрагментов
Связывание фрагментов
Макеты для устройств



Класс Object
(`java.lang.Object`)

Класс Fragment
(`android.app.Fragment`)

Класс `Fragment` реализует версии методов жизненного цикла по умолчанию. Он также определяет другие методы, необходимые фрагменту, — такие, как `getView()`.

Класс YourFragment
(`com.hfad.foo`)

Большая часть поведения фрагмента обеспечивается методами суперкласса. Вам остается лишь переопределить методы, нужные вам.

Хотя у фрагментов много общего с активностями, класс `Fragment` не расширяет класс `Activity`. Это означает, что некоторые методы, доступные для активностей, недоступны для фрагментов.

Обратите внимание на то, что класс `Fragment` не реализует класс `Context`. В отличие от активности, фрагмент не является специализацией контекста, а следовательно, не имеет прямого доступа к глобальной информации о среде выполнения приложения. Вместо этого фрагменту придется обращаться к такой информации через контексты других объектов — например, его родительской активности.

Заполнение представлений в методе onStart() фрагмента

Класс `WorkoutDetailFragment` должен обновить свои представления подробной информацией о комплексе упражнений. Это необходимо сделать при запуске активности, поэтому мы воспользуемся методом `onStart()` фрагмента. Код выглядит так:

```
package com.hfad.workout;

import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

public class WorkoutDetailFragment extends Fragment {
    private long workoutId;

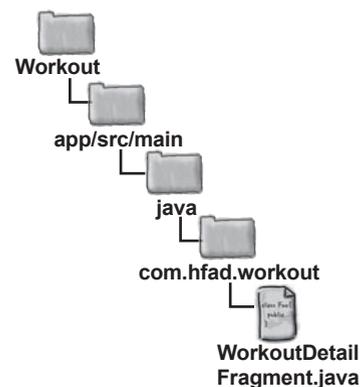
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment_workout_detail, container, false);
    }

    @Override
    public void onStart() {
        super.onStart();
        View view = getView();
        if (view != null) {
            TextView title = (TextView) view.findViewById(R.id.textTitle);
            Workout workout = Workout.workouts[(int) workoutId];
            title.setText(workout.getName());
            TextView description = (TextView) view.findViewById(R.id.textDescription);
            description.setText(workout.getDescription());
        }
    }

    public void setWorkout(long id) {
        this.workoutId = id;
    }
}
```

Этот класс используется в методе `onStart()`.

Метод `getView()` получает корневой объект `View` фрагмента. Далее полученный объект используется для получения ссылок на надписи, предназначенные для названия и описания комплекса упражнений.



Как упоминалось на предыдущей странице, фрагменты отличаются от активностей и поэтому не поддерживают некоторые методы, доступные для активностей. Например, у фрагментов отсутствует метод `findViewById()`. Чтобы получить ссылку на представления фрагмента, сначала необходимо получить ссылку на корневое представление фрагмента методом `getView()` и по этой ссылке найти дочерние представления.

Итак, теперь фрагмент обновляет свои представления. Давайте опробуем приложение в деле.

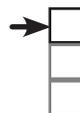
Всегда вызывайте версию суперкласса в реализации любых методов жизненного цикла фрагментов.



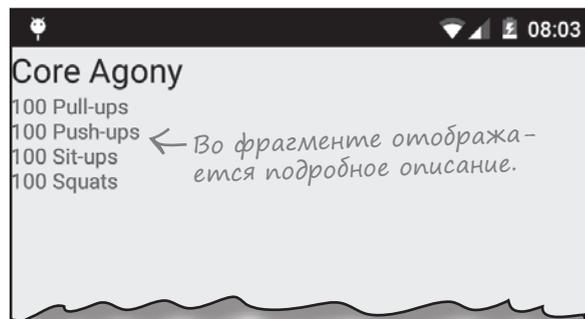
Тест-драйв

При запуске приложения на экране устройства появляется подробная информация о комплексе упражнений.

Приложение выглядит точно так же, как если бы информация выводилась в активности. Но поскольку активность использует *фрагмент* для вывода информации, при желании мы можем задействовать этот фрагмент в другой активности.

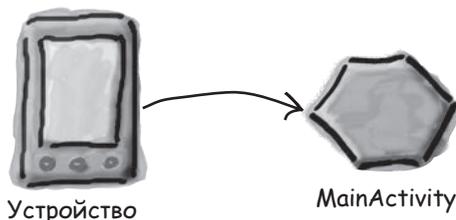


Создание фрагментов
Связывание фрагментов
Макеты для устройств

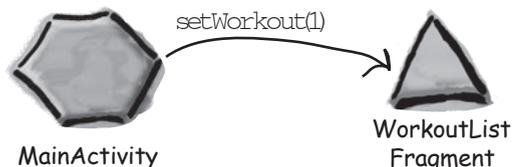


Что происходит при запуске приложения

- 1 При запуске приложения создается активность `MainActivity`.



- 2 `MainActivity` передает идентификатор комплекса классу `WorkoutDetailFragment` в своем методе `onCreate()`, вызывая метод `setWorkout()` фрагмента.

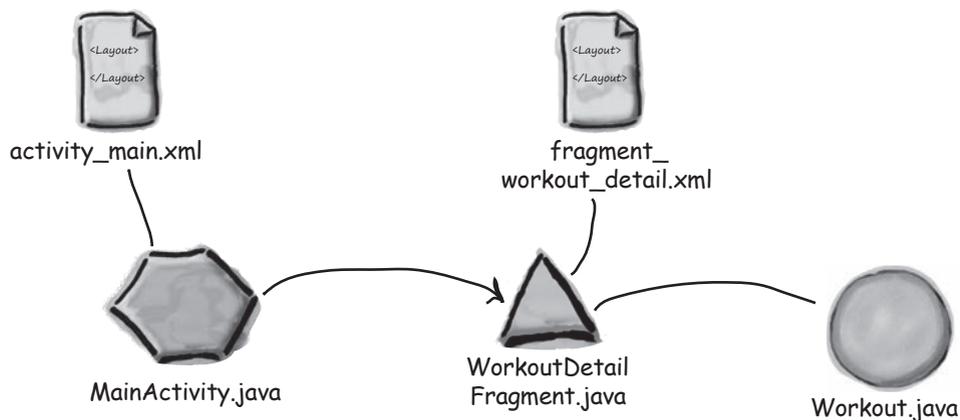


- 3 Фрагмент использует полученное значение в своем методе `onStart()` для заполнения своих надписей.



Что было сделано

На данный момент мы создали активность *MainActivity.java*, ее макет *activity_main.xml*, фрагмент *WorkoutDetailFragment.java*, его макет *fragment_workout_detail.xml*, а также файл обычного класса Java *Workout.java*. *MainActivity* использует *WorkoutListFragment* для вывода подробной информации о комплексе упражнений, а фрагмент получает данные от класса *Workout*.



Следующее, что нужно сделать — создать фрагмент *WorkoutListFragment* для вывода списка комплексов упражнений.

Часто задаваемые вопросы

В: Почему активность не может получить фрагмент вызовом `findViewById()`?

О: Потому что `findViewById()` всегда возвращает объект `View` — а фрагменты, как ни странно, не наследуют от этого класса.

В: Почему активность не содержит метода `findFragmentById()`, аналогичного `findViewById()`?

О: Резонный вопрос. Дело в том, что в ранних версиях Android фрагменты не поддерживались. Диспетчер фрагментов позволяет добавить большой объем полезного кода для управления фрагментами без его внедрения в базовый класс активности.

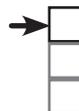
В: Почему у фрагментов нет метода `findViewById()`?

О: Потому что фрагменты не являются представлениями или активностями. Вместо вызова этого метода приходится использовать метод `getView()` фрагмента для получения ссылки на корневое представление фрагмента, а затем вызывать метод `findViewById()` представления для получения дочерних представлений.

В: Чтобы активность могла использоваться в приложении, она должна быть зарегистрирована в файле `AndroidManifest.xml`. А фрагменты нужно регистрировать?

О: Нет. Действительно, активности должны регистрироваться в файле `AndroidManifest.xml`, но фрагменты в нем не регистрируются.

Создание фрагмента со списком



Создание фрагментов
Связывание фрагментов
Макеты для устройств

Теперь, когда фрагмент `WorkoutDetailFragment` заработал, необходимо создать второй фрагмент со списком разных комплексов упражнений. После этого фрагменты можно будет использовать для создания разных пользовательских интерфейсов для телефонов и планшетов.



Вы уже видели, как списковое представление добавляется в активность. Мы создадим фрагмент, содержащий только списковое представление, и заполним его названиями комплексов упражнений.



Значит, фрагмент будет содержать только списковое представление. Интересно... Для активностей, содержащих только список, мы использовали активность `ListActivity`. Нет ли чего-нибудь похожего для фрагментов?

А он прав! Мы можем воспользоваться специальной разновидностью фрагмента — `ListFragment`.

Эта тема рассматривается на следующей странице.

ListFragment — фрагмент, содержащий только списковое представление

Списковый фрагмент — разновидность фрагментов, предназначенная для работы со списками. Как и списковая активность, такой фрагмент автоматически связывается со списковым представлением, и вам не придется создавать компонент самостоятельно. Вот как выглядит код:

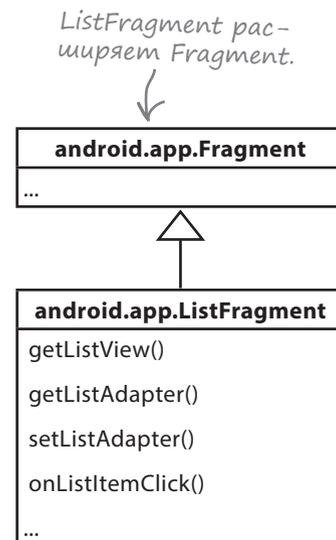
Списковый фрагмент содержит собственное списковое представление, так что вам не придется создавать его самостоятельно: достаточно предоставить данные.



Как и в случае со списковой активностью, использование спискового фрагмента для отображения категорий обладает парой преимуществ:

- ★ **Вам не придется строить макет самостоятельно.**
Списковые фрагменты определяют свой макет на программном уровне, поэтому вам не придется создавать или заниматься сопровождением разметки XML. Макет, генерируемый списковым фрагментом, содержит одно списковое представление. Для обращения к списковому представлению из кода активности используется метод `getListView()` спискового фрагмента. Такое обращение необходимо для того, чтобы вы могли задать данные, которые должны выводиться в списковом представлении.
- ★ **Вам не нужно реализовать собственного слушателя событий.**
Класс `ListFragment` регистрируется как слушатель события для спискового представления и отслеживает щелчки на вариантах спискового представления. Чтобы фрагмент реагировал на щелчки, достаточно реализовать метод `onListItemClick()` спискового фрагмента. Скоро вы увидите, как это делается.

Как же выглядит код спискового фрагмента?



ListFragment — разновидность Fragment, специализированная для работы со списковым представлением. В макете по умолчанию этого фрагмента содержится компонент ListView.

Создание спискового фрагмента

Списковые фрагменты добавляются в проект точно так же, как и обычные фрагменты. Выберите команду File→New...→Fragment→Fragment (Blank). Присвойте фрагменту имя “WorkoutListFragment”, снимите флажки создания XML макета, а также флажки включения фабричных методов и интерфейсных обратных вызовов. Списковые фрагменты определяют свои макеты на программном уровне, поэтому вам не нужно, чтобы среда Android Studio создавала макет. При щелчке на кнопке Finish среда Android Studio создает новый списковый фрагмент в файле с именем *WorkoutListFragment.java* в папке *app/src/main/java*.

Ниже приведен типичный код создания спискового фрагмента. Как видите, он очень похож на код обычного фрагмента. Замените им код *WorkoutListFragment*:

```
package com.hfad.workout;

import android.os.Bundle;
import android.app.ListFragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class WorkoutListFragment extends ListFragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        return super.onCreateView(inflater, container, savedInstanceState);
    }
}
```

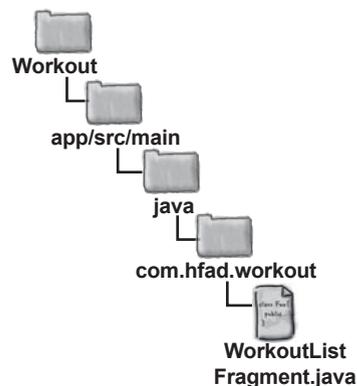
фрагмент должен расширять ListFragment, а не Fragment.



Вызов метода onCreateView() суперкласса предоставляет макет по умолчанию для ListFragment.



Создание фрагментов
Связывание фрагментов
Макеты для устройств



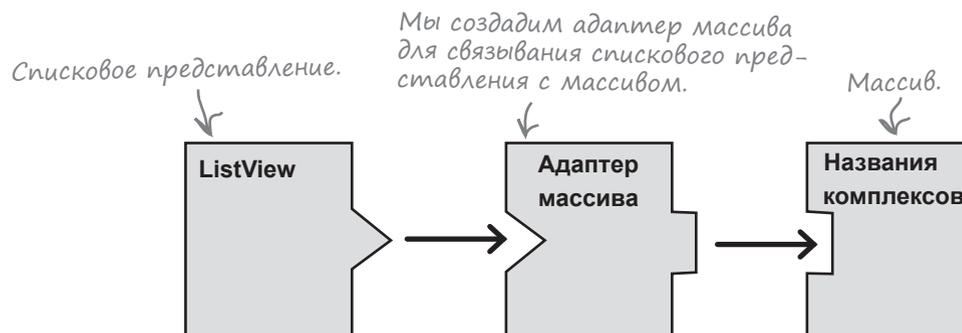
Приведенный выше код создает простейший списковый фрагмент с именем *WorkoutListFragment*. Так как фрагмент является списковым, он должен расширять класс *ListFragment* вместо *Fragment*. Метод *onCreateView()* не является обязательным. Этот метод вызывается при создании представления фрагмента. Мы включаем его в свой код, так как хотим, чтобы списковое представление фрагмента заполнялось данными сразу же после его создания. Если ваш код ничего не должен делать в этот момент, включать этот метод не обязательно.

Давайте посмотрим, как списковое представление заполняется данными.

Использование ArrayAdapter для заполнения ListView

Как упоминалось в главе 6, для связывания данных со списковым представлением можно воспользоваться адаптером. Это относится и к списковым представлениям, заключенным в фрагмент; `ListView` расширяет `AdapterView`, и именно этот класс обеспечивает работу представления с адаптерами.

Так как информация для спискового представления в `WorkoutListFragment` будет передаваться в массиве названий, мы воспользуемся адаптером массива для связывания данных со списковым представлением.



Fragment не является специализацией Context

Как было показано ранее, для создания адаптера массива, работающего со списковым представлением, используется синтаксис:

```
ArrayAdapter<DataType> listAdapter = new ArrayAdapter<DataType>(
    context, android.R.layout.simple_list_item_1, array);
```

где `DataType` — тип данных, `array` — массив, а `context` — текущий контекст.

При использовании этого синтаксиса в активности для передачи текущего контекста можно было использовать ключевое слово `this`. Это было возможно, поскольку активность является специализацией контекста — класс `Activity` является субклассом `Context`. Как упоминалось ранее, класс `Fragment` не является субклассом `Context`, так что `this` не подходит. Вместо этого текущий контекст придется получать другим способом. Если адаптер используется в методе `onCreateView()` фрагмента, как в нашем примере, для получения контекста используется метод `getContext()` объекта `LayoutInflater`:

```
ArrayAdapter<DataType> listAdapter = new ArrayAdapter<DataType>(
    inflater.getContext(), android.R.layout.simple_list_item_1, array);
```

Возвращает текущий контекст. →

Когда адаптер будет создан, его следует связать с `ListView` при помощи метода `setListAdapter()` фрагмента:

```
setListAdapter(listAdapter);
```

Давайте воспользуемся адаптером массива для заполнения спискового представления во фрагменте названиями комплексов упражнений.



Обновленный код WorkoutListFragment

Мы обновили свою версию кода `WorkoutListFragment.java` так, чтобы она заполняла списковое представление названиями комплексов. Внесите изменения в свой код и сохраните их:

```
package com.hfad.workout;

import android.os.Bundle;
import android.app.ListFragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;

public class WorkoutListFragment extends ListFragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        String[] names = new String[Workout.workouts.length];
        for (int i = 0; i < names.length; i++) {
            names[i] = Workout.workouts[i].getName();
        }
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(
            inflater.getContext(), android.R.layout.simple_list_item_1,
            names);
        setListAdapter(adapter);

        return super.onCreateView(inflater, container, savedInstanceState);
    }
}
```

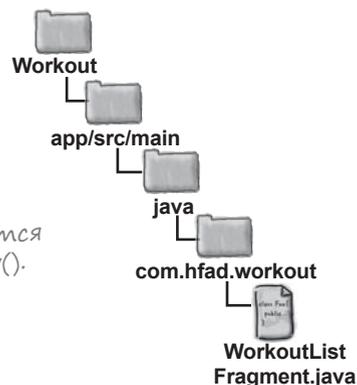
← Этот класс используется в методе `onCreateView()`.

Создать адаптер массива. ↓

Создать массив строк с названиями комплексов упражнений.

Получить контекст от `LayoutInflater`.

← Связать адаптер массива со списковым представлением.



Включение фрагмента WorkoutListFragment в макет MainActivity

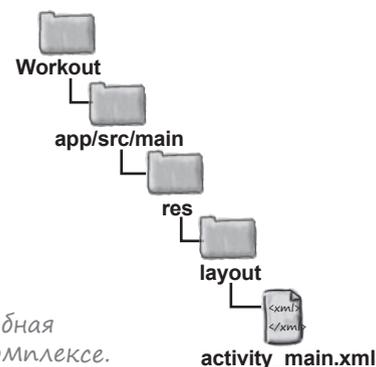
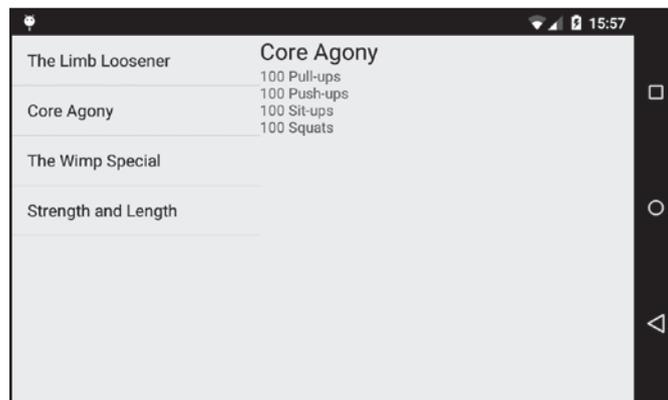
Мы добавим новый фрагмент `WorkoutListFragment` в макет `MainActivity` так, чтобы он отображался слева от `WorkoutDetailFragment`. Размещение фрагментов рядом друг с другом — типичный вариант дизайна приложений для планшетов. Чтобы добиться нужного результата, мы воспользуемся линейным макетом с горизонтальной ориентацией. Для управления распределением горизонтального пространства между фрагментами будет использована система весов. Ниже приведена полученная разметка (внесите изменения в свою версию из `activity_main.xml`):

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <fragment
        class="com.hfad.workout.WorkoutListFragment"
        android:id="@+id/list_frag"
        android:layout_width="0dp"
        android:layout_weight="2"
        android:layout_height="match_parent"/>
    <fragment
        class="com.hfad.workout.WorkoutDetailFragment"
        android:id="@+id/detail_frag"
        android:layout_width="0dp"
        android:layout_weight="3"
        android:layout_height="match_parent" />
</LinearLayout>
```

Сначала выводится список комплексов.

Затем выводится подробная информация об одном комплексе.

Атрибут `layout_weight` управляет распределением пространства между фрагментами.

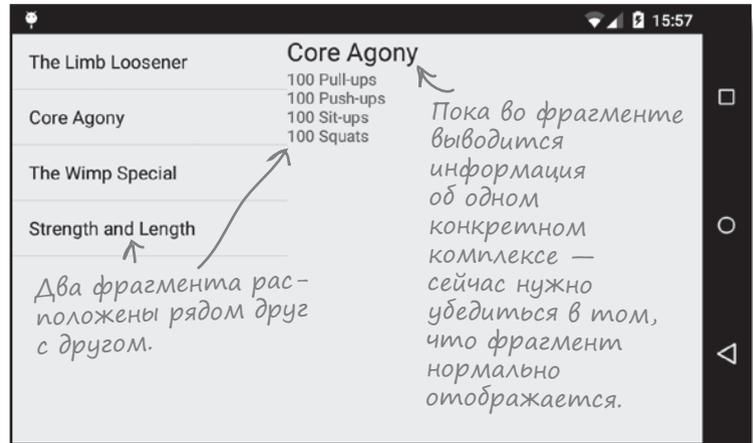


Посмотрим, как теперь выглядит приложение.



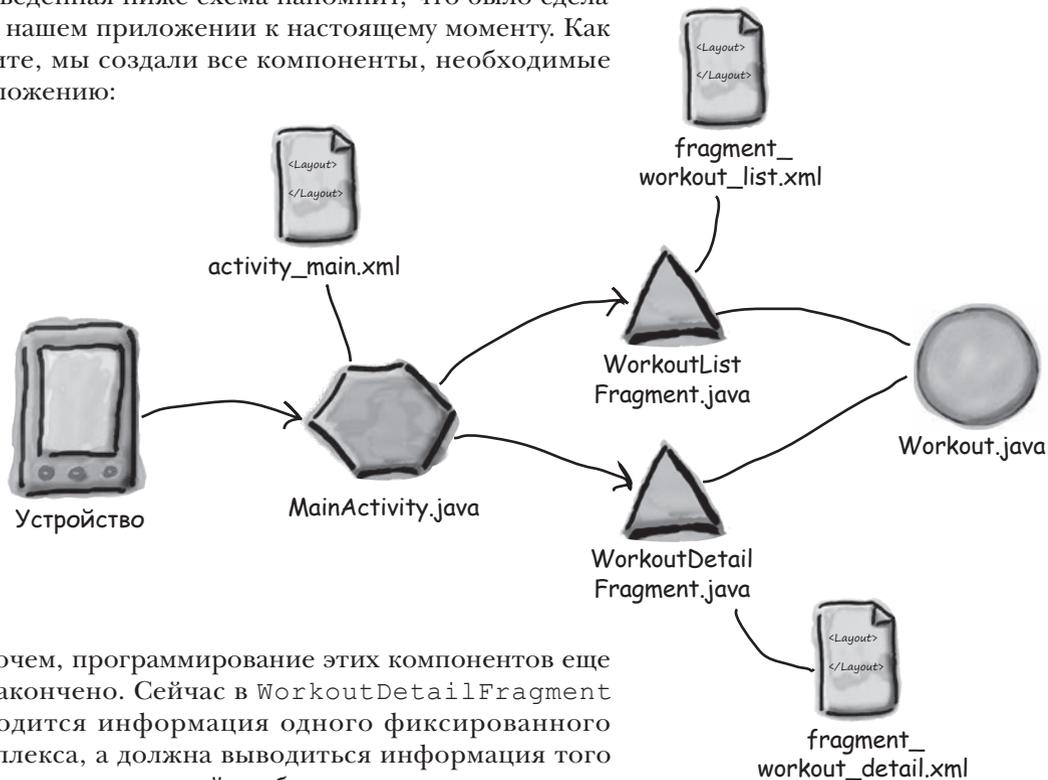
Тест-драйв

При запуске приложения в левой части экрана появляется список комплексов, а справа — подробная информация об одном комплексе. Пока этот комплекс жестко запрограммирован в коде *MainActivity.java*, поэтому как бы пользователь ни пытался выбирать другие варианты, справа ничего не изменится.



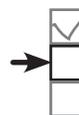
Теперь нужно добиться того, чтобы фрагмент *WorkoutDetailFragment* реагировал на щелчку в *WorkoutListFragment*

Приведенная ниже схема напомним, что было сделано в нашем приложении к настоящему моменту. Как видите, мы создали все компоненты, необходимые приложению:



Впрочем, программирование этих компонентов еще не закончено. Сейчас в *WorkoutDetailFragment* выводится информация одного фиксированного комплекса, а должна выводиться информация того комплекса, который выбирается пользователем в *WorkoutListFragment*.

Связывание списка с детализацией



Создание фрагментов
Связывание фрагментов
Макеты для устройств

Как же добиться того, чтобы подробная информация изменялась при щелчке на варианте в списке? Есть несколько возможных решений. Мы будем действовать примерно так:

- 1 Добавить в `WorkoutListFragment` код, который ожидает щелчка на комплексе упражнений.
- 2 Когда этот код выполняется, мы вызываем некий код из `MainActivity.java`, который...
- 3 ...заполняет фрагмент детализации новой информацией.

Однако включать в `WorkoutListFragment` код, который *напрямую* взаимодействует с `WorkoutDetailFragment`, было бы нежелательно. Как вы думаете, почему?

Из-за возможности *повторного использования*. Наши фрагменты должны располагать минимумом информации о среде, содержащей их. Чем больше фрагмент должен знать об активности, использующей его, тем меньше он пригоден для повторного использования.



Один момент! Вы говорите, что фрагмент не должен знать об активности, которая его содержит? А как насчет второго пункта? Разве мы не заставляем его зависеть от `MainActivity`? И разве это не означает, что он не может использоваться в другой активности?

Необходимо использовать интерфейс для отделения фрагмента от активности.

Логическое отделение фрагмента от активности

Имеются два объекта, которые должны взаимодействовать друг с другом, — фрагмент и активность. Мы хотим, чтобы они взаимодействовали, располагая минимумом информации о другой стороне. В Java для решения подобных задач используются *интерфейсы*. При определении интерфейса формулируются *минимальные требования к объекту для его осмысленного взаимодействия с другим объектом*. Это означает, что фрагмент сможет взаимодействовать практически с любой активностью — при условии, что эта активность реализует необходимый интерфейс.

Мы создадим интерфейс с именем `WorkoutClickListener`, который выглядит примерно так:

```
interface WorkoutClickListener {
    void itemClicked(long id);
};
```

Если активность реализует этот интерфейс, мы сможем сообщить ей, что на варианте в списковом фрагменте был сделан щелчок. Во время выполнения будет происходить следующая последовательность событий:

- ❶ Объект `WorkoutClickListener` сообщает фрагменту о своем желании прослушивать события щелчков.
- ❷ Пользователь щелкает на комплексе упражнений в списке.
- ❸ Вызывается метод `onListItemClicked()` в списковом фрагменте.
- ❹ Затем этот метод вызывает метод `itemClicked()` класса `WorkoutClickListener` с передачей идентификатора варианта, на котором был сделан щелчок.

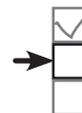
Но когда активность должна сообщать о прослушивании?

Когда активность будет сообщать фрагменту, что она готова к получению уведомлений о щелчках на вариантах в списке? Взглянув на схему жизненного цикла фрагмента, вы увидите, что при присоединении фрагмента к активности вызывается метод `onAttach()` фрагмента с передачей объекта активности:

```
@Override
public void onAttach(Activity activity) {
    ...
}
```

Этот метод можно использовать для регистрации активности во фрагменте. Давайте рассмотрим код реализации.

Добавление интерфейса к списковому фрагменту



Создание фрагментов
Связывание фрагментов
Макеты для устройств

Мы изменили свой код `WorkoutListFragment.java`, добавив в него слушателя события (внесите изменения и сохраните свою работу):

```
package com.hfad.workout;

import android.os.Bundle;
import android.app.ListFragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.app.Activity;
import android.widget.ListView;

public class WorkoutListFragment extends ListFragment {

    static interface WorkoutListListener {
        void itemClicked(long id);
    };

    private WorkoutListListener listener;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        String[] names = new String[Workout.workouts.length];
        for (int i = 0; i < names.length; i++) {
            names[i] = Workout.workouts[i].getName();
        }
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(
            inflater.getContext(), android.R.layout.simple_list_item_1,
            names);
        setListAdapter(adapter);
        return super.onCreateView(inflater, container, savedInstanceState);
    }

    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);
        this.listener = (WorkoutListListener) activity;
    }

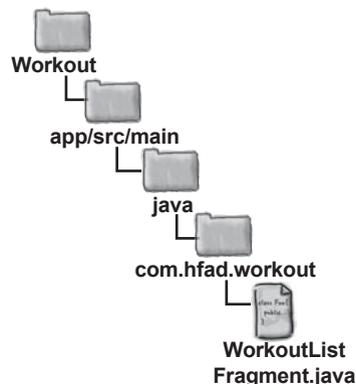
    @Override
    public void onItemClick(ListView l, View v, int position, long id) {
        if (listener != null) {
            listener.itemClicked(id);
        }
    }
}
```

Импортировать эти классы.

Добавить слушателя к фрагменту.

Вызывается при при-соединении фрагмента к активности.

Сообщить слушателю о том, что на одном из вариантов ListView был сделан щелчок.



Реализация интерфейса активностью

Теперь активность *MainActivity.java* должна реализовать только что созданный нами интерфейс *WorkoutListListener*. Внесите в свой код следующие изменения:

```

package com.hfad.workout;

import android.app.Activity;
import android.os.Bundle;

public class MainActivity extends Activity
    implements WorkoutListFragment.WorkoutListListener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        WorkoutDetailFragment frag = (WorkoutDetailFragment)
            getSupportFragmentManager().findFragmentById(R.id.detail_frag);
        frag.setWorkout(1);
    }

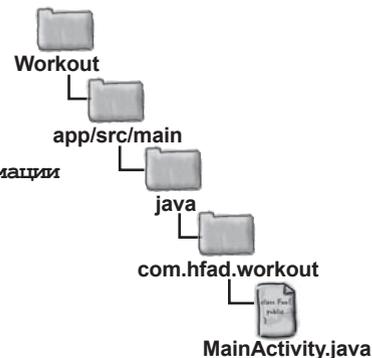
    @Override
    public void itemClicked(long id) {
        //Здесь размещается код отображения подробной информации
    }
}

```

Реализовать слушателя, определенного в *WorkoutListFragment*.

Эти строки удаляются — они больше не нужны.

Этот метод определяется в слушателе.



Когда пользователь щелкает на варианте в фрагменте, вызывается метод *itemClicked()* активности. В этот метод включается код вывода подробной информации о только что выбранном комплексе.

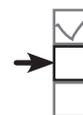
Но как обновить фрагмент с детализацией?

WorkoutDetailFragment обновляет свои представления при старте фрагмента. Но как добиться того, чтобы фрагмент обновил свои представления после его появления на экране?

Возможно, вы думаете, что мы можем воспользоваться методами жизненного цикла фрагмента, чтобы заставить его обновить информацию. Однако мы поступим иначе — **фрагмент детализации будет заменяться новым фрагментом детализации каждый раз, когда потребуется изменить его содержимое.**

И для этого есть достаточно веская причина...

Фрагменты должны поддерживать кнопку возврата



Создание фрагментов
Связывание фрагментов
Макеты для устройств

Предположим, пользователь выбирает один комплекс, а потом другой. При нажатии кнопки Назад он рассчитывает вернуться к первому из выбранных им комплексов.



Во всех приложениях, которые мы строили до сих пор, кнопка Назад возвращала пользователя к предыдущей активности. Однако теперь, когда мы начали работать с фрагментами, необходимо лучше понимать, что происходит при нажатии кнопки Назад.

Стек возврата

Стек возврата (back stack) представляет собой список «мест», посещенных на устройстве. Каждое «место» представлено **транзакцией** в стеке возврата. Многие транзакции осуществляют переход от одной активности к другой:

Transaction: Go to inbox activity
Transaction: Go to 'compose new mail' activity
Transaction: Go to sent mail activity

Все это разные транзакции.

Итак, при переходе к новой активности транзакция этого перехода заносится в стек возврата. Если нажать кнопку Назад, транзакция будет отменена, и вы вернетесь к предыдущей активности.

Однако транзакции в стеке возврата *не обязаны* быть переходами между активностями. Они могут быть простыми изменениями фрагментов на экране:

Transaction: Replace the 'Strength and length' detail fragment with a 'Core agony' fragment
Transaction: Replace the 'Core agony' fragment with 'The wimp special'

Это означает, что смена *фрагментов* также может отменяться кнопкой Назад, как и смена *активностей*.

Не обновление, а замена!

Вместо того, чтобы обновлять представления в `WorkoutDetailFragment`, мы заменим весь фрагмент новым экземпляром `WorkoutDetailFragment`, настроенным для вывода информации о следующем выбранном комплексе. При таком подходе замена фрагмента будет зарегистрирована в стеке возврата, а пользователь сможет отменить изменение нажатием кнопки Назад. Но как заменить один фрагмент другим?

Начать следует с внесения изменений в файл макета `activity_main.xml`. Вместо того, чтобы вставлять фрагмент `WorkoutDetailFragment` напрямую, мы воспользуемся **фреймом**.

Фрейм — разновидность группы представлений, используемая для резервирования области экрана. Он определяется элементом `<FrameLayout>` и используется для отображения одиночных объектов — в нашем примере это фрагмент. Мы помещаем фрагмент во фрейм, чтобы иметь возможность управлять его содержимым на программном уровне. Каждый раз, когда пользователь выбирает новый вариант в списковом представлении `WorkoutListFragment`, текущее содержимое фрейма заменяется новым экземпляром `WorkoutDetailFragment` с информацией о выбранном комплексе:

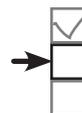
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <fragment
        class="com.hfad.workout.WorkoutListFragment"
        android:id="@+id/list_frag"
        android:layout_width="0dp"
        android:layout_weight="2"
        android:layout_height="match_parent"/>

    <fragment
        class="com.hfad.workout.WorkoutDetailFragment"
        android:id="@+id/detail_frag"
        android:layout_width="0dp"
        android:layout_weight="3"
        android:layout_height="match_parent" />
</LinearLayout>
```

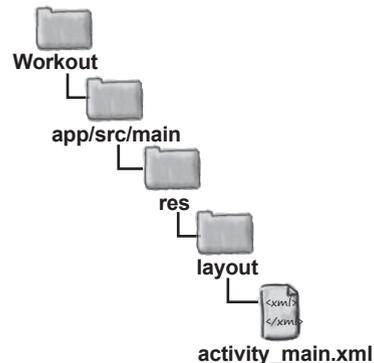
фрагмент будет отображаться в элементе <FrameLayout>.

фрагмент будет добавляться в <FrameLayout> на программном уровне.



Создание фрагментов
Связывание фрагментов
Макеты для устройств

Если реакция на изменения в пользовательском интерфейсе не нужна, добавьте фрагмент в активность с использованием элемента `<fragment>`. В противном случае используйте `<FrameLayout>`.



А теперь напишем код для добавления фрагмента в `<FrameLayout>`.

Транзакции фрагментов

Замена фрагмента во время выполнения происходит в виде **транзакции фрагмента** – набора изменений, относящихся к фрагменту, которые должны применяться как единое целое.

Создание транзакции фрагмента начинается с получения объекта `FragmentManager` от диспетчера фрагментов:

```
WorkoutDetailFragment fragment = new WorkoutDetailFragment();
FragmentManager transaction = fragmentManager.beginTransaction();
```

Затем указываются все действия, которые должны быть сгруппированы в транзакции. В нашем случае требуется заменить фрагмент во фрейме; замена осуществляется методом `replace()` макета:

```
transaction.replace(R.id.fragment_container, fragment);
```

где `R.id.fragment_container` – идентификатор контейнера, содержащего фрагмент. Вы также можете добавить фрагмент в контейнер методом `add()` или удалить его методом `remove()`:

```
transaction.add(R.id.fragment_container, fragment);
transaction.remove(fragment);
```

Метод `setTransition()` используется для определения анимации перехода, сопровождающей транзакцию.

```
transaction.setTransition(transition);
```

← Назначать переход не обязательно.

где `transition` – тип анимации. Допустимые значения – `TRANSIT_FRAGMENT_CLOSE` (фрагмент удаляется из стека), `TRANSIT_FRAGMENT_OPEN` (фрагмент добавляется), `TRANSIT_FRAGMENT_FADE` (фрагмент растворяется или проявляется) и `TRANSIT_NONE` (анимация отсутствует).

После определения всех действий, которые должны выполняться в составе транзакции, вызов метода `addToBackStack()` помещает транзакцию в стек возврата. Это делается для того, чтобы пользователь мог вернуться к предыдущему состоянию фрагмента нажатием кнопки Назад. Метод `addToBackStack()` получает один параметр – строку с именем, используемую для идентификации транзакции:

```
transaction.addToBackStack(null);
```

← В большинстве случаев получать транзакцию вам не придется, поэтому при вызове передается `null`.

Чтобы закрепить изменения в активности, вызовите метод `commit()`:

```
transaction.commit();
```

Метод `commit()` закрепляет изменения.

↑
Начало транзакции фрагмента.

← Замена фрагмента, содержащегося в контейнере.

← При желании вы можете добавлять или удалять фрагменты. В нашем примере это не нужно.

Обновленный код MainActivity

Итак, новый экземпляр `WorkoutDetailFragment` должен вывести данные правильного комплекса упражнений, отображать фрагмент в активности в составе транзакции и добавлять транзакцию в стек возврата. Полный код реализации выглядит так:

```
package com.hfad.workout;

import android.app.Activity;
import android.os.Bundle;
import android.app.FragmentTransaction;

public class MainActivity extends Activity
    implements WorkoutListFragment.WorkoutListListener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

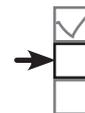
    @Override
    public void itemClicked(long id) {
        WorkoutDetailFragment details = new WorkoutDetailFragment();
        FragmentTransaction ft = getFragmentManager().beginTransaction();
        details.setWorkout(id);
        ft.replace(R.id.fragment_container, details);
        ft.addToBackStack(null);
        ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);
        ft.commit();
    }
}
```

← Чтобы использовать транзакцию фрагмента, необходимо импортировать класс `FragmentTransaction`.

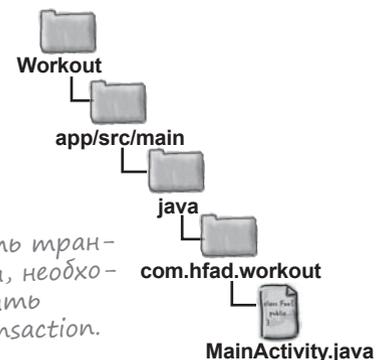
← Начало транзакции фрагмента.

← Заменить фрагмент и добавить его в стек возврата.

← Включить анимацию растворения и проявления фрагментов.



Создание фрагмента
Связывание фрагментов
Макеты для устройств



Давайте посмотрим, что происходит при выполнении кода.



Тест-драйв

При запуске приложения в левой части экрана появляется список. Если выбрать один из комплексов, справа появляется подробная информация о выбранном комплексе. Если же выбрать другой комплекс, а затем нажать кнопку Назад, на экране снова отображается информация того комплекса, который был выбран первым.



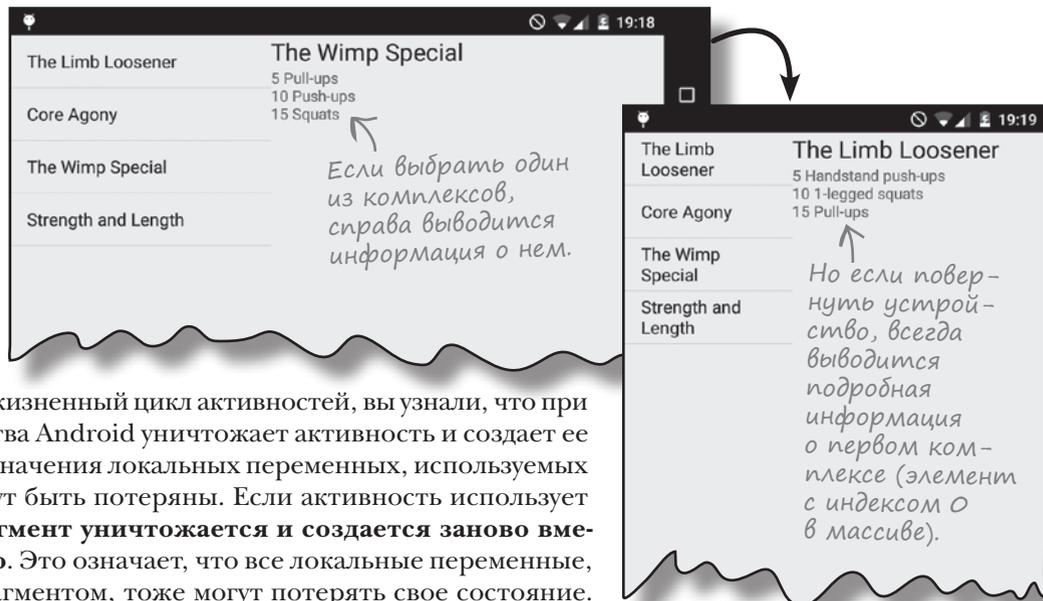
На первый взгляд приложение работает нормально... до того, как вы повернете устройство. При изменении ориентации экрана возникает проблема. Давайте посмотрим, что происходит.

Поворот устройства нарушает работу приложения

Если повернуть устройство, возникает проблема. Какой бы комплекс вы ни выбрали, при повороте приложение всегда выводит информацию о первом комплексе.



Создание фрагментов
Связывание фрагментов
Макеты для устройств



Когда мы изучали жизненный цикл активностей, вы узнали, что при повороте устройства Android уничтожает активность и создает ее заново. При этом значения локальных переменных, используемых активностью, могут быть потеряны. Если активность использует фрагмент, то **фрагмент уничтожается и создается заново вместе с активностью**. Это означает, что все локальные переменные, используемые фрагментом, тоже могут потерять свое состояние. В коде фрагмента `WorkoutDetailFragment` локальная переменная `workoutId` используется для хранения идентификатора комплекса, выбранного пользователем в списковом представлении `WorkoutListFragment`. Когда пользователь поворачивает устройство, переменная `workoutId` теряет свое текущее значение и возвращается к значению по умолчанию 0. Затем фрагмент выводит подробную информацию о комплексе с идентификатором 0 — первом в списке.

Для фрагментов эта проблема решается примерно так же, как и для активностей. Сначала вы переопределяете метод `onSaveInstanceState()` фрагмента и помещаете локальную переменную, значение которой требуется сохранить, в параметр `Bundle` метода:

```
public void onSaveInstanceState(Bundle savedInstanceState) {
    savedInstanceState.putLong("workoutId", workoutId);
}
```

После этого значение извлекается из `Bundle` в методе `onCreateView()` фрагмента:

```
if (savedInstanceState != null) {
    workoutId = savedInstanceState.getLong("workoutId");
}
```

Переработанный код приведен на следующей странице.

← Метод `onSaveInstanceState()` вызывается перед уничтожением фрагмента.

← Он может использоваться для восстановления предыдущего состояния переменной `workoutId`.

Kog WorkoutDetailFragment

```
package com.hfad.workout;
```

... ← Новые директивы импорта не нужны, эту часть можно опустить.

```
public class WorkoutDetailFragment extends Fragment {
    private long workoutId;
```

```
@Override
```

```
public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
```

```
    if (savedInstanceState != null) {
```

```
        workoutId = savedInstanceState.getLong("workoutId");
```

```
    }
```

```
    return inflater.inflate(R.layout.fragment_workout_detail, container, false);
```

```
}
```

```
@Override
```

```
public void onStart() {
```

```
    super.onStart();
```

```
    View view = getView();
```

```
    if (view != null) {
```

```
        TextView title = (TextView) view.findViewById(R.id.textTitle);
```

```
        Workout workout = Workout.workouts[(int) workoutId];
```

```
        title.setText(workout.getName());
```

```
        TextView description = (TextView) view.findViewById(R.id.textDescription);
```

```
        description.setText(workout.getDescription());
```

```
    }
```

```
}
```

```
@Override
```

```
public void onSaveInstanceState(Bundle savedInstanceState) {
```

```
    savedInstanceState.putLong("workoutId", workoutId);
```

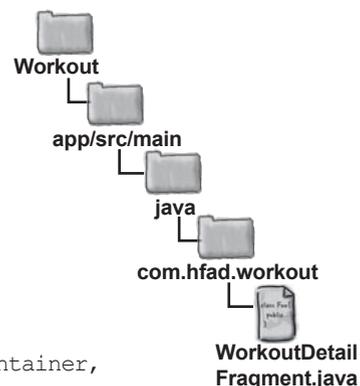
```
}
```

```
public void setWorkout(long id) {
```

```
    this.workoutId = id;
```

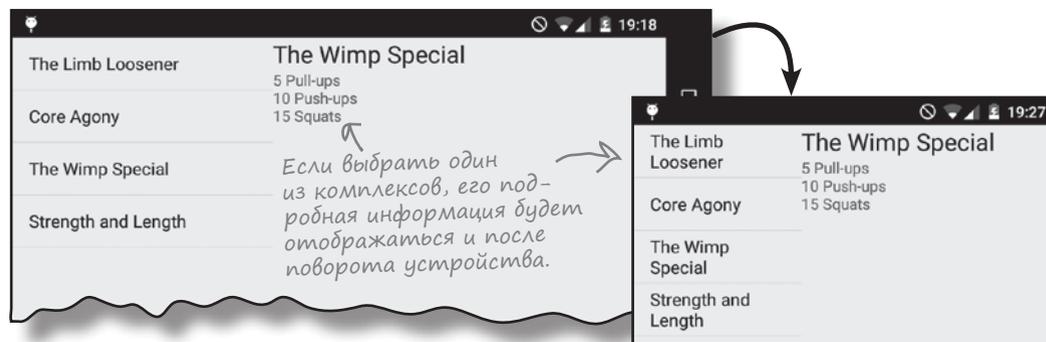
```
}
```

```
}
```



← Задать значение workoutId.

Сохранить значение workoutId в объекте savedInstanceState типа Bundle перед уничтожением фрагмента. Позднее сохраненное значение читается в методе onCreateView().



Телефон и планшет

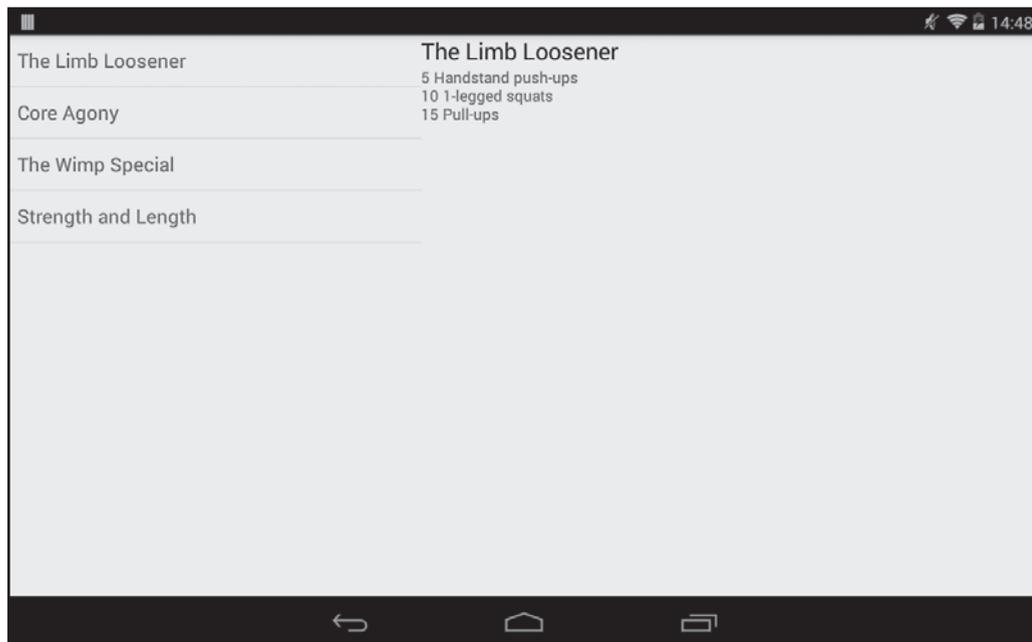


Создание фрагментов
Связывание фрагментов
Макеты для устройств

В приложении Workout осталось решить еще одну задачу. Мы хотим, чтобы приложение по-разному работало в зависимости от того, на каком устройстве оно выполняется — на планшете или на телефоне.

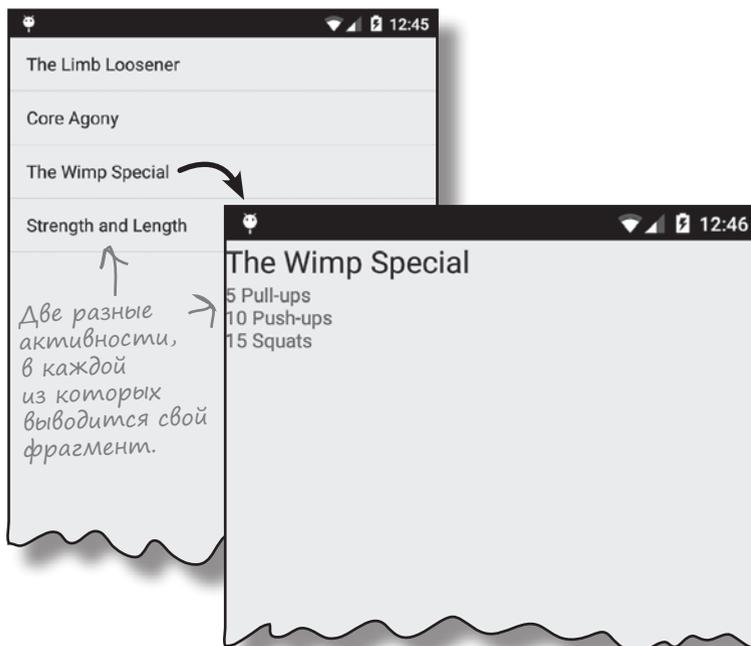
На планшете

Если приложение выполняется на планшете, оно должно выглядеть и вести себя так, как в его текущей версии. Списки комплексов и подробная информация о них должны выводиться рядом друг с другом в одной активности. Если щелкнуть на комплексе, подробная информация выводится на этом же экране.



На телефоне

Если приложение выполняется на телефоне, оно должно работать иначе. Список комплексов упражнений должен отображаться в одной активности и занимать полный экран на устройстве. Щелчок на варианте списка запускает вторую активность с подробной информацией о комплексе.

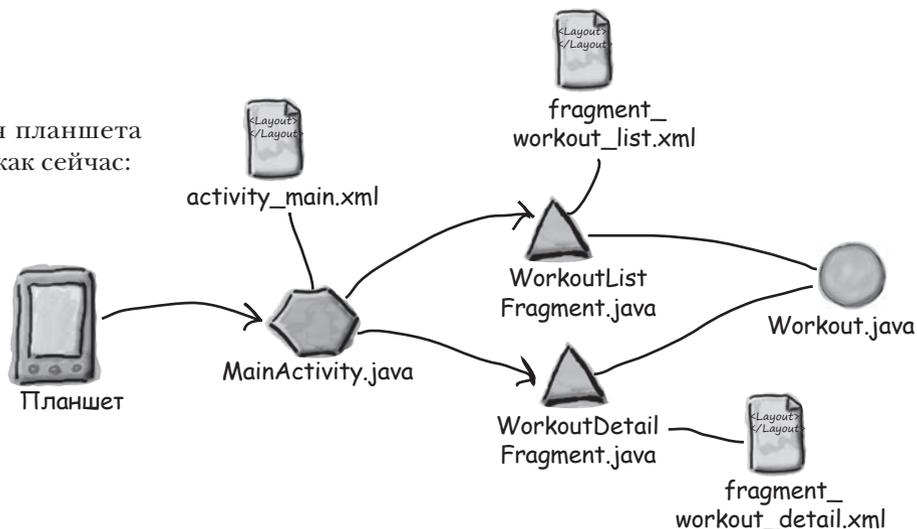


Структура приложения для планшета и телефона

Версии приложения для планшета и телефона должны работать так:

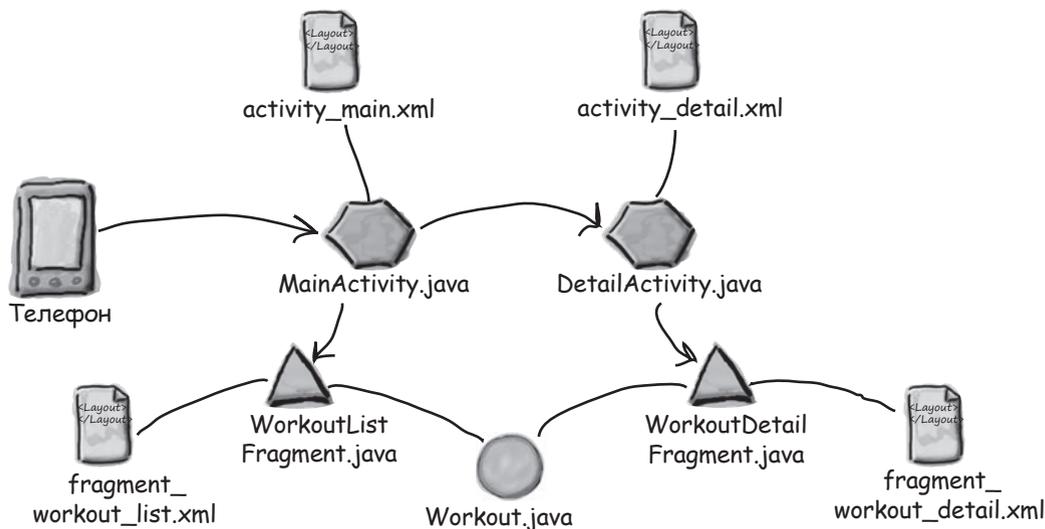
На планшете

Версия приложения для планшета должна работать так же, как сейчас:



На телефоне

Вместо того, чтобы использовать оба фрагмента внутри MainActivity, MainActivity будет использовать WorkoutListFragment, а DetailActivity будет использовать WorkoutDetailFragment. MainActivity запускает DetailActivity при выборе комплекса пользователем.



Итак, приложение должно выглядеть и работать по-разному в зависимости от того, где оно выполняется – на телефоне или на планшете. Давайте посмотрим, как обеспечить выбор макета в зависимости от типа устройства, на котором выполняется приложение.

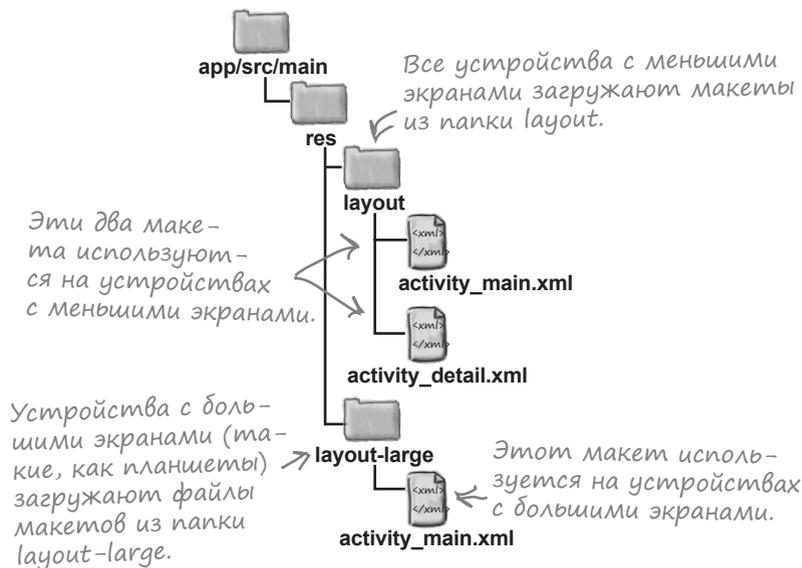


Размещение ресурсов для конкретного типа экрана в специальных папках

Ранее в книге было показано, как использовать разные ресурсы изображений в зависимости от размера экрана — для этого ресурсы размещаются в разных папках *drawable*. Например, изображения, которые должны использоваться на устройствах с большим экраном, находятся в папке *drawable-hdpi*.

Примерно так же можно поступать и с другими ресурсами — макетами, меню и значениями. Если вы хотите создать несколько версий одного ресурса для разных характеристик экрана, достаточно создать несколько папок ресурсов с соответствующими именами. После этого устройство во время выполнения загружает ресурсы из той папки, которая больше всего соответствует характеристикам экрана.

Например, если вы хотите создать один макет для устройств с большим экраном и еще пару макетов для других устройств, макет для устройств с большим экраном помещается в папку *app/src/main/res/layout-large*, а макеты для других устройств — в папку *app/src/main/res/layout*. Когда приложение выполняется на устройстве с большим экраном, устройство использует макет из папки *layout-large*.



На следующей странице описаны различные схемы назначения имен папок ресурсов.

Выбор имен папок

Любые ресурсы (графику, макеты, меню и значения) можно размещать в разных папках, чтобы указать, на каких типах устройств они должны использоваться. Имя папки, предназначенной для конкретного экрана, может включать информацию о размере экрана, плотности пикселей, ориентации и пропорциях; компоненты имени разделяются дефисами. Например, чтобы создать макет, который должен использоваться только на планшетах с очень большим экраном в горизонтальной ориентации, следует создать папку с именем *layout-xlarge-land* и поместить файл макета в эту папку. Некоторые значения, которые могут использоваться в именах папок:

Resource type	Screen size	Screen density	Orientation	Aspect ratio
drawable	-small	-ldpi	-land	-long
layout	-normal	-mdpi	-port	-notlong
menu	-large	-hdpi		↑ long — для экранов с очень большим значением высоты.
mipmap	-xlarge	-xhdpi		
values		-xxhdpi		
		-xxxhdpi		
		-nodpi		
		-tvdpi		

Тип ресурса должен быть указан обязательно.

Плотность пикселей зависит от количества точек на дюйм.

Ресурс *mipmap* используется для значков приложений. В старых версиях Android Studio вместо него используется имя *drawable*.

Для ресурсов, не зависящих от плотности. Используйте *-nodpi* для любых ресурсов изображений, которые не должны масштабироваться (например, папка может называться *drawable-nodpi*).

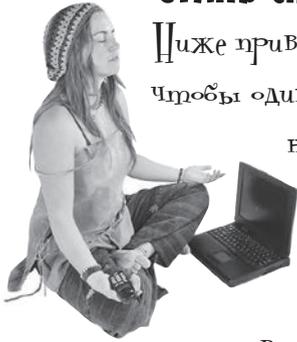
Android во время выполнения решает, какие ресурсы следует использовать, подбирая наиболее близкое соответствие. Если точного совпадения нет, используются ресурсы для размера экрана меньше текущего. Если все ресурсы предназначены только для экранов с размером больше текущего, Android их не использует, и приложение аварийно завершается. Если вы хотите, чтобы приложение работало только на устройствах с конкретным размером экрана, укажите этот факт в файле *AndroidManifest.xml* при помощи атрибута `<supports-screens>`. Например, чтобы запретить запуск приложения на устройствах с маленькими экранами, используйте

```
<supports-screens android:smallScreens="false"/>
```

Используя разные имена папок, можно создавать макеты, предназначенные для телефонов или планшетов. Начнем с планшетной версии приложения.

Дополнительную информацию по этой теме можно найти на странице https://developer.android.com/guide/practices/screens_support.html

СТАНЬ структурой папок



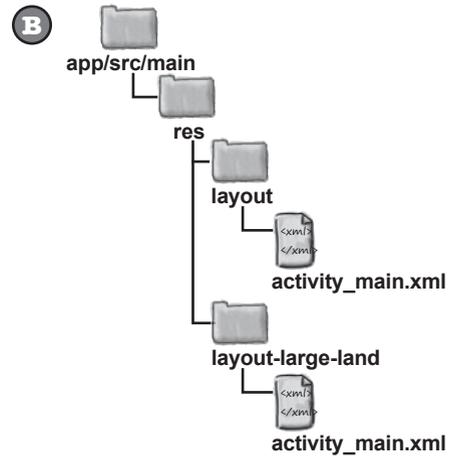
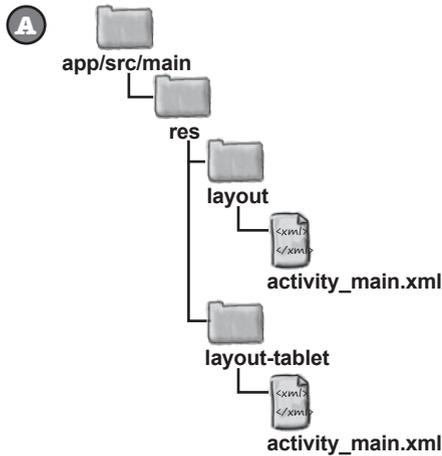
Ниже приведен код активности. Нужно, чтобы один макет отображался при запуске на устройствах с большим размером экрана, а другой — на устройствах с нормальным размером экрана. (какой из структур папок будет выполняться это требование?)

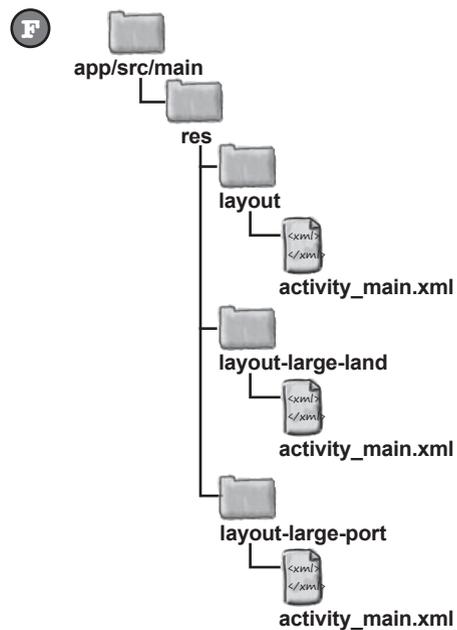
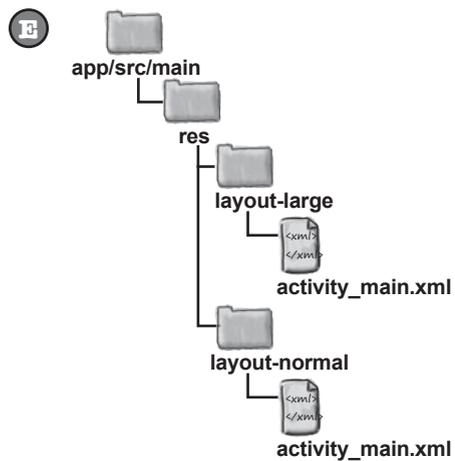
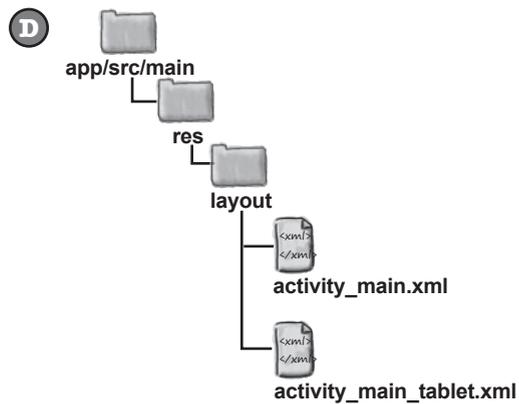
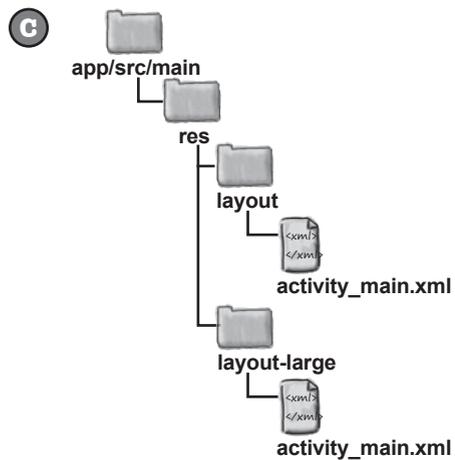
```
import android.app.Activity;
import android.os.Bundle;

public class MainActivity extends Activity {

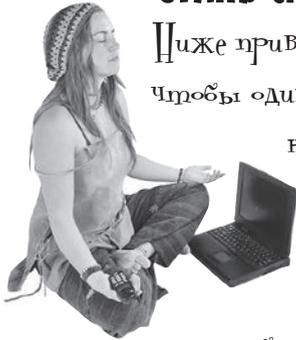
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ...
    }
}
```

← Это активность.





СТАНЬ структурой папок. Решение

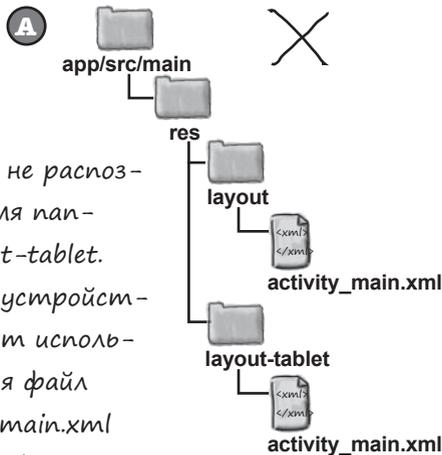


Ниже приведен код активности. Нужно, чтобы один макет отображался при запуске на устройствах с большим размером экрана, а другой — на устройствах с нормальным размером экрана. (Какой из двух структур папок будет выполняться это требование?)

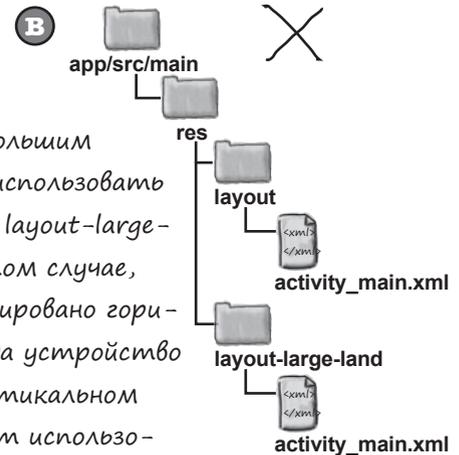
```
import android.app.Activity;
import android.os.Bundle;

public class MainActivity extends Activity {

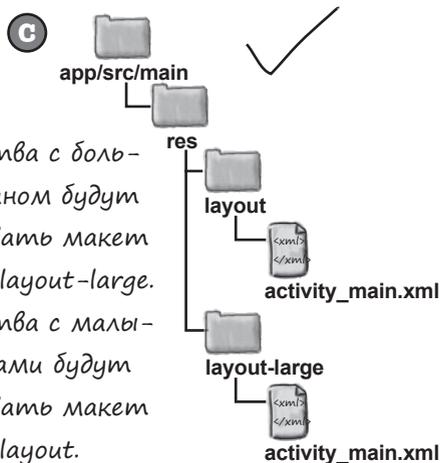
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ...
    }
}
```



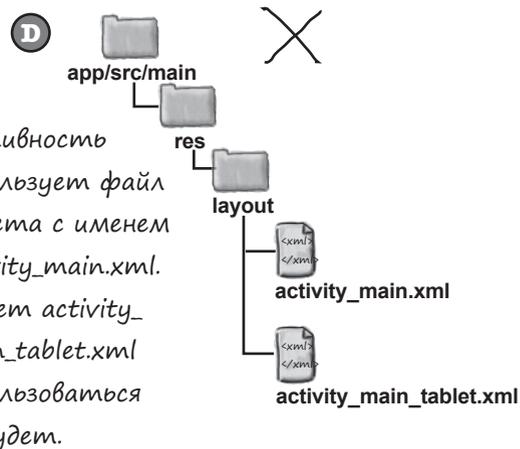
Android не распознает имя папки `layout-tablet`. На всех устройствах будет использоваться файл `activity_main.xml` из папки `layout`.



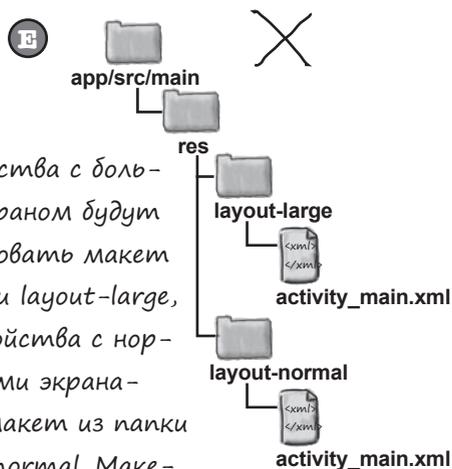
Устройство с большим экраном будет использовать макет из папки `layout-large-land` только в том случае, если оно ориентировано горизонтально. Когда устройство находится в вертикальном положении, будет использоваться макет из папки `layout`.



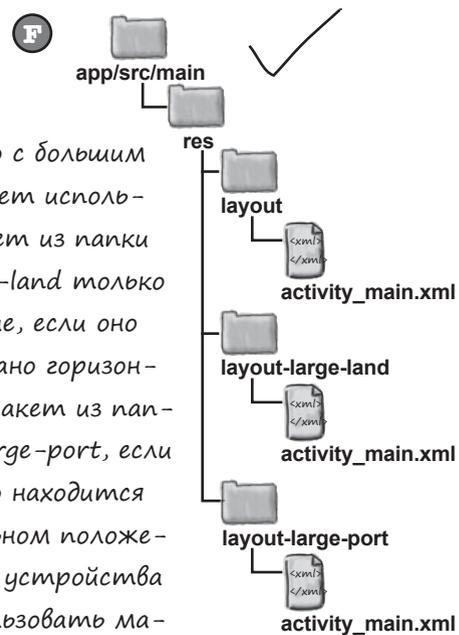
Устройства с большим экраном будут использовать макет из папки `layout-large`. Устройства с маленькими экранами будут использовать макет из папки `layout`.



Активность использует файл макета с именем `activity_main.xml`. Макет `activity_main_tablet.xml` использоваться не будет.



Устройства с большим экраном будут использовать макет из папки `layout-large`, а устройства с нормальными экранами — макет из папки `layout-normal`. Макета для малых экранов нет, поэтому на них приложение работать не будет.



Устройство с большим экраном будет использовать макет из папки `layout-large-land` только в том случае, если оно ориентировано горизонтально, и макет из папки `layout-large-port`, если устройство находится в вертикальном положении. Другие устройства будут использовать макет из папки `layout`.

Планшеты используют макеты из папки layout-large

Получить работоспособную версию приложения для планшета несложно — достаточно поместить существующий файл макета `activity_main.xml` в папку `app/src/main/res/layout-large`. Макет из этой папки будет использоваться только на устройствах с большим экраном.

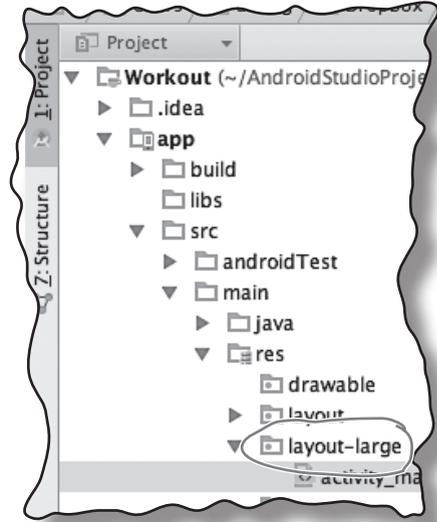
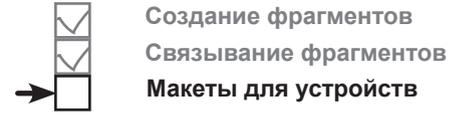
Если папка `app/src/main/res/layout-large` не существует в вашем проекте Android Studio, ее необходимо создать. Для этого перейдите к структуре папок Project, выделите папку `app/src/main/res` на панели структуры и выберите команду `File→New...→Directory`. Когда вам будет предложено ввести имя, введите “`layout-large`”. Если щелкнуть на кнопке ОК, Android Studio создаст папку `app/src/main/res/layout-large`. Чтобы скопировать файл макета `activity_main.xml`, выделите файл на панели структуры и выберите команду `Copy` из меню `Edit`. Выделите новую папку `layout-large` и выберите команду `Paste` из меню `Edit`. Android Studio скопирует файл `activity_main.xml` в папку `app/src/main/res/layout-large`.

Если вы откроете этот файл, он будет выглядеть примерно так:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <fragment
        class="com.hfad.workout.WorkoutListFragment"
        android:id="@+id/list_frag"
        android:layout_width="0dp"
        android:layout_weight="2"
        android:layout_height="match_parent"/>

    <FrameLayout
        android:id="@+id/fragment_container"
        android:layout_width="0dp"
        android:layout_weight="3"
        android:layout_height="match_parent" />
</LinearLayout>
```



Этот макет используется на устройствах с большим экраном, поэтому на планшете два фрагмента будут отображаться рядом друг с другом. Теперь можно заняться макетами для телефона.

Маем MainActivity для телефона

При запуске приложения на телефоне активность MainActivity должна отображать WorkoutListFragment, а не WorkoutDetailFragment. Для этого мы обновим разметку `activity_main.xml` в папке `app/src/main/res/layout`, чтобы макет содержал только `WorkoutListFragment`. Телефоны, на которых запускается приложение, будут использовать макет из папки `layout`, тогда как планшеты будут использовать макет из папки `layout-large`. Откройте файл `activity_main.xml` из папки `app/src/main/res/layout` и замените его содержимое следующей разметкой:

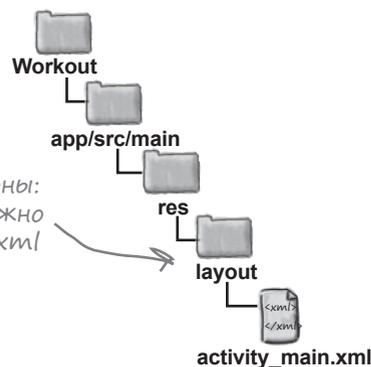
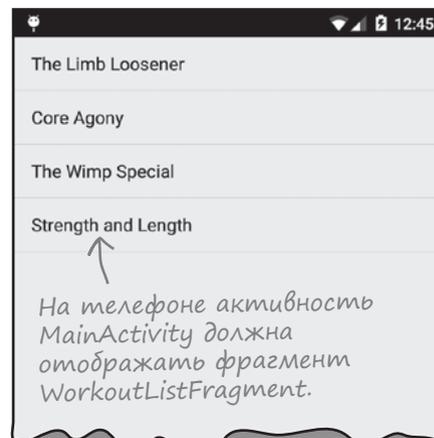
```
<?xml version="1.0" encoding="utf-8"?>
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    class="com.hfad.workout.WorkoutListFragment"
    android:id="@+id/list_frag"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

Будьте внимательны: редактировать нужно файл `activity_main.xml` из папки `layout`.

Так как на телефоне активность MainActivity должна содержать только `WorkoutListFragment`, создавать отдельный макет с элементом `<fragment>` не нужно. Это необходимо только при отображении нескольких фрагментов.

Обратите внимание: версия `activity_main.xml` из папки `layout` не содержит фрейм `fragment_container`, в отличие от версии `activity_main.xml` из папки `layout-large`. Дело в том, что фрагмент `WorkoutDetailFragment` должен отображаться только версией `activity_main.xml` из папки `layout-large`. Позднее мы воспользуемся этим фактом для определения того, какой макет используется приложением на устройстве пользователя.

Следующее, что нужно сделать, — создать вторую активность, использующую `WorkoutDetailFragment`.



РАССЛАБЬТЕСЬ Чтобы приложение по-разному выглядело на телефонах и планшетах, мы используем два разных макета с одинаковыми именами. На нескольких ближайших страницах будьте особенно внимательны и следите за тем, чтобы изменения вносились в нужную версию макета.

Телефоны используем *DetailActivity* для вывода подробной информации

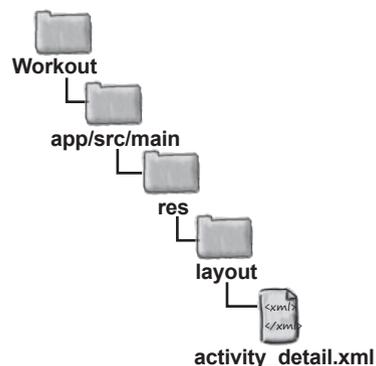
Переходим к созданию второй активности с именем *DetailActivity*. Эта активность содержит фрагмент *WorkoutDetailFragment* и используется телефонами для вывода информации о комплексе упражнений, выбранном пользователем.

Воспользуйтесь мастером Android Studio New Activity для создания новой пустой активности с именем *DetailActivity.java* и макета с именем *activity_detail.xml*. Макет должен находиться в папке *app/src/main/res/layout*, чтобы он был доступен для любого устройства. Макет содержит только фрагмент *WorkoutDetailFragment*. Обновите разметку *activity_detail.xml* следующим образом:

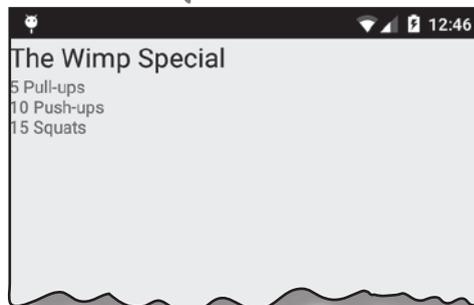
```
<?xml version="1.0" encoding="utf-8"?>
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    class="com.hfad.workout.WorkoutDetailFragment"
    android:id="@+id/detail_frag"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

Кроме обновления макета *activity_detail*, также необходимо внести изменения в код *DetailActivity*. Если приложение выполняется на телефоне, активность *MainActivity* должна запускать *DetailActivity* при помощи интента. Интент включает идентификатор комплекса упражнений, выбранного пользователем, в составе дополнительной информации. Активность *DetailActivity* должна передать его фрагменту *WorkoutDetailFragment* при помощи метода *setWorkout()*.

- Создание фрагментов
- Связывание фрагментов
- Макеты для устройств



← *DetailActivity* выводит только *WorkoutDetailFragment*.



Также необходимо изменить обе эти активности. Начнем с *DetailActivity*.

У бассейна



Выловите сегменты кода из бассейна и расставьте их в пропусках в коде `DetailActivity.java`. Каждый фрагмент может использоваться **только один** раз; использовать все фрагменты не обязательно. Ваша **задача** — получить идентификатор комплекса из интента и передать его `WorkoutDetailFragment`.

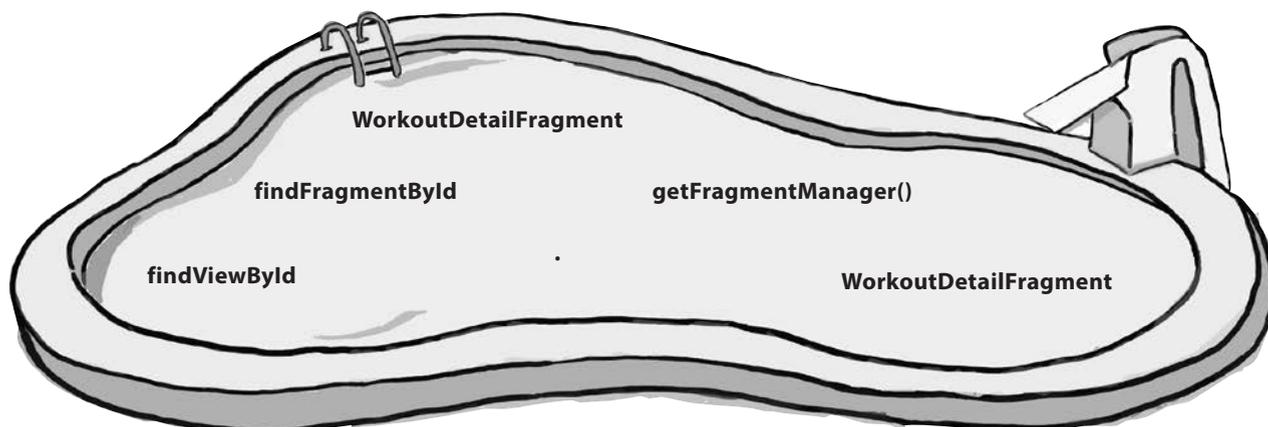
```
package com.hfad.workout;

import android.app.Activity;
import android.os.Bundle;

public class DetailActivity extends Activity {
    public static final String EXTRA_WORKOUT_ID = "id";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_detail);
        ..... workoutDetailFragment = (.....)
        ..... (R.id.detail_frag);
        int workoutId = (int) getIntent().getExtras().get(EXTRA_WORKOUT_ID);
        workoutDetailFragment.setWorkout(workoutId);
    }
}
```

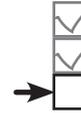
Константа определяет имя дополнительной информации, передаваемой в интенте, чтобы `MainActivity` и `DetailActivity` гарантированно использовали одну и ту же строку.



У бассейна. Решение



Выловите сегменты кода из бассейна и расставьте их в пропусках в коде `DetailActivity.java`. Каждый фрагмент может использоваться **только один** раз; использовать все фрагменты не обязательно. Ваша **задача** — получить идентификатор комплекса из интента и передать его `WorkoutDetailFragment`.



Создание фрагментов
Связывание фрагментов
Макеты для устройств

```
package com.hfad.workout;

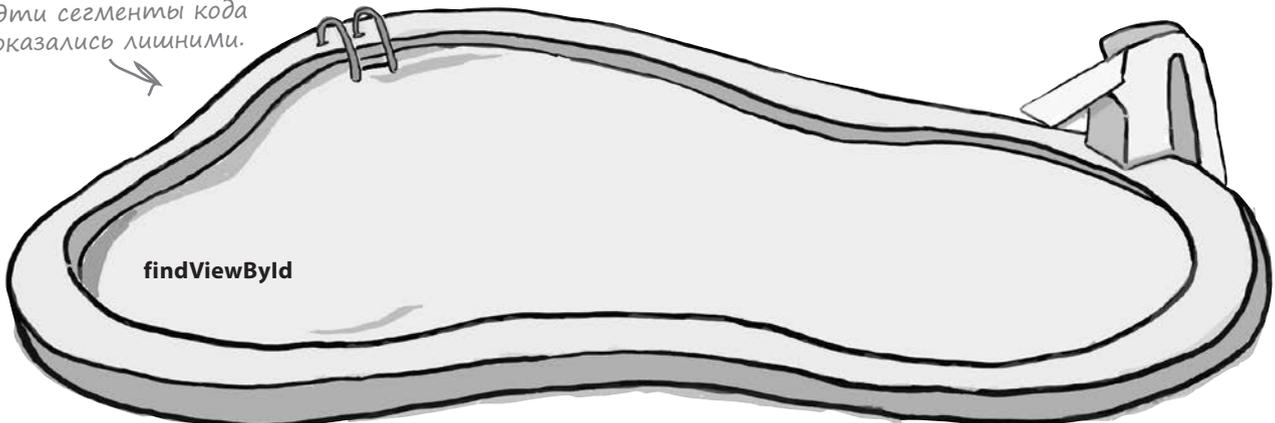
import android.app.Activity;
import android.os.Bundle;

public class DetailActivity extends Activity {
    public static final String EXTRA_WORKOUT_ID = "id";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_detail);
        WorkoutDetailFragment workoutDetailFragment = (WorkoutDetailFragment)
            getFragmentManager().findFragmentById(R.id.detail_frag);
        int workoutId = (int) getIntent().getExtras().get(EXTRA_WORKOUT_ID);
        workoutDetailFragment.setWorkout(workoutId);
    }
}
```

Чтобы получить ссылку на фрагмент, вызываем метод `findFragmentById()` диспетчера фрагментов.

Эти сегменты кода оказались лишними.



Полный код DetailActivity

Ниже приведен полный код `DetailActivity` (замените им код, сгенерированный Android Studio):

```
package com.hfad.workout;

import android.app.Activity;
import android.os.Bundle;

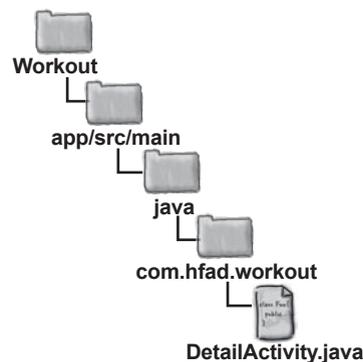
public class DetailActivity extends Activity {
    public static final String EXTRA_WORKOUT_ID = "id";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_detail);
        WorkoutDetailFragment workoutDetailFragment = (WorkoutDetailFragment)
            getSupportFragmentManager().findFragmentById(R.id.detail_frag);
        int workoutId = (int) getIntent().getExtras().get(EXTRA_WORKOUT_ID);
        workoutDetailFragment.setWorkout(workoutId);
    }
}
```

↑
*Передать идентификатор
 комплекса фрагменту.*

↓
*Получить ссылку
 на фрагмент.*

↑
*Получить идентификатор
 комплекса, выбранного поль-
 зователем, из интента.*



Код `DetailActivity` получает идентификатор комплекса из интента, запустившего активность. Следующее, что нужно сделать, — приказать `MainActivity` запустить `DetailActivity`... Но только в том случае, если приложение выполняется на телефоне.

Но как это узнать?

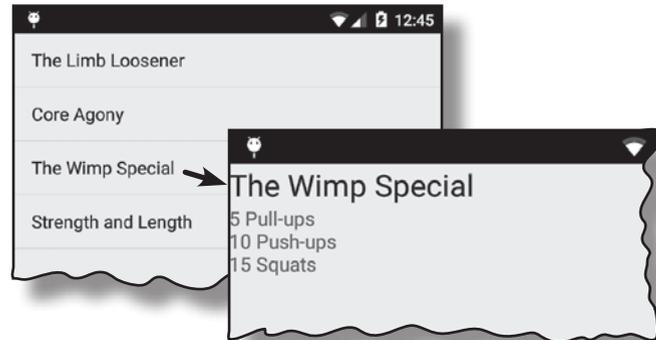
Использование различий в макетах



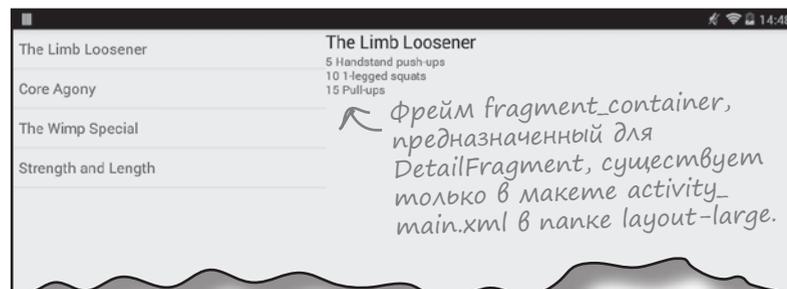
Создание фрагментов
Связывание фрагментов
Макеты для устройств

Активность `MainActivity` должна выполнять разные действия при выборе комплекса упражнений в зависимости от того, какой макет `activity_main.xml` используется приложением — из папки `layout` или из папки `layout-large`.

Если приложение выполняется на телефоне, то устройство использует `activity_main.xml` из папки `layout`. В этом макете отсутствует фрагмент, и если пользователь выбирает вариант в списке, активность `MainActivity` должна запускать `DetailActivity`.



Если приложение выполняется на планшете, то устройство использует `activity_main.xml` из папки `layout-large`. В этот макет входит фрейм с идентификатором `fragment_container`, который используется для отображения `WorkoutDetailFragment`. Если пользователь выбирает вариант в этом случае, нужно вывести новый экземпляр `WorkoutDetailFragment` во фрейме `fragment_container`.



Обе ситуации можно обработать в `MainActivity`; для этого следует проверить, какой макет используется устройством. Такая проверка может быть выполнена поиском представления с идентификатором `fragment_container`.

Если `fragment_container` существует, значит, устройство использует `activity_main.xml` из папки `layout-large`, следовательно при выборе комплекса упражнений следует отобразить новый экземпляр `WorkoutDetailFragment`. Если же `fragment_container` не существует, то устройство использует версию `activity_main.xml` из папки `layout`, поэтому вместо этого запускается `DetailActivity`.

Обновленный код MainActivity

Ниже приведен полный код *MainActivity.java* (внесите изменения, выделенные жирным шрифтом):

```
package com.hfad.workout;

import android.app.Activity;
import android.app.FragmentTransaction;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends Activity
    implements WorkoutListFragment.WorkoutListListener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

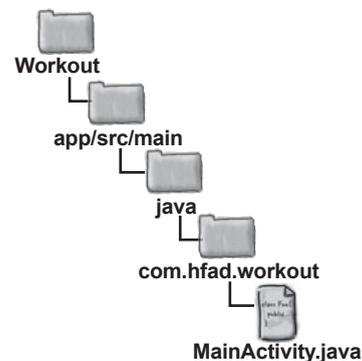
    @Override
    public void itemClicked(long id) {
        View fragmentContainer = findViewById(R.id.fragment_container);
        if (fragmentContainer != null) {
            WorkoutDetailFragment details = new WorkoutDetailFragment();
            FragmentTransaction ft = getFragmentManager().beginTransaction();
            details.setWorkout(id);
            ft.replace(R.id.fragment_container, details);
            ft.addToBackStack(null);
            ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);
            ft.commit();
        } else {
            Intent intent = new Intent(this, DetailActivity.class);
            intent.putExtra(DetailActivity.EXTRA_WORKOUT_ID, (int)id);
            startActivity(intent);
        }
    }
}
```

Этот код должен выполняться только в том случае, если фрейм присутствует в макете.

Эти классы используются в методе *itemClicked()*.

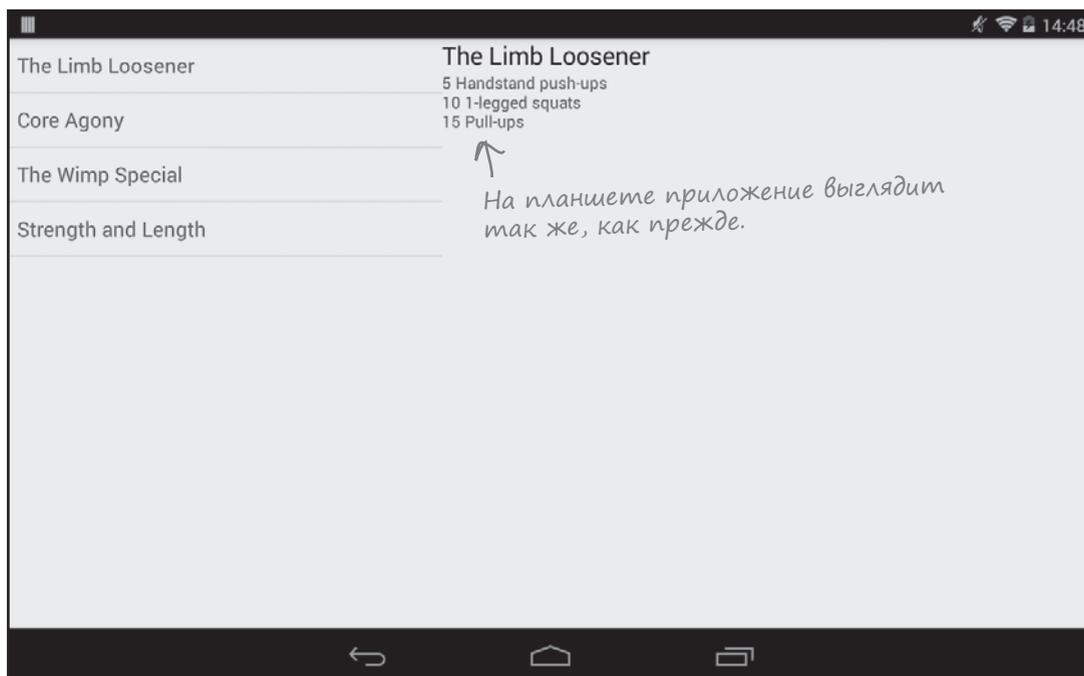
Получить ссылку на фрейм, содержащий *WorkoutDetailFragment*. Он будет существовать, если приложение выполняется на устройстве с большим экраном.

Если фрейм отсутствует, значит, приложение выполняется на устройстве с малым экраном. В этом случае следует запустить активность *DetailActivity* и передать ей идентификатор комплекса.



Посмотрим, что происходит при запуске приложения.

Если запустить приложение на планшете, оно выглядит точно так же, как прежде. В левой части экрана выводится список названий комплексов; если выбрать один из них, справа появляется подробная информация.



Если же запустить приложение на телефоне, на экране появляется список названий комплексов. Если выбрать одно из них, подробная информация отображается в отдельной активности.





Ваш инструментарий Android

Глава 7 осталась позади, а ваш инструментарий пополнился навыками работы с фрагментами.

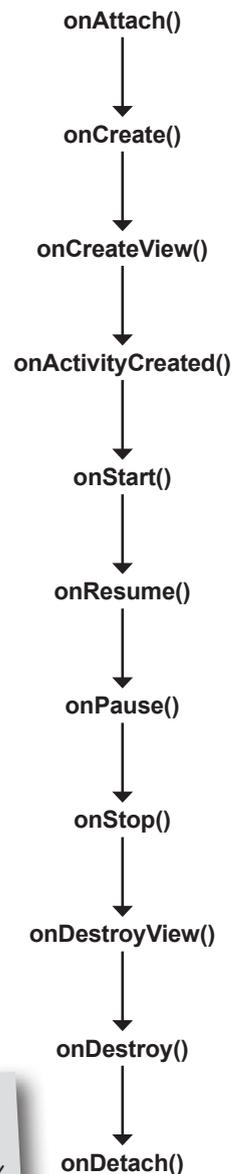
КЛЮЧЕВЫЕ МОМЕНТЫ



- Фрагмент используется для управления частью экрана. Фрагменты подходят для повторного использования в нескольких активностях.
- Фрагмент связывается с макетом.
- Класс фрагмента расширяет класс `android.app.Fragment`.
- Метод `onCreateView()` вызывается каждый раз, когда Android потребуется макет фрагмента.
- Фрагменты добавляются в макет активности при помощи элемента `<fragment>` с добавлением атрибута `class`.
- Методы жизненного цикла фрагмента связываются с состояниями активности, содержащей фрагмент.
- Класс `Fragment` не расширяет класс `Activity` и не реализует класс `Context`.
- У фрагментов отсутствует метод `findViewById()`. Вместо него используйте метод `getView()` для получения ссылки на корневое представление, а затем вызовите метод `findViewById()` представления.
- Списковый фрагмент представляет собой фрагмент, содержащий собственный компонент `ListView`. Списковые фрагменты создаются субклассированием `ListFragment`.
- Если фрагмент должен реагировать на изменения в пользовательском интерфейсе, используйте элемент `<FrameLayout>`.
- Используйте транзакции фрагментов для внесения групп изменений в существующий фрагмент и добавления их в стек возврата.
- Чтобы внешний вид и поведение приложений изменялись в зависимости от устройства, разместите разные макеты в папках, соответствующих типам устройств.

Весь код для этой главы можно загрузить по адресу <https://tinyurl.com/HeadFirstAndroid>.

Методы жизненного цикла фрагментов



Укрощение фрагментов



Кнопка Назад вытворяет не пойми что, в глазах рябит от транзакций.

Тут я вооружаюсь методом `getChildFragmentManager()` — БАБАХ! И все вернулось в норму.

Вы уже видели, что использование фрагментов в активностях способствует повторному использованию кода и делает приложения более гибкими. В этой главе мы покажем, как вложить один фрагмент внутрь другого. Вы научитесь пользоваться диспетчером дочерних фрагментов для укрощения строптивых транзакций фрагментов. А попутно вы узнаете, почему так важно знать различия между активностями и фрагментами.

Создание вложенных фрагментов

В главе 7 вы научились создавать фрагменты, включать их в активности и связывать их друг с другом. Для этого мы создали списковый фрагмент со списком комплексов упражнений и фрагмент с подробной информацией одного комплекса. Однако фрагменты могут содержаться не только в активностях — они могут вкладываться в другие фрагменты. Чтобы показать, как это делается, мы разместим фрагмент с секундомером во фрагменте с подробной информацией комплекса.



WorkoutListFragment содержит список комплексов.

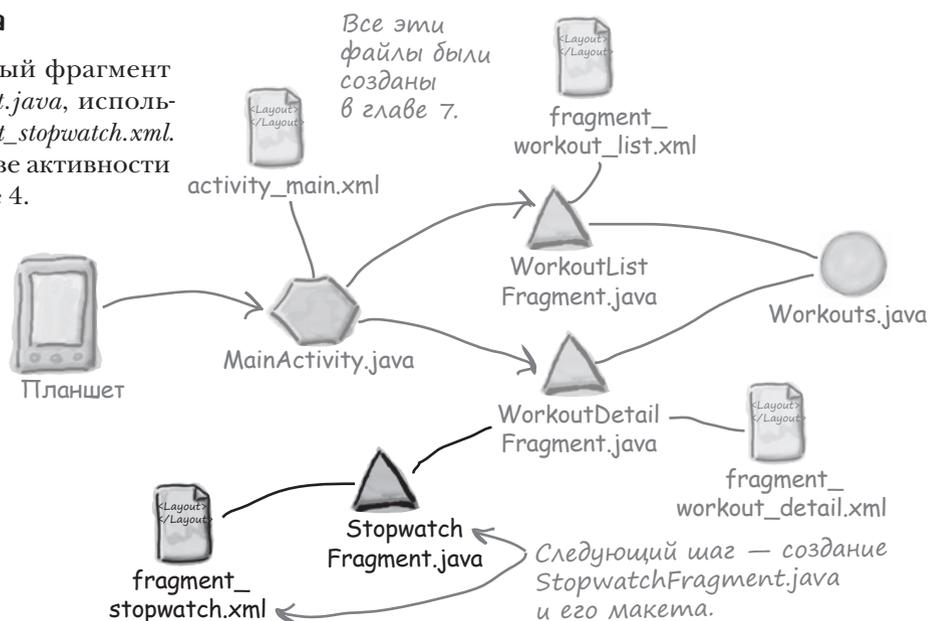
В WorkoutDetailFragment выводится подробная информация о комплексе, выбранном пользователем.

Мы собираемся встроить фрагмент с секундомером в WorkoutDetailFragment.

Добавление нового фрагмента

Мы собираемся добавить новый фрагмент секундомера `StopwatchFragment.java`, использующий макет с именем `fragment_stopwatch.xml`. Фрагмент будет создан на основе активности секундомера, созданной в главе 4.

Мы уже знаем, что у поведения активностей и фрагментов есть много общего, но мы также знаем, что у фрагментов есть своя специфика — они не являются subclasses активностей. **Возможно ли переписать код активности так, чтобы он заработал как фрагмент?**



Жизненные циклы фрагментов и активностей похожи...

Чтобы понять, как переписать активность в виде фрагмента, необходимо немного подумать над тем, чем они похожи, а в чем различны. Рассматривая жизненные циклы фрагментов и активностей, мы видим, что они очень похожи:

Метод жизненного цикла	Активность?	Фрагмент?
<code>onAttach()</code>		✓
<code>onCreate()</code>	✓	✓
<code>onCreateView()</code>		✓
<code>onActivityCreated()</code>		✓
<code>onStart()</code>	✓	✓
<code>onPause()</code>	✓	✓
<code>onResume()</code>	✓	✓
<code>onStop()</code>	✓	✓
<code>onDestroyView()</code>		✓
<code>onRestart()</code>	✓	
<code>onDestroy()</code>	✓	✓
<code>onDetach()</code>		✓

...но методы немного отличаются

Методы жизненного цикла фрагментов почти совпадают с методами жизненного цикла активностей, однако существует одно принципиальное различие: методы жизненного цикла активностей объявлены **защищенными** (`protected`), а методы жизненного цикла фрагментов объявлены **открытыми** (`public`). И мы уже видели, что способ создания макета фрагментами на основе ресурсного файла макета тоже отличается.

Кроме того, во фрагменте нельзя напрямую вызывать такие методы, как `findViewById()`. Вместо этого приходится получать ссылку на объект `View`, а затем вызывать `view.findViewById()`.

Принимая во внимание эти сходства и различия, можно переходить к написанию кода...

Возьми в руку карандаш



ниже приведен код `StopwatchActivity`, написанный нами ранее. Мы преобразуем этот код во фрагмент с именем `StopwatchFragment`. Возьмите карандаш и отметьте все необходимые изменения. Учтите следующие обстоятельства:

- Вместо файла макета с именем `activity_stopwatch.xml` будет использоваться макет с именем `fragment_stopwatch.xml`.
- Убедитесь в том, что методам назначены правильные ограничения доступа.
- Как задать макет?
- Метод `runTimer()` не сможет вызывать `findViewById()`, поэтому стоит рассмотреть возможность передачи объекта представления при вызове `runTimer()`.

```
public class StopwatchActivity extends Activity {
    //Количество секунд на секундомере.
    private int seconds = 0;
    //Секундомер работает?
    private boolean running;
    private boolean wasRunning;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_stopwatch);
        if (savedInstanceState != null) {
            seconds = savedInstanceState.getInt("seconds");
            running = savedInstanceState.getBoolean("running");
            wasRunning = savedInstanceState.getBoolean("wasRunning");
            if (wasRunning) {
                running = true;
            }
        }
        runTimer();
    }

    @Override
    protected void onPause() {
        super.onPause();
        wasRunning = running;
        running = false;
    }
}
```

← Сколько прошло секунд.

← Переменная `running` указывает, работает ли секундомер, а `wasRunning` — работал ли секундомер перед остановкой секундомера.

← Если активность была уничтожена и создана заново — восстановить значения переменных из объекта `savedInstanceState` типа `Bundle`.

← Запустить метод `runTimer()`.

← Остановить секундомер, если активность была приостановлена.

```

@Override
protected void onResume() {
    super.onResume();
    if (wasRunning) {
        running = true;
    }
}

@Override
protected void onSaveInstanceState(Bundle savedInstanceState) {
    savedInstanceState.putInt("seconds", seconds);
    savedInstanceState.putBoolean("running", running);
    savedInstanceState.putBoolean("wasRunning", wasRunning);
}

public void onClickStart(View view) {
    running = true;
}

public void onClickStop(View view) {
    running = false;
}

public void onClickReset(View view) {
    running = false;
    seconds = 0;
}

private void runTimer() {
    final TextView timeView = (TextView) findViewById(R.id.time_view);
    final Handler handler = new Handler();
    handler.post(new Runnable() {
        @Override
        public void run() {
            int hours = seconds / 3600;
            int minutes = (seconds % 3600) / 60;
            int secs = seconds % 60;
            String time = String.format("%d:%02d:%02d",
                hours, minutes, secs);
            timeView.setText(time);
            if (running) {
                seconds++;
            }
            handler.postDelayed(this, 1000);
        }
    });
}
}

```

← Запустить секундомер, если активность продолжает выполнение.

Сохранить состояние активности перед ее уничтожением.

← Запустить, остановить или обнулить секундомер в зависимости от того, какую кнопку нажал пользователь.

Использовать Handler для передачи на ежесекундное выполнение кода, который увеличивает количество секунд и обновляет надпись.

Возьми в руку карандаш



Решение

Ниже приведен код `StopwatchActivity`, написанный нами ранее. Мы преобразуем этот код во фрагмент с именем `StopwatchFragment`. Возьмите карандаш и отметьте все необходимые изменения. Учтите следующие обстоятельства:

- Вместо файла макета с именем `activity_stopwatch.xml` будет использоваться макет с именем `fragment_stopwatch.xml`.
- Убедитесь в том, что методам назначены правильные ограничения доступа.
- Как задать макет?
- Метод `runTimer()` не сможет вызывать `findViewById()`, поэтому стоит рассмотреть возможность передачи объекта представления при вызове `runTimer()`.

Новое имя.

```
public class StopwatchActivity StopwatchFragment extends Activity Fragment {
```

```
    //Количество секунд на секундомере.
```

```
    private int seconds = 0;
```

```
    //Секундомер работает?
```

```
    private boolean running;
```

```
    private boolean wasRunning;
```

```
@Override
```

```
protected public void onCreate(Bundle savedInstanceState) {
```

```
    super.onCreate(savedInstanceState);
```

```
    setContentView(R.layout.activity_stopwatch);
```

```
    if (savedInstanceState != null) {
```

```
        seconds = savedInstanceState.getInt("seconds");
```

```
        running = savedInstanceState.getBoolean("running");
```

```
        wasRunning = savedInstanceState.getBoolean("wasRunning");
```

```
        if (wasRunning) {
```

```
            running = true;
```

```
        }
```

```
    } runTimer();
```

```
}
```

```
@Override
```

```
public View onCreateView(LayoutInflater inflater, ViewGroup container,
```

```
    Bundle savedInstanceState) {
```

```
    View layout = inflater.inflate(R.layout.fragment_stopwatch, container, false);
```

```
    runTimer(layout);
```

```
    return layout;
```

```
}
```

```
@Override
```

```
protected public void onPause() {
```

```
    super.onPause();
```

```
    wasRunning = running;
```

```
    running = false;
```

```
}
```

Расширяет класс `Fragment`, а не `Activity`.

Метод должен быть открытым.

Макет фрагмента уже не назначается в методе `onCreate()`.

Этот код можно оставить в методе `onCreate()`.

Метод `runTimer()` еще не вызывается, потому что макет еще не задан — никаких представлений еще нет.

Макет фрагмента назначается в методе `onCreateView()`.

Представление макета передается при вызове метода `runTimer()`.

Метод должен быть открытым.

```

@Override
protected public void onResume() {
    super.onResume();
    if (wasRunning) {
        running = true;
    }
}

@Override
protected public void onSaveInstanceState(Bundle savedInstanceState) {
    savedInstanceState.putInt("seconds", seconds);
    savedInstanceState.putBoolean("running", running);
    savedInstanceState.putBoolean("wasRunning", wasRunning);
}

public void onClickStart(View view) {
    running = true;
}

public void onClickStop(View view) {
    running = false;
}

public void onClickReset(View view) {
    running = false;
    seconds = 0;
}

private void runTimer(View view) {
    final TextView timeView = (TextView) view.findViewById(R.id.time_view);
    final Handler handler = new Handler();
    handler.post(new Runnable() {
        @Override
        public void run() {
            int hours = seconds / 3600;
            int minutes = (seconds % 3600) / 60;
            int secs = seconds % 60;
            String time = String.format("%d:%02d:%02d",
                hours, minutes, secs);
            timeView.setText(time);
            if (running) {
                seconds++;
            }
            handler.postDelayed(this, 1000);
        }
    });
}
}

```

Метод должен быть открытым.

Метод должен быть открытым.

Теперь методу runTimer()
передается объект View.

Параметр view используется
для вызова findViewById().

Код StopwatchFragment

Добавим фрагмент StopwatchFragment в проект Workout, чтобы использовать его в приложении. Это делается точно так же, как в главе 7: командой File→New...→Fragment→Fragment (Blank). Введите имя фрагмента “StopwatchFragment”, имя макета “fragment_stopwatch” и снимите флажки включения фабричных методов и интерфейсных методов обратного вызова.

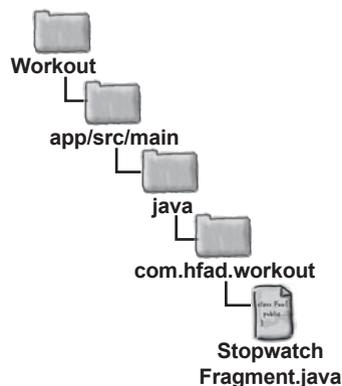
При нажатии кнопки Finish Android Studio создает новый фрагмент в файле с именем *StopwatchFragment.java* в папке *app/src/main/java*. Замените код фрагмента, сгенерированный Android Studio, следующим (это тот самый код, который вы изменяли в упражнении на предыдущей странице):

```
package com.hfad.workout;

import android.os.Bundle;
import android.os.Handler;
import android.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

public class StopwatchFragment extends Fragment {
    //Количество секунд на секундомере.
    private int seconds = 0;
    //Секундомер работает?
    private boolean running;
    private boolean wasRunning;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if (savedInstanceState != null) {
            seconds = savedInstanceState.getInt("seconds");
            running = savedInstanceState.getBoolean("running");
            wasRunning = savedInstanceState.getBoolean("wasRunning");
            if (wasRunning) {
                running = true;
            }
        }
    }
}
```



← Сколько прошло секунд?

← Переменная *running* указывает, работает ли секундомер, а *wasRunning* — работал ли секундомер перед приостановкой секундомера.

← Восстановить значения переменных из объекта *savedInstanceState* муня *Bundle*.

Код StopwatchFragment (продолжение)

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                        Bundle savedInstanceState) {
    View layout = inflater.inflate(R.layout.fragment_stopwatch, container, false);
    runTimer(layout);
    return layout;
}
```

← Назначить макет фрагмента и передать макет при вызове метода `runTimer()`.

```
@Override
public void onPause() {
    super.onPause();
    wasRunning = running;
    running = false;
}
```

← Если фрагмент приостанавливается, сохранить информацию о том, работал ли секундомер на момент приостановки, и остановить отсчет времени.

```
@Override
public void onResume() {
    super.onResume();
    if (wasRunning) {
        running = true;
    }
}
```

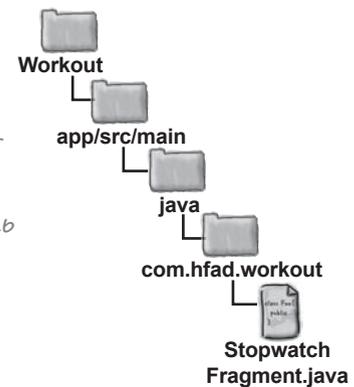
← Если секундомер работал до приостановки, снова запустить отсчет времени.

```
@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
    savedInstanceState.putInt("seconds", seconds);
    savedInstanceState.putBoolean("running", running);
    savedInstanceState.putBoolean("wasRunning", wasRunning);
}
```

← Сохранить значения переменных в `Bundle` перед уничтожением активности. Эти значения используются при повороте устройства.

```
public void onClickStart(View view) {
    running = true;
}
```

↑ Выполняется при нажатии кнопки `Start`.



Продолжение
на следующей
странице.

Код StopwatchFragment (продолжение)

```

public void onClickStop(View view) {
    running = false;
}

public void onClickReset(View view) {
    running = false;
    seconds = 0;
}

private void runTimer(View view) {
    final TextView timeView = (TextView) view.findViewById(R.id.time_view);
    final Handler handler = new Handler();
    handler.post(new Runnable() {
        @Override
        public void run() {
            int hours = seconds / 3600;
            int minutes = (seconds % 3600) / 60;
            int secs = seconds % 60;
            String time = String.format("%d:%02d:%02d",
                hours, minutes, secs);
            timeView.setText(time);
            if (running) {
                seconds++;
            }
            handler.postDelayed(this, 1000);
        }
    });
}
}

```

Выполняется при нажатии кнопки Stop.

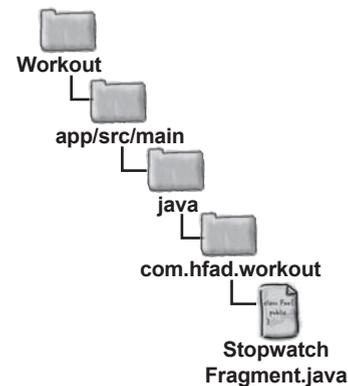
Выполняется при нажатии кнопки Reset.

Код, размещаемый в объекте Handler, может выполняться в фоновом программном потоке.

Вывести количество прошедших секунд.

Если секундомер работает, увеличить число секунд.

Код Handler выполняется каждую секунду.



Это весь код Java, необходимый для StopwatchFragment. Теперь можно сделать следующий шаг — определить внешний вид фрагмента. Для этого мы изменим разметку, сгенерированную Android Studio.

Мажем StopwatchFragment

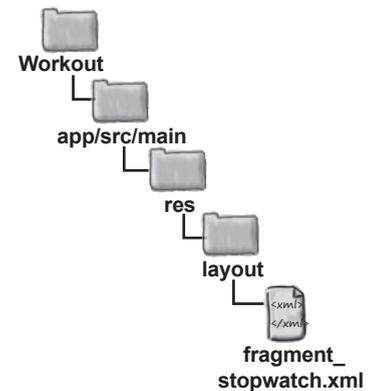
Для фрагмента StopwatchFragment будет использован тот же макет, который использовался в исходном приложении Stopwatch. Замените содержимое `fragment_stopwatch.xml` следующей разметкой:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/time_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="0dp"
        android:text=""
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:textSize="92sp" />

    <Button
        android:id="@+id/start_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/time_view"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="20dp"
        android:onClick="onClickStart"
        android:text="@string/start" />

    <Button
        android:id="@+id/stop_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/start_button"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="10dp"
        android:onClick="onClickStop"
        android:text="@string/stop" />
```



Прошедшее время в часах,
минутах и секундах.

Кнопка Start

Кнопка Stop

Код кнопки
Reset при-
веден на
следующей
странице.



Макет StopwatchFragment (продолжение)

```

<Button
    android:id="@+id/reset_button" ← Кнопка Reset.
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/stop_button"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="10dp"
    android:onClick="onClickReset"
    android:text="@string/reset" />
</RelativeLayout>

```

В макете StopwatchFragment используются строковые значения

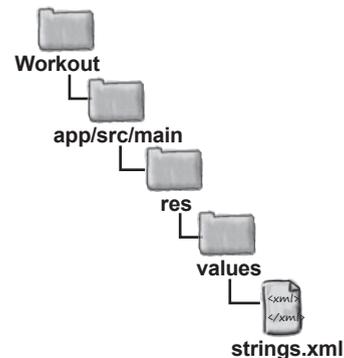
Разметка XML в *fragment_stopwatch.xml* использует строковые ресурсы для текста, выводимого на кнопках Start, Stop и Reset. Эти строки необходимо добавить в *strings.xml*:

```

...
<string name="start">Start</string>
<string name="stop">Stop</string>
<string name="reset">Reset</string>
...

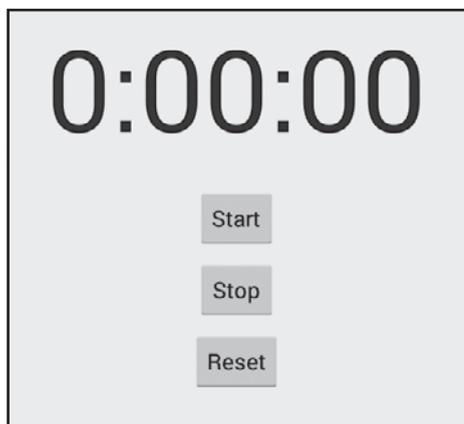
```

Текст, выводимый на кнопках.



По своему внешнему виду фрагмент ничем не отличается от активности. Важно другое: его можно использовать в других активностях и фрагментах.

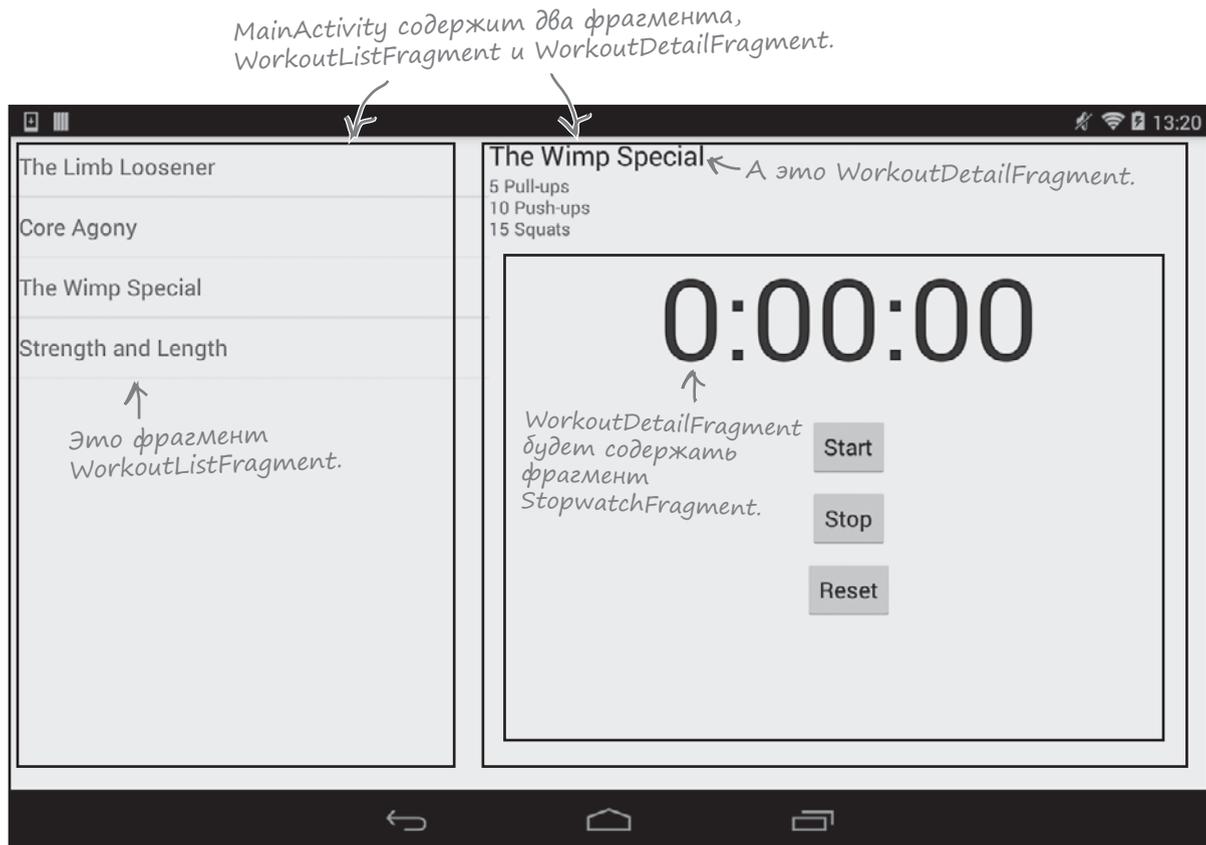
Секундомер выглядит так же, как выглядел в активности. Но теперь он оформлен в виде фрагмента, и его можно будет использовать в других активностях и фрагментах.



Следующее, что нужно сделать, – включить фрагмент при выводе подробной информации о выбранном комплексе упражнений.

Добавление фрагмента в WorkoutDetailFragment

Фрагмент StopwatchFragment будет включен в WorkoutDetailFragment. Пользовательский интерфейс MainActivity на планшете будет выглядеть так:



Фрагмент добавляется на программном уровне

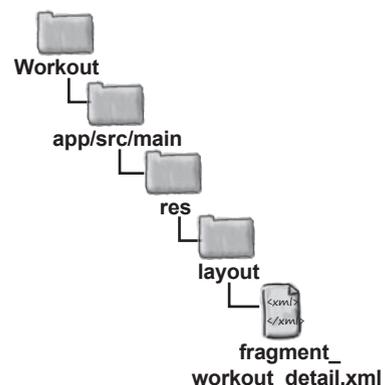
Вы видели два способа добавления фрагментов: с использованием файла макета и из кода Java. Добавление фрагментов в макеты других фрагментов работает не лучшим образом, поэтому для включения StopwatchFragment в WorkoutDetailFragment мы воспользуемся кодом Java. Это означает, что мы будем действовать почти так же, как при добавлении WorkoutDetailFragment в активность. Есть только одно различие, о котором вы вскоре узнаете.

При использовании вложенных фрагментов их добавление должно выполняться на программном уровне.

Добавление фрейма в месте нахождения фрагмента

Как было показано в главе 7, для программного добавления фрагмента из кода Java следует включить фрейм в ту позицию макета, в которой должен располагаться фрагмент.

Фрагмент `StopwatchFragment` нужно добавить в `WorkoutDetailFragment` под названием и описанием комплекса. Мы добавим под надписями с названием и описанием фрейм, в котором будет содержаться фрагмент `StopwatchFragment`:



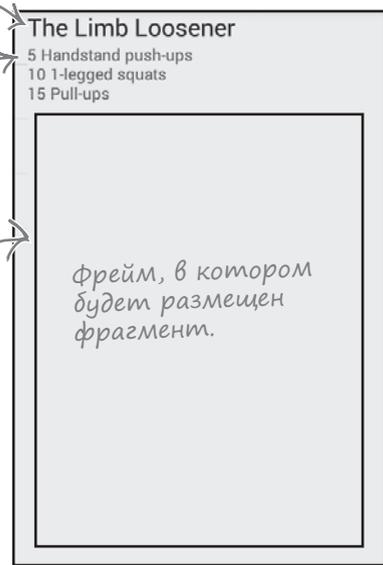
```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_height="match_parent"
    android:layout_width="match_parent"
    android:orientation="vertical">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text=""
        android:id="@+id/textTitle" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text=""
        android:id="@+id/textDescription" />

    <FrameLayout
        android:id="@+id/stopwatch_container"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</LinearLayout>
    
```



После того как фрейм будет включен в макет, необходимо добавить в него фрагмент в коде Java.

Отображение фрагмента в коде Java

Фрагмент `StopwatchFragment` должен добавляться во фрейм при создании представления `WorkoutDetailFragment`. Мы будем делать это так же, как в главе 7: с заменой фрагмента, отображаемого во фрейме, с использованием транзакции фрагмента. Вспомните код, который мы использовали в главе 7:

```
WorkoutDetailFragment details = new WorkoutDetailFragment();
FragmentManager ft = getFragmentManager().beginTransaction();
ft.replace(R.id.fragment_container, details);
ft.addToBackStack(null);
ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);
ft.commit();
```

Создать новый экземпляр фрагмента, который нужно отобразить.

Открыть транзакцию фрагмента.

Заменить фрагмент и добавить его в стек возврата.

Включить растворение/проявление для нового и старого фрагмента.

Закрепить транзакцию.

Приведенный выше код использовался для замены фрагмента, отображаемого в активности, но на этот раз ситуация принципиально иная. Вместо того, чтобы заменять фрагмент, отображаемый в *активности*, мы должны заменить фрагмент, отображаемый в *фрагменте*. Это означает, что в процедуру создания транзакции фрагмента придется внести небольшое изменение.

Когда нам потребовалось отобразить фрагмент в активности, мы создали транзакцию фрагмента при помощи диспетчера фрагмента активности:

```
FragmentManager ft = getFragmentManager().beginTransaction();
```

Возвращает ссылку на диспетчера фрагментов для активности.

Метод `getFragmentManager()` получает диспетчера фрагментов, связанного с *родительской активностью фрагмента*. Это означает, что транзакция фрагмента привязывается к активности.

Чтобы отобразить фрагменты внутри другого фрагмента, вам понадобится немного другой диспетчер фрагментов — а именно диспетчер фрагментов, связанный с *родительским фрагментом*. Это означает, что любые транзакции фрагментов будут привязаны к родительскому фрагменту, а не к активности.

Для получения диспетчера фрагментов, связанного с родительским фрагментом, используется метод `getChildFragmentManager()`. Это означает, что код открытия транзакции выглядит примерно так:

```
FragmentManager ft = getChildFragmentManager().beginTransaction();
```

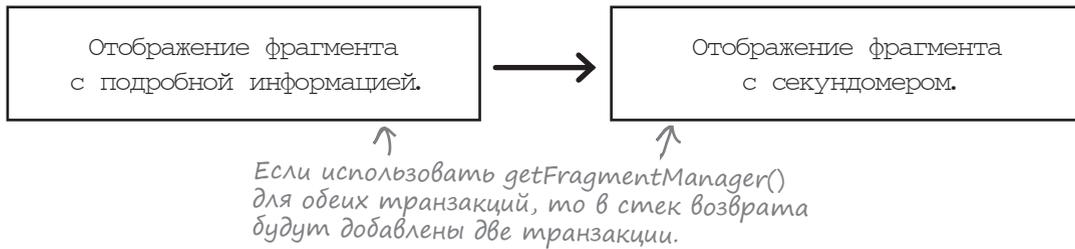
Возвращает ссылку на диспетчера фрагментов для фрагмента.

Что же изменяется при использовании метода `getChildFragmentManager()` с практической точки зрения?

getFragmentManager() создает транзакции на уровне активности

Для начала разберемся, что происходит в том случае, если `WorkoutDetailFragment` использует `getFragmentManager()` для создания транзакции фрагмента, отображающей `StopwatchFragment`.

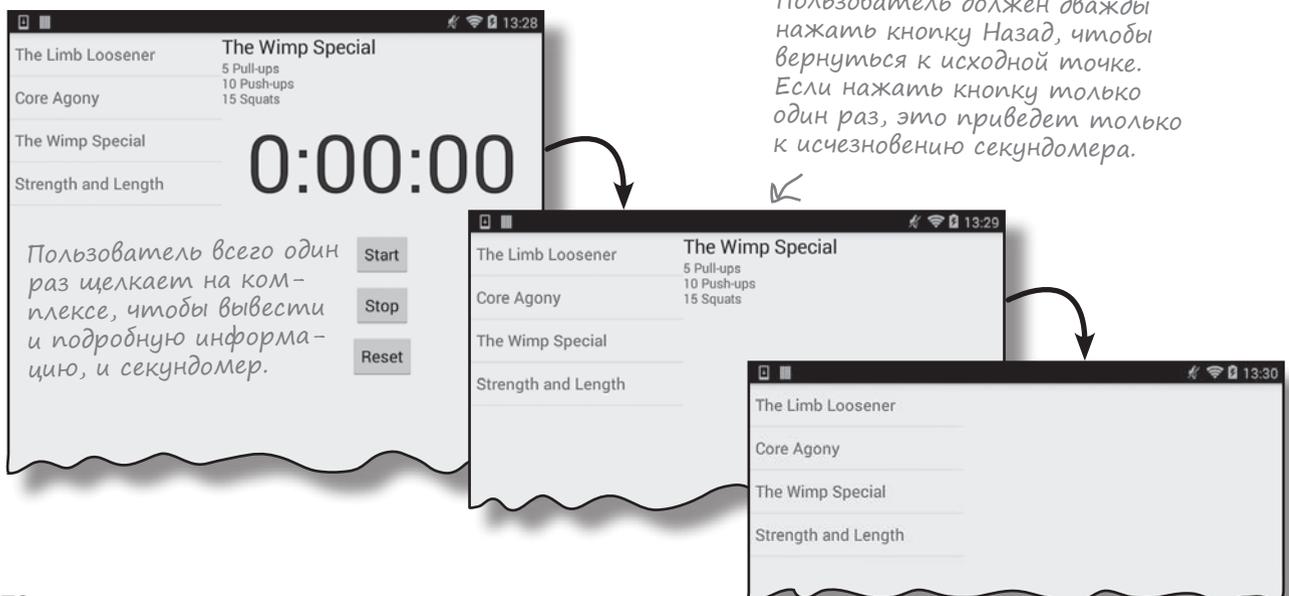
Когда пользователь выбирает комплекс упражнений, приложение должно вывести подробную информацию о комплексе и секундомер. `MainActivity` создает транзакцию для отображения `WorkoutDetailFragment`. Если для отображения `StopwatchFragment` также будет использоваться результат вызова `getFragmentManager()`, в стеке возврата появятся две транзакции.



Внимание: кнопка Назад

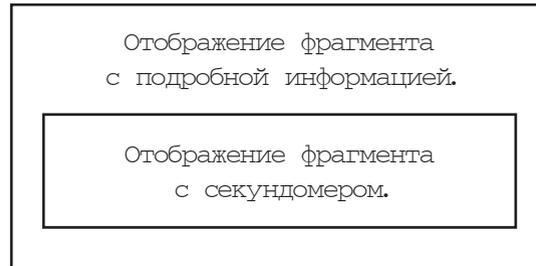
Применение двух транзакций для вывода информации создает одну проблему: при нажатии кнопки Назад происходит нечто странное.

Когда пользователь выбирает комплекс, а затем нажимает кнопку Назад, он ожидает, что экран вернется к предыдущему состоянию. Однако **кнопка Назад отменяет только последнюю транзакцию в стеке возврата**. Это означает, что если создать две транзакции (для вывода подробной информации о комплексе и секундомера), нажатие кнопки Назад приведет только к удалению секундомера. Чтобы удалить раздел подробной информации, пользователю придется нажимать кнопку Назад еще раз.



Вложенным фрагментам — вложенные транзакции

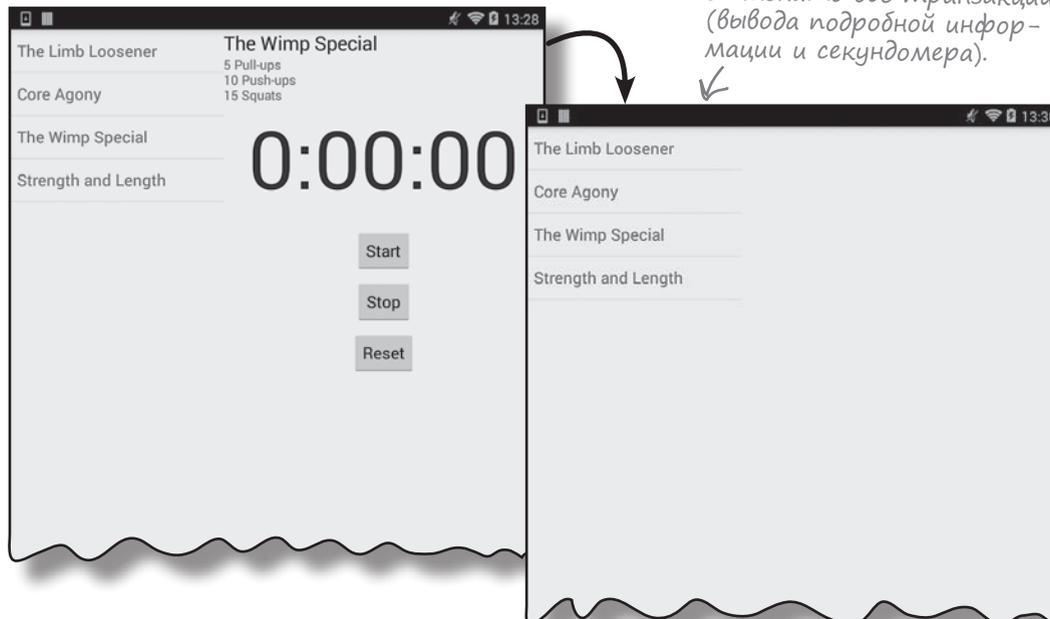
Чтобы понять, почему не стоит создавать несколько транзакций для вложенных фрагментов, следует понять, для чего был создан диспетчер вложенных фрагментов. Транзакции, созданные диспетчером вложенных фрагментов, находятся *внутри* основных транзакций. Таким образом, при добавлении `StopwatchFragment` в `WorkoutDetailFragment` с использованием транзакции, созданной вызовом `getChildFragmentManager().beginTransaction()`, транзакции образуют иерархическую структуру:



← Использование `getChildFragmentManager()` для отображения секундомера означает, что транзакция для фрагмента с секундомером будет вложена в транзакцию фрагмента с подробной информацией.

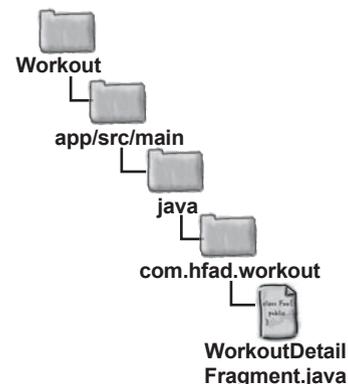
В стек возврата помещается всего одна транзакция, которая содержит вторую вложенную транзакцию. Когда пользователь нажимает кнопку Назад, транзакция фрагмента с подробной информацией отменяется, а это означает, что вместе с ней будет отменена и транзакция фрагмента с секундомером. Теперь при нажатии кнопки Назад приложение ведет себя правильно:

Теперь пользователю достаточно нажать кнопку Назад всего один раз, чтобы отменить обе транзакции (вывода подробной информации и секундомера).



Отображение фрагмента в методе onCreateView() родителя

Фрагмент StopwatchFragment должен добавляться во фрейм при создании представления фрагмента WorkoutDetailFragment. В этот момент вызывается метод onCreateView() фрагмента WorkoutDetailFragment, поэтому мы добавим в метод onCreateView() транзакцию фрагмента для отображения StopwatchFragment. Код реализации выглядит так:



```

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
    if (savedInstanceState != null) {
        workoutId = savedInstanceState.getLong("workoutId");
    }
    FragmentTransaction ft = getChildFragmentManager().beginTransaction();
    StopwatchFragment stopwatchFragment = new StopwatchFragment();
    ft.replace(R.id.stopwatch_container, stopwatchFragment);
    ft.addToBackStack(null);
    ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);
    ft.commit();
    return inflater.inflate(R.layout.detail, container, false);
}
  
```

Добавить транзакцию в стек возврата.

Закрепить транзакцию.

Открыть транзакцию.

← Заменить фрагмент во фрейме.

← Выбрать стиль анимации перехода.

Как видите, код почти идентичен тому, что использовался для отображения фрагмента в активности. Важнейшее различие заключается в том, что фрагмент отображается внутри другого фрагмента, поэтому вместо метода getFragmentManager() должен использоваться метод getChildFragmentManager(). Мы рассмотрим полный код WorkoutDetailFragment на следующей странице, а потом разберемся, как он работает.

Часть Задаваемые Вопросы

В: Я понимаю, что диспетчер вложенных фрагментов справляется с ситуацией, когда один фрагмент помещается внутри другого. Но что, если я вложу один фрагмент в другой, потом еще один внутри него, потом третий, потом четвертый...?

О: Все транзакции образуют цепочку вложений, а на уровне активности останется всего одна транзакция. Таким образом, весь набор вложенных транзакций может быть отменен одним нажатием кнопки Назад.

Полный код WorkoutDetailFragment

Ниже приведен полный код `WorkoutDetailFragment.java`:

```

package com.hfad.workout;

import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;
import android.app.FragmentTransaction;

public class WorkoutDetailFragment extends Fragment {
    private long workoutId;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        if (savedInstanceState != null) {
            workoutId = savedInstanceState.getLong("workoutId");
        }
        FragmentTransaction ft = getChildFragmentManager().beginTransaction();
        StopwatchFragment stopwatchFragment = new StopwatchFragment();
        ft.replace(R.id.stopwatch_container, stopwatchFragment);
        ft.addToBackStack(null);
        ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);
        ft.commit();
        return inflater.inflate(R.layout.fragment_workout_detail, container, false);
    }

    @Override
    public void onStart() {
        super.onStart();
        View view = getView();
        if (view != null) {
            TextView title = (TextView) view.findViewById(R.id.textTitle);
            Workout workout = Workout.workouts[(int) workoutId];
            title.setText(workout.getName());
            TextView description = (TextView) view.findViewById(R.id.textDescription);
            description.setText(workout.getDescription());
        }
    }

    @Override
    public void onSaveInstanceState(Bundle savedInstanceState) {
        savedInstanceState.putLong("workoutId", workoutId);
    }

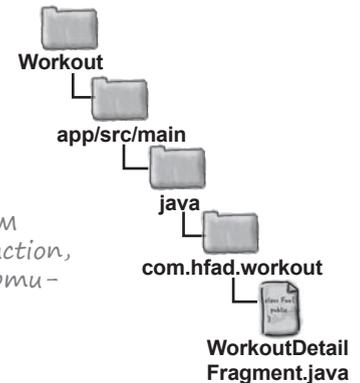
    public void setWorkout(long id) {
        this.workoutId = id;
    }
}

```

Использовать транзакцию фрагмента для добавления фрагмента с секундометром во фрейм.

Так как мы используем класс `FragmentTransaction`, его необходимо импортировать.

Эти методы изменять не нужно.

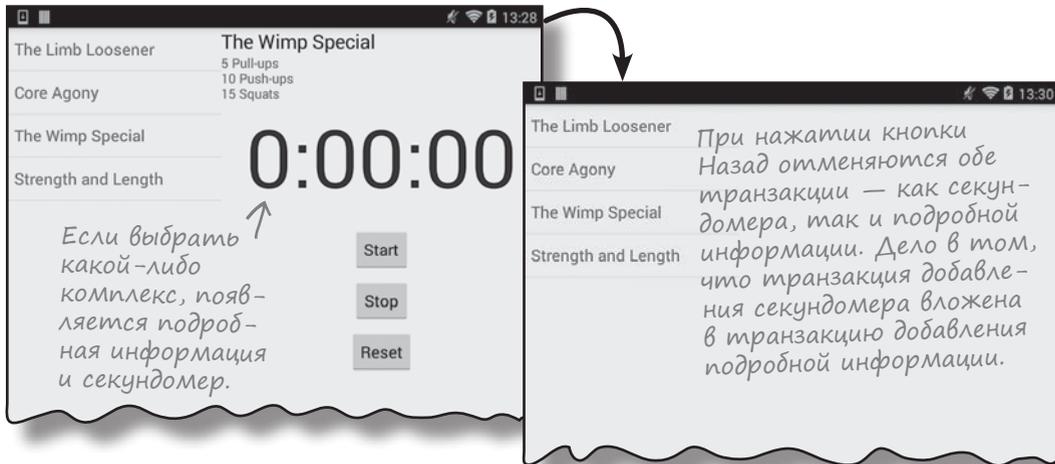




Тест-драйв

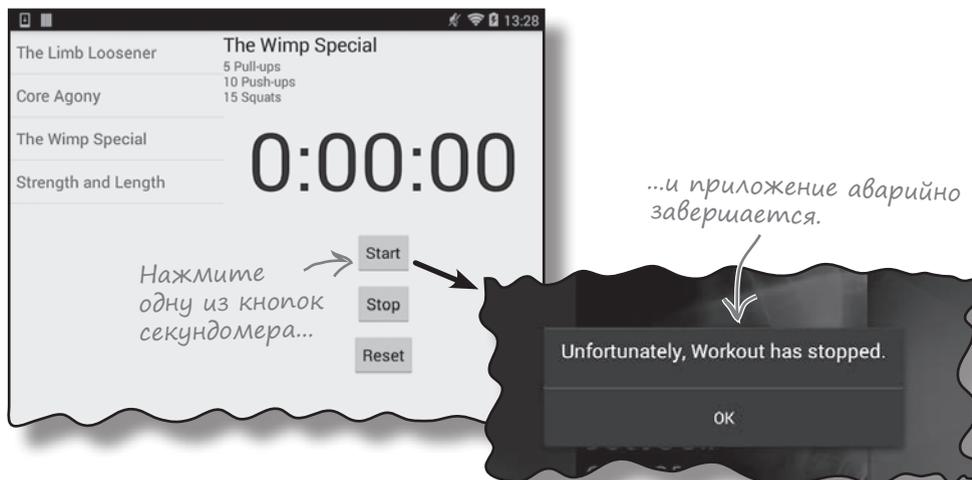
Итак, мы добавили код отображения секундомера. Давайте запустим приложение и посмотрим, как оно работает.

Если выбрать комплекс упражнений в списке, справа появляется подробная информация и секундомер. Если же нажать кнопку Назад, весь экран возвращается к предыдущему состоянию:



Но если вы попытаетесь взаимодействовать с секундомером, обнаруживается проблема.

Если нажать одну из кнопок секундомера, происходит нечто неожиданное — приложение аварийно завершается:



Давайте разберемся, что же пошло не так.

Почему при нажатии кнопки происходит сбой?

Преобразуя активность секундомера во фрагмент, мы не изменили никакой код, связанный с кнопками. Этот код прекрасно работал, когда он был активностью, — почему же во фрагменте он приводит к сбою приложения?

Ниже приведен отладочный вывод Android Studio. Удастся ли вам найти причину возникшей проблемы?

Комментарий
↓

```
01-24 17:37:00.326    2400-2400/com.hfad.fraghack E/AndroidRuntime: FATAL EXCEPTION: main
    Process: com.hfad.fraghack, PID: 2400
    java.lang.IllegalStateException: Could not find a method onClickStart(View) in the activity
    class com.hfad.fraghack.MainActivity for onClick handler on view class android.widget.
    Button with id 'start_button'
        at android.view.View$1.onClick(View.java:3994)
        at android.view.View.performClick(View.java:4756)
        at android.view.View$PerformClick.run(View.java:19749)
        at android.os.Handler.handleCallback(Handler.java:739)
        at android.os.Handler.dispatchMessage(Handler.java:95)
        at android.os.Looper.loop(Looper.java:135)
        at android.app.ActivityThread.main(ActivityThread.java:5221)
        at java.lang.reflect.Method.invoke(Native Method)
        at java.lang.reflect.Method.invoke(Method.java:372)
        at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:899)
        at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:694)
    Caused by: java.lang.NoSuchMethodException: onClickStart [class android.view.View]
        at java.lang.Class.getMethod(Class.java:664)
        at java.lang.Class.getMethod(Class.java:643)
        at android.view.View$1.onClick(View.java:3987)
        at android.view.View.performClick(View.java:4756)
        at android.view.View$PerformClick.run(View.java:19749)
        at android.os.Handler.handleCallback(Handler.java:739)
        at android.os.Handler.dispatchMessage(Handler.java:95)
        at android.os.Looper.loop(Looper.java:135)
        at android.app.ActivityThread.main(ActivityThread.java:5221)
        at java.lang.reflect.Method.invoke(Native Method)
        at java.lang.reflect.Method.invoke(Method.java:372)
        at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:899)
        at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:694)
```

Обратимся к разметке макета StopwatchFragment

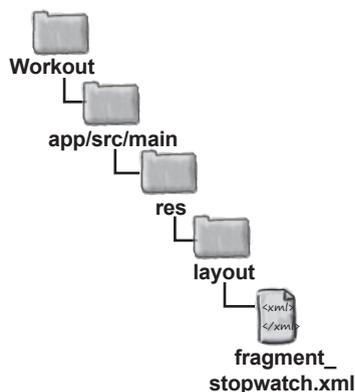
В разметке StopwatchFragment методы связываются с кнопками точно так же, как это делалось для активностей, — атрибут android:onClick определяет, какой метод должен вызываться при нажатии каждой из кнопок:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    ...
    <Button
        android:id="@+id/start_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/time_view"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="20dp"
        android:onClick="onClickStart"
        android:text="@string/start" />

    <Button
        android:id="@+id/stop_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/start_button"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="10dp"
        android:onClick="onClickStop"
        android:text="@string/stop" />

    <Button
        android:id="@+id/reset_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/stop_button"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="10dp"
        android:onClick="onClickReset"
        android:text="@string/reset" />
</RelativeLayout>
```

Для секундомера, реализованного в форме фрагмента, используется тот же макет, который мы использовали для активности.



Для назначения методов, которые должны вызываться при щелчках на кнопках, в макете используются атрибуты android:onClick.

Почему же при использовании фрагмента возникают проблемы?

Атрибут `onClick` вызывает методы активности, а не фрагмента

Использование атрибута `android:onClick` для назначения метода, который должен вызываться при щелчке на представлении, создает одну большую проблему. Атрибут указывает, какой метод должен вызываться в **текущей активности**. Это нормально, когда представления находятся в макете *активности*. Но когда представления находятся во *фрагменте*, возникает проблема. Вместо вызова методов фрагмента Android вызывает методы родительской активности. Если такие методы в активности отсутствуют, в приложении происходит фатальная ошибка. Проблема возникает независимо от того, включен ли фрагмент в активность или же он вложен в другой фрагмент. Она типична для *всех* фрагментов.

Данная проблема проявляется не только при работе с кнопками. Атрибут `android:onClick` может использоваться с любыми представлениями, расширяющими класс `Button`. К этой категории относятся флажки, переключатели, выключатели и двухпозиционные кнопки.

В принципе методы *можно* вынести из фрагмента в активность, но у такого решения есть один серьезный недостаток. Оно означает, что фрагмент перестает быть автономным — если мы захотим использовать фрагмент в другой активности, то код придется также включить и в *эту* активность. Проблему нужно решать на уровне фрагмента.

Как добиться того, чтобы при нажатии кнопок вызывались методы фрагмента

Чтобы кнопки во фрагменте вызывали методы фрагмента, а не методы активности, необходимо сделать две вещи:

- 1 **Удалить упоминания `android:onClick` из макета фрагмента.**
При использовании атрибута `android:onClick` кнопки пытаются вызывать методы активности, поэтому эти атрибуты следует удалить из макета фрагмента.
- 2 **Свяжите кнопки с методами фрагмента посредством реализации интерфейса `OnClickListener`.**
Это гарантирует, что при щелчках на кнопках будут вызываться правильные методы.

Давайте проделаем это во фрагменте `StopwatchFragment`.



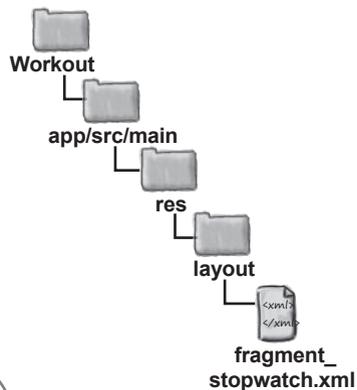
Удаление атрибутов onClick из макета фрагмента

Начнем с удаления строк с android:onClick из макета фрагмента. Тогда Android не будет пытаться вызывать методы активности при нажатиях кнопок:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    ...
    <Button
        android:id="@+id/start_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/time_view"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="20dp"
        android:onClick="onClickStart"
        android:text="@string/start" />

    <Button
        android:id="@+id/stop_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/start_button"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="10dp"
        android:onClick="onClickStop"
        android:text="@string/stop" />

    <Button
        android:id="@+id/reset_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/stop_button"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="10dp"
        android:onClick="onClickReset"
        android:text="@string/reset" />
</RelativeLayout>
```



Удалить атрибуты onClick для каждой из кнопок секундомера.

Следующий шаг — заставить фрагмент реагировать на щелчки на кнопках.

Реализация OnClickListener фрагментом

Чтобы при щелчках на кнопках вызывались методы StopwatchFragment, следует реализовать во фрагменте интерфейс View.OnClickListener:

Фрагмент реализует интерфейс OnClickListener.

```
public class StopwatchFragment extends Fragment implements View.OnClickListener {
    ...
}
```

Фрагмент StopwatchFragment превращается в специализацию View.OnClickListener, что позволяет ему реагировать на щелчки на его представлениях.

Чтобы фрагмент знал, как нужно реагировать на щелчки, реализуйте метод onClick() интерфейса View.OnClickListener. Этот метод вызывается каждый раз, когда пользователь щелкает на представлении во фрагменте.

```
@Override
public void onClick(View v) {
    ...
}
```

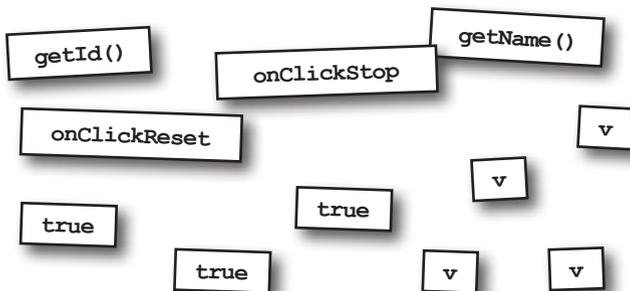
← Метод onClick() должен быть переопределен в коде фрагмента.

Метод onClick() имеет один параметр View. В нем передается представление, на котором щелкнул пользователь. Чтобы узнать, на каком представлении был сделан щелчок, и определить, как на него следует реагировать, используйте метод getId() класса View.



Развлечения с Магнитами

Удастся ли вам завершить код метода onClick() фрагмента StopwatchFragment? При щелчке на кнопке Start должен вызываться метод onClickStart(), на кнопке Stop — метод onClickStop() и на кнопке Reset — метод onClickReset().



```
@Override
public void onClick(View v) {

    switch (..... :.....) {
        case R.id.start_button:

            onClickStart(.....);
            break;
        case R.id.stop_button:

            ..... (.....);
            break;
        case R.id.reset_button:

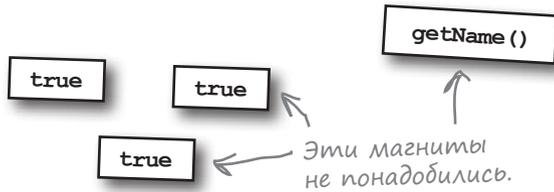
            ..... (.....);

    }
}
```



Развлечения с магнитами. Решение

Удастся ли вам завершить код метода `onClick()` фрагмента `StopwatchFragment`? При щелчке на кнопке `Start` должен вызываться метод `onClickStart()`, на кнопке `Stop` — метод `onClickStop()` и на кнопке `Reset` — метод `onClickReset()`.



`@Override`

```
public void onClick(View v) {
```

```
    switch (v.getId()) {
```

```
        case R.id.start_button:
```

```
            onClickStart(v);
```

```
            break;
```

```
        case R.id.stop_button:
```

```
            onClickStop(v);
```

```
            break;
```

```
        case R.id.reset_button:
```

```
            onClickReset(v);
```

```
    }
```

Метод `onClick()` фрагмента `StopwatchFragment`

Ниже приведен код реализации метода `onClick()` фрагмента `StopwatchFragment`, обеспечивающий вызов правильного метода при щелчке на каждой кнопке:

```
@Override
public void onClick(View v) {
    switch (v.getId()) {
        case R.id.start_button:
            onClickStart(v);
            break;
        case R.id.stop_button:
            onClickStop(v);
            break;
        case R.id.reset_button:
            onClickReset(v);
            break;
    }
}
```

Представление, на котором сделан щелчок.

Проверить, на каком представлении щелкнул пользователь.

Если щелчок сделан на кнопке `Start`, вызвать метод `onClickStart()`.

Если на кнопке `Stop` — вызвать метод `onClickStop()`.

Если на кнопке `Reset` — вызвать метод `onClickReset()`.

Осталось сделать последний шаг: связать слушателя с кнопками фрагмента.

Связывание OnClickListener с кнопками

Чтобы представления реагировали на щелчки, необходимо вызвать метод `setOnClickListener()` каждого представления. В параметре метода `setOnClickListener()` передается объект `OnClickListener`. Так как `StopwatchFragment` реализует интерфейс `OnClickListener`, мы можем использовать `this` для передачи фрагмента в качестве `OnClickListener`.

Например, вот как происходит связывание `OnClickListener` с кнопкой `Start`:

```
Button startButton = (Button) layout.findViewById(R.id.start_button);
startButton.setOnClickListener(this);
```

Получить ссылку на кнопку.
↓

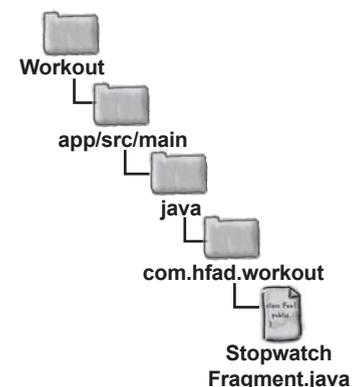
Назначение слушателя для кнопки. ←

Вызовы методов `setOnClickListener()` всех представлений должны совершаться после создания представлений фрагментов. Это означает, что они могут располагаться в методе `onCreateView()` фрагмента `StopwatchFragment`:

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
    View layout = inflater.inflate(R.layout.stopwatch, container, false);
    runTimer(layout);
    Button startButton = (Button) layout.findViewById(R.id.start_button);
    startButton.setOnClickListener(this);
    Button stopButton = (Button) layout.findViewById(R.id.stop_button);
    stopButton.setOnClickListener(this);
    Button resetButton = (Button) layout.findViewById(R.id.reset_button);
    resetButton.setOnClickListener(this);
    return layout;
}
```

Всем кнопкам назначаются слушатели. ↑

Полный код `StopwatchFragment` приведен на следующей странице.



Код StopwatchFragment

Ниже приведен обновленный код `StopwatchFragment.java`:

```

package com.hfad.workout;
    Мы используем класс Button, его необходимо
    ... импортировать.
import android.widget.Button;

public class StopwatchFragment extends Fragment implements View.OnClickListener {
    //Количество секунд на секундомере.
    private int seconds = 0;
    //Секундомер работает?
    private boolean running;
    private boolean wasRunning;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if (savedInstanceState != null) {
            seconds = savedInstanceState.getInt("seconds");
            running = savedInstanceState.getBoolean("running");
            wasRunning = savedInstanceState.getBoolean("wasRunning");
            if (wasRunning) {
                running = true;
            }
        }
    }

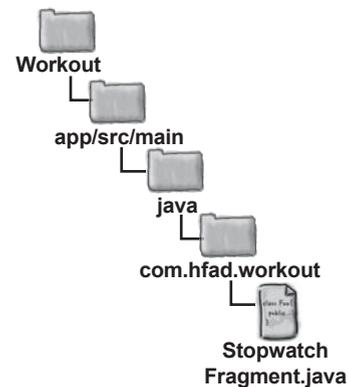
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        View layout = inflater.inflate(R.layout.stopwatch, container, false);
        runTimer(layout);
        Button startButton = (Button) layout.findViewById(R.id.start_button);
        startButton.setOnClickListener(this);
        Button stopButton = (Button) layout.findViewById(R.id.stop_button);
        stopButton.setOnClickListener(this);
        Button resetButton = (Button) layout.findViewById(R.id.reset_button);
        resetButton.setOnClickListener(this);
        return layout;
    }
}

```

Фрагмент должен реализовать интерфейс `View.OnClickListener`.

Метод `onCreate()` остается без изменений.

Изменяется метод `onCreateView()`: слушатели связываются с кнопками.



Код StopwatchFragment (продолжение)

```

@Override
public void onClick(View v) {
    switch (v.getId()) {
        case R.id.start_button:
            onClickStart(v);
            break;
        case R.id.stop_button:
            onClickStop(v);
            break;
        case R.id.reset_button:
            onClickReset(v);
            break;
    }
}
...

public void onClickStart(View view) {
    running = true;
}

public void onClickStop(View view) {
    running = false;
}

public void onClickReset(View view) {
    running = false;
    seconds = 0;
}
...
}

```

← Реализация интерфейса `OnClickListener` требует переопределения метода `onClick()`.

← Вызвать соответствующий метод из фрагмента для той кнопки, на которой был сделан щелчок.

```

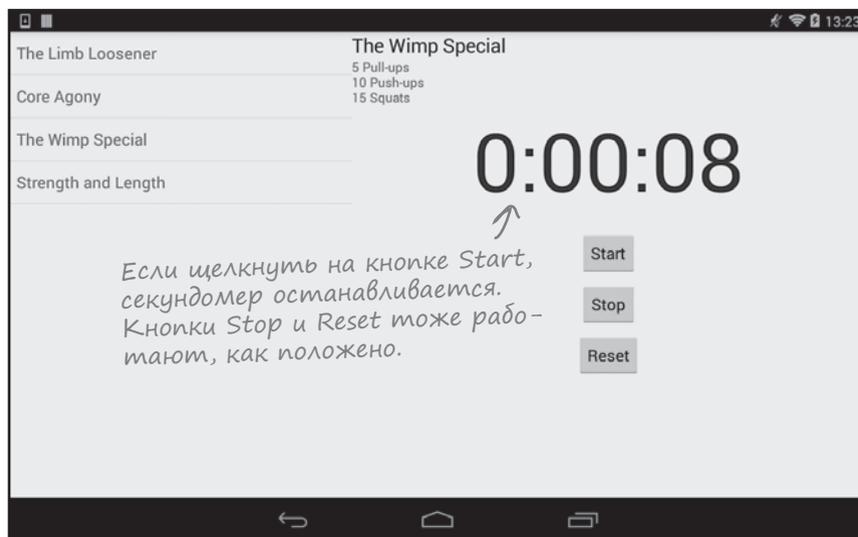
graph TD
    Workout[Workout] --> app_src_main[app/src/main]
    app_src_main --> java[java]
    java --> com_hfad_workout[com.hfad.workout]
    com_hfad_workout --> StopwatchFragment[StopwatchFragment.java]

```

← Те же методы, что и прежде. Они вызываются при щелчках на кнопках.

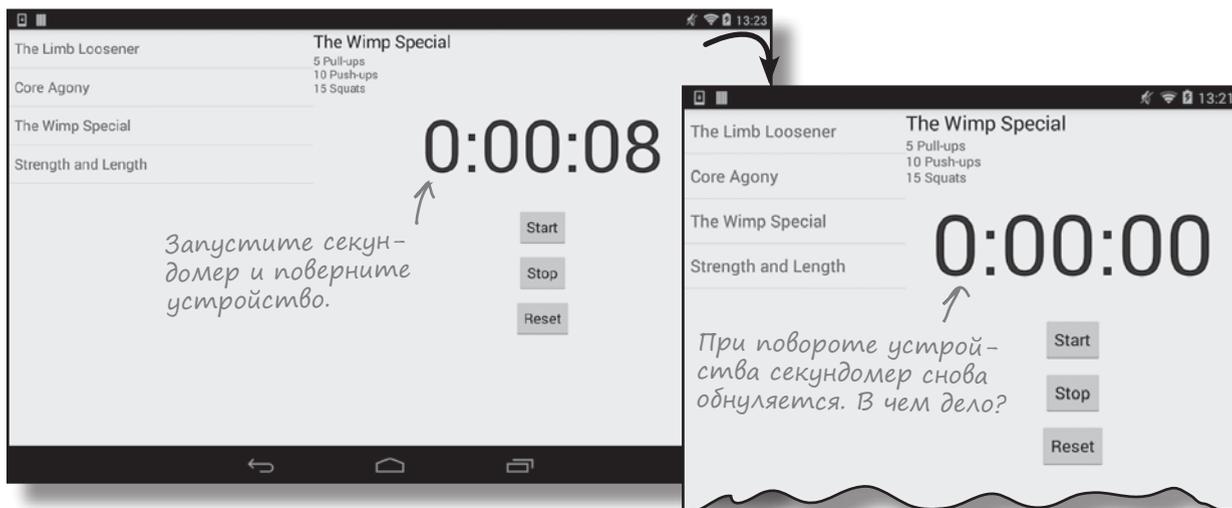
Запустим приложение и посмотрим, что получится.

Теперь при запуске приложения кнопки секундомера работают правильно.



Но при повороте устройства возникает проблема

Если запустить секундомер и повернуть устройство, происходит нечто странное. Показания секундомера снова обнуляются:



Мы уже видели, что изменение ориентации экрана может привести к сбросу представлений. Что же происходит с фрагментами при изменении ориентации?

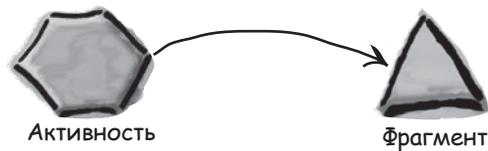
При повороте устройства активность создается заново

Как вам уже известно, если запустить приложение и повернуть устройство, текущая активность будет уничтожена и создана заново. Всем переменным в коде активности возвращаются значения по умолчанию; чтобы сохранить эти значения перед уничтожением активности, необходимо воспользоваться методом `onSaveInstanceState()` активности.

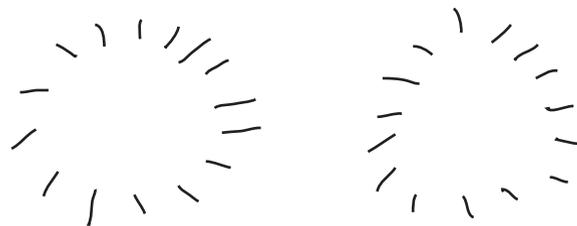
Но что, если активность содержит фрагмент? Вы уже знаете, что жизненные циклы активностей и фрагментов тесно связаны, но что происходит с фрагментом при повороте устройства?

Что происходит с фрагментом при повороте устройства:

- 1 Активность содержит фрагмент.



- 2 Когда пользователь поворачивает устройство, активность уничтожается вместе с фрагментом.



История продолжается...

- 3 **Активность создается заново, вызывается ее метод onCreate().**
Метод onCreate () теперь включает вызов setContentView ().



- 4 **В ходе выполнения метод setContentView() активности читает макет активности и воспроизводит транзакции ее фрагментов.**
Фрагмент создается заново в соответствии с его последней транзакцией.



Когда вы поворачиваете устройство, фрагмент *должен* вернуться к тому состоянию, в котором он находился до поворота. Почему же в нашем случае показания секундомера обнуляются? Чтобы понять это, необходимо обратиться к методу onCreateView () фрагмента WorkoutDetailFragment .

Метод `onCreateView()` выполняется ПОСЛЕ воспроизведения транзакций

Метод `onCreateView()` выполняется после того, как активность воспроизведет все свои транзакции фрагментов. Перед вами код метода. Сможете ли вы объяснить, почему секундомер возвращается к 0 секунд при повороте устройства?

```

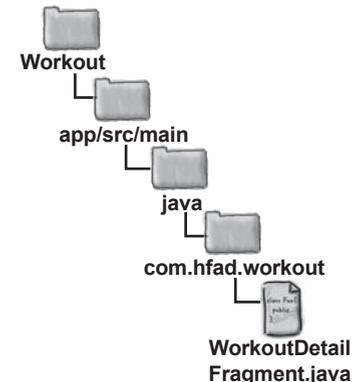
...
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                        Bundle savedInstanceState) {
    if (savedInstanceState != null) {
        workoutId = savedInstanceState.getLong("workoutId");
    }
    Фрагмент с секундомером заменяется новым фрагментом.
    Метод onCreateView() из фрагмента выполняется после того, как активность воспроизведет все свои транзакции фрагментов.
    Выполняется, если фрагмент WorkoutDetailFragment сохранил свое состояние перед уничтожением.
    FragmentTransaction ft = getChildFragmentManager().beginTransaction();
    StopwatchFragment stopwatchFragment = new StopwatchFragment();
    ft.replace(R.id.stopwatch_container, stopwatchFragment);
    ft.addToBackStack(null);
    ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);
    ft.commit();
    return inflater.inflate(R.layout.fragment_workout_detail, container, false);
}
...

```

Метод `onCreateView()` включает транзакцию фрагмента, которая заменяет фрагмент с секундомером новым фрагментом. А это означает, что происходят два события:

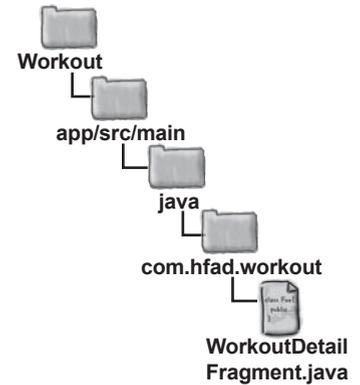
- 1 Активность воспроизводит свои транзакции фрагментов, переводя фрагмент с секундомером в то состояние, в котором он находился перед поворотом устройства.
- 2 Метод `onCreateView()` уничтожает фрагмент с секундомером при воссоздании активности и заменяет его новым фрагментом. Естественно, в новом экземпляре фрагмента показания секундомера возвращаются к 0.

Как же избежать ненужной замены? Замена фрагмента должна происходить только в том случае, если объект с сохраненным состоянием `savedInstanceState` равен `null`. Это будет означать, что новый фрагмент `StopwatchFragment` отображается только при первом создании активности.



Код WorkoutDetailFragment

Ниже приведен полный код *WorkoutDetailFragment.java*:



```

package com.hfad.workout;

...

public class WorkoutDetailFragment extends Fragment {
    private long workoutId;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        if (savedInstanceState != null) {
            workoutId = savedInstanceState.getLong("workoutId");
        } else {
            FragmentTransaction ft = getChildFragmentManager().beginTransaction();
            StopwatchFragment stopwatchFragment = new StopwatchFragment();
            ft.replace(R.id.stopwatch_container, stopwatchFragment);
            ft.addToBackStack(null);
            ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);
            ft.commit();
        }
        return inflater.inflate(R.layout.fragment_workout_detail, container, false);
    }

    @Override
    public void onStart() {
        super.onStart();
        View view = getView();
        if (view != null) {
            TextView title = (TextView) view.findViewById(R.id.textTitle);
            Workout workout = Workout.workouts[(int) workoutId];
            title.setText(workout.getName());
            TextView description = (TextView) view.findViewById(R.id.textDescription);
            description.setText(workout.getDescription());
        }
    }

    @Override
    public void onSaveInstanceState(Bundle savedInstanceState) {
        savedInstanceState.putLong("workoutId", workoutId);
    }

    public void setWorkout(long id) {
        this.workoutId = id;
    }
}

```

Необходимо внести всего одно изменение: поместить транзакцию в команду *else*. Транзакция будет выполняться только в том случае, если ссылка *savedInstanceState* содержит *null*.

Посмотрим, что произойдет при выполнении кода.



Тест-драйв

Запустите приложение, приведите в действие секундомер и поверните устройство. Посмотрим, что произойдет с секундомером.



Секундомер продолжает работать. И хотя при повороте устройства активность уничтожается, транзакции фрагментов воспроизводятся успешно. В этом случае `StopwatchFragment` уже не заменяется новым фрагментом.

Часто задаваемые вопросы

В: Если я оставлю атрибут `android:onClick` в разметке макета фрагмента, будет ли Android пытаться вызывать метод в моей активности?

О: Да, будет. Вместо того, чтобы использовать атрибут `android:onClick` для организации обработки щелчков на представлениях, реализуйте `OnClickListener`.

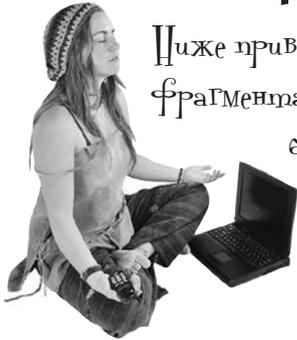
В: Относится ли это только к вложенным фрагментам или же к фрагментам вообще?

О: Это стандартное поведение всех фрагментов независимо от того, вложены они в другие фрагменты или нет.

В: Стоит ли мне использовать фрагменты в своих приложениях?

О: Это зависит от приложения и того, чего вы хотите добиться. Одно из главных преимуществ использования фрагментов — возможность их использования для поддержки разных размеров и пропорций экранов. Например, на планшетах фрагменты могут отображаться рядом друг с другом, а на меньших устройствах — на разных экранах. Другие полезные возможности фрагментов будут описаны в следующих главах...

СТАНЬ фрагментом



Ниже приведены два блока разметки макета фрагмента, а на следующей странице — два блока кода Java. Представьте себя на месте фрагмента и определите, какая комбинация выведет сообщение при установленном выключателе.

A `<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
 xmlns:tools="http://schemas.android.com/tools"
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 android:orientation="vertical"
 tools:context="com.hfad.ch10ex.SwitchFragment">`

```

  <Switch
    android:id="@+id/switch_view"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
</LinearLayout>

```

`</LinearLayout>`

*↑
Две части разметки макета
фрагмента.
↓*

B `<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
 xmlns:tools="http://schemas.android.com/tools"
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 android:orientation="vertical"
 tools:context="com.hfad.ch10ex.SwitchFragment">`

```

  <Switch
    android:id="@+id/switch_view"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="onClick" />
</LinearLayout>

```

`</LinearLayout>`

```

C public class SwitchFragment extends Fragment implements View.OnClickListener{

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment_switch, container, false);
    }

    @Override
    public void onClick(View v) {
        if (v.getId() == R.id.switch_view) {
            if (((Switch) v).isChecked()) {
                Toast.makeText(v.getContext(), "On", Toast.LENGTH_SHORT).show();
            }
        }
    }
}

```

Два блока кода Java
фрагмента.

```

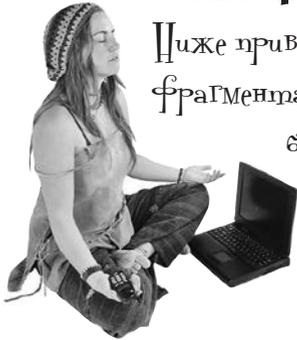
D public class SwitchFragment extends Fragment implements View.OnClickListener{

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View layout = inflater.inflate(R.layout.fragment_switch, container, false);
        Switch switchView = (Switch) layout.findViewById(R.id.switch_view);
        switchView.setOnClickListener(this);
        return layout;
    }

    @Override
    public void onClick(View v) {
        if (v.getId() == R.id.switch_view) {
            if (((Switch) v).isChecked()) {
                Toast.makeText(v.getContext(), "On", Toast.LENGTH_SHORT).show();
            }
        }
    }
}

```

СТАНЬ фрагментом. Решение



Ниже приведены два блока разметки макета фрагмента, а на следующей странице — два блока кода Java. Представьте себя на месте фрагмента и определите, какая комбинация выведет сообщение при установленном выключателе.

A

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.hfad.ch10ex.SwitchFragment">

    <Switch
        android:id="@+id/switch_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</LinearLayout>
```



Правильная разметка.

B

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.hfad.ch10ex.SwitchFragment">

    <Switch
        android:id="@+id/switch_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="onClick" />

</LinearLayout>
```



Выключатель в этом варианте разметки использует атрибут `android:onClick`. Это приведет к тому, что вызываться будет код активности, а не код фрагмента.

C `public class SwitchFragment extends Fragment implements View.OnClickListener{`

```

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                        Bundle savedInstanceState) {
    return inflater.inflate(R.layout.fragment_switch, container, false);
}

```

X

```

@Override
public void onClick(View v) {
    if (v.getId() == R.id.switch_view) {
        if (((Switch) v).isChecked()) {
            Toast.makeText(v.getContext(), "On", Toast.LENGTH_SHORT).show();
        }
    }
}
}

```

Этот код реализует `View.OnClickListener`, но не назначает слушателя для выключателя. Следовательно, метод `onClick()` вызываться не будет.

D `public class SwitchFragment extends Fragment implements View.OnClickListener{`

```

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                        Bundle savedInstanceState) {
    View layout = inflater.inflate(R.layout.fragment_switch, container, false);
    Switch switchView = (Switch) layout.findViewById(R.id.switch_view);
    switchView.setOnClickListener(this);
    return layout;
}

```

✓

```

@Override
public void onClick(View v) {
    if (v.getId() == R.id.switch_view) {
        if (((Switch) v).isChecked()) {
            Toast.makeText(v.getContext(), "On", Toast.LENGTH_SHORT).show();
        }
    }
}
}

```

Правильный код Java. При щелчке на выключателе выполняется метод `onClick()`.



Ваш инструментарий Android

Глава 8 осталась позади, а ваш инструментарий пополнился вложенными фрагментами.

Весь код для этой главы можно загрузить по адресу <https://tinyurl.com/HeadFirstAndroid>.

КЛЮЧЕВЫЕ МОМЕНТЫ



- Фрагменты могут содержать другие фрагменты.
- Если фрагмент вкладывается в другой фрагмент, добавление вложенного фрагмента должно выполняться на программном уровне в коде Java.
- При выполнении транзакций с вложенным фрагментом при создании транзакции следует использовать метод `getChildFragmentManager()`.
- Если использовать атрибут `android:onClick` в фрагменте, Android будет искать метод с этим именем в родительской активности фрагмента.
- Вместо того, чтобы использовать атрибут `android:onClick` во фрагменте, реализуйте во фрагменте интерфейс `View.OnClickListener` и реализуйте его метод `onClick()`.
- При изменении конфигурации устройства активность и ее фрагменты уничтожаются. Когда активность создается заново, транзакции ее фрагментов воспроизводятся при вызове `setContentView()` из метода `onCreate()`.
- Метод `onCreateView()` фрагмента выполняется после того, как активность воспроизведет свои транзакции фрагментов.

В поисках короткого пути



Вот видишь —
я же говорила,
что мы рано или
поздно вернемся
домой.

Да мы бы вернулись
уже час назад, если бы
она знала о существова-
нии кнопки Вверх...

Все мы предпочитаем короткие пути к цели. В этой главе вы узнаете, как ускорить выполнение команд ваших приложений при помощи **панелей действий**. Вы узнаете, как запускать другие активности из **элементов действий** на панели действий, как передавать данные другим приложениям при помощи **провайдера передачи информации** и как перемещаться в иерархии приложения с использованием **кнопки Вверх на панели действий**. Попутно вы узнаете, как обеспечить единый стиль внешнего вида и поведения приложений с использованием **тем оформления**, и познакомитесь с **пакетом библиотеки поддержки Android**.

Хорошее приложение имеет четкую структуру

В главе 6 мы рассматривали способы определения структуры приложений, удобных для пользователя. Напомним, что при создании приложения в основном используются экраны трех видов:

Экраны верхнего уровня

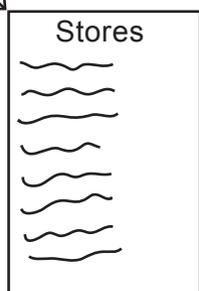
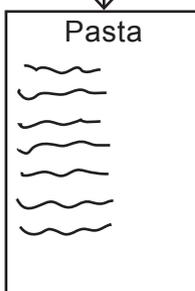
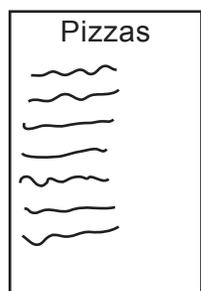
Как правило, это первая активность приложения, которую видит пользователь.



Примитивный эскиз приложения для продажи пиццы. Приложение должно выдавать подробную информацию о разных видах пиццы, о пастах и торговых точках. Также в нем должна быть реализована возможность оформления заказа.

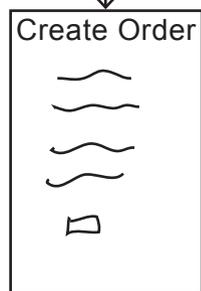
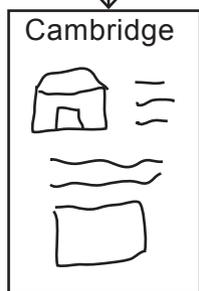
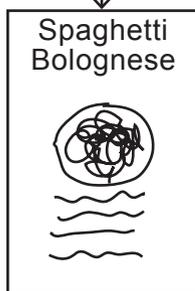
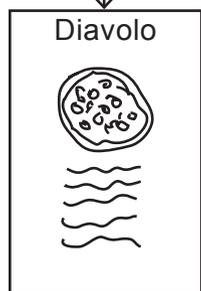
Экраны категорий

На экранах категорий выводятся данные, относящиеся к определенной категории, — часто в виде списка. На них пользователь может перейти к экранам детализации/редактирования.



Экраны детализации/редактирования

На этих экранах выводится подробная информация о конкретных записях, пользователь может редактировать существующие или создавать новые записи.

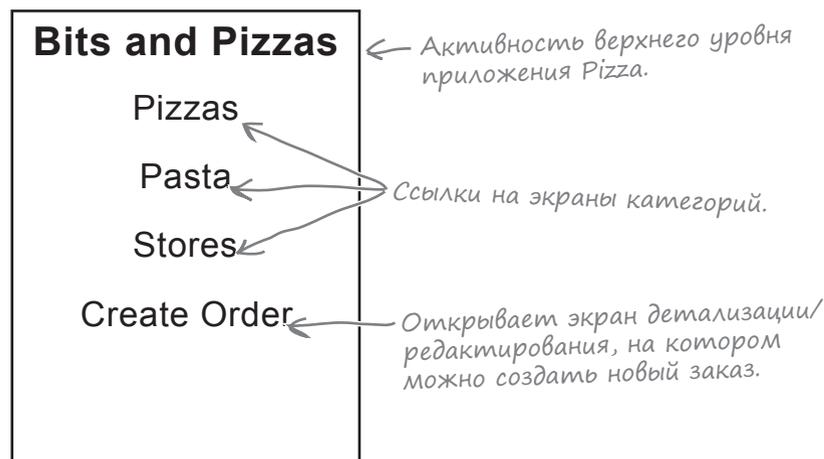


Удобные средства навигации

Любой пользователь, часто работающий с вашим приложением, оценит средства, которые сделают его работу более эффективной. Мы рассмотрим навигационные представления, ускоряющие навигацию в приложении и оставляющие больше места для реального контента приложения. Для начала стоит поближе присмотреться к экрану верхнего уровня в представленном выше приложении Pizza.

Типы навигации

На экране верхнего уровня приложения Pizza находится список разделов приложения, доступных для пользователя.

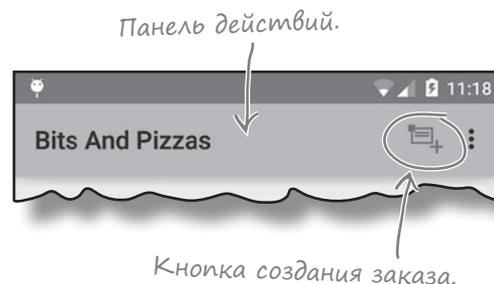


Первые три пункта связываются с активностями категорий; первая открывает список видов пиццы, вторая — список видов пасты, а третья — список магазинов. Активности категорий можно рассматривать как **пассивные**: они выводят информацию и помогают перейти к нужному месту.

Четвертый пункт связан с активностью детализации/редактирования, которая используется для создания заказа. Эта активность является **активной**: она позволяет создать новые данные. Обычно разработчик по-разному обходится с активными и пассивными средствами навигации в приложении. В этой главе рассматривается использование активных средств навигации.

Применение действий для навигации

В приложениях Android активные средства навигации обычно добавляются на панель действий. Эта панель часто располагается в верхней части многих активностей. На ней размещаются наиболее часто используемые действия, поэтому обычно кнопки на панели лучше всего описываются всевозможными глаголами: «Создать», «Найти», «Изменить» и т. д. В приложении Pizza можно упростить задачу размещения заказа для пользователя, добавив панель действий в верхнюю часть любой активности. На панели действий размещается кнопка создания заказа Create Order, которая будет доступна для пользователя в любой точке приложения, а не только в активности верхнего уровня. Давайте посмотрим, как добавить в приложение панель действий.



Начнем с панели действий

Панель действий выполняет в приложениях несколько функций:

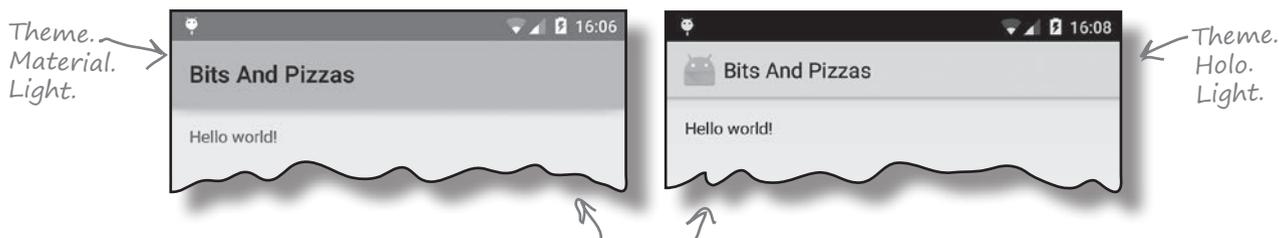
- ★ Она используется для вывода имени приложения или активности, чтобы пользователь знал, в какой точке приложения он находится. Например, в почтовом клиенте на панели действий может выводиться текущая папка (Входящие, Корзина и т. д.).
- ★ Она привлекает внимание пользователя к размещенным на ней ключевым действиям — например, публикации контента или выполнению поиска.
- ★ Панель действий используется для перехода к другим активностям, в которых выполняются нужные действия.

Чтобы добавить в приложение панель действий, необходимо выбрать **тему**, включающую панель действий. Тема (theme) представляет собой визуальный стиль, применяемый ко всей активности или приложению и обеспечивающий целостность его внешнего вида и поведения. Тема управляет такими аспектами, как цвет фона активности и панели действий, оформление текста и т. д. Android содержит набор встроенных тем, которые вы можете использовать.

*Android включает набор встроенных тем.
Полный список приведен в справочной документации Android R.style:
<http://developer.android.com/reference/android/R.style.html>*

API уровня 11 и выше

Если ваши приложения должны работать в API уровня 11 и выше, добавьте панель действий, применяя класс темы `Theme.Holo` или один из его subclasses. В большинстве случаев следует использовать именно этот вариант. Для API уровня 21 и выше имеется дополнительная возможность использования одной из более современных тем `Theme.Material`. Выберите одну из нескольких тем в зависимости от того, какой внешний вид вы хотите придать своему приложению. Например, в результате применения темы `Theme.Material.Light.DarkActionBar` активности будут отображаться со светлым фоном и темной панелью действий.



API уровня 7 и выше

Примеры назначения двух разных тем.

Если ваше приложение должно поддерживать старые устройства с API уровня 7 и выше, вы все равно сможете добавить панель действий, но делается это несколько иначе. Прежде всего придется изменить активности так, чтобы вместо класса `android.app.Activity` они расширяли класс `android.support.v7.app.AppCompatActivity`. После этого применяется одна из тем семейства `Theme.AppCompat`.

Класс `ActionBarActivity` и темы `Theme.AppCompat` входят в библиотеки поддержки **Android**. Познакомимся с ними поближе.

← *Этот способ используется только в том случае, если вы намерены поддерживать старые устройства с API уровнями 7, 8, 9 и 10. На большинстве современных устройств поддерживаются более высокие уровни API.*

Библиотеку поддержку Android

С течением времени в Android появляются новые возможности. Но что, если вы захотите использовать новейшие виджеты Android на устройстве двух-трехлетней давности? В таких случаях можно воспользоваться библиотеками поддержки Android. В основном эти библиотеки обеспечивают обратную совместимость, то есть возможность использования новых возможностей Android на старых устройствах.

Некоторые возможности Android реализованы только в библиотеках поддержки, и если вы захотите использовать их в своих приложениях, без библиотек поддержки не обойтись. Например, `DrawerLayout API` позволяет создать навигационную выдвижную панель, которую можно «вытащить» из-за края экрана; на момент написания книги эта возможность была доступна только в библиотеке поддержки `v4`.

Пакет библиотек поддержки Android содержит несколько библиотек, каждая из которых ориентирована на определенный базовый уровень API и включает конкретную функциональность. В имени библиотеки поддержки содержится минимальный номер версии Android, с которым совместима библиотека. Например, библиотека поддержки `v4` может использоваться с API уровня 4 и выше; библиотека `v7` может использоваться с API уровня 7 и выше. Для каждой из этих библиотек выходят обновления, в которых реализуются новые возможности и исправления ошибок.

Классы библиотек поддержки хранятся в пакетах с именами `android.support.v*`. Например, классы библиотеки `v4` хранятся в пакете `android.support.v4`.

Несколько примеров библиотек из пакета библиотек поддержки Android:

v4 support library

Включает расширенный набор возможностей — например, поддержку компонентов приложений и средств пользовательского интерфейса.

v7 appcompat library

Включает поддержку панелей действий для API уровня 7 и выше, а также поддержку создания и использования дизайна Material.

v7 cardview library

Добавляет поддержку виджета `CardView` для вывода информации в форме карточек.

v7 gridlayout library

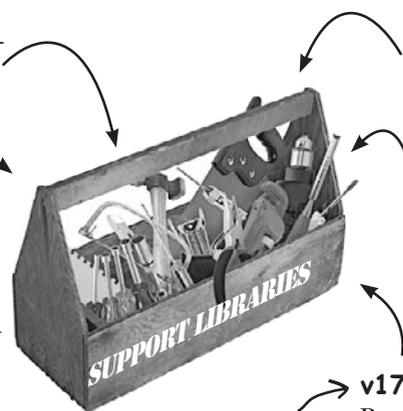
Добавляет поддержку класса `GridLayout`.

v7 recyclerview library

Добавляет поддержку виджета `RecyclerView`.

v17 leanback library

Включает API для построения пользовательских интерфейсов для телевизоров.



Примеры библиотек поддержки.

Android Studio часто включает библиотеки поддержки в проект по умолчанию. Чтобы убедиться в этом, создадим новый проект для прототипа приложения `Pizza` и посмотрим, включает ли проект какие-либо ссылки на них.

Библиотеки поддержки можно включить в ваш проект

Сейчас мы построим прототип приложения Pizza, поддерживающий API уровня 17 и выше. Создайте новый проект Android с именем “Bits and Pizzas” с пустой активностью и именем пакета `com.hfad.bitsandpizzas`. Выберите минимальный уровень API уровня 17. Присвойте активности имя “MainActivity”, макету — имя “activity_main”, а ресурсу меню — имя “menu_main”.

Вероятно, ваш новый проект уже использует библиотеку поддержки по умолчанию. Для начала загляните в файл `MainActivity.java`. Ниже приведен код, сгенерированный Android Studio. По умолчанию MainActivity расширяет класс `android.support.v7.app.AppCompatActivity` — то есть используется библиотека поддержки v7:

```
package com.hfad.bitsandpizzas;

import android.support.v7.app.AppCompatActivity;
...
public class MainActivity extends AppCompatActivity {
    ...
}
```

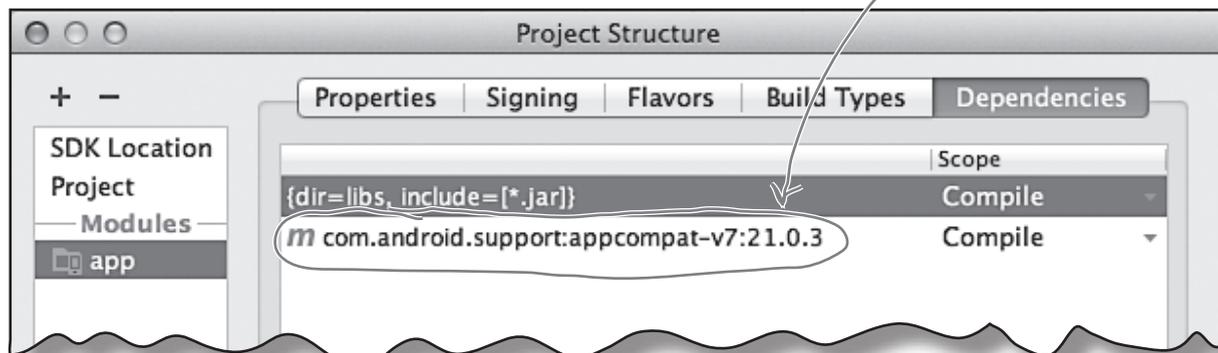
Префикс `android.support.v7` в директиве импортирования `ActionBarActivity` сообщает, что класс принадлежит библиотеке v7 `appcompat`.

Возможно, ваш код `MainActivity.java` выглядит иначе. Это зависит от поведения используемой среды разработки.

Класс `ActionBarActivity` используется в сочетании с темами `Theme.AppCompat` для добавления панелей действий в приложения, поддерживающие уровни API с 7 по 10. Если вы используете `ActionBarActivity` как суперкласс для своих активностей, вы должны выбрать одну из этих тем; в противном случае приложение работать не будет. Более современные темы — такие, как `Material`, — использовать нельзя.

Даже если вы удалите из приложения упоминания `ActionBarActivity`, библиотека поддержки v7 все равно останется среди зависимостей вашего проекта. Чтобы убедиться в этом, выполните команду `File→Project Structure`. Щелкнув на модуле `app` и выбрав команду `Dependencies`, вы найдете среди зависимостей ссылку на библиотеку v7 `appcompat`:

Android Studio автоматически добавляет библиотеку v7 `appcompat` в состав зависимостей. Впрочем, ее здесь может и не быть — это зависит от используемой версии Android Studio.



Использование современных тем в приложении themes

В нашем приложении должны использоваться панели действий. Приложение поддерживает устройства, работающие с уровнем API от 17 и выше, поэтому обеспечивать обратную совместимость использованием `ActionBarActivity` и `Theme.AppCompat` не придется. Чтобы придать приложению более современный вид, мы будем использовать тему `Holo` по умолчанию и переключимся на тему `Material`, если приложение работает на устройстве с API уровня 21. Для этого необходимо:

- 1 **Убедиться в том, что в коде активности отсутствуют упоминания `ActionBarActivity`.**
В противном случае использоваться может только тема `Theme.AppCompat`.
- 2 **Применить темы.**
Приложение должно выбрать тему, соответствующую уровню API, на котором оно работает.

Мы оставим зависимость от библиотеки `v7-appcompat`, так как она повлияет на код, который будет написан позднее.

Наследование `MainActivity` от `Activity`

Для начала нужно убедиться в том, что `MainActivity.java` использует класс `Activity`, а не `ActionBarActivity`. Ваш код должен выглядеть так:

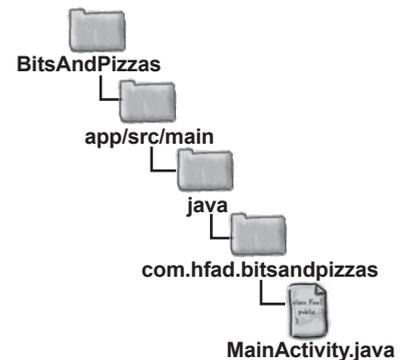
```
package com.hfad.bitsandpizzas;

import android.app.Activity;
import android.os.Bundle;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Проверьте, что активность расширяет `Activity`, а не `ActionBarActivity`. Если вы используете `ActionBarActivity`, то не сможете выбрать темы `Holo` или `Material` — Android заставляет использовать тему `AppCompat`.



Итак, мы знаем, что `MainActivity` не использует `ActionBarActivity`; теперь посмотрим, как применить тему.

Применение темы в AndroidManifest.xml

Как вы уже видели, файл *AndroidManifest.xml* приложения содержит важнейшую информацию о приложении — например, о содержащихся в нем активностях. Также файл включает ряд атрибутов, влияющих на поведение панелей действий.

Ниже приведена разметка *AndroidManifest.xml*, сгенерированная Android Studio us (ключевые места выделены жирным шрифтом):



```

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.hfad.bitsandpizzas" >
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            ...
        </activity>
    </application>
</manifest>

```

Значок приложения. Android Studio предоставляет значок по умолчанию.

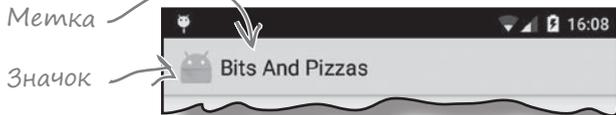
Имя приложения (в форме, удобной для пользователя).

Тема.

Имя активности (в форме, удобной для пользователя).

Метка

Значок



Атрибут **android:icon** определяет значок приложения. Значок используется на экране устройства для запуска приложения; кроме того, он будет выводиться на панели действий приложения, если это предусмотрено выбранной темой приложения. Значок может быть либо обычным графическим объектом (drawable), либо ресурсом mipmap. Изображения типа mipmap используются для значков приложения и хранятся в папках *mipmap** в *app/src/main/res*. Как и в случае с объектами drawables, вы можете добавить несколько версий изображений для разных вариантов плотности экрана, разместив их в папках *mipmap* с соответствующими именами. Например, значок в папке *mipmap-hdpi* будет использоваться устройствами с экранами высокой плотности. Для ссылок на ресурсы mipmap в макете используется запись @mipmap. Атрибут **android:label** назначает метку — имя, удобное для пользователя, — приложению или активности (в зависимости от того, используется ли он в элементе <application> или <activity>). На панели действий выводится метка текущей активности. Если текущей активности не назначена метка, вместо нее выводится метка приложения.

Android Studio создает значки приложения по умолчанию для разных вариантов плотности экрана. В старых версиях Android Studio значки размещаются в папках drawable, а в новых версиях они находятся в папках mipmap.

Атрибут **android:theme** задает тему. При включении в элемент <application> этот атрибут применяется ко всему приложению. В элементе <activity> он применяется только к одной активности.

В нашем примере атрибуту **android:theme** присвоено значение "@style/AppTheme". Префикс @style означает, что тема определяется в **файле стилевых ресурсов**. Что же это такое — файл стилевых ресурсов?

Определение стилей в файлах стилевых ресурсов

Файл стилевых ресурсов содержит подробную информацию обо всех темах, которые вы собираетесь использовать. При создании проекта в Android Studio среда разработки создает файл стилевых ресурсов по умолчанию с именем *styles.xml*, находящийся в папке *app/src/main/res/values*. Этот файл выглядит примерно так:

```
<resources>
  <!-- Базовая тема приложения. -->
  <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
    <!-- Здесь производится настройка темы. -->
  </style>
</resources>
```

Не беспокойтесь, если у вас Android Studio использует другую тему — все равно мы сменим ее на следующей странице.

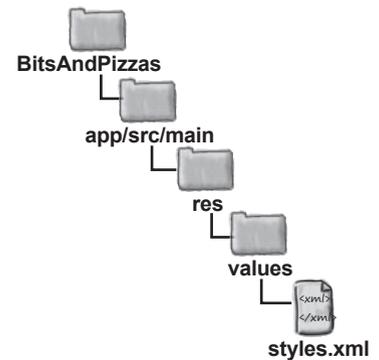
Файл стилевых ресурсов содержит один или несколько стилей. Каждый стиль определяется элементом `<style>`.

Каждому стилю должно быть присвоено имя, которое определяется атрибутом `name`. Стиль должен иметь имя, чтобы атрибут `android:theme` в файле *AndroidManifest.xml* мог сослаться на него. В нашем случае стилю присвоено имя "AppTheme", что позволяет *AndroidManifest.xml* сослаться на него с использованием синтаксиса "`@style/AppTheme`". Атрибут `parent` указывает, от кого стиль должен унаследовать свои свойства. В приведенном примере родителем является тема "`Theme.AppCompat.Light.DarkActionBar`".

Файл стилевых ресурсов также может использоваться для настройки внешнего вида приложения посредством изменения свойств существующей темы. Для этого в элемент `<style>` добавляется элемент `<item>`, который описывает вносимые изменения. Например, вот как выглядит модификация темы, в которой все активности имеют красный фон:

```
<resources>
  <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
    <item name="android:background" color="#FF0000"/>
  </style>
</resources>
```

С этой строкой все активности выводятся на красном фоне.

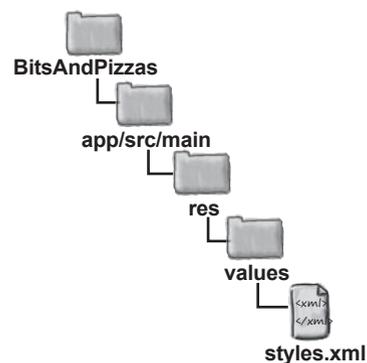


Мы не будем подробно рассматривать процесс настройки тем. Если вас интересует эта тема, обращайтесь к электронной справочной документации: <http://developer.android.com/guide/topics/ui/themes.html>. На следующей странице мы изменим тему, используемую в приложении.

Назначение темы по умолчанию в styles.xml

Мы собираемся изменить приложение так, чтобы оно по умолчанию использовало тему `Theme.Holo.Light` и переключалось на тему `Theme.Material.Light`, если приложение работает на устройстве с API уровня 21. Начнем с изменения темы по умолчанию. Откройте файл стилевых ресурсов `styles.xml`, находящийся в папке `app/src/main/res/values`. Это стандартный файл стилевых ресурсов. По умолчанию приложение должно использовать тему `Theme.Holo.Light`, и этот факт должен быть отражен в атрибуте `<style>` следующим образом:

```
<resources>
  <style name="AppTheme" parent="android:Theme.Holo.Light">
    <!-- Здесь производится настройка темы. -->
  </style>
</resources>
```



Использование темы Material на современных устройствах

Как было показано в главе 8, вы можете использовать разные структуры папок для того, чтобы приложение выбирало разные ресурсы во время выполнения. Например, вы видели, как организовать использование разных файлов макетов в зависимости от размера экрана устройства.

На этот раз приложение должно выбирать разные стили ресурсов в зависимости от уровня API, на котором оно работает. Чтобы приложение выбирало ресурс, если оно работает на устройстве с API уровня 21, мы создадим новую папку `values-v21` и добавим файл ресурсов в эту папку.

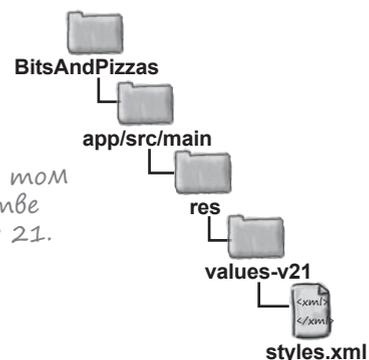
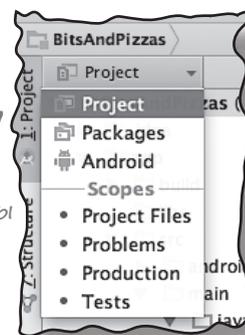
Создайте в `app/src/main/res` новую папку с именем `values-v21`. Затем скопируйте файл `styles.xml` из папки `values` и вставьте его в папку `values-v21`.

Приложение должно использовать тему `Material`, если оно работает на устройстве с API уровня 21. Отредактируйте файл `styles.xml` в папке `values-v21`, чтобы в нем использовалась тема `Theme.Material.Light`:

```
<resources>
  <style name="AppTheme" parent="android:Theme.Material.Light">
    <!-- Здесь производится настройка темы. -->
  </style>
</resources>
```

Эта тема выбирается в том случае, если на устройстве используется API уровня 21.

Новую папку будет проще добавить, если вы переключитесь в режим Project.



Что происходит при запуске приложения

- 1 Когда вы запускаете приложение, Android видит, что в приложении следует применить тему, описанную значением `@style/AppTheme`.



Android

Нужно использовать стиль с именем `AppTheme`, который лучше всего подходит для данного устройства.

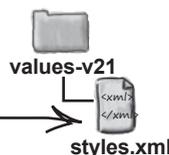
- 2 Если приложение работает на устройстве с API уровня 21, Android использует стиль с именем `AppTheme` из папки `values-21`.
Стиль определяет тему `Theme.Material.Light`; эта тема применяется в приложении.



API 21?
Идеальное совпадение.



Android



Name: `AppTheme`
Parent: `Theme.Material.Light`

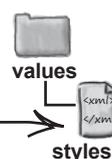
- 3 Если приложение работает на устройстве с API уровня ниже 21, используется стиль с именем `AppTheme` из папки `values`.
Стиль определяет тему `Theme.Holo.Light`, поэтому именно эта тема будет применена в приложении.



Использую стиль из папки `values`.



Android

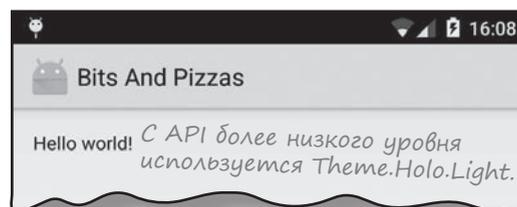
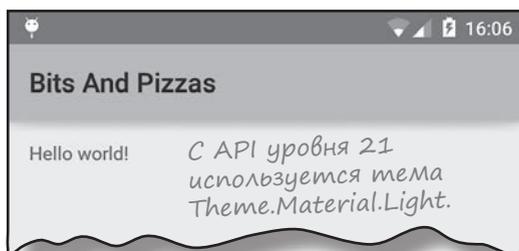


Name: `AppTheme`
Parent: `Theme.Holo.Light`



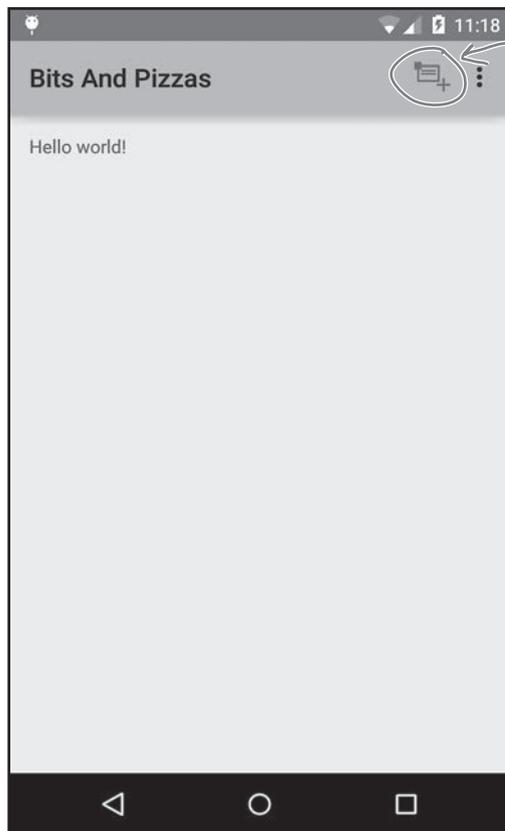
Тест-драйв

При запуске приложения активность `MainActivity` содержит панель действий. Если приложение запускается на устройстве с API уровня 21, используется тема `Theme.Material.Light`. На устройствах с API более низкого уровня используется тема `Theme.Holo.Light`.



Добавление элементов действий на панель действий

Как правило, на панель действий добавляются элементы действий (action items). Они представляют собой кнопки или текст, при щелчке на которых что-то происходит. Для примера мы добавим на панель действий кнопку “Create Order”.



Мы создадим новую кнопку действия для создания заказа.

Процесс добавления элемента действия на панель действий состоит из трех шагов:

- 1** **Определение элементов действий в файле ресурсов меню.**
- 2** **Заполнение ресурса меню активностью.**
Для этого следует реализовать метод `onOptionsItemSelected()`.
- 3** **Добавление кода, который определяет, что должно происходить при щелчке на каждом элементе действия.**
Для этого следует реализовать метод `onOptionsItemSelected()`.

Начнем с файла ресурсов меню.

Файл ресурсов меню

Когда вы создаете проект, содержащий активность, Android Studio автоматически создает файл ресурсов меню по умолчанию. Мы приказали Android Studio присвоить этому файлу имя `menu_main.xml` и разместить его в папке `app/src/main/res/menu`. Все файлы ресурсов меню помещаются в эту папку.

Ниже приведен файл ресурсов меню, сгенерированный Android Studio. Он описывает единственный элемент действия `Settings`, который располагается в дополнительной области (overflow):

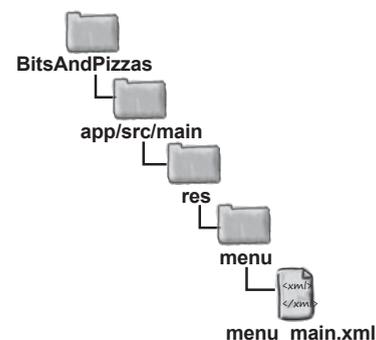
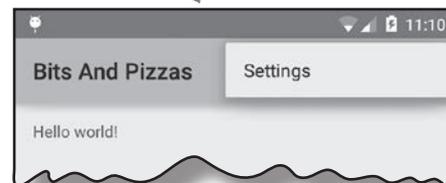
```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    tools:context=".MainActivity">

    <item android:id="@+id/action_settings"
        android:title="@string/action_settings"
        android:orderInCategory="100"
        app:showAsAction="never" />

</menu>
```

Элемент
действия
`Settings`.

Панель действий
по умолчанию.



Каждый файл ресурсов меню содержит корневой элемент `<menu>`. Файл ресурсов меню определяет одно меню, или набор элементов, добавляемых на панель действий. Приложение может содержать несколько файлов ресурсов; это может быть удобно в тех случаях, если панели действий разных активностей содержат разные наборы элементов действий. Для добавления элементов в меню используется элемент `<item>`. Каждый элемент действия описывается отдельным элементом `<item>`. Элемент `<item>` обладает рядом атрибутов, которые вы можете использовать; ниже перечислены наиболее часто используемые атрибуты:

<code>android:id</code>	Задаёт уникальный идентификатор элемента действия. Идентификатор необходим для обращений к элементу действия из кода активности.
<code>android:icon</code>	Значок элемента: ресурс <code>drawable</code> или <code>miplmar</code> .
<code>android:title</code>	Текст элемента действия. Может не отображаться, если элементу действия присвоен значок, а на панели действий не хватает места для значка и текста. Если элемент действия находится в дополнительной области панели, то выводится только текст.
<code>android:orderInCategory</code>	Целочисленное значение, которое помогает Android определить порядок размещения элементов на панели действий.

В приведенной выше разметке используется еще один атрибут: `showAsAction`. Он рассматривается на следующей странице.

Атрибут меню showAsAction

Атрибут `showAsAction` указывает, как элемент действия должен отображаться на панели действий. Например, с его помощью можно разместить элемент в дополнительной области, а не на основной панели действий или разместить элемент на основной панели действий только при наличии места. Атрибут может принимать следующие значения:

"ifRoom"	Элемент размещается на панели действий, если позволяет место. Если места не хватает, элемент размещается в дополнительной области.
"withText"	Включить текст названия элемента.
"never"	Элемент размещается в дополнительной области и никогда — на основной панели действий.
"always"	Элемент всегда размещается в основной части панели. Будьте сдержанны; если таких элементов окажется слишком много, они начнут перекрываться.

Еще раз обратимся к атрибуту `showAsAction` в разметке ресурсов меню. Обратите внимание: атрибут `showAsAction` снабжается префиксом `app:` вместо префикса `android:`:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto"
      ...>
  <item android:id="@+id/action_settings"
        android:title="@string/action_settings"
        android:orderInCategory="100"
        app:showAsAction="never" />
</menu>
```

← Добавляет пространство имен app.

← Атрибуты `id`, `title` и `orderInCategory` используют пространство имен `android`.

← `showAsAction` использует пространство имен `app`.

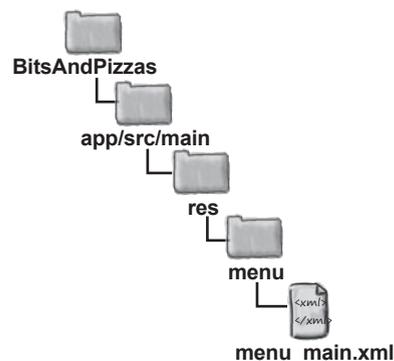
Ранее в этой главе вы уже видели, что наш проект содержит зависимость от библиотеки `v7 appcompat`. В библиотеке `v7 appcompat` атрибут `showAsAction` не входит в пространство имен `android`. Если ваш проект содержит зависимость от библиотеки `v7 appcompat`, атрибут `showAsAction` должен иметь префикс `app:`, а элемент `<menu>` должен включать атрибут

```
xmlns:app="http://schemas.android.com/apk/res-auto"
```

Если ваш проект не содержит зависимости от библиотеки `v7 appcompat`, атрибут `showAsAction` должен быть снабжен префиксом `android:` вместо `app:`. Также возможно не указывать атрибут

```
xmlns:app="http://schemas.android.com/apk/res-auto"
```

в элементе `<menu>`.



Добавление нового элемента действия

Мы добавим на панель действий новый элемент для создания заказов. Элементу назначен пояснительный текст “Create Order” и значок. Для значков на панели действий можно использовать как собственные изображения, так и значки из пакета значков Android. В пакет включено много стандартных значков, которые вы можете использовать в своих приложениях.

Мы воспользуемся значком `ic_action_new_event` из пакета значков. Сначала загрузите пакет по адресу <https://developer.android.com/design/downloads/index.html>. Распаковав его, вы обнаружите в нем много разных значков для разных тем и размеров экрана.

Значки `ic_action_new_event` находятся в папке `Action Bar Icons/holo_light/05_content_new_event`. Пакет включает несколько разных версий для разных размеров экрана (они упорядочены по именам папок). Вы должны скопировать значки в соответствующие папки проекта. Скопируйте значок из папки `drawable-hdpi` в папку `drawable-hdpi` вашего проекта и т. д.

После добавления значков включите в `strings.xml` новый строковый ресурс `action_create_order`:

```
<string name="action_create_order">Create Order</string>
```

Затем добавьте элементы действий в `menu_main.xml`:

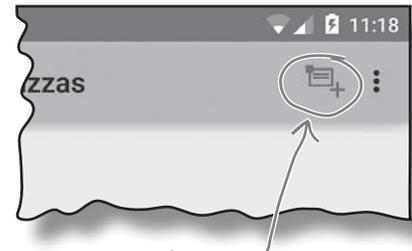
```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  tools:context=".MainActivity">

  <item android:id="@+id/action_create_order"
    android:title="@string/action_create_order"
    android:icon="@drawable/ic_action_new_event"
    android:orderInCategory="1"
    app:showAsAction="ifRoom" />

  <item android:id="@+id/action_settings"
    android:title="@string/action_settings"
    android:orderInCategory="100"
    app:showAsAction="never" />

</menu>
```

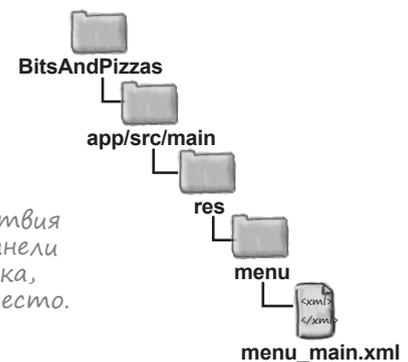
Новый элемент действия отображается на панели действий в виде значка, если для него есть место.



Новый элемент действия.

← Если среда Android Studio не создала эти папки автоматически, вы должны создать их самостоятельно.

← Текст, связанный с элементом действия.



После того как элементы действий будут включены в файл ресурсов меню, необходимо добавить их на панель действий из кода активности. Давайте посмотрим, как это делается.

Заполнение меню в активности методом onCreateOptionsMenu()

После того как вы создадите файл ресурсов меню, содержащиеся в нем элементы добавляются на панель действий; для этого следует реализовать метод onCreateOptionsMenu() активности. Этот метод выполняется при создании меню панели действий и получает один параметр — объект Menu, представляющий панель действий.

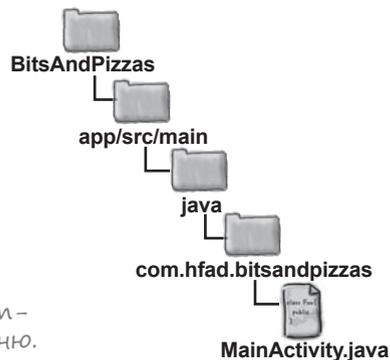
Наша реализация метода onCreateOptionsMenu() выглядит так:

```
package com.hfad.bitsandpizzas;

import android.view.Menu; ← Метод onCreateOptionsMenu()
...                          использует класс Menu.

public class MainActivity extends Activity {
    ...
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Заполнение меню; элементы (если они есть) добавляются на панель действий.
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return super.onCreateOptionsMenu(menu);
    }
}
```

Реализация этого метода добавляет на панель действий все элементы действий, содержащиеся в файле ресурсов меню.



Для добавления элементов на панель действий используется вызов

```
getMenuInflater().inflate(R.menu.menu_main, menu);
```

↑
Файл ресурсов меню.

←
Объект Menu, представляющий панель действий.

Этот вызов берет элементы действий из файла ресурсов меню menu_main.xml и добавляет их в объект Menu панели действий.

Обработка выбора элементов действий с использованием метода `onOptionsItemSelected()`

Реакция активности на выбор элементов на панели действий программируется в методе `onOptionsItemSelected()`. Этот метод выполняется каждый раз, когда выбирается элемент на панели действий.

Метод `onOptionsItemSelected()` получает один параметр: объект `MenuItem`, который представляет элемент на панели действий, выбранный пользователем. Метод `getItemId()` объекта `MenuItem` используется для получения идентификатора элемента, выбранного пользователем, чтобы вы могли выполнить соответствующее действие — например, запустить новую активность.

В нашем примере код метода `onOptionsItemSelected()` выглядит так:

```
package com.hfad.bitsandpizzas;
```

```
import android.view.MenuItem;
```

```
...
```

```
public class MainActivity extends Activity {
```

```
...
```

```
@Override
```

```
public boolean onOptionsItemSelected(MenuItem item) {
```

```
    switch (item.getItemId()) {
```

```
        case R.id.action_create_order:
```

```
            //Код, выполняемый при выборе элемента Create Order
```

```
            return true;
```

```
        case R.id.action_settings:
```

```
            //Код, выполняемый при выборе элемента Settings
```

```
            return true;
```

```
        default:
```

```
            return super.onOptionsItemSelected(item);
```

```
    }
```

```
}
```

```
}
```

Метод `onOptionsItemSelected()` использует этот класс.

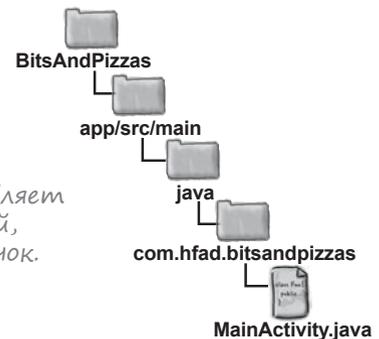
Объект `MenuItem` представляет элемент на панели действий, на котором был сделан щелчок.

Проверить, какой элемент был выбран пользователем.

Элемент `Create Order` должен делать что-то полезное.

Возвращаемое значение `true` сообщает Android, что щелчок на элементе обработан.

Android Studio автоматически создает элемент `Settings`. Здесь размещается код, который выполняет полезные действия.



При выборе элемента действия `Create Order` будет запускаться новая активность с именем `OrderActivity`.

Создание `OrderActivity`

Мы создадим новую активность с именем `OrderActivity`, которая будет запускаться элементом действия `Create Order`.

Начнем с создания пустой активности с именем “`OrderActivity`” и макетом “`activity_order`”, текстом “`Create Order`” и ресурсом меню с именем “`menu_order`”.

Ниже приведен код `OrderActivity.java`. Убедитесь в том, что ваш код не отличается от нашего. В частности, проследите за тем, чтобы класс `OrderActivity` расширял класс `Activity`, а не `ActionBarActivity`. Это важно, потому что с `ActionBarActivity` могут использоваться только темы семейства `Theme.AppCompat`, а мы хотим использовать темы `Holo` и `Material`.

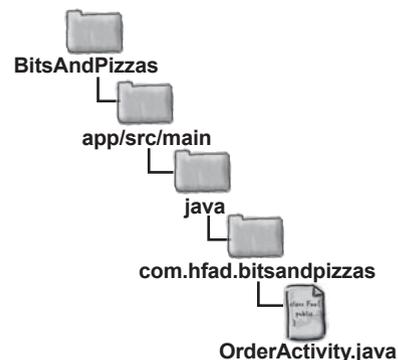
```
package com.hfad.bitsandpizzas;

import android.app.Activity;
import android.os.Bundle;

public class OrderActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_order);
    }
}
```

Убедитесь в том, что `OrderActivity` расширяем `Activity`, а не `ActionBarActivity`.



Мы не включили методы `onCreateOptionsMenu()` и `onOptionsItemSelected()` в свой код `OrderActivity`, так как нам не нужно, чтобы активность `OrderActivity` отображала на своей панели инструментов элементы меню из файла ресурсов меню. Эти методы могут понадобиться, если в будущей версии понадобится добавить поддержку меню.

Итак, активность `OrderActivity` успешно создана. Теперь нужно заставить элемент действия `Create Order` из `MainActivity` запускать ее.

Запуск OrderActivity элементом действия Create Order

При щелчке на элементе действия Create Order на панели действий MainActivity должна запускаться активность OrderActivity. Для этого необходимо внести изменения в метод `onOptionsItemSelected()` класса MainActivity. Для запуска OrderActivity будет использоваться интент.

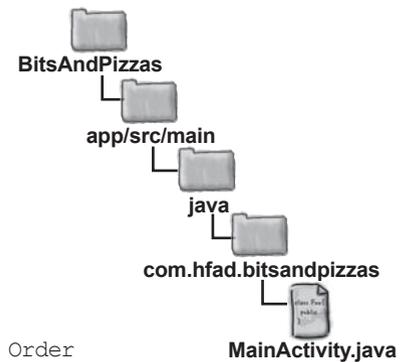
Перед вами код с необходимыми изменениями:

```
package com.hfad.bitsandpizzas;

import android.content.Intent;
...

public class MainActivity extends Activity {
...
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.action_create_order:
                //Код, выполняемый при выборе элемента Create Order
                Intent intent = new Intent(this, OrderActivity.class);
                startActivity(intent);
                return true;
            case R.id.action_settings:
                //Код, выполняемый при выборе элемента Settings
                return true;
            default:
                return super.onOptionsItemSelected(item);
        }
    }
}
```

← Мы собираемся использовать класс Intent.



← Интент используется для запуска OrderActivity при выборе элемента действия Create Order.

Когда пользователь щелкает на элементе действия Create Order, создается интент для запуска OrderActivity.

Полный код `MainActivity.java` приведен на следующей странице.

Полный код MainActivity.java

```

package com.hfad.bitsandpizzas;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;

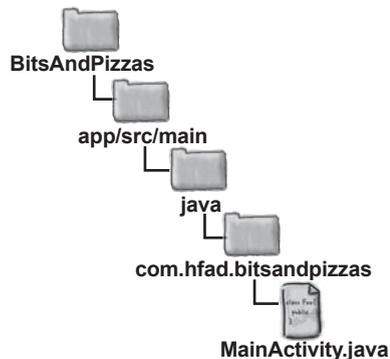
public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Заполнение меню; элементы (если они есть) добавляются на панель действий.
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return super.onCreateOptionsMenu(menu);
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.action_create_order:
                //Код, выполняемый при выборе элемента Create Order
                Intent intent = new Intent(this, OrderActivity.class);
                startActivity(intent);
                return true;
            case R.id.action_settings:
                //Код, выполняемый при выборе элемента Settings
                return true;
            default:
                return super.onOptionsItemSelected(item);
        }
    }
}

```



Добавляет элементы
на панель действий.



При выборе элемента Create Order
запускается OrderActivity.





Тест-драйв

При запуске приложения элемент действия Create Order отображается на панели действий MainActivity. Если щелкнуть на этом элементе действия, он запускает OrderActivity.



РАССЛАБЬТЕСЬ

Не огорчайтесь, если элемент действия не появится на главной панели действий.

Из-за ошибки в некоторых реализациях библиотеки v7 appcompat он может оказаться в дополнительной области. Если в вашем приложении возникнет такая проблема, сообщите о ней в Google.

Часто задаваемые вопросы

В: У моего приложения уже есть значок и имя. Откуда они взялись?

О: При создании проекта Android в среде разработки (такой, как Android Studio) среда генерирует часть кода за вас. В частности, этот код включает такие атрибуты, как значок и имя приложения.

В: Возможно ли использовать панели действий, если вы планируете поддерживать API уровня ниже 7?

О: Нет, невозможно. Впрочем, это не создает проблем, потому что очень малая часть устройств работает с API уровня 7 и ниже.

В: Почему мне приходится использовать ActionBarActivity, если я хочу поддерживать API уровня ниже 11?

О: Потому что в этом случае для добавления панели действий необходимо использовать библиотеку поддержки Android.

В: Мне когда-нибудь понадобится использовать разные темы для разных уровней API?

О: Не исключено. Тема Material появилась в API уровня 21; возможно, вы захотите использовать эту тему в приложении, если она доступна.

В: Вы говорите, что темы могут применяться на уровне не только приложения, но и отдельных активностей. Считаете, что это может пригодиться?

О: Да, может. Темы Holo и Material содержат несколько субклассов тем, которые слегка изменяют внешний вид активностей. Если вы хотите, чтобы одна активность отличалась от других, ей можно назначить другую тему.

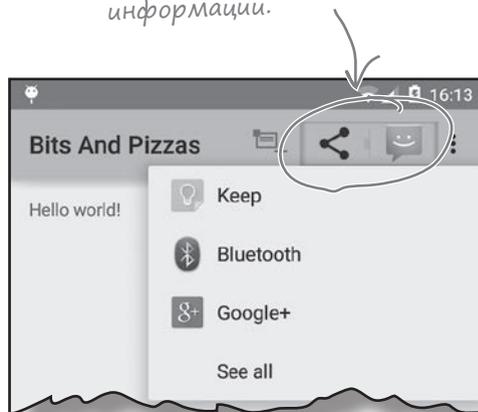
Передача информации с панели действий

А теперь посмотрим, как использовать провайдера действий на панели действий. Провайдер действий — элемент, который добавляется на панель действий и сам управляет своим внешним видом и поведением.

Сейчас мы сосредоточимся на использовании провайдера действия передачи информации. С его помощью пользователи могут передавать информацию из вашего приложения в другие приложения — например, в Gmail. Скажем, пользователь может отправить подробное описание определенного вида пиццы одному из своих контактов.

Провайдер действия передачи информации имеет собственный значок, так что вам не придется определять значок самостоятельно. При щелчке провайдер предоставляет список приложений, которым он умеет передавать информацию.

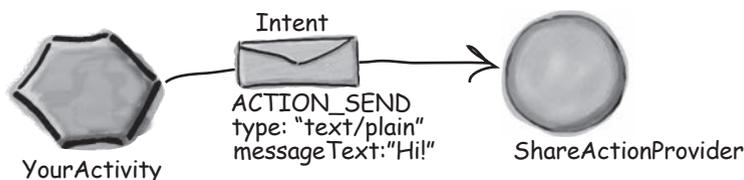
Так выглядит действие передачи информации на панели действий. Если щелкнуть на нем, на экране появляется список приложений, которые могут использоваться для передачи информации.



Использование интента для передачи информации

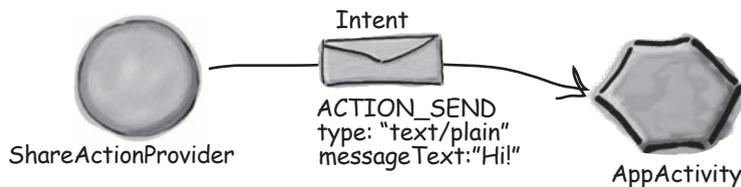
Чтобы обмениваться информацией через провайдера передачи информации, следует передать ему интент. Переданный интент определяет передаваемую информацию и ее тип. Например, можно определить интент, который передает текст с действием ACTION_SEND; действие передачи информации откроет список приложений на устройстве, способных передавать данные такого типа.

- 1** Активность создает интент и передает его провайдеру действия передачи информации. Интент описывает передаваемую информацию, ее тип и выполняемое действие.



- 2** Когда пользователь щелкает на действии передачи информации, оно использует интент для того, чтобы вывести список приложений, работающих с информацией такого типа.

Пользователь выбирает приложение, а провайдер действия передачи информации передает интент активности приложения, которая может его обработать.



Добавление провайдера в файл menu_main.xml

Чтобы добавить действие передачи информации на панель действий, следует включить его в файл ресурсов меню.

Для начала добавьте новую строку `action_share` в файл `strings.xml`. Она будет использоваться для вывода названия действия в том случае, если элемент окажется в дополнительной области:

```
<string name="action_share">Share</string>
```

Действие передачи информации добавляется в файл ресурсов меню, как обычно, при помощи элемента `<item>`. Однако на этот раз необходимо указать, что элемент определяет провайдера действия передачи информации. Для этого добавьте в элемент атрибут `android:actionProviderClass` и присвойте ему значение `android.widget.ShareActionProvider`.

Код, добавляющий действие передачи информации, выглядит так:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto"
      xmlns:tools="http://schemas.android.com/tools"
      tools:context=".MainActivity">

  <item android:id="@+id/action_create_order"
        ... />

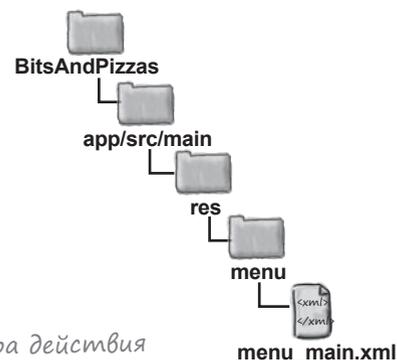
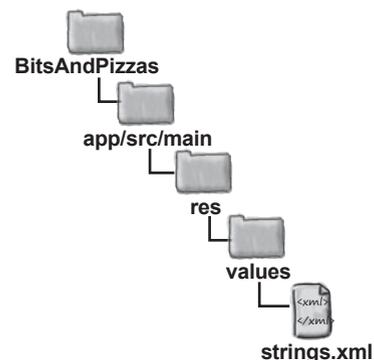
  <item android:id="@+id/action_share"
        android:title="@string/action_share"
        android:orderInCategory="2"
        app:showAsAction="ifRoom"
        android:actionProviderClass="android.widget.ShareActionProvider" />

  <item android:id="@+id/action_settings"
        ... />

</menu>
```

Вывести провайдера действия передачи информации на панели действий, если хватит места.

Класс провайдера действия передачи информации.



Когда вы включаете действие передачи информации в файл ресурсов меню, включать значок не нужно: провайдер уже определяет его за вас. Теперь, когда вы добавили действие передачи информации на панель действий, нужно указать, какая именно информация передается.

Информация задается при помощи интента

Чтобы при щелчке действие активизировало передачу информации, необходимо сообщить ему тип передаваемой информации в коде активности. Для этого провайдеру назначается интент при помощи его метода `setShareIntent()`. В следующем примере действие передачи информации настраивается для передачи некоторого текста по умолчанию:

```
package com.hfad.bitsandpizzas;

...
import android.widget.ShareActionProvider;

public class MainActivity extends Activity {

    private ShareActionProvider shareActionProvider;

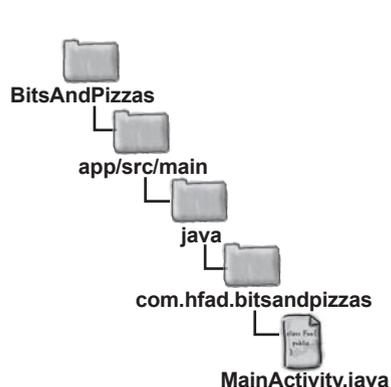
    ...
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.menu_main, menu);
        MenuItem menuItem = menu.findItem(R.id.action_share);
        shareActionProvider = (ShareActionProvider) menuItem.getActionProvider();
        setIntent("This is example text");
        return super.onCreateOptionsMenu(menu);
    }

    private void setIntent(String text) {
        Intent intent = new Intent(Intent.ACTION_SEND);
        intent.setType("text/plain");
        intent.putExtra(Intent.EXTRA_TEXT, text);
        shareActionProvider.setShareIntent(intent);
    }
}
```

↑
Добавить приватную переменную `ShareActionProvider`.

↑ Получить ссылку на провайдера действия передачи информации и присвоить ее приватной переменной. Затем вызвать метод `setIntent()`.

↑ Мы создаем метод `setIntent()`, который создает интент и передает его провайдеру действия передачи информации при помощи его метода `setShareIntent()`.



При любых изменениях информации, которая должна передаваться, необходимо вызывать метод `setShareIntent()` провайдера. Например, если пользователь пролистывает изображения в фотогалерее, вы должны проследить за тем, чтобы передавалась текущая фотография.

Полный код активности приведен на следующей странице. Просмотрите его, а потом мы разберемся, что же происходит при выполнении приложения.

Полный код MainActivity.java

Ниже приведен полный код активности *MainActivity.java*:

```

package com.hfad.bitsandpizzas;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.ShareActionProvider;

public class MainActivity extends Activity {

    private ShareActionProvider shareActionProvider;

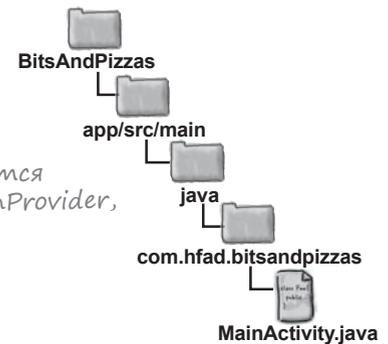
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Заполнение меню; элементы (если они есть) добавляются на панель действий.
        getMenuInflater().inflate(R.menu.menu_main, menu);
        MenuItem menuItem = menu.findItem(R.id.action_share);
        shareActionProvider = (ShareActionProvider) menuItem.getActionProvider();
        setIntent("This is example text");
        return super.onCreateOptionsMenu(menu);
    }

    private void setIntent(String text) {
        Intent intent = new Intent(Intent.ACTION_SEND);
        intent.setType("text/plain");
        intent.putExtra(Intent.EXTRA_TEXT, text);
        shareActionProvider.setShareIntent(intent);
    }
}

```

В коде используется класс *ShareActionProvider*, его необходимо импортировать.



Здесь задается текст по умолчанию, который должен передаваться провайдером действия передачи информации.

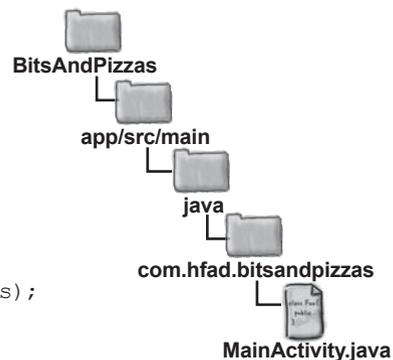
Продолжение на следующей странице.

Код MainActivity.java (продолжение)

```

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.action_create_order:
            //Код, выполняемый при выборе элемента Create Order
            Intent intent = new Intent(this, OrderActivity.class);
            startActivity(intent);
            return true;
        case R.id.action_settings:
            //Код, выполняемый при выборе элемента Settings
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}

```



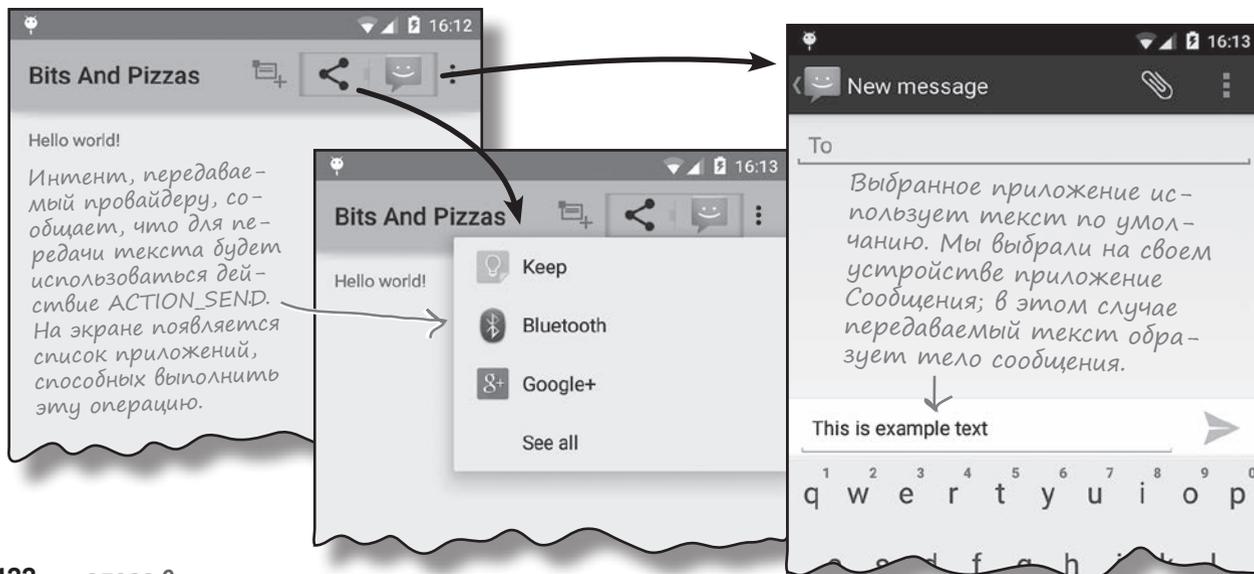
↑
Этот метод
не изменился.



Тест-драйв

При запуске приложения на панели действий отображается элемент действия передачи информации. Если щелкнуть на нем, на экране появляется список приложений, способных принять интент, который мы собираемся передать. Если выбрать приложение в списке, оно получает текст по умолчанию.

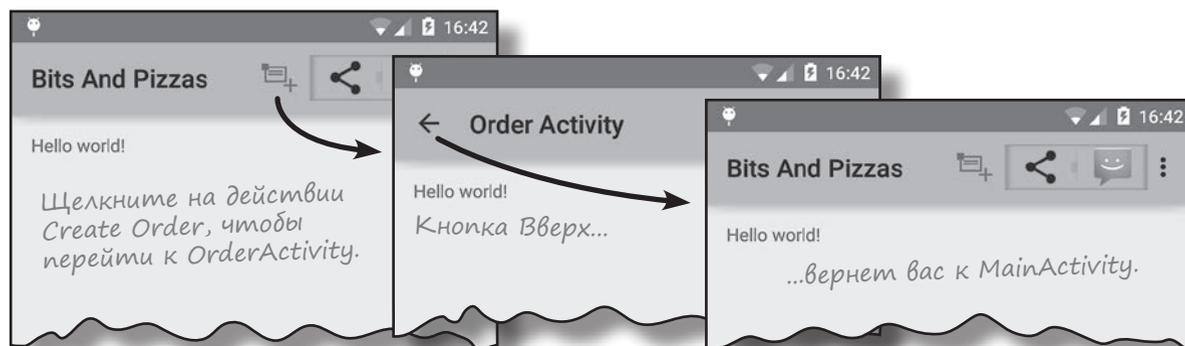
Помните: действие передачи информации может находиться не в главной, а в дополнительной области панели действий.



Навигация с кнопкой Вверх

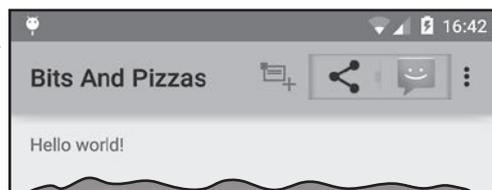
Если в вашем приложении используется иерархия активностей, вы можете добавить на панель действий кнопку Вверх, упрощающую переходы в иерархиях. Например, на панели действия активности `MainActivity` в нашем приложении находится элемент действия для запуска второй активности `OrderActivity`. Если на панели действий `OrderActivity` будет доступна кнопка Вверх, то пользователь сможет вернуться к `MainActivity`, щелкнув на этой кнопке.

Кнопка Вверх.



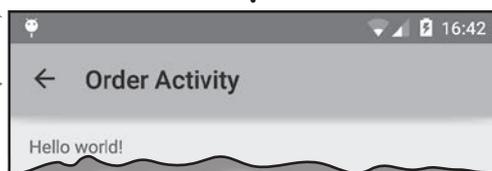
На первый взгляд навигация с кнопкой Вверх напоминает навигацию с кнопкой Назад, но между ними существуют важные различия. Кнопка Назад позволяет пользователю пройти в обратном направлении по истории активностей, с которыми он работал. С другой стороны, переходы по кнопке Вверх основаны исключительно на иерархической структуре приложения.

Родительская активность.



Дочерняя активность.

Нажатие кнопки Вверх в дочерней активности вернет вас вверх по иерархии к родительской активности.



Кнопка Назад возвращает пользователя к предыдущей активности.

Кнопка Вверх используется для навигации по иерархии приложения.

Теперь вы знаете, как работает кнопка Вверх, и мы поместим ее на панель действий `OrderActivity`. Щелкнув на этой кнопке, пользователь переходит к активности `MainActivity`.

Назначение родителя активности

Кнопка Вверх осуществляет переход вверх по иерархии активностей приложения. Эта иерархия объявляется в разметке *AndroidManifest.xml* – вы указываете родителя каждой активности. Например, при нажатии кнопки Вверх пользователь должен переходить от *OrderActivity* к *MainActivity*; это означает, что *MainActivity* является родителем *OrderActivity*.

Начиная с API уровня 16 родительская активность задается атрибутом `android:parentActivityName`. В более старых версиях Android приходится включать элемент `<meta-data>` с именем родительской активности. Вот как выглядят оба способа в нашем файле *AndroidManifest.xml*:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.hfad.bitsandpizzas" >
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >

        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity
            android:name=".OrderActivity"
            android:label="@string/title_activity_order"
            android:parentActivityName=".MainActivity">
            <meta-data
                android:name="android.support.PARENT_ACTIVITY"
                android:value=".MainActivity" />
            </activity>

    </application>
</manifest>
```



В приложениях с API уровня 16 и выше используется эта строка. Она сообщает, что родителем *OrderActivity* является *MainActivity*.

← Элемент `<meta-data>` добавляется только в том случае, если вы поддерживаете приложения с уровнем API ниже 16. Мы приводим его только для того, чтобы вы знали, как он выглядит; впрочем, вреда от него все равно не будет.

Наконец, необходимо включить кнопку Вверх в *OrderActivity*.

Добавление кнопки Вверх

Кнопка Вверх добавляется в коде активности. Сначала вы получаете ссылку на панель действий, используя метод `getActionBar()` активности, а затем вызываете метод `setDisplayHomeAsUpEnabled()` панели действий и передаете при вызове значение `true`.

```
ActionBar actionBar = getActionBar();
actionBar.setDisplayHomeAsUpEnabled(true);
```

Мы хотим, чтобы кнопка Вверх присутствовала в `OrderActivity`, поэтому включаем соответствующий код в метод `onCreate()` файла `OrderActivity.java`. Полный код активности выглядит так:

```
package com.hfad.bitsandpizzas;

import android.app.ActionBar;
import android.app.Activity;
import android.os.Bundle;

public class OrderActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_order);
        ActionBar actionBar = getActionBar();
        actionBar.setDisplayHomeAsUpEnabled(true);
    }
}
```

← Класс `ActionBar`, используемый в программе, необходимо импортировать.

↑ Включить кнопку Вверх на панели действий.

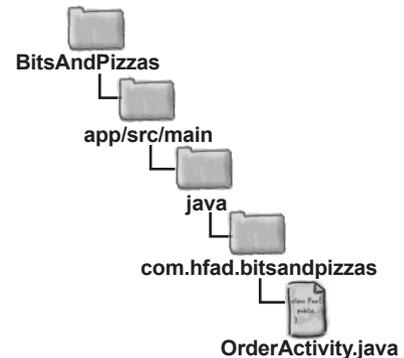
Посмотрим, что происходит при запуске приложения.



Будьте осторожны!

Если для активности включается кнопка Вверх, у нее должен быть задан родитель.

Если этого не сделать, то при вызове метода `setDisplayHomeAsUpEnabled()` будет выдано исключение `null-указателя`.





Тест-драйв

Когда вы запустите приложение и щелкнете на элементе действия Create Order, как и прежде, отображается активность `OrderActivity`.



На панели действий `OrderActivity` присутствует кнопка `Вверх`. При нажатии этой кнопки отображается иерархический родитель текущей активности — `MainActivity`.



Ваш инструментарий Android

Глава 9 осталась позади, а ваш инструментарий пополнился панелями действий.

Весь код для этой главы можно загрузить по адресу <https://tinyurl.com/HeadFirstAndroid>.

КЛЮЧЕВЫЕ МОМЕНТЫ



- Чтобы добавить панель действий в приложения с поддержкой API уровня 11 и выше, примените одну из тем Holo или Material.
- Чтобы добавить панель действий в приложение с поддержкой API уровня 7 и выше, примените тему AppCompatActivity и используйте класс ActionBarActivity. При использовании ActionBarActivity необходимо применить тему AppCompatActivity.
- Класс ActionBarActivity и темы AppCompatActivity находятся в библиотеке v7 appcompat.
- Атрибут `android:theme` в *AndroidManifest.xml* указывает, какая тема должна применяться.
- Стили определяются в файле стилевых ресурсов при помощи элемента `<style>`. Атрибут `name` определяет имя стиля. Атрибут `parent` определяет, от кого стиль должен наследовать свои свойства.
- Для файлов стилевых ресурсов должна использоваться папка по умолчанию *app/src/main/res/values*. Если вы хотите, чтобы файл стилевых ресурсов использовался с API уровня 21, поместите его в папку *app/src/main/res/values-v21*.
- Чтобы добавить элементы действий на панель действий, включите их в файл ресурсов меню.
- Чтобы добавить элементы из файла ресурсов меню на панель действий, реализуйте метод `onOptionsItemSelected()` активности.
- Чтобы указать, что должно происходить при щелчке на элементе, реализуйте метод `onOptionsItemSelected()` активности.
- Вы можете организовать обмен данными из своего приложения; для этого добавьте на панель действий провайдера действия передачи информации. Описание провайдера включается в файл ресурсов меню. Вызовите метод `setShareIntent()` провайдера для передачи ему интента с описанием передаваемой информации.
- Добавьте кнопку Вверх на панель действий, чтобы подняться вверх по иерархии приложения. Иерархия определяется в файле *AndroidManifest.xml*. Чтобы включить кнопку Вверх, используйте метод `setDisplayHomeAsUpEnabled()` класса ActionBar.

Подальше положишь...

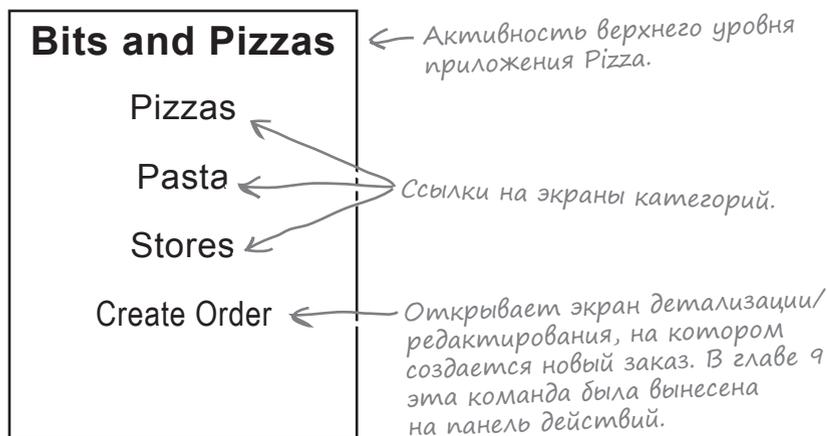


У меня все необходимое для навигации разложено по местам — просто нужно выдвинуть панельку.

С приложением намного приятнее работать, когда в нем хорошо организована навигация. В этой главе мы представим **выдвижные панели (drawers)** — чтобы открыть их, следует провести пальцем по экрану или прикоснуться к значку на панели действий. Вы увидите, как использовать выдвижные панели для вывода **списка ссылок**, ведущих к **основным разделам приложения**. Также вы увидите, как **переключение фрагментов** упрощает переход к этим разделам и ускоряет их отображение.

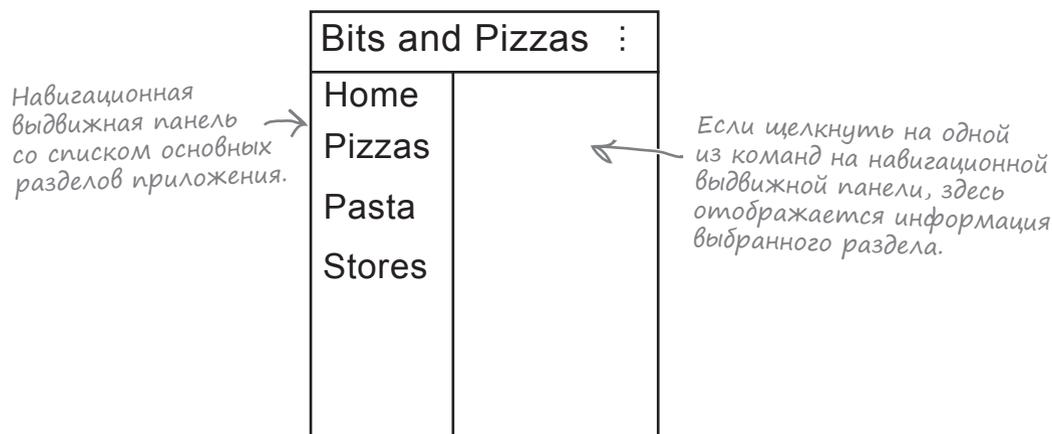
Возвращаемся к приложению Pizza

В главе 7 был приведен эскиз экрана верхнего уровня приложения Pizza. На экране выводится список разделов приложения, к которым пользователь может перейти. Первые три пункта связываются с экранами категорий (пицца, паста, магазины), а последний пункт ведет на экран детализации/редактирования, на котором пользователь может создать заказ.



В предыдущей главе было показано, как добавлять элементы действий на панель действий. Это решение лучше всего подходит для активных команд (например, создания заказа), но как насчет экранов категорий? Так как эти экраны выполняют пассивные функции и используются для навигации по приложению, мы пойдем другим путем.

Команды Pizzas, Pasta и Stores будут вынесены на **навигационную выдвигающую панель**, содержащую ссылки на основные навигационные точки приложения. Эти навигационные точки, часто называемые **разделами**, обычно представляют основные компоненты приложения — экраны верхнего уровня и категорий:

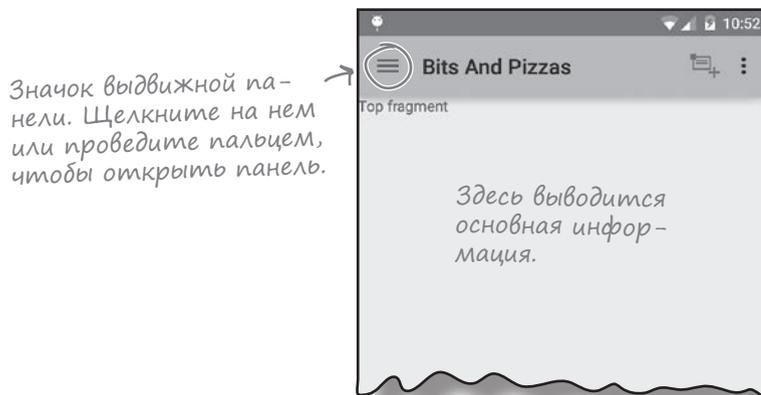


Подробнее о выдвижных панелях

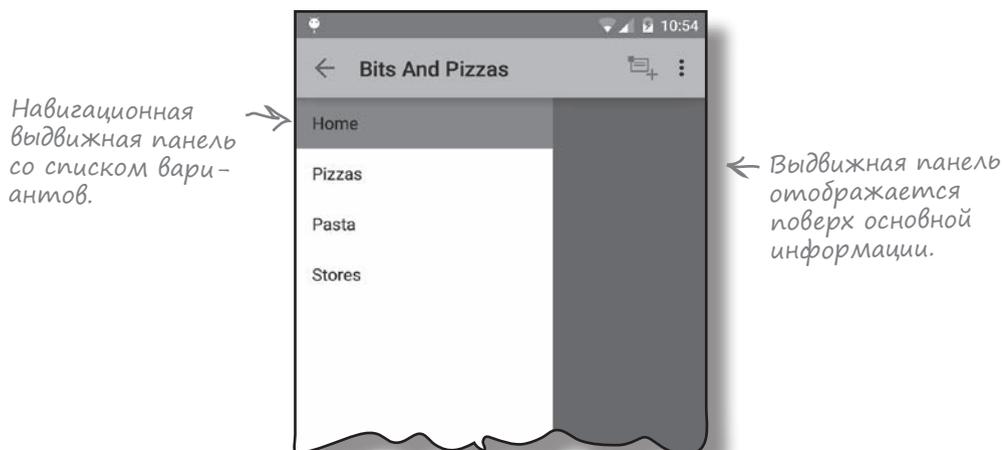
Выдвижная панель реализуется при помощи особой разновидности макетов **DrawerLayout**. `DrawerLayout` управляет двумя представлениями:

- ★ Представление для основного контента. Обычно здесь используется фрейм `FrameLayout`, чтобы вы могли отображать и переключать фрагменты.
- ★ Представление для выдвижной панели (обычно списковое представление `ListView`).

По умолчанию `DrawerLayout` отображает представление, которое почти не отличается от обычной активности:



Если щелкнуть на значке выдвижной панели или провести пальцем от края экрана, представление выдвижной панели «накрывает» основную информацию:

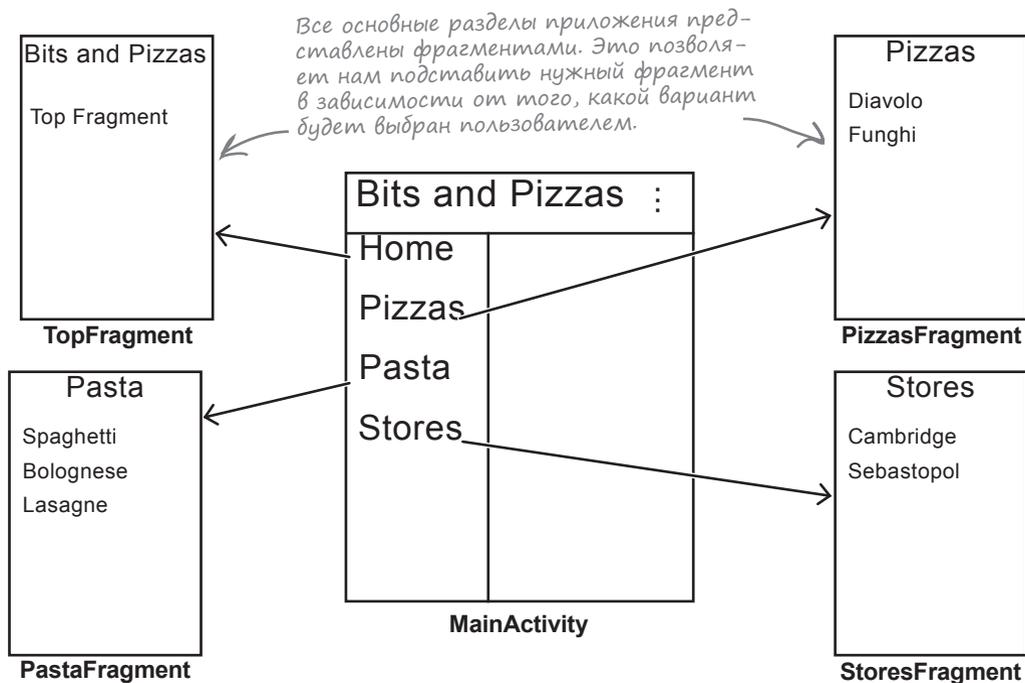


Затем содержимое панели используется для навигации по приложению. Как это влияет на структуру приложения Pizza?

Структура приложения Pizza

Мы изменим код активности `MainActivity` так, чтобы в ней использовалась выдвигающая панель. Активность будет содержать фрейм для отображения фрагментов и списковое представление для вывода списка вариантов.

Списковое представление содержит команды `Home`, `Pizzas`, `Pasta` и `Stores`, при помощи которых пользователь может легко переходить к основным навигационным точкам приложения. Мы создадим фрагменты для всех этих вариантов. Отсюда следует, что фрагменты можно будет переключать во время выполнения, а навигационная выдвигающая панель будет доступна на каждом из этих экранов.



Реализация этой схемы состоит из нескольких этапов:

- 1 **Создание фрагментов для основных разделов.**
- 2 **Создание и инициализация выдвигающей панели.**
Выдвигающая панель содержит компонент `ListView` с перечнем вариантов.
- 3 **Обработка выбора вариантов в компоненте `ListView`.**
Это позволит пользователю переходить к основным разделам приложения.
- 4 **Добавление объекта `ActionBarDrawerToggle`.**
С этим объектом пользователь может вызвать выдвигающую панель с панели действий, а активность может реагировать на действия открытия и закрытия активности.



РАССЛАБЬТЕСЬ

Добавление выдвигающей панели потребует большого объема кода.

В оставшейся части главы мы покажем, как добавить выдвигающую панель в приложение, а в конце главы будет приведен полный код `MainActivity.java`.

Начнем с создания фрагментов.

Создание фрагмента TopFragment

Фрагмент TopFragment будет использоваться для вывода контента верхнего уровня. Пока в нем будет выводиться текст “Top fragment” — просто чтобы вы не забыли, какой фрагмент находится на экране. Создайте новый пустой фрагмент с именем TopFragment и именем макета fragment_top. Код TopFragment.java:

```
package com.hfad.bitsandpizzas;

import android.os.Bundle;
import android.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class TopFragment extends Fragment {

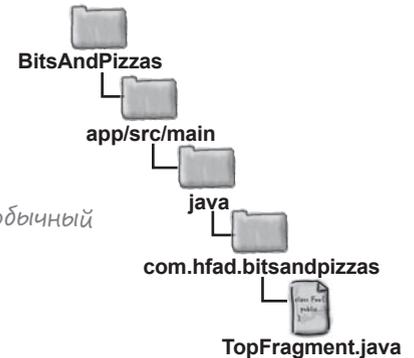
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment_top, container, false);
    }
}
```

TopFragment.java — обычный базовый фрагмент.



- Добавление фрагментов
- Создание выдвижной панели
- Выбор вариантов
- ActionBarDrawerToggle

Мы строим все свои фрагменты на базе пустого фрагмента, так как собираемся заменить весь код, сгенерированный Android Studio.



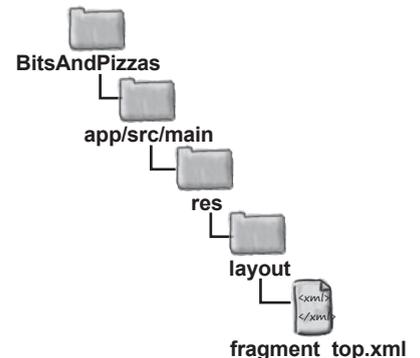
Добавьте следующий строковый ресурс в strings.xml; он будет использоваться в макете фрагмента:

```
<string name="title_top">Top fragment</string>
```

Добавьте эту строку в strings.xml. Мы используем ее в макете, чтобы пользователь знал, что на экране отображается TopFragment.

Разметка fragment_top.xml выглядит так:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <TextView
        android:text="@string/title_top"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</RelativeLayout>
```



Создание фрагмента *PizzaFragment*

Для вывода списка видов пиццы будет использоваться списковый фрагмент *ListFragment* с именем *PizzaFragment*. Создайте новый пустой фрагмент с именем *PizzaFragment*, снимите флажок создания макета. Напомним, что списковым фрагментам макет не нужен — они используют собственный макет. Теперь добавьте в *strings.xml* новый ресурс массива с именем “pizzas” (в этом массиве хранятся названия пиццы):

```
<string-array name="pizzas">
    <item>Diavolo</item>
    <item>Funghi</item>
</string-array>
```

← Добавьте в *strings.xml* массив *pizzas*.

Затем измените код *PizzaFragment.java* так, чтобы в нем использовался класс *ListFragment*. Его списковое представление должно заполняться названиями пиццы. Код выглядит так:

```
package com.hfad.bitsandpizzas;

import android.app.ListFragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;

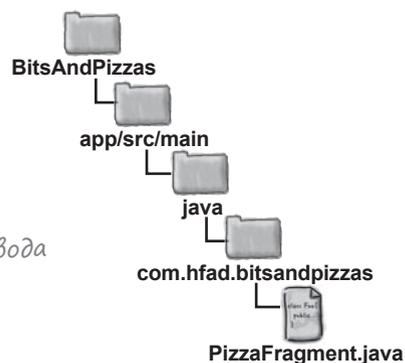
public class PizzaFragment extends ListFragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(
            inflater.getContext(),
            android.R.layout.simple_list_item_1,
            getResources().getStringArray(R.array.pizzas));
        setListAdapter(adapter);
        return super.onCreateView(inflater, container, savedInstanceState);
    }
}
```

Мы используем *ListFragment* для вывода списка видов пиццы.



Добавление фрагментов
Создание выдвигной панели
Выбор вариантов
ActionBarDrawerToggle



Создание фрагмента PastaFragment

Списковый фрагмент `ListFragment` с именем `PastaFragment` будет использоваться для отображения списка видов пасты. Создайте новый пустой фрагмент с именем `PastaFragment`. Флажок создания макета можно снять, так как списковые фрагменты используют собственные макеты. Добавьте в `strings.xml` новый ресурс массива строк с именем “pasta” (в нем содержатся названия видов пасты):

```
<string-array name="pasta">
  <item>Spaghetti Bolognese</item>
  <item>Lasagne</item>
</string-array>
```

← Добавьте в `strings.xml` массив `pasta`.

Затем измените код `PastaFragment.java` так, чтобы в нем использовался класс `ListFragment`. Его списковое представление должно заполняться названиями пасты. Код выглядит так:

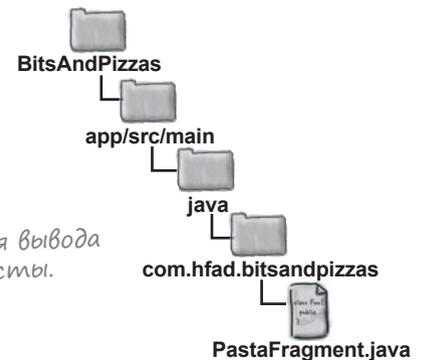
```
package com.hfad.bitsandpizzas;

import android.app.ListFragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.AdapterView;

public class PastaFragment extends ListFragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(
            inflater.getContext(),
            android.R.layout.simple_list_item_1,
            getResources().getStringArray(R.array.pasta));
        setListAdapter(adapter);
        return super.onCreateView(inflater, container, savedInstanceState);
    }
}
```

Мы используем `ListFragment` для вывода списка видов пасты.



Создание фрагмента StoresFragment

Для вывода списка видов пасты будет использоваться списковый фрагмент `ListFragment` с именем `StoresFragment`. Создайте новый пустой фрагмент с именем “`StoresFragment`”. Снимите флажок создания макета, так как списковые фрагменты определяют собственный макет.

Теперь добавьте в `strings.xml` новый ресурс массива с именем “`stores`” (в этом массиве хранятся названия магазинов):

```
<string-array name="stores">
  <item>Cambridge</item>
  <item>Sebastopol</item>
</string-array>
```

← Добавьте в `strings.xml` массив `stores`.

Затем измените код `StoresFragment.java` так, чтобы в нем использовался класс `ListFragment`. Его списковое представление должно заполняться названиями магазинов. Код выглядит так:

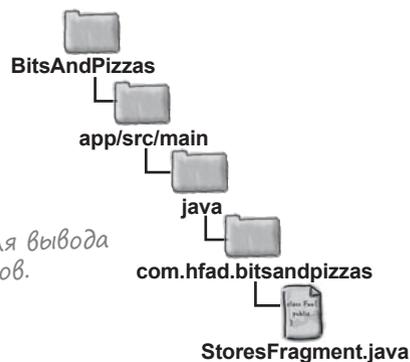
```
package com.hfad.bitsandpizzas;

import android.app.ListFragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;

public class StoresFragment extends ListFragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(
            inflater.getContext(),
            android.R.layout.simple_list_item_1,
            getResources().getStringArray(R.array.stores));
        setListAdapter(adapter);
        return super.onCreateView(inflater, container, savedInstanceState);
    }
}
```

Мы используем `ListFragment` для вывода списка магазинов.



Добавление фрагментов
Создание выдвигной панели
Выбор вариантов
ActionBarDrawerToggle

Добавление DrawerLayout

Теперь мы изменим макет *MainActivity.java* так, чтобы в нем использовался класс *DrawerLayout*. Как говорилось ранее, в нем будет содержаться фрейм *FrameLayout* для вывода фрагментов и списковое представление *ListView* для выдвижной панели.

Для создания *DrawerLayout* используется разметка следующего вида:

```
<android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        ... />
    <ListView
        android:layout_width="240dp"
        android:layout_height="match_parent"
        ... />
</android.support.v4.widget.DrawerLayout>
```

В макете используется класс DrawerLayout из библиотеки v4 support. Библиотека v7 appcompat включает библиотеку v4 support.

Элемент FrameLayout будет использоваться для вывода фрагментов.

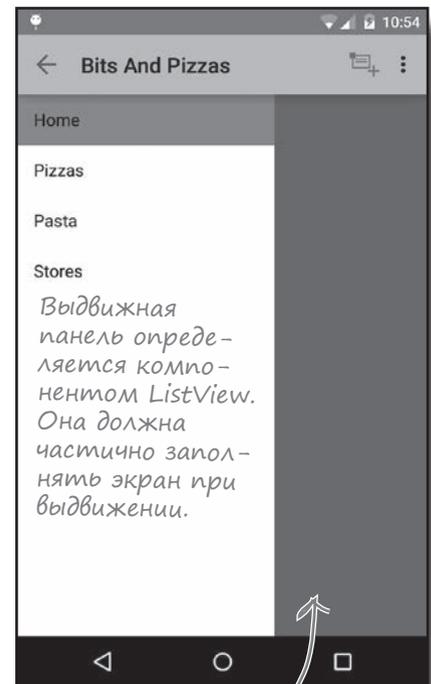
Элемент ListView описывает выдвижную панель.

DrawerLayout — корневой компонент нового макета. Это объясняется тем, что он должен управлять всем, что появляется на экране. Класс *DrawerLayout* определяется в библиотеке *v4 support*, поэтому мы используем его полное имя *android.support.v4.widget.DrawerLayout*.

Первый элемент в *DrawerLayout* используется для вывода информации. В нашем примере это фрейм, в котором будут выводиться фрагменты. Он должен занимать как можно больше места, поэтому мы присваиваем его атрибутам *layout_width* и *layout_height* значение "match_parent".

Второй элемент в *DrawerLayout* определяет саму выдвижную панель. Если вы используете *ListView*, то на выдвижной панели будет отображаться список вариантов. Обычно панель должна частично заполнять экран по горизонтали при выдвижении, поэтому атрибуту *layout_height* присваивается значение "match_parent", а атрибуту *layout_width* — фиксированная ширина.

Полный код *activity_main.xml* приведен на следующей странице.



Информация выводится во фрейме, при этом она заполняет экран. В настоящий момент фрейм частично скрыт выдвижной панелью.

Полная разметка activity_main.xml

Ниже приведена полная разметка `activity_main.xml`:

```
<android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
```

```
<FrameLayout
    android:id="@+id/content_frame"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

```
<ListView android:id="@+id/drawer"
    android:layout_width="240dp"
    android:layout_height="match_parent"
    android:layout_gravity="start"
    android:choiceMode="singleChoice"
    android:divider="@android:color/transparent"
    android:dividerHeight="0dp"
    android:background="#ffffff" />
```

```
</android.support.v4.widget.DrawerLayout>
```

ListView описывает выдвижную панель.

Фрагменты будут отображаться во фрейме FrameLayout.

← Ширина панели.

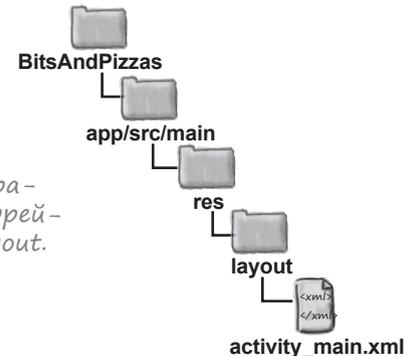
← Где должна размещаться выдвижная панель.

← Варианты будут выбираться по одному.

← Запретить разделительные линии между вариантами и назначить цвет фона выдвижной панели.



Добавление фрагментов
Создание выдвижной панели
Выбор вариантов
ActionBarDrawerToggle



Внимательно изучите значения, заданные в элементе `<ListView>`, — скорее всего, все выдвижные панели, которые вы будете создавать, будут оформляться сходным образом. Для задания размеров выдвижной панели используются атрибуты `layout_width` и `layout_height`. Мы присвоили `layout_width` значение "240dp", чтобы открытая панель имела ширину 240dp.

Если атрибуту `layout_gravity` присвоено значение "start", выдвижная панель размещается слева в тех языках, в которых текст пишется слева направо, и справа в тех странах, где пишут справа налево.

Атрибуты `divider`, `dividerHeight` и `background` используются для отключения разделительных линий между вариантами и назначения цвета фона.

Наконец, атрибут `choiceMode` со значением "singleChoice" сообщает, что варианты в списке могут выбираться только по одному (множественное выделение невозможно).



Будьте осторожны!

Если в вашем проекте отсутствует зависимость от библиотеки поддержки

v7 appcompat, код выдвижной панели из этой главы работать не будет.

Для управления зависимостями выполните команду `File→Project Structure→App→Dependencies`.

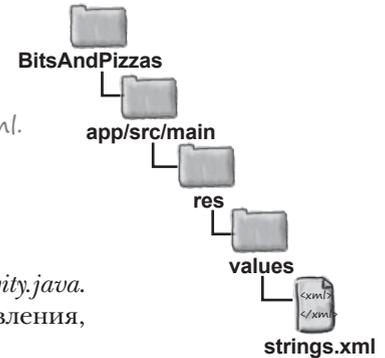
Инициализация списка на выдвижной панели

После того как макет выдвижной панели будет добавлен в `activity_main.xml`, необходимо задать его поведение в `MainActivity.java`. Начнем с заполнения спискового представления. Для этого в файл `strings.xml` добавляется массив вариантов, а затем список заполняется при помощи адаптера массива.

Перед вами массив строк, который следует добавить в `strings.xml` (каждый элемент массива представляет фрагмент, который должен отображаться при выборе данного элемента):

```
<string-array name="titles">
    <item>Home</item>
    <item>Pizzas</item>
    <item>Pasta</item>
    <item>Stores</item>
</string-array>
```

← Варианты, отображаемые на выдвижной панели. Этот массив добавляется в `strings.xml`.



Списковое представление заполняется в методе `onCreate()` из `MainActivity.java`. Мы определяем приватные переменные для массива и спискового представления, так как они понадобятся нам позднее. Код выглядит так:

```
...
import android.widget.ArrayAdapter;
import android.widget.ListView;

public class MainActivity extends Activity {
    ...
    private String[] titles;
    private ListView drawerList;

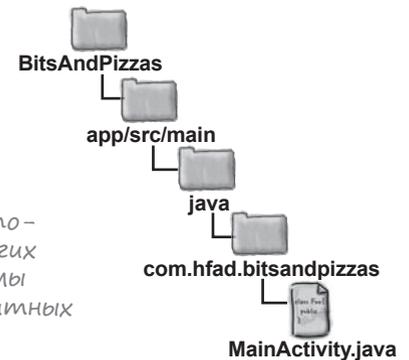
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        titles = getResources().getStringArray(R.array.titles);
        drawerList = (ListView) findViewById(R.id.drawer);
        drawerList.setAdapter(new ArrayAdapter<String>(this,
            android.R.layout.simple_list_item_activated_1, titles));
    }
    ...
}
```

← Эти классы используются в коде, их необходимо импортировать.

← Эти данные также понадобятся нам в других методах, поэтому мы сохраняем их в приватных переменных класса.

← Для заполнения ListView используется класс ArrayAdapter.

↑ Режим `simple_list_item_activated_1` означает, что вариант, на котором щелкнул пользователь, выделяется подсветкой.



Итак, списковое представление заполнено вариантами. Теперь можно заняться программированием реакции списка на выбор его элементов.

OnClickListener и обработка щелчков на вариантах спискового представления



Добавление фрагментов
Создание выдвигной панели
Выбор вариантов
ActionBarDrawerToggle

Реакция спискового представления на щелчки программируется так же, как это делалось в главе 6, — с использованием `OnClickListener`. Мы создаем слушателя, реализуем его метод `onClick()` и связываем слушателя со списковым представлением. Код выглядит так:

```
...
import android.view.View;
import android.widget.AdapterView;

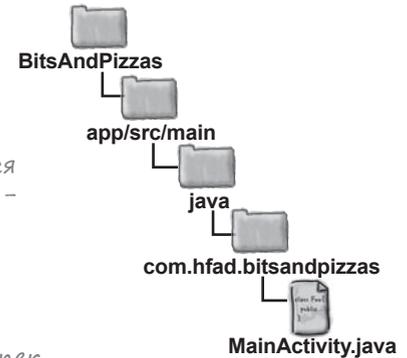
public class MainActivity extends Activity {
    ...
    private class DrawerItemClickListener implements ListView.OnItemClickListener {
        @Override
        public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
            //Код, выполняемый при щелчке на элементе списка.
        }
    };
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        drawerList.setOnItemClickListener(new DrawerItemClickListener());
    }
};
```

Эти классы используются в программе, их необходимо импортировать.

Реализуем OnClickListener.

Когда пользователь щелкает на одном из вариантов на выдвигной панели, вызывается метод onItemClick().

Добавить новый экземпляр OnClickListener к списковому представлению выдвигной панели.



Метод `onClick()` должен содержать код, который должен выполняться при щелчке на одном из элементов спискового представления. В нашем примере он будет вызывать новый метод `selectItem()`, передавая ему позицию выбранного элемента. Сам метод будет написан позднее. Метод должен решать три задачи:

- ★ Переключать фрагмент во фрейме.
- ★ Изменять заголовок на панели действий в соответствии с выбранным макетом.
- ★ Закрывать выдвигную панель.

Вы уже знаете все, что необходимо знать для выполнения первой из этих операций. Посмотрим, как вы справитесь с упражнением на следующей странице.



Развлечения с Магнитами

Когда пользователь щелкает на одном из элементов спискового представления на выдвижной панели, во фрейме `content_frame` должен появляться правильный фрагмент. Попробуйте заполнить пропуски в следующем коде.

```
private void selectItem(int position) {
    Fragment fragment;

    switch( ..... ) {
        case 1:

            fragment = .....;
            break;
        case 2:

            fragment = .....;
            break;
        case 3:

            fragment = .....;
            break;
        default:

            fragment = .....;
    }

    FragmentTransaction ft = getFragmentManager(). .....;

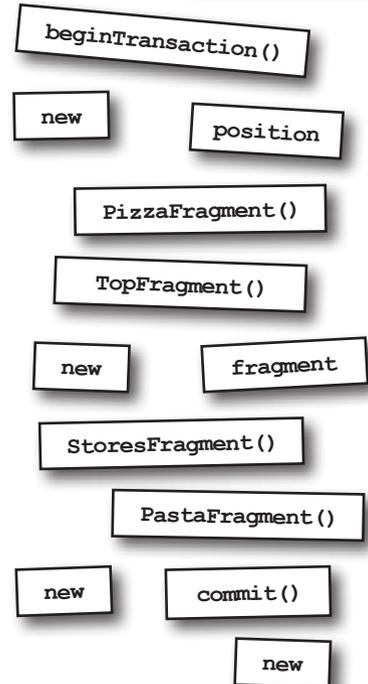
    ft.replace(R.id.content_frame, .....);

    ft.addToBackStack(null);
    ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);

    ft. ....;
}

}
```

Элементы спискового представления `ListView`.





Развлечения с магнитами. Решение

Когда пользователь щелкает на одном из элементов спискового представления на выдвижной панели, во фрейме `content_frame` должен появляться правильный фрагмент. Попробуйте заполнить пропуски в следующем коде.



Элементы
спискового
представления.

```
private void selectItem(int position) {
```

```
    Fragment fragment;
```

```
    switch ( position ) {
```

← Проверить позицию элемента, на котором был сделан щелчок, в списковом представлении.

```
        case 1:
```

```
            fragment = new PizzaFragment();
```

← Создать экземпляр фрагмента, соответствующего позиции. Например, если пользователь щелкнул на варианте "Pizzas", создается экземпляр `PizzaFragment`.

```
            break;
```

```
        case 2:
```

```
            fragment = new PastaFragment();
```

```
            break;
```

```
        case 3:
```

```
            fragment = new StoresFragment();
```

```
            break;
```

```
        default:
```

```
            fragment = new TopFragment();
```

← По умолчанию создается экземпляр `TopFragment`.

```
    }
```

```
    FragmentTransaction ft = getFragmentManager().
```

```
    ft.replace(R.id.content_frame, fragment);
```

← Создать транзакцию для замены отображаемого фрагмента.

```
    ft.addToBackStack(null);
```

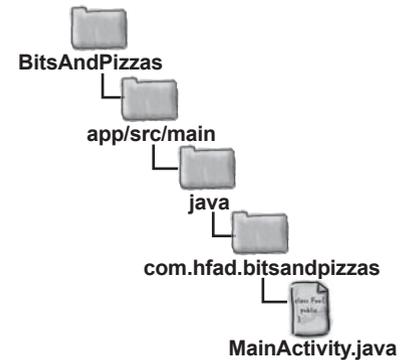
```
    ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);
```

```
    ft.commit(); ← Закрепить транзакцию.
```

```
}
```

Метод `selectItem()` — что было сделано

Ниже приведен обновленный код `MainActivity.java` (при щелчке на одном из элементов на панели действий вызывается метод `selectItem()`, который отображает нужный фрагмент):



```

...
import android.app.Fragment;
import android.app.FragmentTransaction;

public class MainActivity extends Activity {
    ...
    private class DrawerItemClickListener implements ListView.OnItemClickListener {
        @Override
        public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
            selectItem(position);
        }
    };

    private void selectItem(int position) {
        Fragment fragment;
        switch(position) {
            case 1:
                fragment = new PizzaFragment();
                break;
            case 2:
                fragment = new PastaFragment();
                break;
            case 3:
                fragment = new StoresFragment();
                break;
            default:
                fragment = new TopFragment();
        }
        FragmentTransaction ft = getFragmentManager().beginTransaction();
        ft.replace(R.id.content_frame, fragment);
        ft.addToBackStack(null);
        ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);
        ft.commit();
    }
}

```

← По щелчку на элементе вызывается метод `selectItem()`.

← Проверить позицию элемента, на котором был сделан щелчок.

← В зависимости от позиции создается фрагмент правильного типа. Например, вариант "Pizzas" находится в позиции 1, поэтому в этом случае создается фрагмент `PizzaFragment`.

← По умолчанию создается экземпляр `TopFragment`.

↑ Для замены текущего фрагмента используется транзакция фрагмента.

Итак, метод `selectItem()` выводит правильный фрагмент, и мы можем заняться изменением заголовка панели действий.



Обновление фрагментов
Создание выдвжной панели
Выбор вариантов
ActionBarDrawerToggle

Изменение заголовка на панели действий

Кроме переключения отображаемого фрагмента также необходимо изменить заголовок на панели действий, чтобы он соответствовал текущему фрагменту. По умолчанию на панели действий должно отображаться имя приложения, но если пользователь, скажем, выбирает команду Pizzas, заголовок на панели действий должен замениться текстом "Pizzas". Это поможет пользователю сориентироваться, в какой точке приложения он находится.

Для этого мы воспользуемся позицией выбранного варианта для выборки текста, который должен отображаться, из массива titles. Затем текст на панели действий обновляется методом `setTitle()` класса `ActionBar`. Эта операция будет оформлена в виде отдельного метода, так как она еще понадобится нам позднее. Код выглядит так:

```
private void selectItem(int position) {
    ...
    //Назначение заголовка панели действий.
    setActionBarTitle(position);
}
```

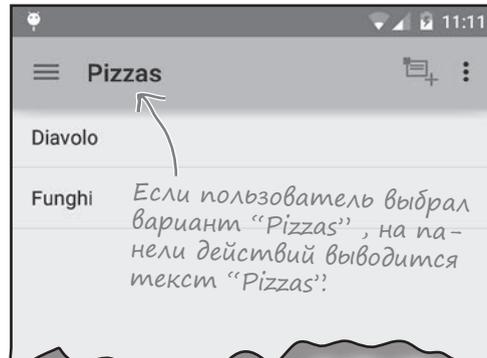
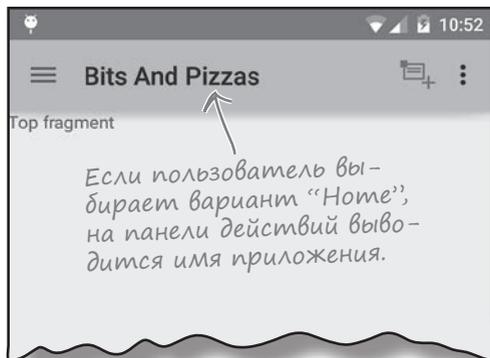
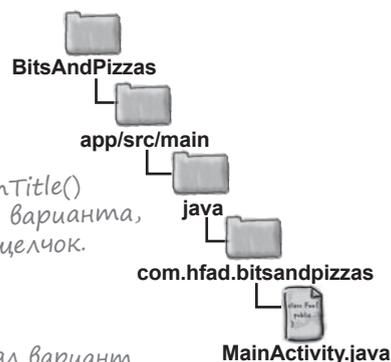
← Вызвать метод `setActionBarTitle()` и передать ему позицию варианта, на котором был сделан щелчок.

```
private void setActionBarTitle(int position) {
    String title;
    if (position == 0) {
        title = getResources().getString(R.string.app_name);
    } else {
        title = titles[position];
    }
    getActionBar().setTitle(title);
}
```

← Если пользователь выбрал вариант "Home", в качестве текста заголовка используется имя приложения.

← В противном случае получить из массива `titles` строку, соответствующую позиции выбранного варианта, и использовать ее.

← Вывести заголовок на панели действий.



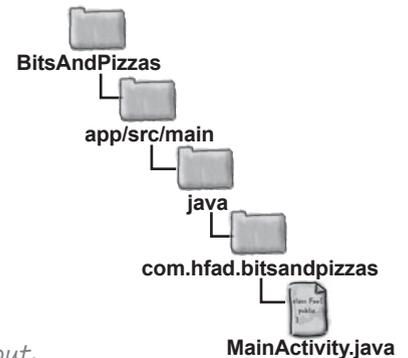
Заккрытие выдвижной панели

Последнее, что нужно сделать в коде `selectItem()`, — закрыть выдвижную панель, чтобы не заставлять пользователя закрывать ее вручную. Чтобы закрыть выдвижную панель, следует получить ссылку на объект `DrawerLayout` и вызвать его метод `closeDrawer()`. Метод `closeDrawer()` получает один параметр — объект `View`, используемый для выдвижной панели. В нашем случае это компонент `ListView`, в котором выводится список вариантов:

```
private void selectItem(int position) {
    ...
    //Закреть выдвижную панель
    DrawerLayout drawerLayout = (DrawerLayout) findViewById(R.id.drawer_layout);
    drawerLayout.closeDrawer(drawerList);
}
```

Получить ссылку на DrawerLayout.

drawerList — выдвижная панель, связанная с DrawerLayout. Вызов приказывает объекту DrawerLayout закрыть панель drawerList.



Экран представлен объектом `DrawerLayout`. Он содержит фрейм для вывода информации и списковое представление, используемое для выдвижной панели.

Необходимо приказать объекту `DrawerLayout` закрыть выдвижную панель `ListView`.



Мы рассмотрели все компоненты, необходимые для написания кода `selectItem()`. Рассмотрим полный код метода и его использование в `MainActivity`.

Обновленный код MainActivity.java

Ниже приведен обновленный код MainActivity.java:

```
package com.hfad.bitsandpizzas;
```

```
...
```

```
import android.support.v4.widget.DrawerLayout;
```

```
public class MainActivity extends Activity {
```

```
...
```

```
private DrawerLayout drawerLayout;
```

```
private class DrawerItemClickListener implements ListView.OnItemClickListener {
```

```
    @Override
```

```
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
```

```
        //Code to run when an item in the navigation drawer gets clicked
```

```
        selectItem(position);
```

```
    }
```

```
};
```

```
@Override
```

```
protected void onCreate(Bundle savedInstanceState) {
```

```
    super.onCreate(savedInstanceState);
```

```
    setContentView(R.layout.activity_main);
```

```
    titles = getResources().getStringArray(R.array.titles);
```

```
    drawerList = (ListView) findViewById(R.id.drawer);
```

```
    drawerLayout = (DrawerLayout) findViewById(R.id.drawer_layout);
```

```
    //Заполнение ListView
```

```
    drawerList.setAdapter(new ArrayAdapter<String>(this,
```

```
        android.R.layout.simple_list_item_activated_1, titles));
```

```
    drawerList.setOnItemClickListener(new DrawerItemClickListener());
```

```
    if (savedInstanceState == null) {
```

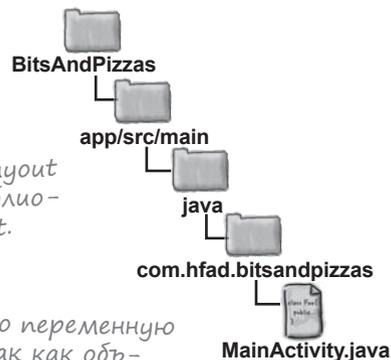
```
        selectItem(0);
```

```
    }
```

```
}
```



Добавление фрагментов
Создание выдвижной панели
Выбор вариантов
ActionBarDrawerToggle



← Класс DrawerLayout находится в библиотеке v4 support.

← Добавить приватную переменную для DrawerLayout, так как объект будет использоваться в других методах.

← Вызвать метод selectItem().

← Получить ссылку на DrawerLayout.

← При исходном создании MainActivity использовать метод selectItem() для отображения TopFragment.

→ Продолжение на следующей странице.

Код MainActivity.java (продолжение)

```

private void selectItem(int position) {
    // Обновление информации на экране посредством замены фрагментов
    Fragment fragment;
    switch(position) {
        case 1:
            fragment = new PizzaFragment();
            break;
        case 2:
            fragment = new PastaFragment();
            break;
        case 3:
            fragment = new StoresFragment();
            break;
        default:
            fragment = new TopFragment();
    }
    FragmentTransaction ft = getFragmentManager().beginTransaction();
    ft.replace(R.id.content_frame, fragment);
    ft.addToBackStack(null);
    ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);
    ft.commit();
    //Назначение заголовка панели действий.
    setActionBarTitle(position);
    //Закрытие выдвижной панели.
    drawerLayout.closeDrawer(drawerList);
}

private void setActionBarTitle(int position) {
    String title;
    if (position == 0){
        title = getResources().getString(R.string.app_name);
    } else {
        title = titles[position];
    }
    getActionBar().setTitle(title);
}
...
}

```

Определить, какой фрагмент должен отображаться.

Вывести фрагмент с использованием транзакции фрагмента.

Задать заголовок панели действий.

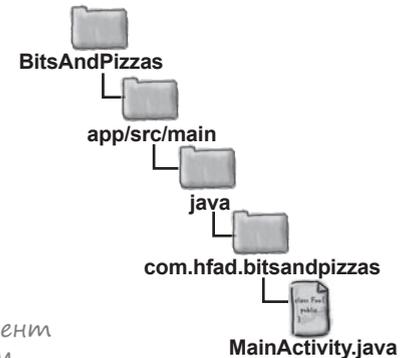
Закрывать выдвижную панель.

Если пользователь выбирает вариант "Home", в качестве текста заголовка используется имя приложения.

Иначе получить из массива titles элемент, соответствующий позиции выбранного элемента, и использовать его текст.

Вывести заголовок на панели действий.

Методы onCreateOptionsMenu() и onOptionsItemSelected() из исходного кода MainActivity не приводятся, так как они не изменялись.



Открытие и закрытие выдвигной панели

К настоящему моменту мы добавили в `MainActivity` выдвигную панель, заполнили ее списком основных разделов приложения и научили активность реагировать на выбор варианта. Пора сделать следующий шаг — разобраться в том, как открывать и закрывать панель и как реагировать на ее состояние.

Есть пара причин, по которым может возникнуть необходимость в реагировании на состояние выдвигной панели. Во-первых, при открытии и закрытии выдвигной панели может изменяться текст на панели действий. Допустим, при открытой выдвигной панели может выводиться имя приложения, а при закрытой — название выбранного фрагмента. Другая причина связана с составом элементов на панели действий. При открытой выдвигной панели некоторые (или все) из этих элементов можно скрыть, чтобы пользователь мог щелкать на них только при закрытой выдвигной панели. На нескольких ближайших страницах мы покажем вам, как настроить объект `DrawerListener` для прослушивания событий `DrawerLayout`. Мы воспользуемся этой возможностью для того, чтобы скрыть действие передачи информации на панели действий при открытой навигационной панели и снова сделать его видимым при закрытии выдвигной панели.



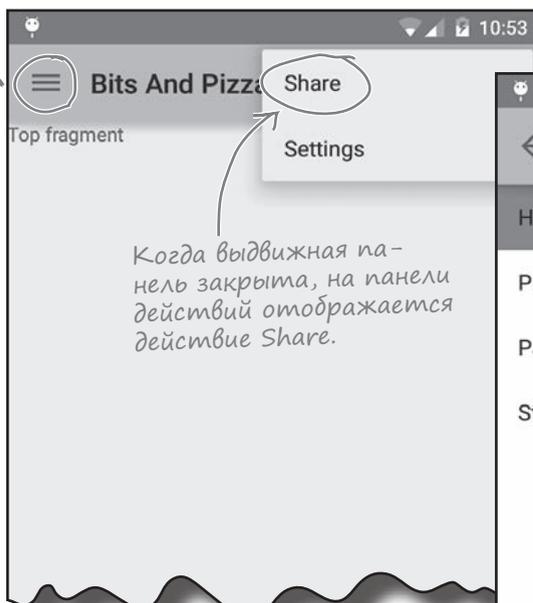
РАССЛАБЬТЕСЬ

Да, мы знаем — вам кажется, что для простого создания выдвигной

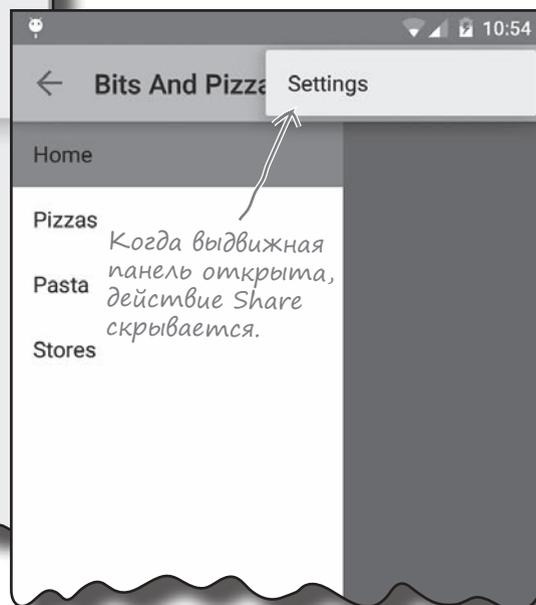
панели приходится принимать во внимание слишком много факторов.

Даже если код покажется вам слишком сложным, придерживайтесь этой схемы, и все будет нормально.

Выдвигная панель будет открываться и закрываться кнопкой на панели действий.



Когда выдвигная панель закрыта, на панели действий отображается действие Share.



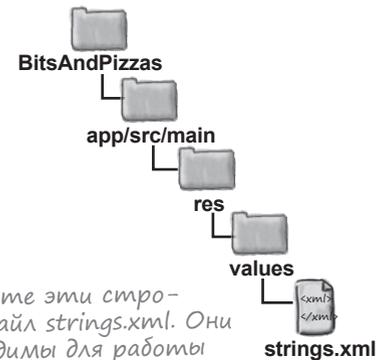
Когда выдвигная панель открыта, действие Share скрывается.

Использование ActionBarDrawerToggle

Для настройки `DrawerListener` лучше всего воспользоваться **ActionBarDrawerToggle** — особой разновидностью `DrawerListener`, работающей с панелью действий. Она обеспечивает прослушивание событий `DrawerLayout`, как и обычный объект `DrawerListener`, а также позволяет открывать и закрывать выдвижную панель, щелкая на значке на панели действий.

Начните с создания в `strings.xml` двух ресурсов строк, описывающих действия «открытия выдвижной панели» и «закрытия выдвижной панели». Эти строки необходимы для улучшения доступности приложения:

```
<string name="open_drawer">Open drawer</string>
<string name="close_drawer">Close drawer</string>
```



Добавьте эти строки в файл `strings.xml`. Они необходимы для работы `ActionBarDrawerToggle`.

Затем создайте новый объект `ActionBarDrawerToggle`, вызвав его конструктор с четырьмя параметрами: `Context` (обычно `this` для текущего контекста), `DrawerLayout` и двумя строковыми ресурсами. Наконец, переопределите методы `onDrawerClosed()` и `onDrawerOpened()` класса `ActionBarDrawerToggle`:

Создание объекта `ActionBarDrawerToggle`.

```
ActionBarDrawerToggle drawerToggle = new ActionBarDrawerToggle(this, drawerLayout,
    R.string.open_drawer, R.string.close_drawer) {
```

```
// Вызывается при переходе выдвижной панели в полностью закрытое состояние.
```

```
@Override
```

```
public void onDrawerClosed(View view) {
    super.onDrawerClosed(view);
    //Код, выполняемый при закрытии выдвижной панели
}
```

Метод вызывается при закрытии выдвижной панели.

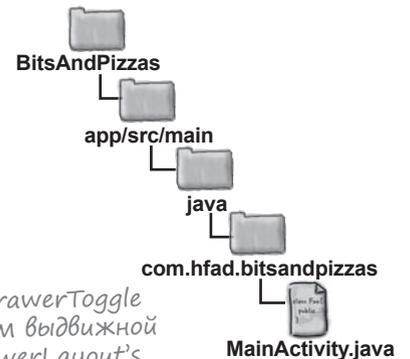
Метод вызывается при открытии выдвижной панели.

```
//Вызывается при переходе выдвижной панели в полностью открытое состояние.
```

```
@Override
```

```
public void onDrawerOpened(View drawerView) {
    super.onDrawerOpened(drawerView);
    //Код, выполняемый при открытии выдвижной панели
}
```

```
};
```



Назначим `ActionBarDrawerToggle` слушателем выдвижной панели `DrawerLayout`'s.

После того как объект `ActionBarDrawerToggle` будет создан, он связывается с `DrawerLayout` при помощи метода `setDrawerListener()` объекта `DrawerLayout`:

```
drawerLayout.setDrawerListener(drawerToggle);
```

`invalidateOptionsMenu()`

Изменение элементов панели действий Во время выполнения

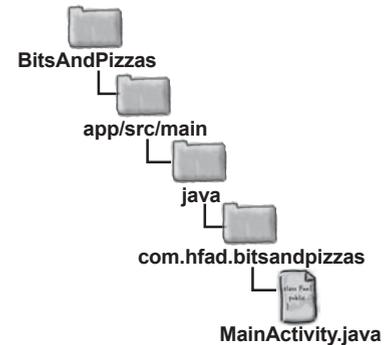


Добавление фрагментов
Создание выдвигной панели
Выбор вариантов
ActionBarDrawerToggle

Если на панели действий присутствуют элементы, привязанные к содержанию конкретного фрагмента, их можно скрыть при открытой выдвигной панели, а потом снова отобразить, когда выдвигная панель закрывается. Если вам потребуется изменить содержимое панели действий в подобной ситуации, это можно сделать следующим образом.

Сначала необходимо вызвать метод `invalidateOptionsMenu()` активности. Он сообщает Android, что элементы меню, которые должны находиться на панели действий, изменились и их необходимо создать заново. При вызове метода `invalidateOptionsMenu()` вызывается метод `onPrepareOptionsMenu()` активности. Вы можете переопределить этот метод, чтобы указать, как должны измениться элементы меню.

В нашем примере видимость действия передачи данных на панели действий будет изменяться в зависимости от того, закрыта или открыта выдвигная панель. Следовательно, мы должны вызвать метод `invalidateOptionsMenu()` в методах `onDrawerClosed()` и `onDrawerOpened()` класса `ActionBarDrawerToggle`.



```
public void onDrawerClosed(View view) {
    super.onDrawerClosed(view);
    invalidateOptionsMenu();
}

public void onDrawerOpened(View drawerView) {
    super.onDrawerOpened(drawerView);
    invalidateOptionsMenu();
}
```

Метод `invalidateOptionsMenu()` приказывает Android создать заново команды меню. Так как мы собираемся изменять видимость действия передачи информации в зависимости от того, открыта или закрыта выдвигная панель, этот метод должен вызываться в методах `onDrawerOpened()` и `onDrawerClosed()`.

Затем метод `onPrepareOptionsMenu()` активности используется для управления видимостью действия Share:

```
//Вызывается при каждом вызове invalidateOptionsMenu()
@Override
public boolean onPrepareOptionsMenu(Menu menu) {
    // Если выдвигная панель открыта, скрыть элементы, связанные с контентом
    boolean drawerOpen = drawerLayout.isDrawerOpen(drawerList);
    menu.findItem(R.id.action_share).setVisible(!drawerOpen);
    return super.onPrepareOptionsMenu(menu);
}
```

Метод `onPrepareOptionsMenu()` будет вызываться при каждом вызове `invalidateOptionsMenu()`.

↑
Действие Share становится невидимым, если выдвигная панель открыта, и видимым при закрытой панели.

Полный код приведен на следующей странице.

Обновленный код MainActivity.java

Ниже приведен обновленный код MainActivity.java:

```

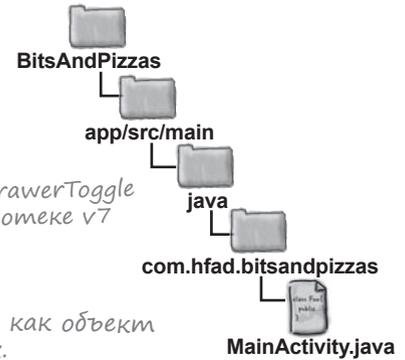
...
import android.support.v7.app.ActionBarDrawerToggle;

public class MainActivity extends Activity {
    ...
    private ActionBarDrawerToggle drawerToggle;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        //Создание ActionBarDrawerToggle
        drawerToggle = new ActionBarDrawerToggle(this, drawerLayout,
            R.string.open_drawer, R.string.close_drawer) {
            //Вызывается при переходе выдвижной панели в полностью закрытое состояние.
            public void onDrawerClosed(View view) {
                super.onDrawerClosed(view);
                invalidateOptionsMenu();
            }
            //Вызывается при переходе выдвижной панели в полностью открытое состояние.
            public void onDrawerOpened(View drawerView) {
                super.onDrawerOpened(drawerView);
                invalidateOptionsMenu();
            }
        };
        drawerLayout.setDrawerListener(drawerToggle);

        //Вызывается при каждом вызове invalidateOptionsMenu()
        @Override
        public boolean onPrepareOptionsMenu(Menu menu) {
            // Если выдвижная панель открыта, скрыть элементы, связанные с контентом
            boolean drawerOpen = drawerLayout.isDrawerOpen(drawerList);
            menu.findItem(R.id.action_share).setVisible(!drawerOpen);
            return super.onPrepareOptionsMenu(menu);
        }
        ...
    }
}

```



Класс ActionBarDrawerToggle находится в библиотеке v7 appcompat.

Добавить приватную переменную, так как объект будет использоваться в других методах.

Вызывать invalidateOptionsMenu() при открытии или закрытии выдвижной панели.

Назначить ActionBarDrawerToggle слушателем для DrawerLayout.

Задать видимость действия Share при открытии и закрытии выдвижной панели.

Управление открытием и закрытием выдвигающей панели

Мы добавили в `MainActivity` выдвигающую панель, заполнили ее списком вариантов, научили активность реагировать на выбор вариантов и разобрались в том, как скрыть элементы действий при открытии выдвигающей панели. Остается последний шаг — сделать так, чтобы пользователь мог открывать и закрывать выдвигающую панель при помощи кнопки на панели действий.

Как было сказано ранее, эта функциональность является одним из преимуществ использования `ActionBarDrawerToggle`. Чтобы включить ее, необходимо добавить немного дополнительного кода. Мы последовательно рассмотрим изменения, которые нужно внести, а в конце будет приведен полный код `MainActivity.java`.

Сначала нужно включить значок на панели действий. Для этого в методе `onCreate()` активности используются вызовы двух методов:

```
getActionBar().setDisplayHomeAsUpEnabled(true);
getActionBar().setHomeButtonEnabled(true);
```



Добавление фрагментов
Создание выдвигающей панели
Выбор вариантов
`ActionBarDrawerToggle`

Класс `ActionBarDrawerToggle` позволяет использовать кнопку Вверх на панели действий для открытия и закрытия выдвигающей панели.



Включить кнопку Вверх, чтобы ее можно было использовать для управления выдвигающей панелью.

Эти строки кода включают кнопку Вверх активности. Так как мы используем объект `ActionBarDrawerToggle`, кнопка Вверх будет использоваться для управления выдвигающей панелью (вместо перемещения вверх по иерархии приложения). Затем необходимо позаботиться о том, чтобы объект `ActionBarDrawerToggle` реагировал на щелчки. Для этого следует вызвать его метод `onOptionsItemSelected()` из метода `onOptionsItemSelected()` активности:

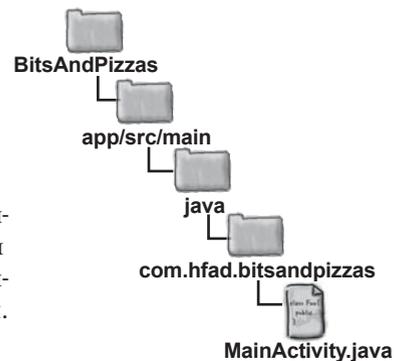
```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    if (drawerToggle.onOptionsItemSelected(item)) {
        return true;
    }
    //Код для остальных элементов действий
    ...
}
```

Чтобы объект `ActionBarDrawerToggle` реагировал на щелчки, необходимо добавить эти строки кода в метод `onOptionsItemSelected()`.

Вызов

```
drawerToggle.onOptionsItemSelected(item)
```

возвращает `true`, если объект `ActionBarDrawerToggle` обработал щелчок. Если вызов возвращает `false`, это означает, что щелчок был сделан на другом элементе действия на панели действий; в этом случае будет выполнен остальной код в методе `onOptionsItemSelected()` активности.



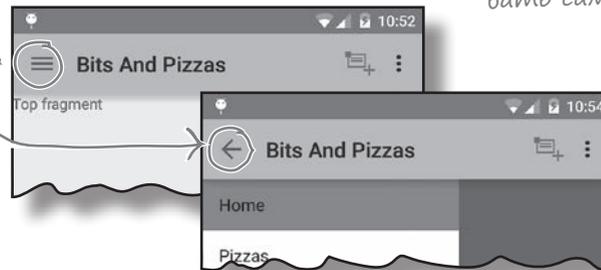
Синхронизация состояния ActionBarDrawerToggle

Чтобы выдвижная панель ActionBarDrawerToggle работала правильно, необходимо решить еще две задачи.

Во-первых, необходимо вызвать метод `syncState()` класса ActionBarDrawerToggle из метода `postCreate()` активности. Метод `syncState()` синхронизирует состояние значка выдвижной панели с состоянием DrawerLayout.

Хорошо бы, конечно, чтобы выдвижная панель делала это за вас... но она этого не делает. Вам придется действовать самостоятельно.

«Синхронизация состояния» означает, что значок выдвижной панели выглядит по-разному в зависимости от того, открыта или закрыта выдвижная панель.



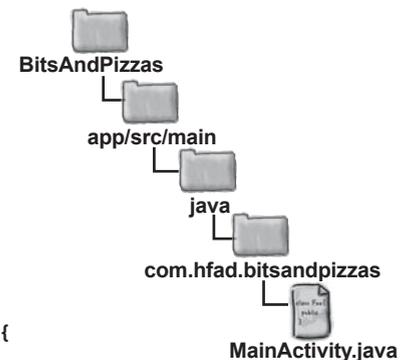
Метод `syncState()` должен вызываться в методе `onPostCreate()` активности, чтобы объект ActionBarDrawerToggle находился в правильном состоянии после создания активности:

```
@Override
protected void onPostCreate(Bundle savedInstanceState) {
    super.onPostCreate(savedInstanceState);
    // Синхронизировать состояние выключателя после onRestoreInstanceState.
    drawerToggle.syncState();
}
```

Этот метод необходимо добавить в активность, чтобы состояние ActionBarDrawerToggle синхронизировалось с состоянием выдвижной панели.

Во-вторых, при изменении конфигурации устройства необходимо передать ActionBarDrawerToggle подробную информацию об изменении. Для этого мы вызываем метод `onConfigurationChanged()` объекта ActionBarDrawerToggle из метода `onConfigurationChanged()` активности:

```
@Override
public void onConfigurationChanged(Configuration newConfig) {
    super.onConfigurationChanged(newConfig);
    drawerToggle.onConfigurationChanged(newConfig);
}
```



Добавьте этот метод в активность, чтобы сведения о любых изменениях в конфигурации передавались ActionBarDrawerToggle.

На следующей странице показано, как выглядит MainActivity.java после внесения этих изменений, а после этого мы посмотрим, что происходит при запуске приложения.

Обновленный код MainActivity.java

Ниже приведена новая версия кода MainActivity.java:

```
...
import android.content.res.Configuration;

public class MainActivity extends Activity {
    ...
    private ActionBarDrawerToggle drawerToggle;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        getActionBar().setDisplayHomeAsUpEnabled(true);
        getActionBar().setHomeButtonEnabled(true);
    }

    @Override
    protected void onPostExecute(Bundle savedInstanceState) {
        super.onPostExecute(savedInstanceState);
        drawerToggle.syncState();
    }

    @Override
    public void onConfigurationChanged(Configuration newConfig) {
        super.onConfigurationChanged(newConfig);
        drawerToggle.onConfigurationChanged(newConfig);
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        if (drawerToggle.onOptionsItemSelected(item)) {
            return true;
        }
        //Код для остальных элементов действий
        switch (item.getItemId()) {
            ...
        }
    }
}
```

Импортировать этот класс, так как он используется в методе onConfigurationChanged().

Включить кнопку Вверх, чтобы она могла использоваться объектом ActionBarDrawerToggle.

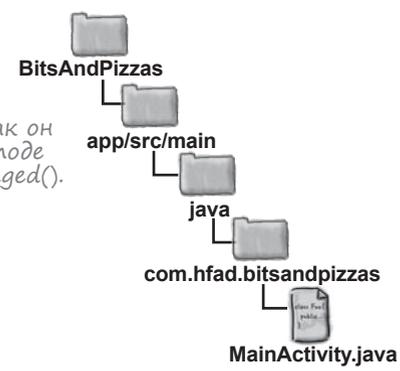
Синхронизировать состояние ActionBarDrawerToggle с состоянием выдвигной панели.

Для передачи информации о любых изменениях конфигурации ActionBarDrawerToggle.

Поручить обработку ActionBarDrawerToggle.



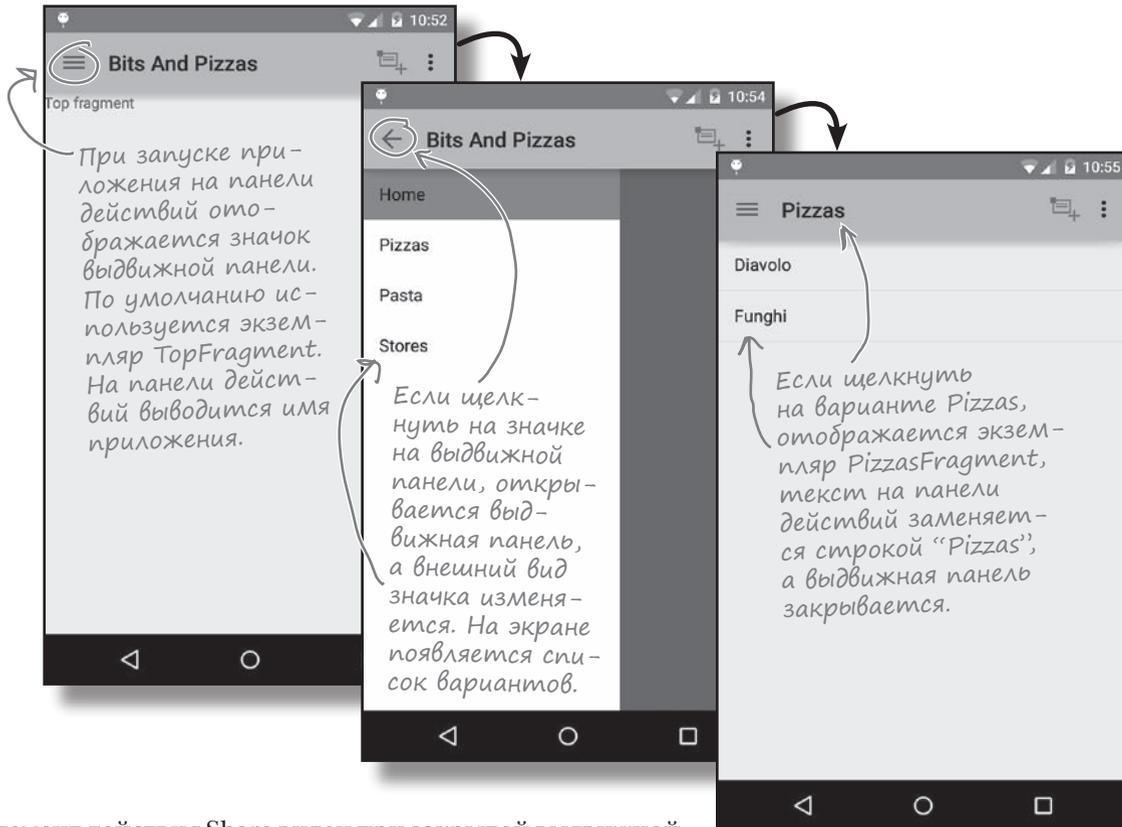
- Добавление фрагментов
- Создание выдвигной панели
- Выбор вариантов
- ActionBarDrawerToggle





Тест-драйв

При запуске приложения отображается активность `MainActivity`. Она включает полностью работоспособную выдвижную панель:



Элемент действия `Share` виден при закрытой выдвижной панели и скрывается, когда она открыта:

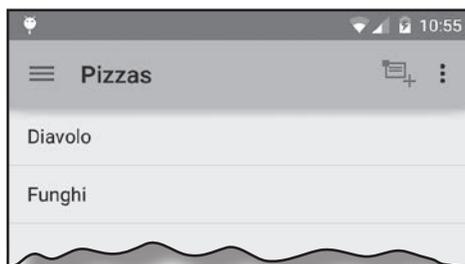


Осталось разобраться еще с одним моментом: необходимо проследить за тем, чтобы при повороте устройства или нажатии кнопки Назад на панели действий отображался правильный текст. Что происходит в текущей версии приложения?

Текст не соответствует фрагменту

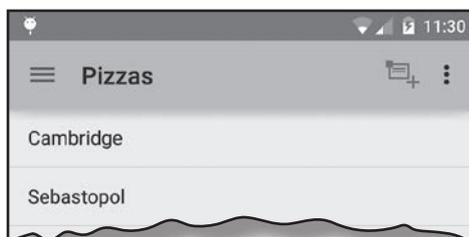
Если щелкнуть на одной из команд на выдвигной панели, заголовок на панели действий меняется в соответствии с отображаемым фрагментом. Например, если щелкнуть на варианте Pizzas, на панели действий появляется текст “Pizzas”:

При выборе вариантов на выдвигной панели текст изменяется в соответствии с выбором.



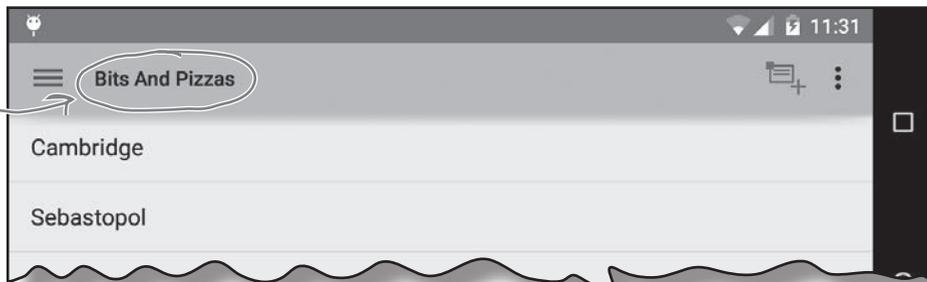
Но если нажать кнопку Назад, заголовок не изменяется в соответствии с отображаемым фрагментом. Допустим, вы выбрали вариант Stores, а затем вариант Pizzas. Сначала на экране появляется список видов пиццы; этот факт отражен в тексте заголовка. Если затем нажать кнопку Назад, приложение возвращается к фрагменту StoresFragment, но в заголовке по-прежнему остается текст “Pizzas”:

Текст в заголовке не изменяется при нажатии кнопки Назад. В данном примере при отображении списка магазинов по-прежнему выводится текст “Pizzas”.



А если повернуть устройство, в заголовок возвращается текст “Bits and Pizzas” независимо от того, какой фрагмент отображается в данный момент:

При повороте устройства происходит сброс текста в панели действий.



Давайте разберемся с этими проблемами. Начнем с синхронизации текста на панели действий при повороте устройства.

Обработка изменений конфигурации

Как вам уже известно, при повороте устройства текущая активность уничтожается и создается заново. Это означает, что любые изменения, внесенные в интерфейс, будут потеряны — включая изменения в тексте заголовка панели действий.

Как и в одной из предыдущих глав, мы воспользуемся методом `onSaveInstanceState()` активности для сохранения позиции текущего выбранного варианта на выдвижной панели. Сохраненное значение будет использовано в методе `onCreate()` для обновления текста на панели действий.

Изменения в коде выделены жирным шрифтом:

```

...
public class MainActivity extends Activity {
    ...
    private int currentPosition = 0;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        //Вывести правильный текст.
        if (savedInstanceState != null) {
            currentPosition = savedInstanceState.getInt("position");
            setActionBarTitle(currentPosition);
        } else {
            selectItem(0);
        }
        ...
    }

    private void selectItem(int position) {
        currentPosition = position;
        ...
    }

    @Override
    public void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        outState.putInt("position", currentPosition);
    }
    ...
}

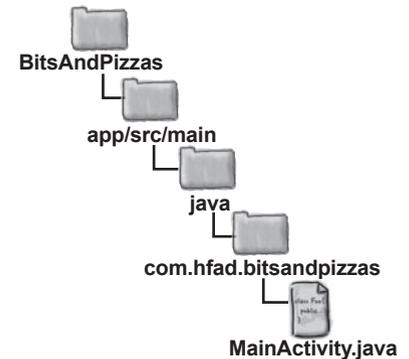
```

По умолчанию `currentPosition` присваивается 0.

Если активность только что создана, использовать `TopFragment`.

Обновить `currentPosition` при выборе варианта.

Сохранить состояние `currentPosition`, если активность готовится к уничтожению.



Если активность была уничтожена и создается заново, взять значение `currentPosition` из предыдущего состояния активности и использовать его для назначения заголовка панели действий.

Реакция на изменения в стеке возврата

И последнее, что нужно сделать — добиться того, чтобы текст на панели действий соответствовал фрагменту, отображаемому при нажатии кнопки Назад. Для этого мы добавим к диспетчеру фрагментов активности слушателя `FragmentManager.OnBackStackChangedListener`. Интерфейс `FragmentManager.OnBackStackChangedListener` отслеживает изменения в стеке возврата. К числу таких изменений относится добавление транзакции фрагмента в стек возврата и нажатие кнопки Назад для возврата к предыдущему элементу стека возврата. Реализация `OnBackStackChangedListener` добавляется к диспетчеру фрагментов, связанному с активностью, следующим образом:

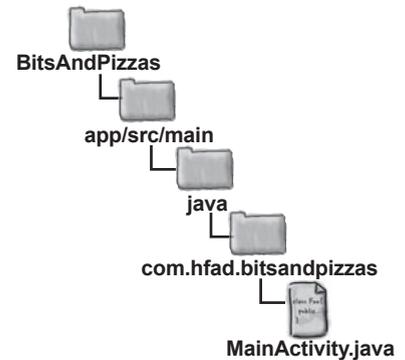
```
getFragmentManager().addOnBackStackChangedListener(
    new FragmentManager.OnBackStackChangedListener() {
        public void onBackStackChanged() {
            //Код, который должен выполняться при изменении
            // в стеке возврата
        }
    }
);
```

При изменении стека возврата вызывается метод `onBackStackChanged()` объекта `OnBackStackChangedListener`. Весь код, который должен выполняться при нажатии кнопки Назад, следует разместить в этом методе.

Метод `onBackStackChanged()` должен:

- ★ Обновить переменную `currentPosition`, чтобы она соответствовала позиции фрагмента, отображаемого в настоящий момент, в списковом представлении.
- ★ Вызвать метод `setActionBarTitle()` с передачей значения `currentPosition`.
- ★ Выделить подсветкой правильный вариант в списковом представлении выдвижной панели; для этого следует вызвать его метод `setItemChecked()`.

В каждом из этих пунктов необходимо знать позицию текущего фрагмента в списковом представлении. Как это проще всего сделать?



Мы добавляем новую реализацию `FragmentManager.OnBackStackChangedListener` и реализуем ее метод `onBackStackChanged()`. Этот метод будет вызываться при каждом изменении в стеке возврата.

Назначение меток фрагментам

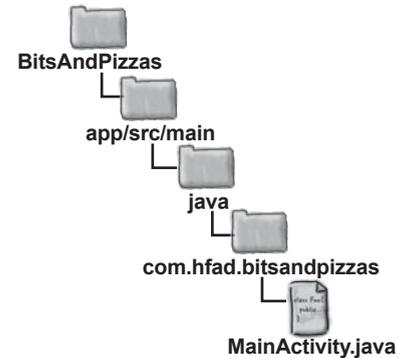
Чтобы определить значение `currentPosition`, мы проверим, фрагмент какого типа в настоящее время связан с активностью. Например, если присоединенный фрагмент является экземпляром `PizzaFragment`, то `currentPosition` будет присвоено значение 1. Для получения ссылки на текущий фрагмент с каждым фрагментом будет связана строковая метка, после чего мы воспользуемся методом `findFragmentByTag()` диспетчера фрагментов для получения фрагмента.

Назначение метки фрагменту происходит в составе транзакции фрагмента. Ниже приведена текущая транзакция фрагмента, которая используется в методе `selectItem()` для замены текущего отображаемого фрагмента:

```
FragmentTransaction ft = getFragmentManager().beginTransaction();
ft.replace(R.id.content_frame, fragment);
ft.addToBackStack(null);
ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);
ft.commit();
```

Для добавления метки к фрагменту при вызове метода `replace()` в транзакции передается дополнительный строковый параметр:

```
FragmentTransaction ft = getFragmentManager().beginTransaction();
ft.replace(R.id.content_frame, fragment, "visible_fragment");
ft.addToBackStack(null);
ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);
ft.commit();
```



При добавлении в стек возврата фрагменту назначается метка "visible_fragment".

В приведенном выше коде к методу `replace()` добавляется метка `"visible_fragment"`. Каждый фрагмент, отображаемый в `MainActivity`, будет помечен этим значением.

Затем метод `findFragmentByTag()` диспетчера фрагментов используется для получения ссылки на фрагмент, связанный с активностью в настоящий момент.

Поиск фрагмента по метке

Чтобы получить фрагмент, связанный с активностью в настоящий момент, мы передадим метку, назначенную в транзакции фрагмента, при вызове `findFragmentByTag()`:

Найти фрагмент с меткой "visible_fragment":

```
FragmentManager fragMan = getFragmentManager();
Fragment fragment = fragMan.findFragmentByTag("visible_fragment");
```

Метод `findFragmentByTag()` сначала проводит поиск по всем фрагментам, связанным с активностью в настоящий момент. Если найти фрагмент с указанным тегом не удалось, он переходит к поиску по всем фрагментам в стеке возврата. Так как в нашем примере всем фрагментам назначается одна и та же метка "visible_fragment", приведенный выше код получит ссылку на фрагмент, который в настоящее время связан с активностью.

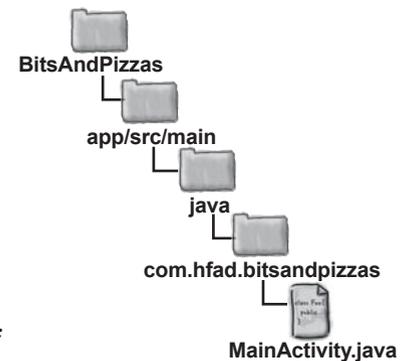
Ниже приведен полный код `OnBackStackListener`. Метод `findFragmentByTag()` используется для получения ссылки на текущий фрагмент. Затем мы проверяем, к какому типу относится фрагмент, чтобы определить значение `currentPosition`:

```
getFragmentManager().addOnBackStackChangeListener(
    new FragmentManager.OnBackStackChangeListener() {
        public void onBackStackChanged() {
            FragmentManager fragMan = getFragmentManager();
            Fragment fragment = fragMan.findFragmentByTag("visible_fragment");
            if (fragment instanceof TopFragment) {
                currentPosition = 0;
            }
            if (fragment instanceof PizzaFragment) {
                currentPosition = 1;
            }
            if (fragment instanceof PastaFragment) {
                currentPosition = 2;
            }
            if (fragment instanceof StoresFragment) {
                currentPosition = 3;
            }
            setActionBarTitle(currentPosition);
            drawerList.setItemChecked(currentPosition, true);
        }
    });
```

Получает фрагмент, в настоящее время связанный с активностью.

Проверить, к какому типу относится фрагмент, и присвоить соответствующее значение `currentPosition`.

Вывести текст на панели действий и выделить правильный вариант в списке на выдвигающей панели.



Вот и все, что необходимо сделать для синхронизации текста на панели действий с отображаемым фрагментом при нажатии кнопки Назад. Прежде чем увидеть, как работает новый код, еще раз просмотрите полный код `MainActivity.java`.

Полный код MainActivity.java

Полный код MainActivity.java выглядит так:

```
package com.hfad.bitsandpizzas;

import android.app.Activity;
import android.app.Fragment;
import android.app.FragmentManager;
import android.app.FragmentTransaction;
import android.content.Intent;
import android.content.res.Configuration;
import android.os.Bundle;
import android.support.v7.app.ActionBarDrawerToggle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.ShareActionProvider;
import android.support.v4.widget.DrawerLayout;
```

Используем класс FragmentManager, его необходимо импортировать.



Все эти классы используются в коде.



```
public class MainActivity extends Activity {
```

```
    private ShareActionProvider shareActionProvider;
    private String[] titles;
    private ListView drawerList;
    private DrawerLayout drawerLayout;
    private ActionBarDrawerToggle drawerToggle;
private int currentPosition = 0;
```

Все эти приватные переменные используются в коде.



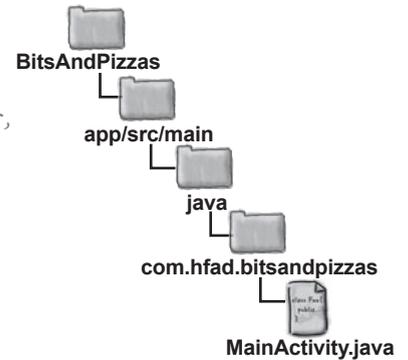
Метод onItemClick() реализации OnItemClickListener вызывается тогда, когда пользователь щелкает на одном из вариантов списка на выдвижной панели.



```
    private class DrawerItemClickListener implements ListView.OnItemClickListener {
        @Override
        public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
            //Код, выполняемый при выборе варианта в списке
            selectItem(position);
        }
    };
```

При выборе варианта в списке на выдвижной панели вызвать метод selectItem().

Продолжение на следующей странице.



MainActivity.java (продолжение)

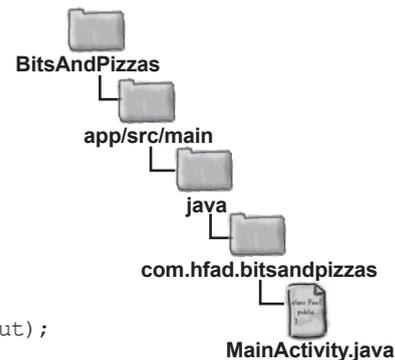
```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    titles = getResources().getStringArray(R.array.titles);
    drawerList = (ListView) findViewById(R.id.drawer);
    drawerLayout = (DrawerLayout) findViewById(R.id.drawer_layout);
    //Инициализация ListView
    drawerList.setAdapter(new ArrayAdapter<String>(this,
        android.R.layout.simple_list_item_activated_1, titles));
    drawerList.setOnItemClickListener(new DrawerItemClickListener());
    //Вывести правильный текст.
    if (savedInstanceState != null) {
        currentPosition = savedInstanceState.getInt("position");
        setActionBarTitle(currentPosition);
    } else {
        selectItem(0);
    }
    //Создание ActionBarDrawerToggle
    drawerToggle = new ActionBarDrawerToggle(this, drawerLayout,
        R.string.open_drawer, R.string.close_drawer) {

        //Вызывается при полном закрытии выдвигной панели
        @Override
        public void onDrawerClosed(View view) {
            super.onDrawerClosed(view);
            invalidateOptionsMenu();
        }

        //Вызывается при полном открытии выдвигной панели.
        @Override
        public void onDrawerOpened(View drawerView) {
            super.onDrawerOpened(drawerView);
            invalidateOptionsMenu();
        }
    };
}

```



← Заполнить список-
ное представление
на выдвигной пане-
ли и обеспечить его
реакцию на щелчки.

← Если активность была
уничтожена и создана заново,
вывести правильный текст
в заголовке панели действий.

← По умолчанию
используется
TopFragment.

← При открытии или закрытии выд-
вигной панели вызывается метод
invalidateOptionsMenu, так как мы
хотим изменить состав элементов
на панели действий.

→ Продолжение
на следующей
странице.

MainActivity.java (продолжение)

Этот код также является частью метода onCreate().

```
drawerLayout.setDrawerListener(drawerToggle);
getActionBar().setDisplayHomeAsUpEnabled(true);
getActionBar().setHomeButtonEnabled(true);
```

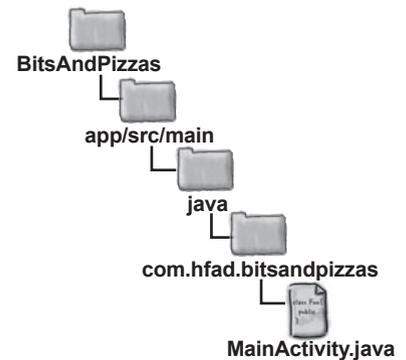
← Включить кнопку Вверх на панели действий, чтобы она могла использоваться для открытия выдвижной панели.

```
getFragmentManager().addOnBackStackChangeListener(
    new FragmentManager.OnBackStackChangeListener() {
        public void onBackStackChanged() {
            FragmentManager fragMan = getFragmentManager();
            Fragment fragment = fragMan.findFragmentByTag("visible_fragment");
            if (fragment instanceof TopFragment) {
                currentPosition = 0;
            }
            if (fragment instanceof PizzaFragment) {
                currentPosition = 1;
            }
            if (fragment instanceof PastaFragment) {
                currentPosition = 2;
            }
            if (fragment instanceof StoresFragment) {
                currentPosition = 3;
            }
            setActionBarTitle(currentPosition);
            drawerList.setItemChecked(currentPosition, true);
        }
    }
);
```

↑
Вызывается при изменении стека возврата.

↑
Проверить, экземпляром какого класса является фрагмент, связанный с активностью, и присвоить currentPosition соответствующее значение.

↑
Вывести текст на панели действий и выделить правильный вариант в списке на выдвижной панели.



→
Продолжение на следующей странице.

MainActivity.java (продолжение)

Метод `selectItem()` вызывается при выборе пользователем варианта в списке на выдвигной панели.

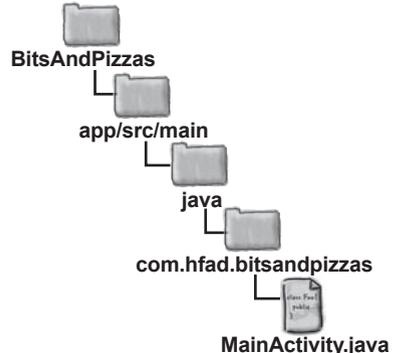
```
private void selectItem(int position) {  
    // Обновить информацию заменой фрагментов  
    currentPosition = position;  
    Fragment fragment;  
    switch(position) {  
        case 1:  
            fragment = new PizzaFragment();  
            break;  
        case 2:  
            fragment = new PastaFragment();  
            break;  
        case 3:  
            fragment = new StoresFragment();  
            break;  
        default:  
            fragment = new TopFragment();  
    }  
    FragmentTransaction ft = getFragmentManager().beginTransaction();  
    ft.replace(R.id.content_frame, fragment, "visible_fragment");  
    ft.addToBackStack(null);  
    ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);  
    ft.commit();  
    //Назначение заголовка панели действий  
    setActionBarTitle(position);  
    //Закрытие выдвигной панели  
    drawerLayout.closeDrawer(drawerList);  
}
```

↑
Закрыть выдвигную панель.

↑
↑
↑
Выбрать отображаемый фрагмент в зависимости от позиции варианта, выбранного пользователем в списке на выдвигной панели.

↓
Отобразить фрагмент.

← Ввести правильный текст заголовка на панели действий.



Продолжение на следующей странице. →

MainActivity.java (продолжение)

```

@Override
public boolean onPrepareOptionsMenu(Menu menu) {
    // Если выдвижная панель открыта, скрыть элементы, связанные с контентом
    boolean drawerOpen = drawerLayout.isDrawerOpen(drawerList);
    menu.findItem(R.id.action_share).setVisible(!drawerOpen);
    return super.onPrepareOptionsMenu(menu);
}

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // Синхронизировать состояние выключателя после onRestoreInstanceState.
    drawerToggle.syncState();
}

@Override
public void onConfigurationChanged(Configuration newConfig) {
    super.onConfigurationChanged(newConfig);
    drawerToggle.onConfigurationChanged(newConfig);
}

@Override
public void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    outState.putInt("position", currentPosition);
}

private void setActionBarTitle(int position) {
    String title;
    if (position == 0){
        title = getResources().getString(R.string.app_name);
    } else {
        title = titles[position];
    }
    getActionBar().setTitle(title);
}

```

Отображать действие Share, если выдвижная панель закрыта; скрыть его, если выдвижная панель открыта.

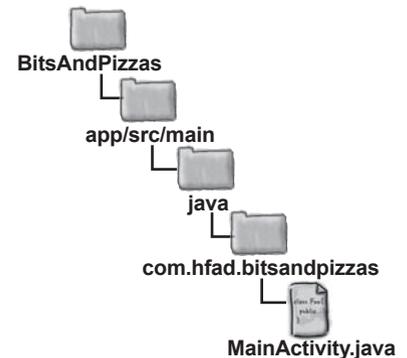
Синхронизировать состояние ActionBarDrawerToggle с состоянием выдвижной панели.

Передать информацию об изменениях конфигурации ActionBarDrawerToggle.

Сохранить состояние currentPosition при уничтожении активности.

Задать текст заголовка панели действий в соответствии с отображаемым фрагментом.

Продолжение на следующей странице.



MainActivity.java (продолжение)

```

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Заполнение меню; элементы добавляются на панель действий (если она есть).
    getMenuInflater().inflate(R.menu.menu_main, menu);
    MenuItem menuItem = menu.findItem(R.id.action_share);
    shareActionProvider = (ShareActionProvider) menuItem.getActionProvider();
    setIntent("This is example text");
    return super.onCreateOptionsMenu(menu);
}

private void setIntent(String text) {
    Intent intent = new Intent(Intent.ACTION_SEND);
    intent.setType("text/plain");
    intent.putExtra(Intent.EXTRA_TEXT, text);
    shareActionProvider.setShareIntent(intent);
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    if (drawerToggle.onOptionsItemSelected(item)) {
        return true;
    }

    switch (item.getItemId()) {
        case R.id.action_create_order:
            //Код, выполняемый при щелчке на действии Create Order
            Intent intent = new Intent(this, OrderActivity.class);
            startActivity(intent);
            return true;
        case R.id.action_settings:
            //Код, выполняемый при щелчке на действии Settings.
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
}
}

```

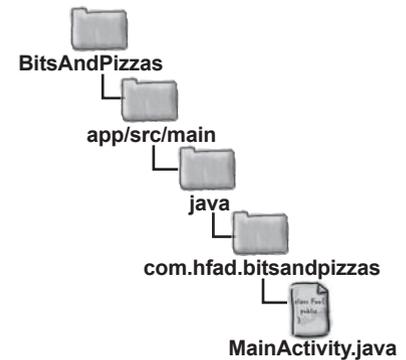
Добавить элементы из файла ресурсов меню на панель действий.

Назначить действию Share интент для передачи информации.

Этот метод вызывается, когда пользователь щелкает на элементе на панели действий.

Если щелчок сделан на ActionBarDrawerToggle, доверить обработку компоненту.

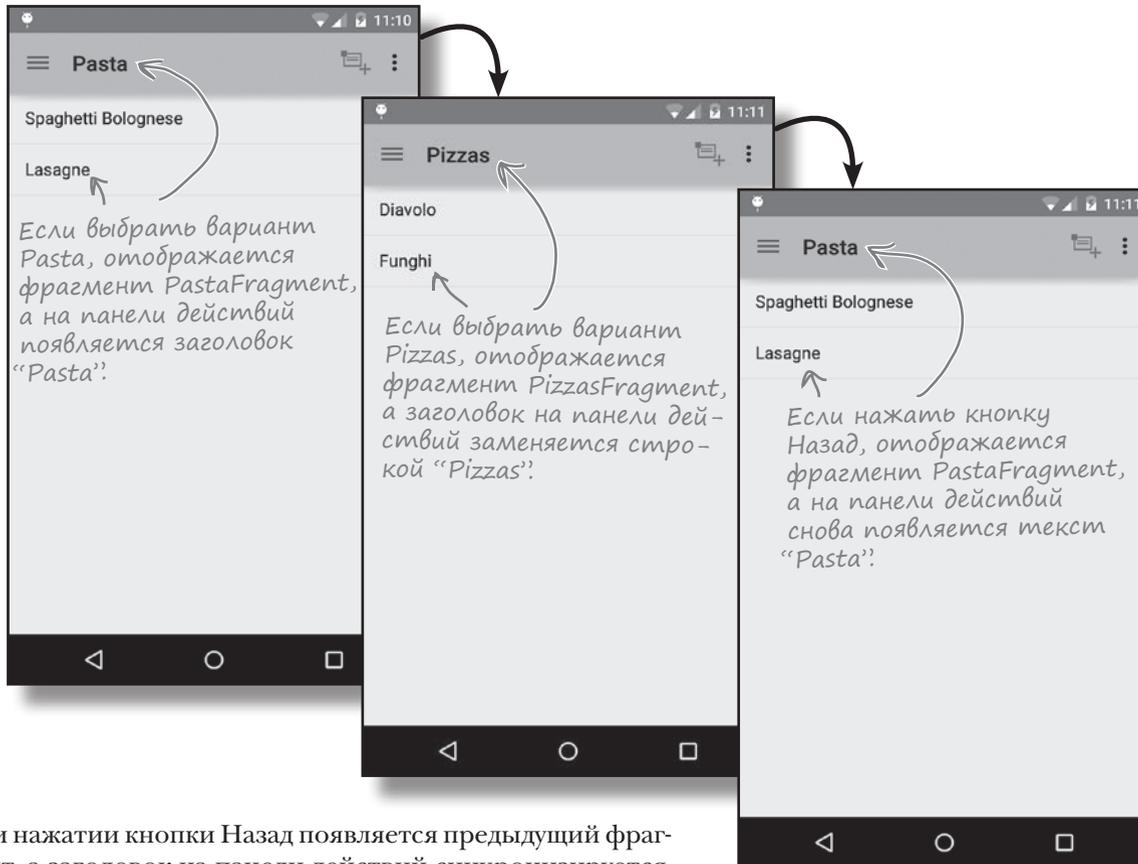
Если щелчок сделан на действии Create Order, запустить OrderActivity.



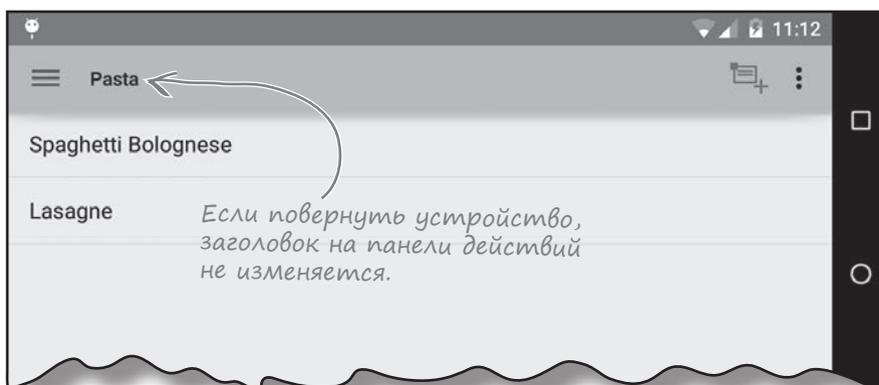


Тест-драйв

Посмотрим, что происходит при запуске приложения.



При нажатии кнопки Назад появляется предыдущий фрагмент, а заголовок на панели действий синхронизируется с изменением. Также синхронизация заголовка сохраняется и при повороте устройства.





Ваш инструментарий Android

Глава 10 осталась позади, а ваш инструментарий пополнился выдвижными панелями.

Весь код для этой главы можно загрузить по адресу <https://tinyurl.com/HeadFirstAndroid>.

КЛЮЧЕВЫЕ МОМЕНТЫ



- Используйте макет `DrawerLayout` для создания активности с навигационной выдвижной панелью. Используйте выдвижную панель для перехода к основным разделам своего приложения.
- Если вы используете панель действий, назначьте `ActionBarDrawerToggle` слушателем `DrawerListener`. Это позволит вам реагировать на открытие и закрытие выдвижной панели, а также добавить на панель действий значок для открытия и закрытия выдвижной панели.
- Чтобы изменить состав элементов панели действий во время выполнения, вызовите `invalidateOptionsMenu()` и включите изменения в метод `onPrepareOptionsMenu()` активности.
- Чтобы реагировать на изменения в стеке возврата, реализуйте `FragmentManager.OnBackStackChangedListener()`.
- Метод `findFragmentByTag()` диспетчера фрагментов ищет фрагменты с заданной меткой.

Работа с базами данных



Говоря о «постоянных отношениях», я имел в виду реляционные базы данных.

Какая бы информация ни использовалась в приложении — рекордные счета или тексты сообщений в социальной сети — эту информацию необходимо где-то хранить. В Android для долгосрочного хранения данных обычно используется база данных SQLite. В этой главе вы узнаете, как создать базу данных, добавить в нее таблицы и заполнить данными — все это делается при помощи удобных вспомогательных объектов SQLite. Затем будет показано, как выполнить безопасное обновление структуры базы данных и как вернуться к предыдущей версии в случае необходимости.

Возвращение в Starbuzz

В главе 6 мы создали приложение для сети кофеен Starbuzz. В этом приложении пользователь переходит между несколькими экранами и может просмотреть информацию о напитках, предлагаемых в Starbuzz.



Приложение Starbuzz получает информацию о напитках от класса `Drink`, содержащего информацию о напитках из меню Starbuzz. Хотя такое решение упрощает построение первой версии приложения, существуют и более совершенные способы хранения и загрузки данных.

В следующих двух главах мы изменим приложение Starbuzz и добьемся того, чтобы данные загружались из базы данных SQLite. В этой главе вы узнаете, как создать базу данных, а в следующей главе мы покажем, как связать с ней активности.

Android хранит информацию в базах данных SQLite

Всем приложениям приходится решать задачи хранения данных. В мире Android для этой цели обычно используется **база данных SQLite**. Почему именно SQLite?



Минимальные затраты ресурсов.

Для работы большинства систем управления базами данных необходим специальный процесс сервера базы данных. SQLite обходится без сервера; база данных SQLite представляет собой обычный файл. Когда база данных не используется, она не расходует процессорное время. Это особенно важно на мобильных устройствах, чтобы избежать разрядки аккумулятора.



Оптимизация для одного пользователя.

С базой данных взаимодействует только наше приложение, поэтому можно обойтись без идентификации с именем пользователя и паролем.



Надежность и быстрота.

Базы данных SQLite невероятно надежны. Они поддерживают транзакции баз данных (другими словами, если при обновлении нескольких блоков данных что-то пойдет не так, SQLite сможет вернуться к исходному состоянию). Кроме того, операции чтения и записи данных реализуются на оптимизированном коде C. Этот код не только быстро работает, но и сокращает объем необходимых вычислительных ресурсов.

В этой главе
изложены
основы SQLite.

Если вы собираетесь
интенсивно работать
с данными в своих
приложениях,
мы рекомендуем
обратиться к учебникам
по SQLite и SQL.

Где хранится база данных?

Android автоматически создает для каждого приложения папку, в которой хранятся базы данных этого приложения. Когда мы создаем базу данных для приложения Starbuzz, она будет храниться в следующей папке:

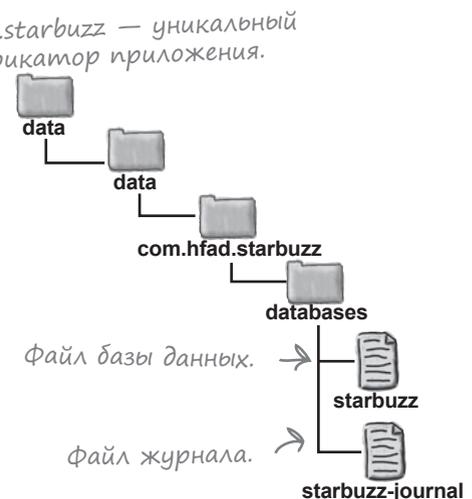
`/data/data/com.hfad.starbuzz/databases`

com.hfad.starbuzz — уникальный идентификатор приложения.

В этой папке приложение может хранить несколько баз данных. Каждая база данных состоит из двух файлов.

Имя первого — **файла базы данных** — соответствует имени базы данных: например, “starbuzz”. Это основной файл баз данных SQLite; в нем хранятся все данные.

Второй файл — **файл журнала**. Его имя состоит из имени базы данных и суффикса “-journal” — например, “starbuzz-journal”. В файле журнала хранится информация обо всех изменениях, внесенных в базу данных. Если в работе с данными возникнет проблема, Android использует данные журнала для отмены (или отката) последних изменений.



Android включает классы SQLite

Система Android включает набор классов для управления базой данных SQLite. Основная часть этой работы выполняется тремя типами объектов.



Помощник SQLite

Помощник SQLite создается расширением класса SQLiteOpenHelper. Он предоставляет средства для создания и управления базами данных.



Класс базы данных SQLite

Класс SQLiteDatabase предоставляет доступ к базе данных. Его можно сравнить с классом SQLConnection в JDBC.

Курсор

Класс Cursor предназначен для чтения в базу данных, но сравнить ResultSet в JDBC.



Мы воспользуемся этими объектами и покажем, как создать в приложении базу данных SQLite, которая заменит класс Drink.

Часто задаваемые вопросы

В: Если при подключении к базе данных не указывается имя пользователя и пароль, то как обеспечивается безопасность данных?

О: Каталог, в котором хранятся базы данных приложения, доступен для чтения только для самого приложения. Безопасность доступа к базе данных обеспечивается на уровне операционной системы.

В: Возможно ли написать приложение Android, которое работает с внешней базой данных — например, Oracle?

О: Ничто не мешает вам работать с другими базами данных по сети, но не стоит забывать об экономии ресурсов, используемых Android. Например, обращение к базе данных через веб-службу может более экономно расходовать заряд аккумулятора. Пока вы не взаимодействуете с базой данных, никакие ресурсы не расходуются.

В: Почему Android не использует JDBC для работы с базами данных SQLite?

О: Если мы знаем, что будем работать с базой данных SQLite, использование JDBC будет явным «перебором». Те уровни драй-

веров баз данных, которые обеспечивают выдающуюся гибкость JDBC, на устройствах Android будут только расходовать заряд аккумулятора.

В: Каталог базы данных находится в каталоге приложения?

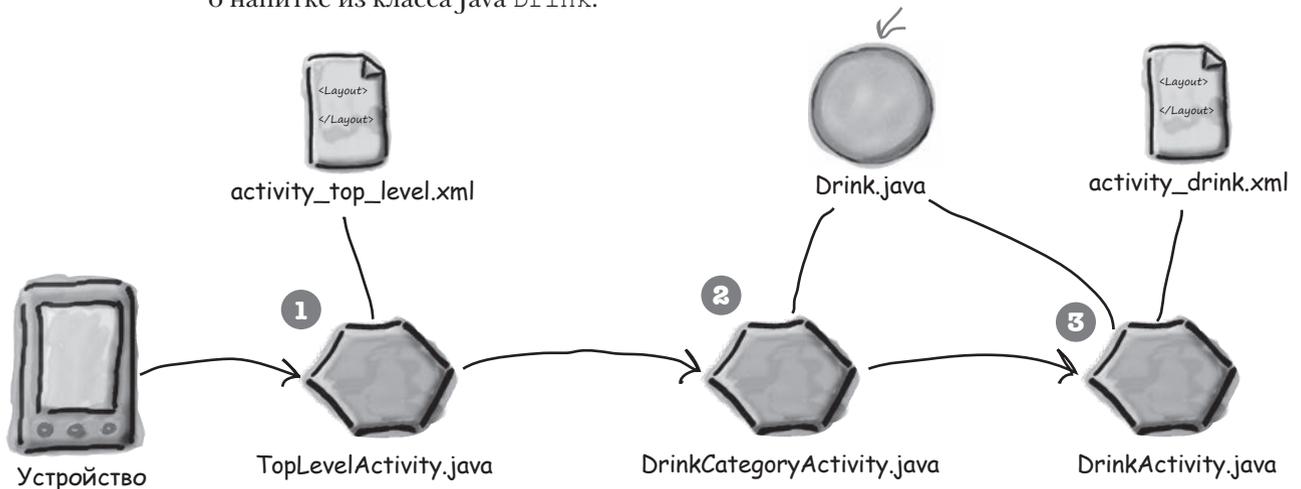
О: Нет. База данных хранится в другом каталоге, отдельно от кода приложения. Это позволяет установить обновленную версию приложения без потери информации в базе данных.

Текущая структура приложения Starbuzz

Вспомним текущую структуру приложения Starbuzz:

- 1 **TopLevelActivity** содержит список вариантов: **Drinks** (напитки), **Food** (блюда) и **Stores** (магазины).
- 2 Когда пользователь выбирает вариант **Drinks**, запускается активность **DrinkCategoryActivity**. Эта активность выводит список напитков, полученных из класса Java **Drink**.
- 3 Когда пользователь щелкает на одном из напитков, подробная информация о нем выводится в **DrinkActivity**. **DrinkActivity** получает подробную информацию о напитке из класса Java **Drink**.

*Приложение в настоящее время получает данные из класса **Drink**.*



Как изменится структура приложения при переходе на базу данных SQLite?

✦ ✦ ✦
Задание!
✦ ✦ ✦

Так как в этой главе мы будем вносить изменения в приложение Starbuzz, откройте исходный проект Starbuzz в Android Studio.

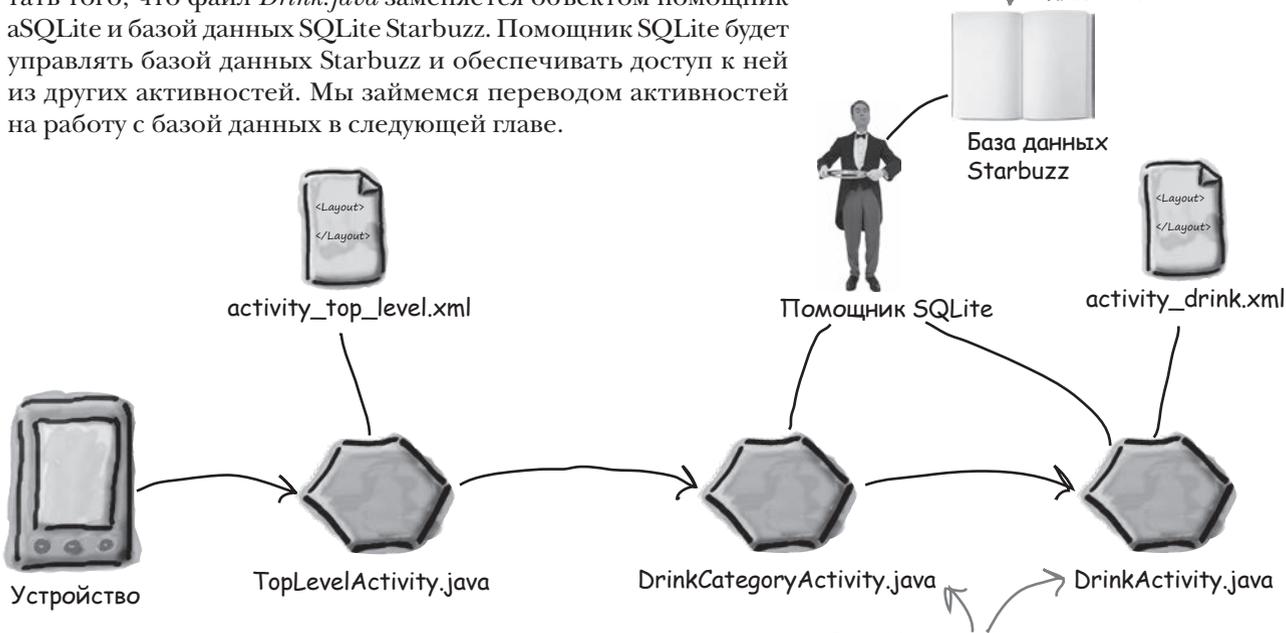
Переход на работу с базой данных

Мы воспользуемся объектом помощника SQLite для создания базы данных SQLite, которая может использоваться нашим приложением Starbuzz. Чтобы заменить класс Java Drink базой данных, помощник SQLite должен сделать следующее:

- 1 **Создание базы данных.**
Прежде чем делать что-либо еще, нужно приказать помощнику SQLite создать версию 1 (первую версию) базы данных Starbuzz.
- 2 **Создание таблицы Drink и заполнение ее информацией о напитках.**
После того как база данных будет создана, в ней создается таблица. Структура таблицы должна соответствовать атрибутам текущего класса Drink; таким образом, в таблице должно храниться название, описание и идентификатор ресурса изображения для каждого напитка. Затем в таблицу будет добавлена информация о трех напитках.

Структура приложения почти не изменяется, если не считать того, что файл *Drink.java* заменяется объектом помощник aSQLite и базой данных SQLite Starbuzz. Помощник SQLite будет управлять базой данных Starbuzz и обеспечивать доступ к ней из других активностей. Мы займемся переводом активностей на работу с базой данных в следующей главе.

Информация о напитках будет храниться в базе данных вместо класса Drink.



Для начала познакомимся поближе с помощником SQLite.

В следующей главе мы внесем изменения в активности, работающие с классом Drink, чтобы вместо класса Java они использовали базу данных.

Помощник SQLite управляет базой данных



Создание базы данных
Создание таблицы

Класс `SQLiteOpenHelper` упрощает задачи создания и сопровождения баз данных. Считайте, что это своего рода личный ассистент, который берет на себя служебные операции по управлению базами данных.

Рассмотрим некоторые типичные задачи, в решении которых вам поспособствует помощник SQLite.

Создание базы данных

При первой установке приложения файл базы данных не существует. Помощник SQLite проследит за тем, чтобы файл базы данных был создан с правильным именем и с правильной структурой таблиц.



Обеспечение доступа к базе данных

Нашему приложению не обязательно знать все подробности о том, где хранится файл базы данных. Помощник SQLite предоставляет удобный объект, представляющий базу данных, и приложение работает с базой через этот объект — тогда, когда сочтет нужным.

Помощник SQLite



Сопровождение баз данных

Может случиться так, что структура базы данных изменится со временем. Положитесь на помощника SQLite — он преобразует старую версию в новенькую и блестящую, с учетом самых последних изменений в структуре базы данных.

Создание помощника SQLite

Чтобы создать помощника SQLite, напишите класс, расширяющий `SQLiteOpenHelper`. При этом вы *должны* переопределить методы `onCreate()` и `onUpgrade()`. Эти методы являются обязательными.

Метод `onCreate()` вызывается при первом создании базы данных на устройстве. Он должен включать весь код, необходимый для создания таблиц, используемых в приложении.

Метод `onUpgrade()` вызывается, когда база данных нуждается в обновлении. Например, если вам потребуется внести изменения в структуру уже используемой базы, код обновления следует разместить именно в этом методе.

В нашем приложении будет использоваться помощник SQLite с именем `StarbuzzDatabaseHelper`. Создайте этот класс в своем проекте Starbuzz: выделите папку `app/src/main/java/com.hfad.starbuzz` на панели структуры проекта и выберите команду `File→New...→Java Class`. Присвойте классу имя “`StarbuzzDatabaseHelper`” и замените его содержимое следующим кодом:

```
package com.hfad.starbuzz;

import android.database.sqlite.SQLiteOpenHelper;
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;

class StarbuzzDatabaseHelper extends SQLiteOpenHelper {

    StarbuzzDatabaseHelper(Context context) {
    }

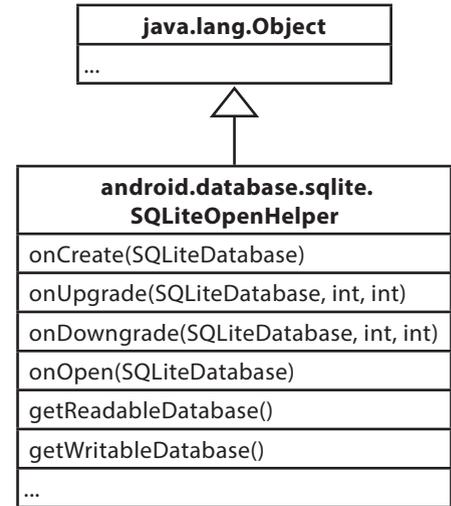
    @Override
    public void onCreate(SQLiteDatabase db) {
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    }
}
```

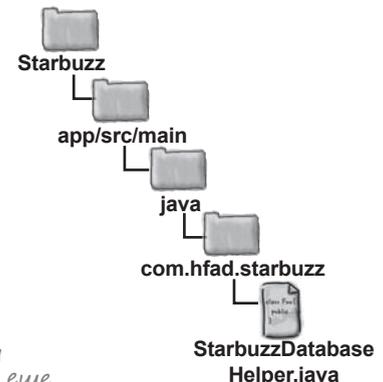
Помощники SQLite должны расширять класс `SQLiteOpenHelper`.

Присутствие методов `onCreate()` и `onUpgrade()` обязательно. Пока мы оставим эти методы пустыми, но еще вернемся к ним позднее в этой главе.

→  Создание базы данных
Создание таблицы



↑
Класс `SQLiteOpenHelper` является subclasses `Object`.



Чтобы метод SQLite делал что-то полезное, необходимо добавить код в его методы. Прежде всего следует сообщить помощнику SQLite, какую базу данных он должен создать.

1. Определение базы данных

Для создания базы данных помощнику SQLite необходимы два важных параметра.

Во-первых, необходимо задать имя базы данных. Присваивание имени гарантирует, что база данных останется на устройстве после закрытия. Если имя не задано, то база данных будет существовать только в памяти, и при закрытии информация пропадет.

Во-вторых, необходимо указать версию базы данных. Номер версии представляет собой целое число, начиная с 1. Помощник SQLite использует номер версии для определения того, нуждается ли база данных в обновлении.

Имя и версия базы данных передаются конструктору суперкласса SQLiteOpenHelper. В нашем примере базе данных присваивается имя "starbuzz", а поскольку это первая версия, ей присваивается номер версии 1. Код создания базы приведен ниже (замените им свою версию *StarbuzzDatabaseHelper.java*):

← Базы данных, хранящиеся только в памяти, могут быть полезны во время тестирования приложения.



Имя: "starbuzz"
Версия: 1

База данных SQLite

...

```
class StarbuzzDatabaseHelper extends SQLiteOpenHelper {

    private static final String DB_NAME = "starbuzz"; // Имя базы данных
    private static final int DB_VERSION = 1; // Версия базы данных
```

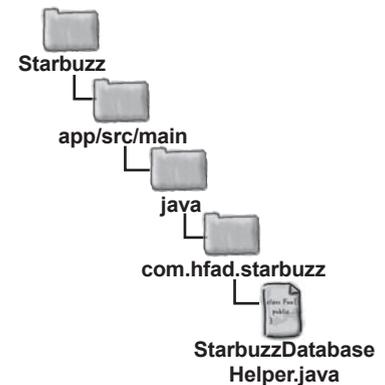
```
    StarbuzzDatabaseHelper(Context context) {
        super(context, DB_NAME, null, DB_VERSION);
    }
```

...

```
}
```

↑ Этот параметр используется для работы с курсорами. Тема курсоров рассматривается в следующей главе.

← Вызываем конструктор супер-класса SQLiteOpenHelper и передаем ему имя и версию базы данных.



Конструктор задает информацию о базе данных, но сама база данных в этой точке не создается. Помощник SQLite ожидает, пока приложение обратится к базе данных, и создает базу данных в этой точке.

После того как помощник SQLite получит информацию о создаваемой базе данных, можно переходить к определению таблиц.

Мы сделали все необходимое для того, чтобы база данных создавалась в нужный момент.



Создание базы данных
Создание таблицы

Внутри базы данных SQLite

Информация в базах данных SQLite хранится в таблицах. Таблица состоит из строк, а строки делятся на столбцы. Один столбец содержит один элемент данных — например, число или блок текста.

Вам нужно создать таблицу для всех видов данных, которые должны храниться в базе. Скажем, в приложении Starbuzz необходимо создать таблицу для информации о напитках. Такая таблица выглядит примерно так:

Таблица состоит из столбцов `_id`, `NAME`, `DESCRIPTION` и `IMAGE_RESOURCE_ID`. Класс `Drink` содержит атрибуты с похожими именами.

_id	NAME	DESCRIPTION	IMAGE_RESOURCE_ID
1	"Latte"	"Espresso and steamed milk"	54543543
2	"Cappuccino"	"Espresso, hot milk and steamed-milk foam"	654334453
3	"Filter"	"Our best drip coffee"	44324234

Некоторые столбцы могут назначаться первичными ключами. Первичный ключ однозначно идентифицирует одну строку данных. Если некоторый столбец является первичным ключом, то база данных не позволит создать строки с одинаковыми значениями этого столбца. Мы рекомендуем создавать в таблицах один целочисленный столбец первичного ключа с именем `_id`. Это связано с тем, что код Android жестко запрограммирован на использование числового столбца с именем `_id`, и его отсутствие может создать проблемы.

Типы данных и класс хранения

Каждый столбец в таблице рассчитан на хранение данных определенного типа. Например, в нашей таблице DRINK в столбце DESCRIPTION могут храниться только текстовые данные. Ниже перечислены основные типы столбцов, используемые в SQLite, и данные, которые в них могут храниться:

INTEGER	Любое целое число
TEXT	Любые символьные данные
REAL	Любое вещественное число
NUMERIC	Логическое значение, дата, дата-время
BLOB	Двоичные большие объекты

В отличие от многих систем баз данных, в SQLite не нужно указывать размер столбца. Во внутренней реализации тип данных преобразуется в намного более универсальный класс хранения. Это означает, что вы можете в общих чертах указать, какие данные собираетесь хранить, но не обязаны указывать их конкретный размер.

В Android принято присваивать столбцу первичного ключа имя `_id`. Код Android ожидает, что в данных присутствует столбец `_id`. Нарушение этого правила затруднит выборку информации из базы данных и ее включение в пользовательский интерфейс.

Таблицы создаются командами SQL

Каждое приложение, взаимодействующее с SQLite, использует стандартный язык баз данных SQL (Structured Query Language). SQL применяется почти во всех видах баз данных. Чтобы создать таблицу DRINK, необходимо выдать соответствующую команду на языке SQL.

Команда SQL для создания таблицы выглядит так:

Столбец `_id` является первичным ключом.

```
CREATE TABLE DRINK ( _id INTEGER PRIMARY KEY AUTOINCREMENT,
NAME TEXT,
DESCRIPTION TEXT,
IMAGE_RESOURCE_ID INTEGER)
```

Имя таблицы → `DRINK`
Столбцы таблицы. → `NAME TEXT`, `DESCRIPTION TEXT`, `IMAGE_RESOURCE_ID INTEGER`

Команда CREATE TABLE сообщает, какие столбцы должны присутствовать в таблице, и данные какого типа должны в этих столбцах храниться. Столбец `_id` является первичным ключом таблицы, а специальное ключевое слово AUTOINCREMENT означает, что при занесении в таблицу новой строки SQLite автоматически сгенерирует для нее уникальный целочисленный идентификатор.

Метод onCreate() вызывается при создании базы данных

Помощник SQLite отвечает за то, чтобы база данных SQLite была создана в момент ее первого использования. Сначала на устройстве создается пустая база данных, после чего вызывается метод onCreate() помощника SQLite. При вызове метода onCreate() передается объект SQLiteDatabase. Мы можем воспользоваться этим объектом для выполнения в методе команды SQL:

```
SQLiteDatabase.execSQL(String sql);
```

← *Выполнить команду SQL, заданную в строковом виде.*

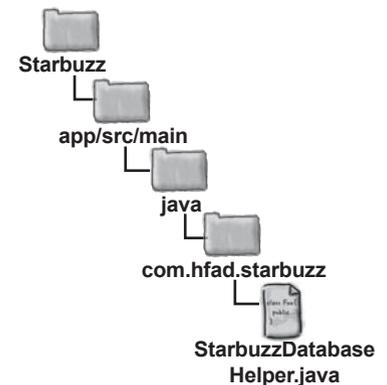
Метод получает один параметр — команду SQL, которую нужно выполнить.

Полный код метода onCreate() выглядит так:

```
@Override
public void onCreate(SQLiteDatabase db) {
    db.execSQL("CREATE TABLE DRINK ( "
        + "_id INTEGER PRIMARY KEY AUTOINCREMENT, "
        + "NAME TEXT, "
        + "DESCRIPTION TEXT, "
        + "IMAGE_RESOURCE_ID INTEGER);");
}
```

Команда создает пустую таблицу DRINK — но что делать, если таблицу потребуется заполнить исходными данными?

**Класс
SQLiteDatabase
предоставляет
доступ к базе
данных.**



`insert()`

Вставка данных методом `insert()`

Класс `SQLiteDatabase` содержит несколько методов для вставки, обновления и удаления данных. Все эти возможности (начиная со вставки данных) будут рассмотрены на нескольких ближайших страницах. Если вам потребуется заполнить таблицу `SQLite` данными, используйте метод `insert()` класса `SQLiteDatabase`. Этот метод вставляет данные в базу и возвращает идентификатор записи. Если метод не смог создать запись, он возвращает значение `-1`.

Чтобы использовать метод `insert()`, необходимо указать таблицу и вставляемые значения. Для определения вставляемых значений создается объект `ContentValues`, в котором данные сохраняются в виде пар «имя/значение»:

```
ContentValues drinkValues = new ContentValues();
```

Для добавления пар «имя/значение» в объект `ContentValues` используется метод `put()` этого объекта. Чтобы вставить строку данных в таблицу `DRINK`, мы указываем имена столбцов в таблице `DRINK` и значения, которые сохраняются в каждом столбце:

```
ContentValues drinkValues = new ContentValues();
drinkValues.put("NAME", "Latte");
drinkValues.put("DESCRIPTION", "Espresso and steamed milk");
drinkValues.put("IMAGE_RESOURCE_ID", R.drawable.latte);
```

Наконец, вызов метода `insert()` объекта `SQLiteDatabase` вставляет значения в таблицу `DRINK`:

```
db.insert("DRINK", null, drinkValues);
```

В результате выполнения этих строк кода в таблицу `DRINK` будет вставлена следующая запись:

<code>_id</code>	<code>NAME</code>	<code>DESCRIPTION</code>	<code>IMAGE_RESOURCE_ID</code>
1	"Latte"	"Espresso and steamed milk"	54543543

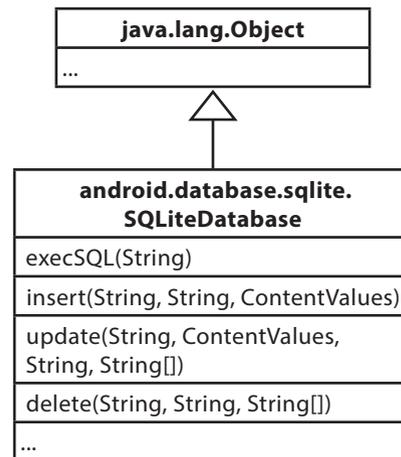
Обобщенная форма метода `insert()` выглядит так:

```
db.insert(String table, String nullColumnHack, ContentValues values);
```

Строковое значение `nullColumnHack` указывать не обязательно; чаще всего в этом параметре передается `null`, как в приведенном примере. Этот параметр присутствует на тот случай, если объект `ContentValues` пуст и вы хотите вставить в таблицу пустую строку. `SQLite` не позволит вставить пустую строку без указания имени хотя бы одного столбца; параметр `nullColumnHack` позволяет сделать это.



Создание базы данных
Создание таблицы



Класс `SQLiteDatabase` является subclasses `Object`.

В столбце `DESCRIPTION` заносится значение "Espresso and steamed milk".

Для каждого вводимого значения требуется отдельный вызов метода `put()`.

В таблице появляется новая запись.

Этот вызов вставляет в таблицу одну строку данных. Чтобы вставить несколько строк, необходимо использовать серию вызовов `insert()`.

Обновление записей методом update()

Для обновления существующей информации в SQLite используется метод `update()` класса `SQLiteDatabase`. Этот метод вносит изменения в записи, хранящиеся в базе данных, и возвращает количество обновленных записей. Чтобы использовать метод `update()`, необходимо указать таблицу, обновляемые значения и условия их обновления. Метод получает следующие параметры:

```
public int update (String table,
                  ContentValues values,
                  String whereClause,
                  String[] whereArgs)
```

В следующем примере значение столбца `DESCRIPTION` заменяется строкой "Tasty", если поле названия напитка содержит "Latte":

```
ContentValues drinkValues = new ContentValues();
drinkValues.put ("DESCRIPTION", "Tasty");
db.update ("DRINK",
          drinkValues,
          "NAME = ?",
          new String[] {"Latte"});
```

← Помещает значение "Tasty" в столбец DESCRIPTION.

← Обновить столбец DESCRIPTION значением "Tasty" в строках таблицы DRINK, для которых NAME = "Latte".

<u>id</u>	<u>NAME</u>	<u>DESCRIPTION</u>	<u>IMAGE_RESOURCE_ID</u>
1	"Latte"	"Espresso and steamed milk" "Tasty"	54543543

Первый параметр метода `update()` содержит имя таблицы, в которой обновляется информация (в данном случае таблица `DRINK`).

Второй параметр указывает, какие значения должны использоваться для обновления. Как и в случае с методом `insert()`, для определения набора значений создается объект `ContentValues` для хранения пар «имя/значение» данных:

```
ContentValues drinkValues = new ContentValues();
drinkValues.put ("DESCRIPTION", "Tasty");
```

Третий параметр задает условия отбора обновляемых записей. В приведенном примере `"NAME = ?"` означает, что столбец `NAME` должен быть равен некоторому значению. Символ `?` обозначает значение столбца, которое определяется содержимым последнего параметра (в данном случае "Latte").

Также возможно указать несколько критериев отбора; на следующей странице мы покажем, как это делается.



Будьте
осторожны!

Если в двух
последних
параметрах
`update()`
передается

значение **null**, будут обновлены **ВСЕ** записи в таблице.

Например, вызов

```
db.update ("DRINK",
          drinkValues,
          null, null);
```

обновит все записи в таблице `DRINK`.

Сложные условия



Создание базы данных
Создание таблицы

Если в запросе должно использоваться сложное условие, проследите за тем, чтобы порядок перечисления условий соответствовал порядку значений. Например, вот как выполняется обновление записей из таблицы DRINK, у которых столбец названия напитка содержит текст “Latte” или столбец описания содержит текст “Our best drip coffee”.+

```
db.update("DRINK",
    drinkValues,
    "NAME = ? OR DESCRIPTION = ?",
    new String[] {"Latte", "Our best drip coffee"});
```

Это означает: Если NAME = "Latte" или DESCRIPTION = "Our best drip coffee".

Значения условий должны относиться к строковому типу String, даже если столбец, к которому относится условие, содержит данные другого типа. В таких случаях значения необходимо преобразовать к типу String. Например, следующий вызов возвращает записи DRINK, в которых столбец _id равен 1:

```
db.update("DRINK",
    drinkValues,
    "_id = ?",
    new String[] {Integer.toString(1)});
```

Целое значение 1 преобразуется в String.

Удаление записей методом delete()

Метод delete() класса SQLiteDatabase работает по тому же принципу, как и только что рассмотренный метод update(). Он имеет следующую форму:

```
public int delete (String table,
    String whereClause,
    String[] whereArgs)
```

Например, удаление из базы данных всех записей, у которых столбец названия содержит текст “Latte”, выполняется следующим образом:

```
db.delete("DRINK",
    "NAME = ?",
    new String[] {"Latte"});
```

Видите, как это похоже на метод update()? Удаляется вся строка данных.

_id	NAME	DESCRIPTION	IMAGE_RESOURCE_ID
1	"Latte"	"Espresso and steamed milk"	54543543

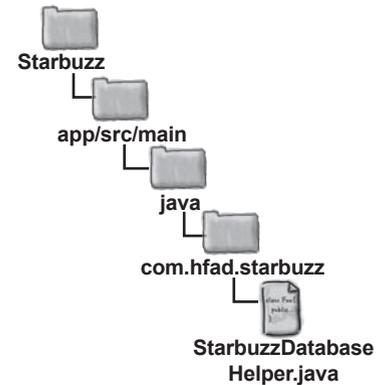
В первом параметре передается имя таблицы, из которой удаляются записи (в данном случае DRINK). Два других параметра позволяют точно описать, какие записи требуется удалить (NAME = “Latte”).

Итак, вы знаете, какие операции могут выполняться при работе с данными в таблицах SQLite. Теперь вам известно все необходимое для создания баз данных SQLite, создания таблиц и их заполнения данными. На следующей странице мы применим эти знания на практике в коде, использующем помощника SQLite.

Kog StarbuzzDatabaseHelper



Create database
Create table



Ниже приведен полный код *StarbuzzDatabaseHelper.java* (внесите изменения в свой код):

```

package com.hfad.starbuzz;

import android.content.ContentValues;
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

class StarbuzzDatabaseHelper extends SQLiteOpenHelper{

    private static final String DB_NAME = "starbuzz"; // Имя базы данных
    private static final int DB_VERSION = 1; // Версия базы данных

    StarbuzzDatabaseHelper(Context context){
        super(context, DB_NAME, null, DB_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db){
        db.execSQL("CREATE TABLE DRINK (_id INTEGER PRIMARY KEY AUTOINCREMENT, "
            + "NAME TEXT, "
            + "DESCRIPTION TEXT, "
            + "IMAGE_RESOURCE_ID INTEGER);");

        insertDrink(db, "Latte", "Espresso and steamed milk", R.drawable.latte);
        insertDrink(db, "Cappuccino", "Espresso, hot milk and steamed-milk foam",
            R.drawable.cappuccino);
        insertDrink(db, "Filter", "Our best drip coffee", R.drawable.filter);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {

    }

    private static void insertDrink(SQLiteDatabase db, String name,
        String description, int resourceId) {
        ContentValues drinkValues = new ContentValues();
        drinkValues.put("NAME", name);
        drinkValues.put("DESCRIPTION", description);
        drinkValues.put("IMAGE_RESOURCE_ID", resourceId);
        db.insert("DRINK", null, drinkValues);
    }
}
  
```

Имя базы данных и номер версии.

Метод onCreate() вызывается при создании базы данных; мы используем его для создания таблицы и вставки данных.

Создать таблицу DRINK.

Вставить данные каждого напитка в отдельную строку.

Метод onUpgrade() вызывается тогда, когда возникает необходимость в обновлении базы данных.

Сейчас мы познакомимся с ним поближе.

Так как нужно добавить несколько напитков, мы создаем отдельный метод.

Что делает код помощника SQLite



Создание базы данных

Создание таблицы

- 1 Пользователь устанавливает приложение и запускает его.**
Когда приложение пытается обратиться к базе данных, помощник SQLite проверяет, существует ли база данных.



SQLite helper

Вам нужна база данных, сэр? Позвольте проверить, существует ли она.

- 2 Если база данных не существует, то она создается.**
Базе данных назначается имя и номер версии, указанные в помощнике SQLite.



Помощник SQLite



База данных SQLite

Имя: "starbuzz"
Версия: 1

- 3 При создании базы данных вызывается метод onCreate() помощника SQLite.**
Метод добавляет в базу данных таблицу DRINK и заполняет ее записями.

Ваша база данных, сэр. Желаете что-нибудь еще?

onCreate()

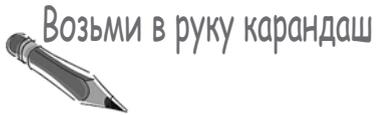


Помощник SQLite



База данных SQLite

Имя: "starbuzz"
Версия: 1



Ниже приведен метод onCreate () класса SQLiteOpenHelper. Укажите, какие значения будут вставлены в столбцы NAME и DESCRIPTION таблицы DRINK после того, как метод onCreate () завершит свою работу.

```
@Override
public void onCreate(SQLiteDatabase db) {
    ContentValues espresso = new ContentValues();
    espresso.put("NAME", "Espresso");
    ContentValues americano = new ContentValues();
    americano.put("NAME", "Americano");
    ContentValues latte = new ContentValues();
    latte.put("NAME", "Latte");
    ContentValues filter = new ContentValues();
    filter.put("DESCRIPTION", "Filter");
    ContentValues mochachino = new ContentValues();
    mochachino.put("NAME", "Mochachino");

    db.execSQL("CREATE TABLE DRINK ( "
        + "_id INTEGER PRIMARY KEY AUTOINCREMENT, "
        + "NAME TEXT, "
        + "DESCRIPTION TEXT);");
    db.insert("DRINK", null, espresso);
    db.insert("DRINK", null, americano);
    db.delete("DRINK", null, null);
    db.insert("DRINK", null, latte);
    db.update("DRINK", mochachino, "NAME = ?", new String[] {"Espresso"});
    db.insert("DRINK", null, filter);
}
```

Указывать
значение
столбца _id
не нужно.

_id	NAME	DESCRIPTION



Возьми в руку карандаш

Решение

Ниже приведен метод onCreate () класса SQLiteOpenHelper. Укажите, какие значения будут вставлены в столбцы NAME и DESCRIPTION таблицы DRINK после того, как метод onCreate () завершит свою работу.

```
@Override
public void onCreate(SQLiteDatabase db) {
    ContentValues espresso = new ContentValues();
    espresso.put("NAME", "Espresso");
    ContentValues americano = new ContentValues();
    americano.put("NAME", "Americano");
    ContentValues latte = new ContentValues();
    latte.put("NAME", "Latte");
    ContentValues filter = new ContentValues();
    filter.put("DESCRIPTION", "Filter");
    ContentValues mochachino = new ContentValues();
    mochachino.put("NAME", "Mochachino");

    db.execSQL("CREATE TABLE DRINK ( "
        + "_id INTEGER PRIMARY KEY AUTOINCREMENT, "
        + "NAME TEXT, "
        + "DESCRIPTION TEXT);");
    db.insert("DRINK", null, espresso);
    db.insert("DRINK", null, americano);
    db.delete("DRINK", null, null);
    db.insert("DRINK", null, latte);
    db.update("DRINK", mochachino, "NAME = ?", new String[] {"Espresso"});
    db.insert("DRINK", null, filter);
}
```

Создать таблицу со столбцами `_id`, `NAME` и `DESCRIPTION`.

← Вставить Espresso в столбец NAME.

← Вставить Americano в столбец NAME.

← Удалить все данные.

← Вставить Latte в столбец NAME.

↑ Вставить Filter в столбец DESCRIPTION.

↑ Записать в столбец NAME текст Mochachino для записей, у которых NAME содержит Espresso. Ни одна запись не обновляется.

<code>_id</code>	<code>NAME</code>	<code>DESCRIPTION</code>
	Latte	
		Filter

А если структура базы данных изменится?

Пока что вы видели, как создать базу данных SQLite, в которой ваше приложение сможет хранить данные. Но что, если когда-нибудь в будущем в структуру базы данных потребуется внести изменения?

Представьте, что многие пользователи уже установили ваше приложение Starbuzz на своих устройствах, и вы решили добавить в таблицу DRINK новый столбец FAVORITE. Как распространить это изменение на устройствах новых и существующих пользователей?



Конечно, можно изменить команду CREATE TABLE в методе onCreate() — но, похоже, проблемы это не решит. Что делать, если на устройстве уже установлена старая версия базы данных?

Когда возникает необходимость в изменении структуры базы данных приложения, приходится учитывать два основных сценария.

Первый — пользователь еще не устанавливал ваше приложение, и база данных на его устройстве не создавалась. В этом случае помощник SQLite создает базу данных при первом обращении к базе данных и выполняет метод onCreate().

Второй — пользователь устанавливает новую версию приложения с другой версией базы данных. Если помощник SQLite обнаруживает, что установленная база данных не соответствует текущей версии приложения, вызывается метод onUpgrade() или onDowngrade().

Как же помощник SQLite проверяет актуальность базы данных?



Номера версий баз данных SQLite

Чтобы определить, нуждается ли база данных SQLite в обновлении, помощник SQLite проверяет ее номер версии. Номер версии присваивается базе данных в помощнике SQLite — он передается при вызове конструктора суперкласса SQLiteOpenHelper. Ранее мы задали номер версии базы данных следующим образом:

```
...
private static final String DB_NAME = "starbuzz";
private static final int DB_VERSION = 1;

StarbuzzDatabaseHelper(Context context) {
    super(context, DB_NAME, null, DB_VERSION);
}
...
```

При создании базы данных ей назначается номер версии из помощника SQLite, и вызывается метод onCreate() помощника SQLite.

Если появится необходимость в модификации базы данных, измените номер версии в коде помощника SQLite. При этом номер версии может как *повышаться*, так и *понижаться*:

```
...
private static final int DB_VERSION = 2;
...
```

← Здесь номер версии увеличивается, так что база данных будет обновлена.

В большинстве случаев номер версии повышается. Понижение встречается только в тех ситуациях, когда нужно отменить изменения, внесенные при предыдущем обновлении.

Когда пользователь устанавливает новейшую версию приложения на своем устройстве и приложение в первый раз обращается к базе данных, помощник SQLite сравнивает свой номер версии с номером версии базы данных на устройстве.

Если номер версии в коде помощника SQLite *выше* номера версии базы, вызывается метод **onUpgrade()** помощника SQLite. Если номер версии в коде помощника SQLite *ниже* номера версии базы, вместо этого вызывается метод **onDowngrade()**.

После выполнения одного из этих методов номер версии базы данных заменяется номером версии из кода помощника SQLite.



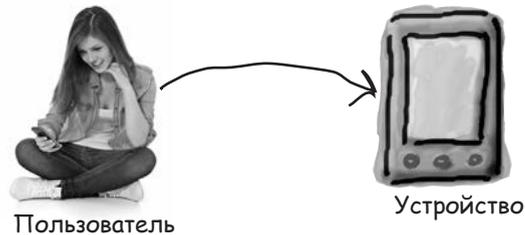
Для любознательных

У баз данных SQLite имеется номер версии, который используется помощником SQLite, и внутренний номер версии схемы. При любом изменении схемы базы данных — например, структуры таблицы — номер версии схемы увеличивается на 1. Вы не можете управлять этим значением, оно используется во внутренней реализации SQLite.

Обновление базы данных: сводка

Вот что происходит при выпуске новой версии приложения, в которой номер версии помощника SQLite изменяется с 1 на 2:

- 1 Пользователь устанавливает новую версию приложения и запускает ее.



- 2 Если пользователь впервые устанавливает это приложение, то база данных не существует, и помощник SQLite создает ее. Помощник SQLite присваивает базе данных имя и номер версии, указанные в его коде.



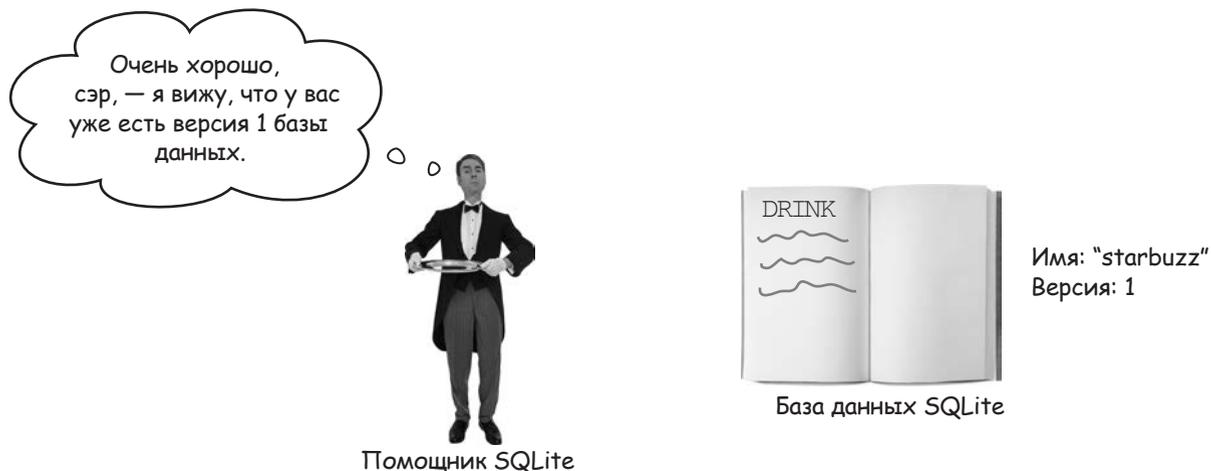
- 3 Когда база данных будет создана, вызывается метод `onCreate()` помощника SQLite. Метод `onCreate()` содержит код, заполняющий базу данных информацией.



История продолжается...

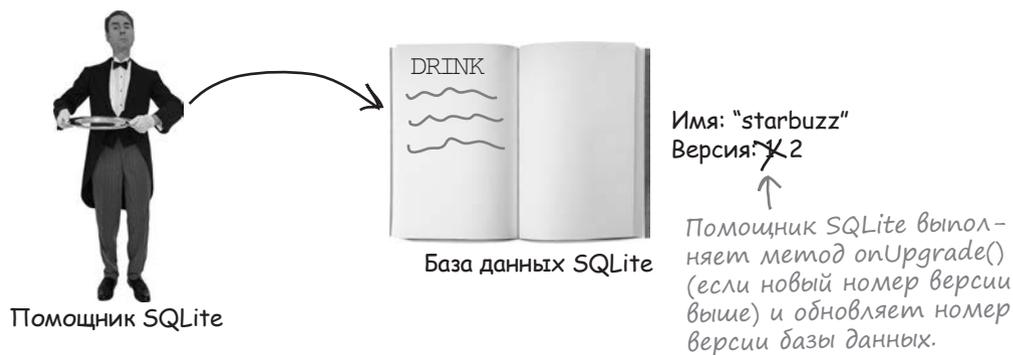
- 4 Если пользователь установил предыдущую версию приложения и обращался к базе данных, то база данных уже существует.

Если база данных существует, помощник SQLite не пытается создавать ее заново.



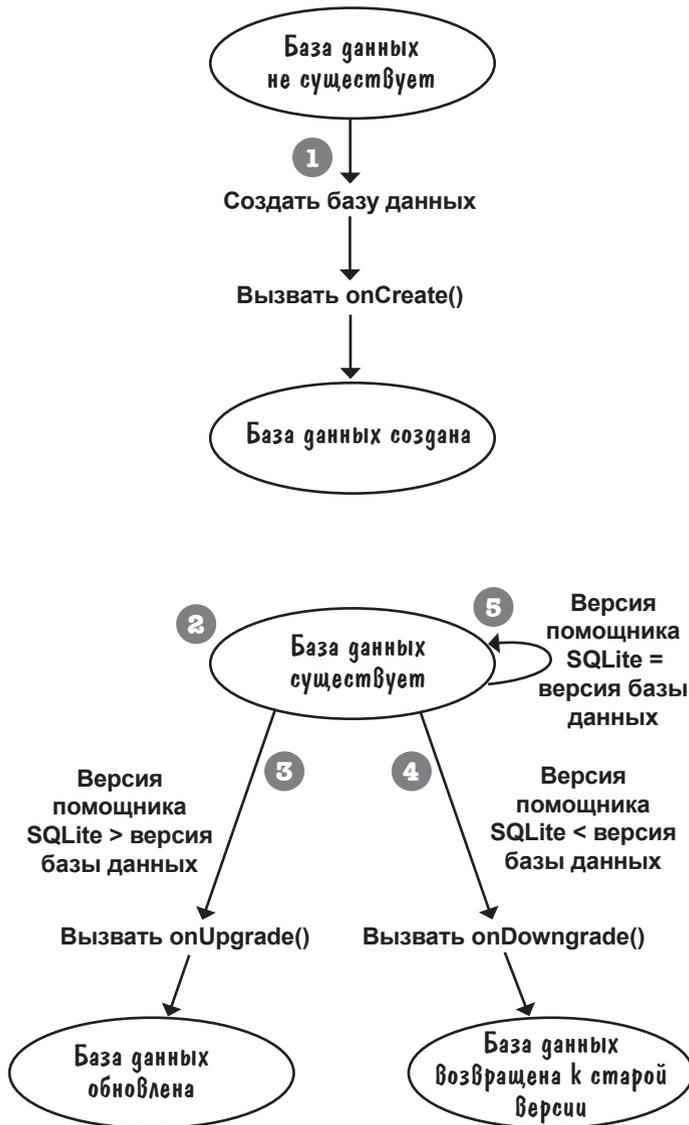
- 5 Помощник SQLite сравнивает номер версии базы данных с номером версии в коде помощника SQLite.

Если номер версии помощника SQLite выше номера версии базы данных, то вызывается метод `onUpgrade()`. Если номер версии помощника SQLite ниже, то вызывается метод `onDowngrade()`. Затем помощник SQLite приводит номер версии базы данных в соответствие с номером версии в своем коде.



Как помощник SQLite принимает решения

На приведенной ниже схеме показано, как действует помощник SQLite в зависимости от того, существует или нет база данных, и от ее номера версии.



Теперь вы знаете, при каких обстоятельствах вызываются методы `onUpgrade()` и `onDowngrade()`. Посмотрим, как использовать их на практике.

1 Если база данных не существует, то помощник SQLite создает базу данных, после чего выполняется метод помощника `onCreate()`.

2 Если база данных уже существует, то помощник SQLite сравнивает номер версии, хранящийся в базе данных, с номером версии в коде помощника.

3 Если номер версии в коде помощника SQLite больше номера версии базы данных, вызывается метод `onUpgrade()`. После обновления помощник SQLite изменяет номер версии базы данных.

4 Если номер версии в коде помощника SQLite меньше номера версии базы данных, вызывается метод `onDowngrade()`. После выполнения возврата помощник SQLite изменяет номер версии базы данных.

5 Если номер версии в коде помощника SQLite равен номеру версии базы данных, не вызывается ни один из методов. База данных уже находится в актуальном состоянии.



Метод onUpgrade()

Метод onUpgrade() получает три параметра — базу данных SQLite, номер версии самой базы и номер версии, переданный суперклассу SQLiteOpenHelper:

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    //Здесь размещается ваш код
}
```

Текущая версия
базы данных. Новая версия из кода
помощника SQLite.

Не забудьте: новая версия
должна быть выше старой.

Номера версий важны: с их помощью можно указать, какие изменения должны вноситься в базу данных в зависимости от того, какая версия базы уже установлена на устройстве пользователя. Допустим, некоторый код должен выполняться только в том случае, если существующая база данных имеет версию 1. Код будет выглядеть примерно так:

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    if (oldVersion == 1) {
        //Код, выполняемый для версии 1
    }
}
```

Этот код выполняется
← только в том случае,
если база данных пользо-
вателя имеет версию 1.

Номера версий также могут использоваться для применения последовательных обновлений:

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    if (oldVersion == 1) {
        //Код, выполняемый для версии 1
    }
    if (oldVersion < 3) {
        //Код, выполняемый для версии 1 или 2
    }
}
```

Этот код выполняется
← только в том случае,
если база данных пользо-
вателя имеет версию 1.

← Этот код выполняется в том случае,
если база данных пользователя имеет
версию 1 или 2. Если у пользователя
установлена база данных с версией 1,
будут выполнены оба блока кода.

В результате вы можете быть уверены в том, что в базу данных будут внесены все необходимые изменения — независимо от того, какая версия базы данных установлена на устройстве пользователя.

Метод onDowngrade() работает аналогичным образом, но выполняется при понижении версии. Он рассматривается на следующей странице.

Метод onDowngrade()

Метод `onDowngrade()` используется реже, чем метод `onUpgrade()`, так как он используется для возврата базы данных к предыдущей версии. Данная возможность может быть полезна, если вы выпустите версию приложения с изменениями в базе данных, а затем обнаружите, что она содержит ошибки. Метод `onDowngrade()` позволяет отменить изменения и вернуть базу данных к предыдущей версии.

Как и `onUpgrade()`, метод `onDowngrade()` получает три параметра: базу данных SQLite, номер версии базы данных и номер версии, переданный суперклассу `SQLiteOpenHelper`:

```
@Override
public void onDowngrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    //Здесь размещается ваш код
}
```

↑ ↑
Для выполнения возврата новая версия должна быть меньше старой.

Как и в случае с методом `onUpgrade()`, номера версий могут использоваться для отмены изменений, относящихся к конкретной версии. Например, если изменения должны вноситься в базу данных с номером версии 3, используйте код следующего вида:

```
@Override
public void onDowngrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    if (oldVersion == 3) {
        //Код, выполняемый для версии 3
    }
}
```

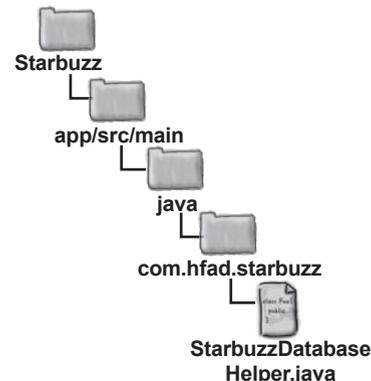
← Этот код выполняется только в том случае, если база данных пользователя имеет версию 3 и вы хотите вернуть ее к меньшей версии.

Посмотрим, как этот способ отмены изменений применяется на практике.



Обновление базы данных

Предположим, мы хотим обновить структуру базы данных с добавлением нового столбца в таблицу DRINK. Так как изменения должны распространяться как на новых, так и на существующих пользователей, соответствующий код должен быть включен как в метод onCreate(), так и в метод onUpgrade(). Метод onCreate() гарантирует, что новый столбец будет присутствовать у всех новых пользователей, а метод onUpgrade() позаботится о том, чтобы он был и у всех существующих пользователей. Вместо того, чтобы повторять похожий код в методах onCreate() и onUpgrade(), мы создадим отдельный метод updateMyDatabase(), который будет вызываться из onCreate() и onUpgrade(). Код, в настоящее время находящийся в onCreate(), будет перемещен в новый метод updateMyDatabase(), и к нему добавится код создания дополнительного столбца. При таком подходе весь код базы данных будет храниться в одном месте, а нам будет проще управлять изменениями, вносимыми при каждом обновлении базы:



```

...
@Override
public void onCreate(SQLiteDatabase db) {
    updateMyDatabase(db, 0, DB_VERSION);
}

```

← Вместо того, чтобы создавать здесь таблицу DRINK, мы создадим ее в методе updateMyDatabase().

```

@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    updateMyDatabase(db, oldVersion, newVersion);
}

```

← Вызвать метод updateMyDatabase() из onUpgrade() с передачей параметров.

```

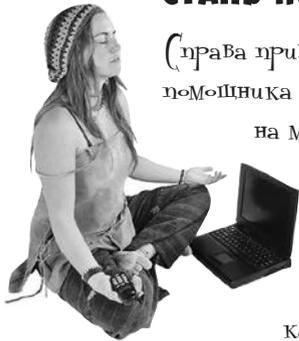
private void updateMyDatabase(SQLiteDatabase db, int oldVersion, int newVersion) {
    if (oldVersion < 1) {
        db.execSQL("CREATE TABLE DRINK (_id INTEGER PRIMARY KEY AUTOINCREMENT, "
            + "NAME TEXT, "
            + "DESCRIPTION TEXT, "
            + "IMAGE_RESOURCE_ID INTEGER);");
        insertDrink(db, "Latte", "Espresso and steamed milk", R.drawable.latte);
        insertDrink(db, "Cappuccino", "Espresso, hot milk and steamed-milk foam",
            R.drawable.cappuccino);
        insertDrink(db, "Filter", "Our best drip coffee", R.drawable.filter);
    }
    if (oldVersion < 2) {
        //Код добавления нового столбца
    }
}
...

```

Ранее этот код находился в методе onCreate().

← Этот код будет выполняться в том случае, если у пользователя уже установлена версия 1 базы данных.

СТАНЬ помощником SQLite



Справа приведены примеры кода помощника SQLite. Представьте себя на месте помощника SQLite и скажите, какой из сегментов кода (из помеченных буквами) будет выполняться для каждого из пользователей, перечисленных ниже. Мы решили одну задачу за вас, чтобы вам было проще взяться за дело.

Пользователь 1 запускает приложение впервые.

Сегмент А. На устройстве пользователя нет базы данных, выполняется метод onCreate().

У пользователя 2 установлена база данных с номером версии 1.

У пользователя 3 установлена база данных с номером версии 2.

У пользователя 4 установлена база данных с номером версии 3.

У пользователя 5 установлена база данных с номером версии 4.

У пользователя 6 установлена база данных с номером версии 5.

...

```
class MyHelper extends SQLiteOpenHelper{

    StarbuzzDatabaseHelper(Context context){
        super(context, "fred", null, 4);
    }

    @Override
    public void onCreate(SQLiteDatabase db){
        A //Выполняется код А
        ...
    }

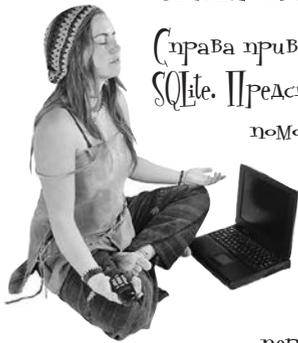
    @Override
    public void onUpgrade(SQLiteDatabase db,
        int oldVersion,
        int newVersion){

        if (oldVersion < 2) {
            B //Выполняется код В
            ...
        }
        if (oldVersion == 3) {
            C //Выполняется код С
            ...
        }
        D //Выполняется код D
        ...
    }

    @Override
    public void onDowngrade(SQLiteDatabase db,
        int oldVersion,
        int newVersion){

        if (oldVersion == 3) {
            E //Выполняется код E
            ...
        }
        if (oldVersion < 6) {
            F //Выполняется код F
            ...
        }
    }
}
```

СТАНЬ помощником SQLite. Решение



Справа приведены примеры кода помощника SQLite. Представьте себя на месте помощника SQLite и скажите, какой из сегментов кода (из помеченных буквами) будет выполняться для каждого из пользователей, перечисленных ниже. Мы решили одну задачу за Вас, чтобы Вам было проще взглянуть за дело.

Пользователь 1 запускает приложение впервые.

Сегмент А. На устройстве пользователя нет базы данных, выполняется метод `onCreate()`.

У пользователя 2 установлена база данных с номером версии 1.

Сегмент В, затем D. База данных обновляется с `oldVersion == 1`.

У пользователя 3 установлена база данных с номером версии 2.

Сегмент D. База данных обновляется с `oldVersion == 2`.

У пользователя 4 установлена база данных с номером версии 3.

Сегмент С, затем D. База данных обновляется с `oldVersion == 3`.

У пользователя 5 установлена база данных с номером версии 4.

Ни один. У пользователя установлена правильная версия базы данных.

У пользователя 6 установлена база данных с номером версии 5.

Сегмент F. База данных возвращается к старой версии с `oldVersion == 5`.

...

```
class MyHelper extends SQLiteOpenHelper{

    StarbuzzDatabaseHelper(Context context){
        super(context, "fred", null, 4);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        A //Выполняется код А
        ...
    }

    @Override
    public void onUpgrade(SQLiteDatabase db,
        int oldVersion,
        int newVersion){
        if (oldVersion < 2) {
            B //Выполняется код В
            ...
        }
        if (oldVersion == 3) {
            C //Выполняется код С
            ...
        }
        D //Выполняется код D
        ...
    }

    @Override
    public void onDowngrade(SQLiteDatabase db,
        int oldVersion,
        int newVersion){
        if (oldVersion == 3) {
            E //Выполняется код E
            ...
        }
        if (oldVersion < 6) {
            F //Выполняется код F
            ...
        }
    }
}
```

Новая версия базы данных 4.

Метод `onCreate()` выполняется только в том случае, если у пользователя не установлена база данных.

Выполняется, если у пользователя установлена версия 1.

Выполняется, если у пользователя установлена версия 3.

Выполняется, если у пользователя установлена версия 1, 2 или 3.

Никогда не выполняется. Если у пользователя установлена версия 3, то вызываться должен метод `onUpgrade`, а не `onDowngrade`.

Выполняется, если у пользователя установлена версия 5. Для выполнения метода `onDowngrade()` у пользователя должна быть установлена версия выше 4 — текущего номера версии в коде помощника.

Обновление существующей базы данных



Обновление базы данных

При обновлении базы данных обычно выполняются два вида действий:

- ★ **Изменение записей базы данных.**
 Ранее в этой главе было показано, как выполнять вставку, изменение и удаление записей в базе данных методами SQLiteDatabase `insert()`, `update()` и `delete()`. Обновление базы данных может сопровождаться добавлением новых записей, изменением или удалением уже существующих записей.
- ★ **Изменение структуры базы данных.**
 Вы уже знаете, как создавать таблицы в базе данных. Также возможны операции добавления столбцов в существующие таблицы, переименования и даже полного удаления таблиц.

На нескольких ближайших страницах вы узнаете, как выполняются эти операции. Начнем с изменения структуры базы данных и добавления столбцов в существующие таблицы.

Добавление новых столбцов средствами SQL

Ранее в этой главе было показано, как создавать таблицы командой SQL `CREATE TABLE`:

```
CREATE TABLE DRINK (_id INTEGER PRIMARY KEY AUTOINCREMENT,
NAME TEXT,
DESCRIPTION TEXT,
IMAGE_RESOURCE_ID INTEGER)
```

← Столбец `_id` является первичным ключом.

↑ Имя таблицы

↑ Столбцы таблицы

Язык SQL также может использоваться для изменения существующих таблиц — эта задача решается командой `ALTER TABLE`. Например, команда добавления столбца в таблицу выглядит так:

```
ALTER TABLE DRINK
ADD COLUMN FAVORITE NUMERIC
```

← Имя таблицы.

← Добавляемый столбец.

В этом примере в таблицу `DRINK` добавляется столбец с именем `FAVORITE`, в котором хранятся числовые значения. На следующей странице мы покажем, как переименовать таблицу или удалить ее из базы данных.



Переименование таблиц

Команда ALTER TABLE может использоваться для переименования таблиц. Например, если вы захотите переименовать таблицу DRINK в FOO, это делается так:

```
ALTER TABLE DRINK ← Текущее имя таблицы.  
RENAME TO FOO ← Новое имя таблицы.
```

Удаление таблиц

Кроме создания и модификации таблиц, также возможно их удаление командой DROP TABLE:

```
DROP TABLE DRINK ← Имя удаляемой таблицы.
```

Эта команда пригодится в том случае, если одна из таблиц в схеме базы данных стала лишней и вы хотите удалить ее для экономии места.

Выполнение команд SQL методом execSQL()

Как было показано ранее, команды SQL выполняются методом execSQL() класса SQLiteDatabase:

```
SQLiteDatabase.execSQL(String sql);
```

Например, команда SQL для добавления нового столбца FAVORITE в таблицу DRINK выполняется так:

```
db.execSQL("ALTER TABLE DRINK ADD COLUMN FAVORITE NUMERIC;");
```

Метод execSQL() может использоваться в любой момент, когда вам потребуется выполнить команды SQL с базой данных. Итак, мы рассмотрели основные действия, которые могут выполняться при обновлении базы данных; применим новые знания в классе *StarbuzzDatabaseHelper.java*.

Полный код помощника SQLite

Ниже приведен полный код класса `StarbuzzDatabaseHelper.java`, который добавляет в таблицу DRINK новый столбец FAVORITE. Приведите свой код в соответствии с нашим (изменения выделены жирным шрифтом):

```
package com.hfad.starbuzz;

import android.content.ContentValues;
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

class StarbuzzDatabaseHelper extends SQLiteOpenHelper{

    private static final String DB_NAME = "starbuzz"; // Имя базы данных
    private static final int DB_VERSION = 2; // Версия базы данных

    StarbuzzDatabaseHelper(Context context){
        super(context, DB_NAME, null, DB_VERSION);
    }

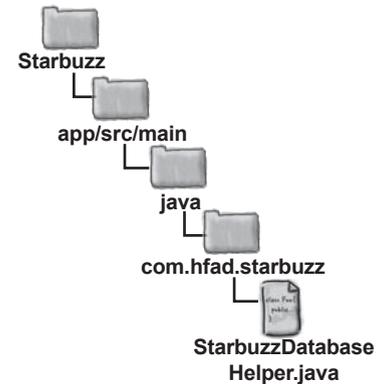
    @Override
    public void onCreate(SQLiteDatabase db){
        updateMyDatabase(db, 0, DB_VERSION);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        updateMyDatabase(db, oldVersion, newVersion);
    }
}
```

↑ Увеличение номера версии сообщает помощнику SQLite, что базу данных следует обновить.

← Код из метода onCreate() выносится в метод updateMyDatabase().

↑ Код обновления базы данных содержится в методе updateMyDatabase().



Продолжение
на следующей странице. →



Код помощника SQLite (продолжение)

```

private void updateMyDatabase(SQLiteDatabase db, int oldVersion, int newVersion) {
    if (oldVersion < 1) {
        db.execSQL("CREATE TABLE DRINK (_id INTEGER PRIMARY KEY AUTOINCREMENT, "
            + "NAME TEXT, "
            + "DESCRIPTION TEXT, "
            + "IMAGE_RESOURCE_ID INTEGER);");
        insertDrink(db, "Latte", "Espresso and steamed milk", R.drawable.latte);
        insertDrink(db, "Cappuccino", "Espresso, hot milk and steamed-milk foam",
            R.drawable.cappuccino);
        insertDrink(db, "Filter", "Our best drip coffee", R.drawable.filter);
    }
    if (oldVersion < 2) {
        db.execSQL("ALTER TABLE DRINK ADD COLUMN FAVORITE NUMERIC;");
    }
}

private static void insertDrink(SQLiteDatabase db, String name,
    String description, int resourceId) {
    ContentValues drinkValues = new ContentValues();
    drinkValues.put("NAME", name);
    drinkValues.put("DESCRIPTION", description);
    drinkValues.put("IMAGE_RESOURCE_ID", resourceId);
    db.insert("DRINK", null, drinkValues);
}
}

```

↑
Добавить числовой столбец
FAVORITE в таблицу DRINK.

Новый код помощника SQLite означает, что при следующем обращении в таблице DRINK базы данных существующих пользователей появится столбец FAVORITE. На устройствах новых пользователей будет полностью создана база данных вместе с новым столбцом. На следующей странице показано, что происходит при выполнении этого кода.

Что происходит при выполнении кода

- 1 При первом обращении к базе данных помощник SQLite проверяет, существует ли база данных.



Помощник SQLite

- 2 Если база данных не существует, помощник SQLite создает ее и выполняет свой метод onCreate().

Наш метод onCreate() вызывает метод updateMyDatabase(). Он создает таблицу DRINK (включая дополнительный столбец) и заполняет таблицу записями.



Помощник SQLite

- 3 Если база данных уже существует, помощник SQLite сравнивает номер версии базы данных с номером версии в коде помощника.

Если номер версии помощника SQLite выше номера версии базы данных, помощник вызывает метод onUpgrade(). Если номер версии помощника SQLite ниже, вызывается метод onDowngrade(). В нашем примере номер версии помощника выше номера версии базы данных, поэтому вызывается метод onUpgrade(). Он вызывает метод updateMyDatabase(), который добавляет в таблицу DRINK столбец с именем FAVORITE.



Помощник SQLite



Ваш инструментарий Android

Глава 11 осталась позади, а ваш инструментарий пополнился навыками создания и обновления баз данных.

Весь код для этой главы можно загрузить по адресу <https://tinyurl.com/HeadFirstAndroid>.

КЛЮЧЕВЫЕ МОМЕНТЫ



- Android использует SQLite для организации хранения данных.
- Класс `SQLiteDatabase` предоставляет доступ к базе данных SQLite.
- Помощник SQLite позволяет создавать базы данных SQLite и управлять ими. Помощник SQLite создается расширением класса `SQLiteOpenHelper`.
- Вы должны реализовать методы `SQLiteOpenHelper onCreate()` и `onUpgrade()`.
- База данных создается при первом обращении к ней. Базе данных необходимо присвоить имя и номер версии, начиная с 1. Если не присвоить базе данных имя, она создается в памяти.
- Метод `onCreate()` вызывается при создании базы данных.
- Метод `onUpgrade()` вызывается при обновлении базы данных.
- Для выполнения команд SQL используется метод `execSQL(String)` класса `SQLiteDatabase`.
- Добавление записей в таблицы производится методом `insert()`.
- Обновление записей производится методом `update()`.
- Удаление записей из таблиц производится методом `delete()`.

Подключение к базам данных

Как хорошо, что есть метод `doInBackground()`. Если бы Мистер Главный Поток делал все по порядку, представляете, как медленно все происходило бы?



Как же подключиться из приложения к базе данных SQLite?

В предыдущей главе было показано, как создать базу данных SQLite с использованием помощника SQLite. Пора сделать следующий шаг — узнать, как работать с базой данных из активностей. В этой главе вы узнаете, как **использовать курсоры для получения информации из базы данных**, как **перемещаться по набору данных с использованием курсора** и как **получить данные из курсора**. Затем вы узнаете, как использовать **адаптеры курсоров** для их связывания со списковыми представлениями. В завершение мы покажем, как написание эффективного **многопоточного кода** с объектами **AsyncTask** ускоряет работу приложений.

что дальше?

Чего мы добились...

В главе 11 вы узнали, как написать помощника SQLite для создания базы данных, как добавить в эту базу данных таблицы и заполнить их данными. Также было показано, как управлять обновлениями базы данных на уровне помощника SQLite, чтобы вы могли модифицировать структуру базы данных и изменять содержащиеся в ней данные.

В этой главе мы покажем, как обеспечить взаимодействие активностей с базой данных, чтобы пользователи могли читать и записывать информацию в базу данных из вашего приложения.

В своем текущем состоянии приложение Starbuzz выглядит так:

Мы создали помощника SQLite и добавили код создания базы данных Starbuzz. Пока база данных не используется активностями приложения.

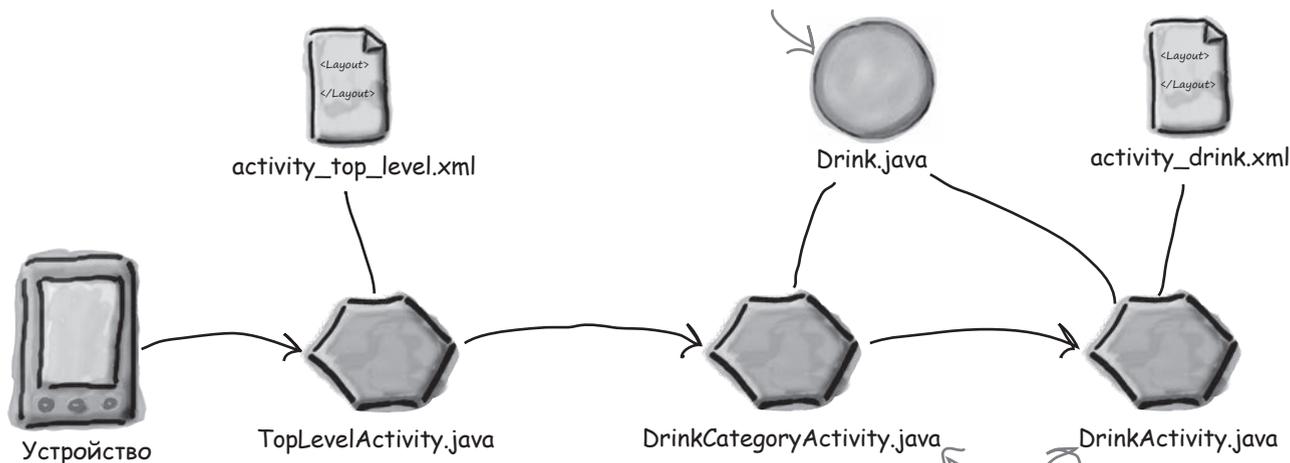


База данных Starbuzz



Помощник SQLite

Как и прежде, в приложении используется класс Drink.



DrinkActivity и DrinkCategoryActivity все еще работают с Drink.java.

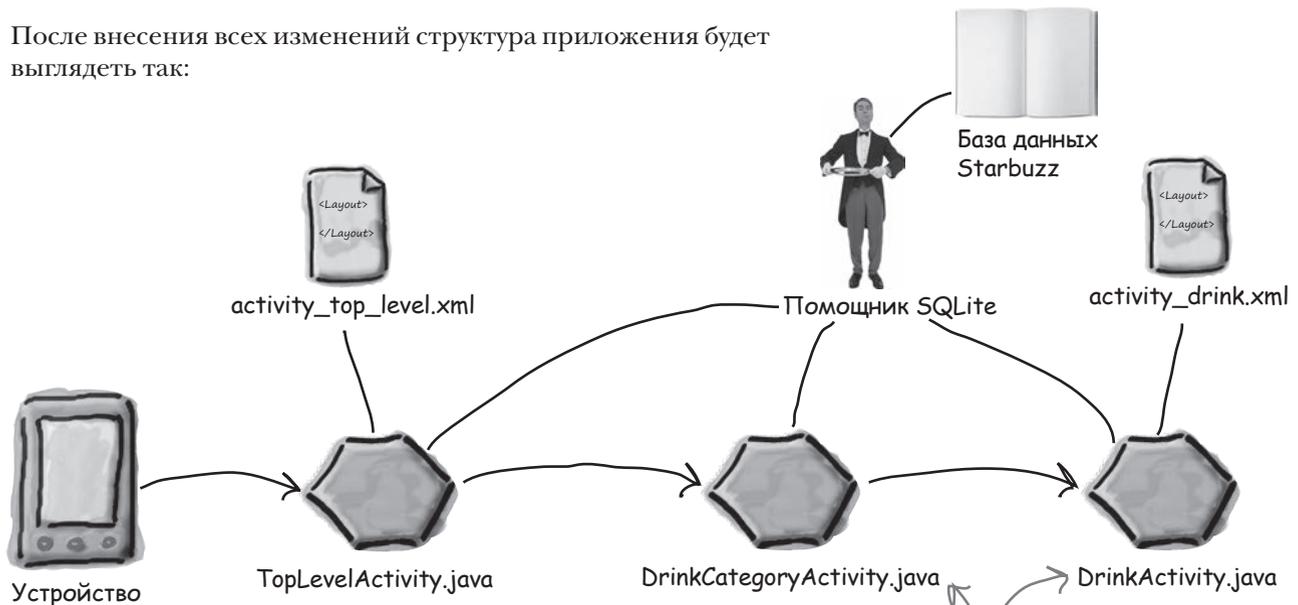
Мы изменим приложение Starbuzz, чтобы вместо класса Java Drink в нем использовалась база данных SQLite Starbuzz.

Переход на работу с базой данных

Класс `Drink` используется двумя активностями. Мы хотим, чтобы эти классы читали данные из базы данных SQLite при содействии помощника SQLite. Вот что для этого нужно сделать:

- 1 **Обновление кода `Drink` в `DrinkActivity`.**
`DrinkActivity` использует класс `Drink` для отображения подробной информации о конкретном напитке. Мы изменим активность так, чтобы информация о напитке загружалась из базы данных Starbuzz.
- 2 **Обновление кода `Drink` в `DrinkCategoryActivity`.**
`DrinkCategoryActivity` использует класс `Drink` для вывода списка всех напитков. Мы изменим ее так, чтобы она выводила список всех записей из таблицы DRINK.
- 3 **Выбор любимых напитков пользователями.**
 В главе 11 мы изменили структуру базы данных и добавили в таблицу DRINK столбец FAVORITE. Мы изменим приложение, чтобы пользователи могли пометить свои любимые напитки; перечень любимых напитков будет отображаться в `TopLevelActivity`.

После внесения всех изменений структура приложения будет выглядеть так:



Начнем с `DrinkActivity.java`.

Мы изменим активности, работающие с классом `Drink`, так, чтобы они вместо класса использовали базу данных.

Текущий код DrinkActivity

Вспомним, как выглядит текущая версия кода *DrinkActivity.java*. Метод `onCreate()` получает номер напитка, выбранного пользователем, получает подробную информацию о нем из класса *Drink*, после чего заполняет представления активности, используя атрибуты напитка:

```
package com.hfad.starbuzz;

import android.app.Activity;
import android.os.Bundle;
import android.widget.ImageView;
import android.widget.TextView;

public class DrinkActivity extends Activity {

    public static final String EXTRA_DRINKNO = "drinkNo";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_drink);

        //Получение напитка из интента
        int drinkNo = (Integer) getIntent().getExtras().get(EXTRA_DRINKNO);
        Drink drink = Drink.drinks[drinkNo];

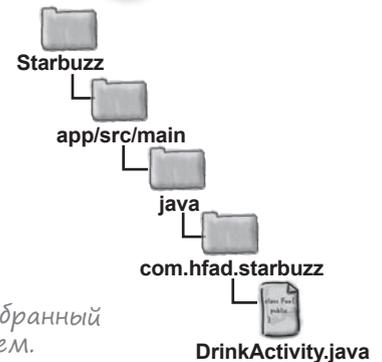
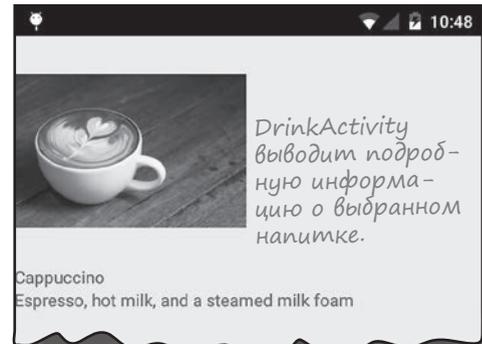
        //Заполнение изображения напитка
        ImageView photo = (ImageView) findViewById(R.id.photo);
        photo.setImageResource(drink.getImageResourceId());
        photo.setContentDescription(drink.getName());

        //Заполнение названия напитка
        TextView name = (TextView) findViewById(R.id.name);
        name.setText(drink.getName());

        //Заполнение описания напитка
        TextView description = (TextView) findViewById(R.id.description);
        description.setText(drink.getDescription());
    }
}
```



DrinkActivity
DrinkCategoryActivity
Любимые напитки



Напиток, выбранный пользователем.

Использовать номер напитка из интента для чтения информации из класса *Drink*. Эту часть нужно будет изменить, чтобы информация загружалась из базы данных.

Значения, которыми заполняются представления в макете, должны загружаться из базы данных, а не из класса *Drink*.

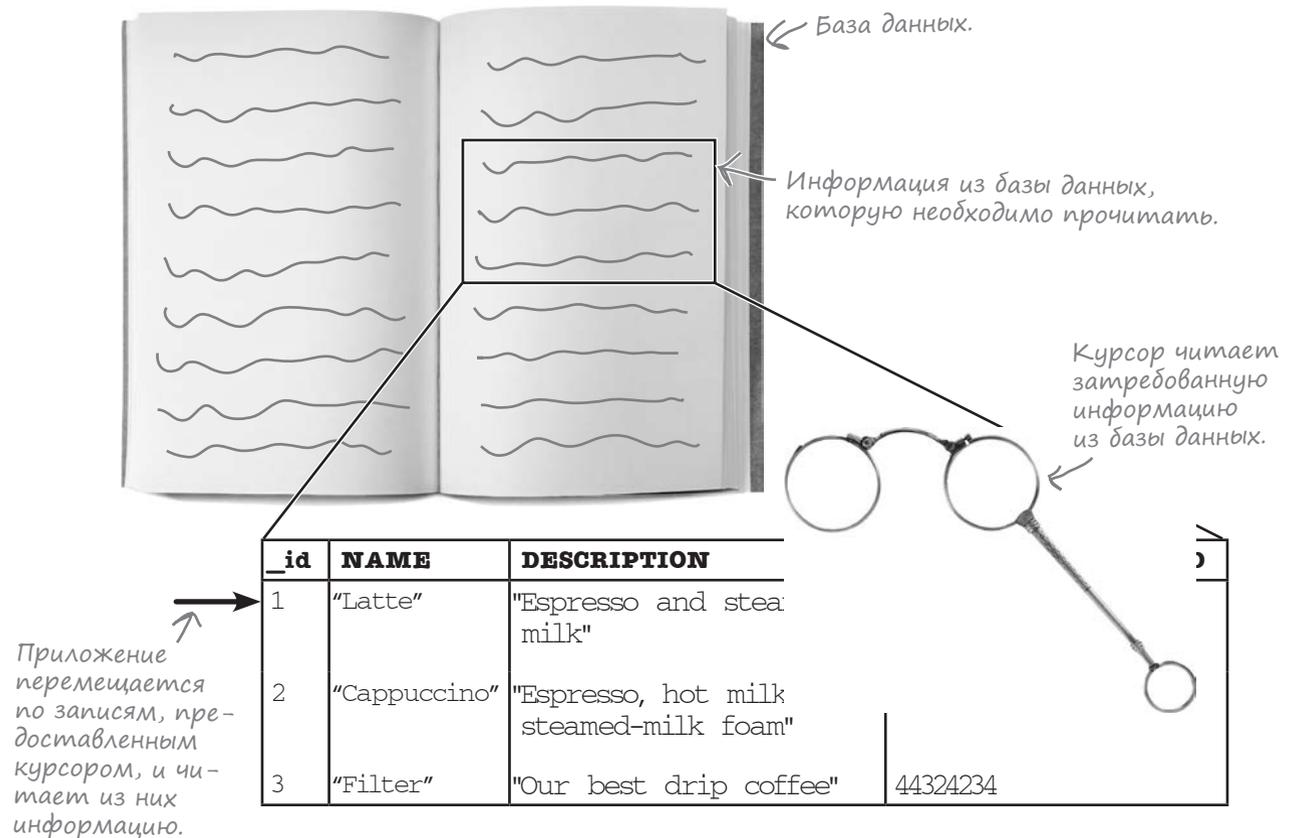
Получение данных через курсор

Наш текущий код `DrinkActivity` получает информацию о конкретном напитке из класса `Drink`. Как изменить его, чтобы информация о напитке загружалась из базы данных Starbuzz? *Как изменить активность, чтобы она читала информацию из базы данных?*

Задача решается при помощи **курсора**.

Для работы с информацией из базы данных используются курсоры

Курсор предоставляет доступ к наборам записей базы данных. Вы указываете, какие данные вам нужны, а курсор возвращает записи из базы данных. Затем вы перебираете записи, предоставленные курсором.



Чтобы создать курсор, вы описываете интересующие вас данные в виде запроса к базе данных. Что же такое «запрос»?



DrinkActivity
 DrinkCategoryActivity
 Любимые напитки

Запрос определяет, какие записи должны быть прочитаны из базы данных

Запрос к базе данных точно определяет набор интересующих вас записей из базы данных. Например, вы можете указать, что вам нужны все данные из таблицы DRINK или только те напитки, названия которых начинаются с буквы “L”. Чем жестче ограничиваются возвращаемые данные, тем эффективнее будет запрос.

Определение таблицы и столбцов

Первое, что необходимо указать в запросе, — из какой таблицы следует получить записи и какие столбцы вам нужны.

Вернуть данные из столбцов NAME и DESCRIPTION таблицы DRINK.

_id	NAME	DESCRIPTION	IMAGE_RESOURCE_ID	FAVORITE
1	"Latte"	"Espresso and steamed milk"	54543543	0
2	"Cappuccino"	"Espresso, hot milk and steamed-milk foam"	654334453	0
3	"Filter"	"Our best drip coffee"	44324234	0

Объявление любых условий, ограничивающих выборку

После указания интересующих вас столбцов можно отфильтровать результаты выборки; для этого необходимо объявить условия, которым должны соответствовать данные. Например, в нашем приложении требуется получить напиток, выбранный пользователем; для этого можно ограничить выборку записями, у которых столбец _id содержит конкретное значение.

Вернуть данные, для которых _id содержит 1.

_id	NAME	DESCRIPTION	IMAGE_RESOURCE_ID	FAVORITE
1	"Latte"	"Espresso and steamed milk"	54543543	0
2	"Cappuccino"	"Espresso, hot milk and steamed-milk foam"	654334453	0
3	"Filter"	"Our best drip coffee"	44324234	0

Другие применения запросов

Если запрос должен вернуть несколько строк данных, возможно, вы захотите указать порядок, в котором должны следовать записи. Например, записи напитков могут быть упорядочены по названию напитков. Кроме того, запросы могут использоваться для группировки данных по некоторому критерию и применения агрегатных функций. Например, вы можете получить количество записей напитков и вывести его в своем приложении.

Как же создаются запросы?

Метод query() класса SQLiteDatabase предоставляет средства для построения запросов SQL

Для построения запроса можно воспользоваться методом query() класса SQLiteDatabase. Метод query() возвращает объект типа Cursor, который может использоваться вашими активностями для обращения к базе данных.

Базовая форма метода query() выглядит так:

```
public Cursor query(String table,
                   String[] columns,
                   String selection,
                   String[] selectionArgs,
                   String groupBy,
                   String having,
                   String orderBy)
```

Метод query() возвращает объект Cursor.

Таблица и столбцы, из которых производится выборка.

Эти параметры используются для применения условий.

Эти параметры используются для группировки данных.

Данные должны следовать в каком-то определенном порядке?

При использовании этой версии метода query() вы можете указать, из какой таблицы должны возвращаться данные, какие столбцы вам нужны, какие условия должны применяться к данным, как должна производиться группировка данных и как должны упорядочиваться результаты. Существуют еще несколько перегруженных версий метода query(), которые позволяют включить в запрос дополнительные данные — например, потребовать, чтобы результаты содержали только уникальные строки, или задать максимальное количество возвращаемых строк. Мы не будем подробно рассматривать все эти версии; если вас заинтересует эта тема, полный список перегруженных методов можно найти в электронной документации Android: <http://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html>

На нескольких следующих страницах представлены типичные примеры использования метода query().

Во внутренней реализации Android использует метод query() для построения команды SQL SELECT.



DrinkActivity
 DrinkCategoryActivity
 Любимые напитки

Определение таблицы и столбцов

Простейший запрос базы данных возвращает значения заданных столбцов всех записей без указания критерия. Для построения такого запроса имя таблицы передается в первом параметре, а массив имен столбцов — во втором. Например, вызов метода `query()` для получения содержимого столбцов `NAME` и `DESCRIPTION` из таблицы `DRINK` выглядит так:

```
Cursor cursor = db.query("DRINK",
    new String[] {"NAME", "DESCRIPTION"},
    null, null, null, null, null);
```

Каждый столбец, который должен быть возвращен запросом, представляется отдельным элементом в массиве строк.

В запросе используются только первые два параметра, поэтому остальные равны `null`.

Запрос возвращает все данные из столбцов `NAME` и `DESCRIPTION` таблицы `DRINK`.

NAME	DESCRIPTION
"Latte"	"Espresso and steamed milk"
"Cappuccino"	"Espresso, hot milk and steamed-milk foam"
"Filter"	"Our best drip coffee"

Ограничение выборки применением условий

Третий и четвертый параметры позволяют применить к запросу условия и указать, какие значения должны содержать интересующие вас столбцы. Например, следующий запрос возвращает только те записи из таблицы `DRINK`, у которых столбец `NAME` содержит значение "Latte":

```
Cursor cursor = db.query("DRINK",
    new String[] {"NAME", "DESCRIPTION"},
    "NAME = ?",
    new String[] {"Latte"},
    null, null, null);
```

← Это означает "у которых столбец NAME содержит 'Latte'".

Третий параметр `"NAME = ?"` означает, что столбец `NAME` должен быть равен некоторому значению. Символ `?` представляет это значение в строке запроса, а само значение представлено содержимым четвертого параметра (в данном примере "Latte").

Запрос возвращает все данные столбцов `NAME` и `DESCRIPTION` таблицы `DRINK` тех записей, у которых столбец `NAME` содержит значение "Latte".

NAME	DESCRIPTION
"Latte"	"Espresso and steamed milk"

Сложные условия

Если вы хотите применить несколько условий в своем запросе, проследите за тем, чтобы порядок перечисления условий соответствовал порядку значений. Например, для получения записей таблицы DRINK, у которых столбец NAME содержит строку "Latte" или столбец DESCRIPTION содержит строку "Our best drip coffee", используется следующий запрос:

```
Cursor cursor = db.query("DRINK",
    new String[] {"NAME", "DESCRIPTION"},
    "NAME = ? OR DESCRIPTION = ?",
    new String[] {"Latte", "Our best drip coffee"},
    null, null, null);
```

Это означает "у которых столбец NAME содержит 'Latte' или столбец DESCRIPTION содержит 'Our best drip coffee'".

Запрос возвращает все данные из столбцов NAME и DESCRIPTION таблицы DRINK, у которых столбец NAME содержит "Latte" или столбец DESCRIPTION содержит "Our best drip coffee".

NAME	DESCRIPTION
"Latte"	"Espresso and steamed milk"
"Filter"	"Our best drip coffee"

Если порядок перечисления условий не соответствует порядку значений, курсор вернет ошибочные данные. Например, если значение "Latte" окажется в паре со столбцом DESCRIPTION вместо столбца NAME, запрос не вернет ни одной записи.

Условия задаются в виде строковых значений

Условия должны быть строками. Если столбец, с которым связывается условие, содержит не текстовые, а другие данные, значения все равно должны быть преобразованы к типу String. Например, следующий запрос возвращает записи DRINK, у которых столбец _id содержит 1:

```
Cursor cursor = db.query("Drink",
    new String[] {"NAME", "DESCRIPTION"},
    "_id = ?",
    new String[] {Integer.toString(1)},
    null, null, null);
```

Целое число 1 преобразуется в строковое значение.

Запрос возвращает все данные из столбцов NAME и DESCRIPTION таблицы DRINK, у которых столбец _id содержит 1.

_id	NAME	DESCRIPTION
1	"Latte"	"Espresso and steamed milk"



Упорядочение данных в запросах

Если вы хотите, чтобы данные выводились в определенном порядке, воспользуйтесь запросом для сортировки данных по нужному столбцу. Например, эта возможность может использоваться для вывода списка названий напитков в алфавитном порядке. По умолчанию данные в таблице упорядочены по значениям `_id`, так как данные вводились именно в этом порядке:

<code>_id</code>	<code>NAME</code>	<code>DESCRIPTION</code>	<code>IMAGE_RESOURCE_ID</code>	<code>FAVORITE</code>
1	"Latte"	"Espresso and steamed milk"	54543543	1
2	"Cappuccino"	"Espresso, hot milk and steamed-milk foam"	654334453	0
3	"Filter"	"Our best drip coffee"	44324234	0

Для получения данных из столбцов `NAME` и `FAVORITE`, упорядоченных по возрастанию `NAME`, используется следующий запрос:

```
Cursor cursor = db.query("DRINK",
    new String[] {"_id", "NAME", "FAVORITE"},
    null, null, null, null,
    "NAME ASC");
```

← Упорядочение по возрастанию NAME.

<code>NAME</code>	<code>FAVORITE</code>
"Cappuccino"	0
"Filter"	0
"Latte"	1

Ключевое слово `ASC` означает, что данные должны упорядочиваться по возрастанию значений столбца. По умолчанию столбцы упорядочиваются по возрастанию, поэтому ключевое слово `ASC` при желании можно опустить. Чтобы данные упорядочивались по убыванию, используйте ключевое слово `DESC`.

Сортировка также может производиться по значениям нескольких столбцов. Например, следующая команда упорядочивает данные по убыванию значений `FAVORITE`, а вторичная сортировка выполняется по возрастанию значений `NAME`:

```
Cursor cursor = db.query("DRINK",
    new String[] {"_id", "NAME", "FAVORITE"},
    null, null, null, null,
    "FAVORITE DESC, NAME");
```

← Упорядочение по убыванию FAVORITE, затем по возрастанию NAME.

<code>NAME</code>	<code>FAVORITE</code>
"Latte"	1
"Cappuccino"	0
"Filter"	0

Мы рассмотрели наиболее типичные варианты использования метода `query()`, однако существуют и другие возможности.

Функции SQL в запросах

Хорошая новость для тех, кто знаком с функциями SQL: вы можете использовать эти функции в запросах. Такие функции используются для вычислений таких характеристик, как количество строк данных в таблице, среднее значение или максимальное значение столбца. Некоторые полезные функции SQL, которые могут использоваться в запросах:

AVG()	Среднее значение
COUNT()	Количество строк
SUM()	Сумма
MAX()	Наибольшее значение
MIN()	Наименьшее значение

Скажем, если вы хотите подсчитать, сколько напитков хранится в таблице DRINK, воспользуйтесь функцией SQL COUNT () для подсчета значений в столбце `_id`:

```
Cursor cursor = db.query("DRINK",
    new String[] {"COUNT(_id) AS count"},
    null, null, null, null, null);
```

Возвращает количество напитков в столбце с именем "count".

Запрос возвращает количество строк в таблице DRINK.

count
3

Если бы таблица DRINK содержала дополнительный столбец PRICE с ценой каждого напитка, то среднюю цену напитка можно было бы вычислить применением функции SQL AVG () для вычисления среднего значения столбца PRICE:

```
Cursor cursor = db.query("DRINK",
    new String[] {"AVG(PRICE) AS price"},
    null, null, null, null, null);
```

price
4.17

В таблице DRINK нет столбца PRICE, но если бы он был — его можно было бы использовать для вычисления средней цены.



DrinkActivity
DrinkCategoryActivity
Любимые напитки

Конструкции SQL GROUP BY и HAVING

Если вы знакомы с конструкциями SQL GROUP BY и HAVING, используйте их в пятом и шестом параметрах метода query().

Предположим, вы хотите узнать, сколько напитков хранится в таблице для каждого значения FAVORITE. Для этого создается запрос, который возвращает значения столбца FAVORITE и количество напитков. Группировка по столбцу FAVORITE используется для получения количества напитков для каждого значения FAVORITE:



Эта книга — не учебник языка SQL. Мы всего лишь даем представление о том, что с ним можно сделать.

Если вы решите, что некоторая возможность окажется полезной в вашей работе, вам стоит обратиться к учебнику — например, *Head First SQL*.

```
Cursor cursor = db.query("DRINK",
                        new String[] {"FAVORITE", "COUNT(_id) AS count"},
                        null, null,
                        "FAVORITE",
                        null, null);
```

Группировать по столбцу FAVORITE. →

↑ ↑
Вернуть столбец FAVORITE и количество напитков.

Если таблица DRINKS содержит следующие данные:

_id	NAME	DESCRIPTION	IMAGE_RESOURCE_ID	FAVORITE
1	"Latte"	"Espresso and steamed milk"	54543543	1
2	"Cappuccino"	"Espresso, hot milk and steamed-milk foam"	654334453	0
3	"Filter"	"Our best drip coffee"	44324234	0

запрос вернет следующий результат:

FAVORITE	count
1	1
0	2

В таблице есть одна запись, у которой столбец FAVORITE содержит значение 1, и две записи, у которых столбец FAVORITE содержит значение 0.

Итак, теперь вы знаете, как создать курсор методом query(). Применим знания на практике и создадим курсор для приложения Starbuzz.



Развлечения с Магнитами. Решение



DrinkActivity
DrinkCategoryActivity
Favorites

Наш код DrinkActivity должен загрузить из базы данных название, описание и идентификатор ресурса изображения для напитка, переданного в интенде. Удастся ли вам построить вызов метода `query()`, который все это делает?

...

```
int drinkNo = (Integer) getIntent().getExtras().get(EXTRA_DRINKNO);

Cursor cursor = db.query(
    "DRINK",
    new String[] { "NAME", "DESCRIPTION", "IMAGE_RESOURCE_ID" },
    "_id = ?",
    new String[] { Integer.toString(drinkNo) },
    null, null, null);
```

*Выдать запрос к таблице DRINK.
Получить столбцы NAME, DESCRIPTION и IMAGE_RESOURCE_ID.*

Для которых значение id равно drinkNo.

drinkNo — целое число, которое необходимо преобразовать в String.

id

...

Получение ссылки на базу данных

На нескольких последних страницах вы узнали, как построить запрос, который возвращает курсор. Метод `query()` определяется в классе `SQLiteDatabase`; это значит, что для его вызова необходимо сначала получить ссылку на базу данных Starbuzz. Класс `SQLiteOpenHelper` реализует пару методов, которые помогут нам в этом: речь идет о методах `getReadableDatabase()` и `getWritableDatabase()`. Каждый из этих методов возвращает объект типа `SQLiteDatabase`, который предоставляет доступ к базе данных. Методы вызываются так:

```
SQLiteOpenHelper starbuzzDatabaseHelper = new StarbuzzDatabaseHelper(this);
SQLiteDatabase db = starbuzzDatabaseHelper.getReadableDatabase();
```

и

```
SQLiteOpenHelper starbuzzDatabaseHelper = new StarbuzzDatabaseHelper(this);
SQLiteDatabase db = starbuzzDatabaseHelper.getWritableDatabase();
```

Чем же отличаются два метода?

Курсоры предоставляют доступ к информации из базы данных.

getReadableDatabase() и getWritableDatabase()

Вероятно, вы подумали, что `getReadableDatabase()` возвращает объект базы данных, доступной только для чтения, а `getWritableDatabase()` возвращает объект базы данных, доступной только для записи? В большинстве случаев `getReadableDatabase()` и `getWritableDatabase()` *возвращают ссылку на один объект базы данных*. Объект базы данных может использоваться как для чтения, так и для записи данных в базу. Для чего тогда нужен метод `getReadableDatabase()`, если он возвращает тот же объект, что и `getWritableDatabase()`?

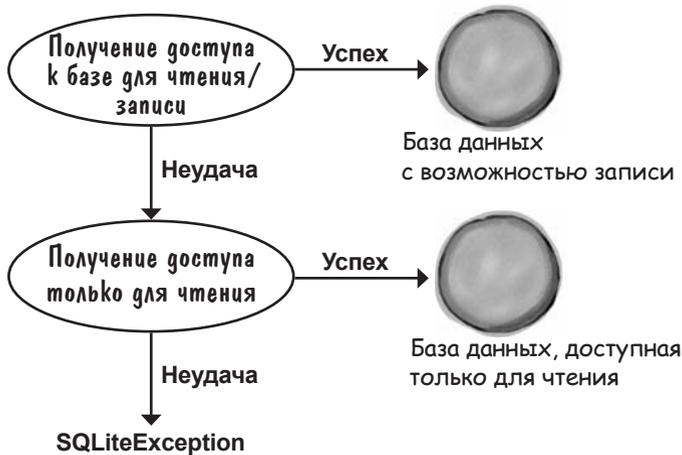
Основное различие между `getReadableDatabase()` и `getWritableDatabase()` проявляется при невозможности записи в базу данных (например, из-за отсутствия свободного места на диске).

При использовании метода `getWritableDatabase()` в этом случае при вызове метода происходит сбой с выдачей исключения `SQLiteException`. Но если использовать метод `getReadableDatabase()`, метод попытается получить ссылку на базу данных, предназначенную только для чтения. Если получить доступ к базе данных только для чтения не удастся, метод все равно выдает исключение `SQLiteException`.

Если вы собираетесь только читать информацию из базы данных, лучше использовать метод `getReadableDatabase()`. Если же существует необходимость в записи в базу данных, используйте метод `getWritableDatabase()`.

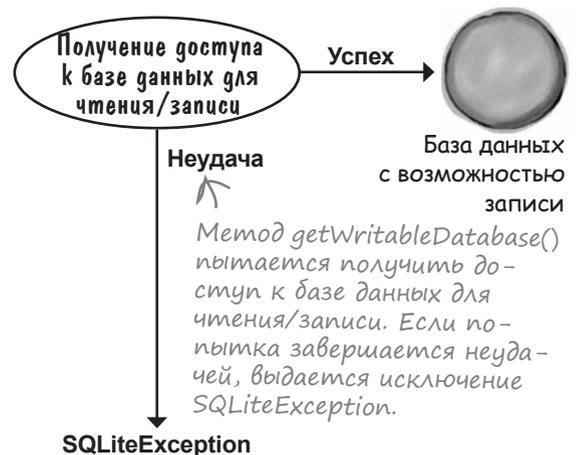
← Скорее всего, при использовании `getReadableDatabase()` вы также сможете записывать данные в базу, но это не гарантировано.

getReadableDatabase()



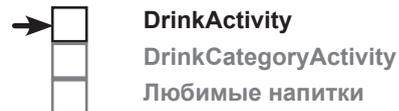
Метод `getReadableDatabase()` сначала пытается получить доступ к базе данных для чтения/записи. Если попытка завершается неудачей, метод пытается получить доступ только для чтения. Если и это невозможно, выдается исключение `SQLiteException`.

getWritableDatabase()



← Метод `getWritableDatabase()` пытается получить доступ к базе данных для чтения/записи. Если попытка завершается неудачей, выдается исключение `SQLiteException`.

Код получения курсора



Объединяя все сказанное, приходим к коду получения курсора. Позднее этот код будет использоваться в методе onCreate () нашей активности.

Так как мы не собираемся записывать данные в базу, используем getReadableDatabase().

```
try {
    SQLiteOpenHelper starbuzzDatabaseHelper = new StarbuzzDatabaseHelper(this);
    SQLiteDatabase db = starbuzzDatabaseHelper.getReadableDatabase();
    Cursor cursor = db.query("DRINK",
        new String[] {"NAME", "DESCRIPTION", "IMAGE_RESOURCE_ID"},
        "_id = ?",
        new String[] {Integer.toString(drinkNo)},
        null, null, null);

    //Код работы с курсором

} catch (SQLException e) {
    Toast toast = Toast.makeText(this, "Database unavailable", Toast.LENGTH_SHORT);
    toast.show();
}
```

Курсор мы получили, но теперь с ним нужно что-то сделать.

Курсор содержит одну запись, так как столбец _id содержит уникальные значения.

Если база данных недоступна, вывести сообщение.

Что делает код

- 1 Сначала создается объект starbuzzDatabaseHelper.



starbuzzDatabaseHelper

- 2 starbuzzDatabaseHelper создает объект SQLiteDatabase с именем db.



starbuzzDatabaseHelper



db

- 2 Вызов метода query() класса SQLiteDatabase создает курсор.



starbuzzDatabaseHelper



db

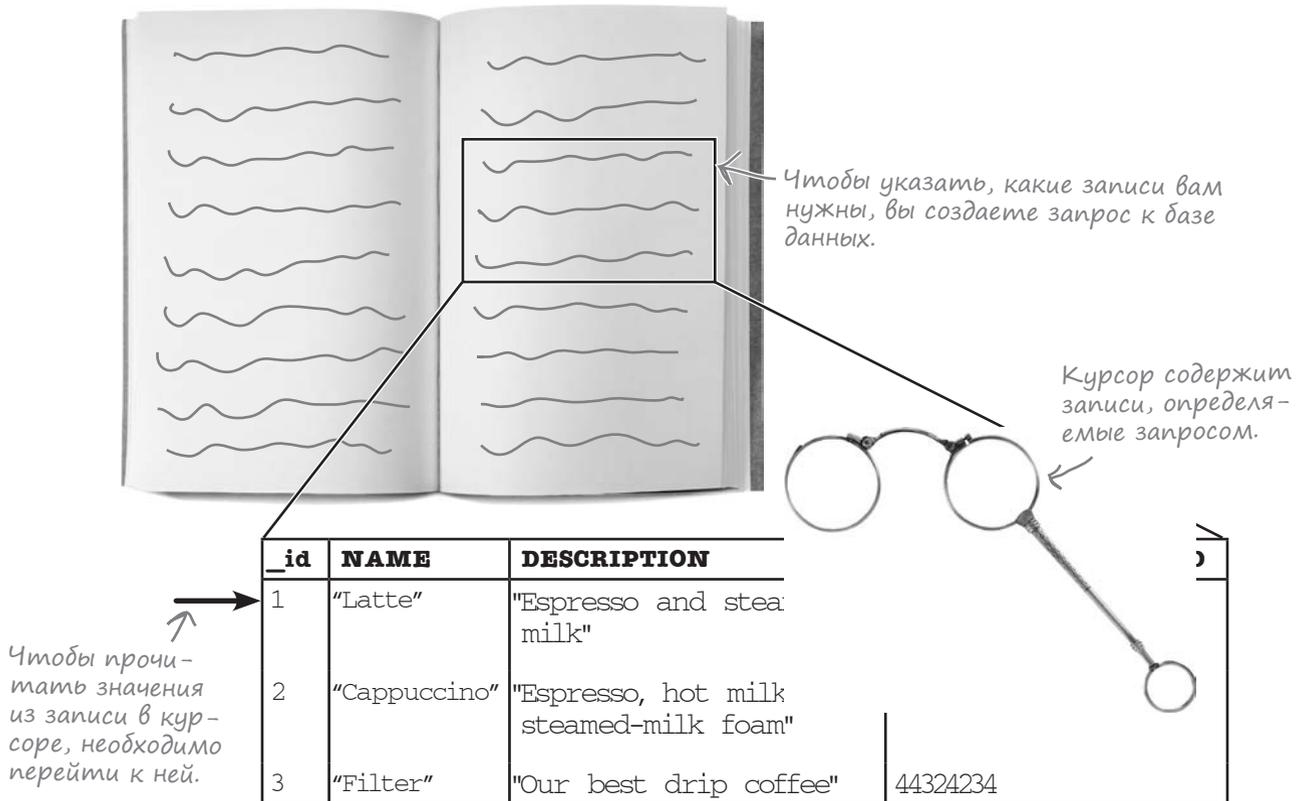
query(...)



курсор

Чтобы прочитать запись из курсора, сначала необходимо перейти к ней

Вы уже знаете, как создавать курсоры; нужно вызвать метод `query()` класса `SQLiteDatabase` для определения данных, которые должен вернуть курсор. Впрочем, это еще не все — из курсора необходимо прочитать данные. Каждый раз, когда вам потребуется прочитать данные из конкретной записи в курсоре, вы сначала должны перейти к этой записи. Это необходимо делать всегда, сколько бы записей ни вернул курсор.



На следующей странице вы узнаете, как происходят перемещения между записями в курсоре.

Переходы между записями

Для перемещения между записями в курсорах используются четыре основных метода: `moveToFirst()`, `moveToLast()`, `moveToPrevious()` и `moveToNext()`.

Чтобы перейти к первой записи набора, возвращенного курсором, используйте метод `moveToFirst()` (метод возвращает `true`, если запись успешно найдена, и `false`, если курсор не вернул ни одной записи):

```
if (cursor.moveToFirst()) {
    //...
};
```

Для перехода к последней записи, возвращенной курсором, используется метод `moveToLast()` (как и `moveToFirst()`, он возвращает `true`, если запись успешно найдена, и `false` в противном случае):

```
if (cursor.moveToLast()) {
    //...
};
```

Для последовательного перебора записей курсора используются методы `moveToPrevious()` и `moveToNext()`.

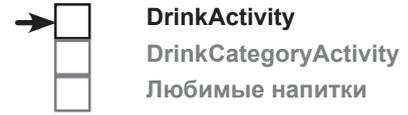
Метод `moveToPrevious()` переходит к предыдущей записи в курсоре (если переход к предыдущей записи выполнен успешно, метод возвращает `true`, а в случае неудачи возвращается `false` — например, если курсор находится на первой записи набора или набор не содержит ни одной записи):

```
if (cursor.moveToPrevious()) {
    //...
};
```

Метод `moveToNext()` аналогичен `moveToPrevious()`, не считая того, что он переходит к следующей записи в курсоре (метод возвращает `true`, если переход выполнен успешно, или `false` в случае неудачи):

```
if (cursor.moveToNext()) {
    //...
};
```

После перехода к нужной записи в курсоре можно прочитать из нее значения. Эта тема рассматривается на следующей странице.



Перейти к первой записи.

NAME	DESCRIPTION
"Latte"	"Espresso and steamed milk"
Cappuccino	"Espresso, hot milk and steamed-milk foam"
Filter	"Our best drip coffee"

NAME	DESCRIPTION
"Latte"	"Espresso and steamed milk"
Cappuccino	"Espresso, hot milk and steamed-milk foam"
Filter	"Our best drip coffee"

Перейти к последней записи.

NAME	DESCRIPTION
"Latte"	"Espresso and steamed milk"
Cappuccino	"Espresso, hot milk and steamed-milk foam"
Filter	"Our best drip coffee"

Перейти к предыдущей записи.

NAME	DESCRIPTION
"Latte"	"Espresso and steamed milk"
Cappuccino	"Espresso, hot milk and steamed-milk foam"
Filter	"Our best drip coffee"

Перейти к следующей записи.

Чтение данных из курсора

После перехода к нужной записи в курсоре из нее читаются значения, которые можно вывести в представлениях активности. Для чтения данных из текущей записи курсора используются методы `get*()`. Точное название метода зависит от типа значения, которое вы собираетесь прочитать. Например, метод `getString()` возвращает значение столбца в формате `String`, а метод `getInt()` возвращает значение столбца в формате `int`. Каждый из методов получает один параметр — индекс столбца.

Вспомните, как выглядел запрос для создания курсора в нашем примере:

```
Cursor cursor = db.query ("Drink",
    new String[] {"NAME", "DESCRIPTION", "IMAGE_RESOURCE_ID"},
    "_id = ?",
    new String[] {Integer.toString(1)},
    null, null, null);
```

Курсор состоит из трех столбцов: `NAME`, `DESCRIPTION` и `IMAGE_RESOURCE_ID`. Первые два столбца, `NAME` и `DESCRIPTION`, содержат данные типа `String`. Третий столбец, `IMAGE_RESOURCE_ID`, содержит данные типа `int`.

Предположим, вы хотите узнать значение столбца `NAME` текущей записи. `NAME` — первый столбец в курсоре — содержит строковые значения. Следовательно, для получения содержимого столбца `NAME` будет использоваться вызов `getString()` следующего вида:

```
String name = cursor.getString(0);
```

← Первый столбец в курсоре.

Другой пример: предположим, вы хотите получить содержимое столбца `IMAGE_RESOURCE_ID`. Это третий столбец в курсоре, он содержит значения `int`, поэтому для получения данных будет использоваться следующий вызов:

```
int imageResource = cursor.getInt(2);
```

Последний шаг: закрытие курсора и базы данных

После того как данные будут прочитаны из курсора, следует закрыть курсор и базу данных, чтобы освободить их ресурсы. Для этого следует вызвать методы `close()` объектов курсора и базы данных:

```
cursor.close();
db.close();
```

Итак, мы рассмотрели все изменения, которые необходимо внести в `DrinkActivity` для получения информации из базы данных `Starbuzz`. Теперь можно переходить к коду реализации.

Столбец 0	Столбец 1	Столбец 2
NAME	DESCRIPTION	IMAGE_RESOURCE_ID
"Latte"	"Espresso and steamed milk"	54543543

Подробные описания get-методов курсоров доступны по адресу <http://developer.android.com/reference/android/database/Cursor.html>.

Код DrinkActivity

Ниже приведен полный код *DrinkActivity.java* (внесите изменения, выделенные жирным шрифтом, затем сохраните результат):

```
package com.hfad.starbuzz;

import android.app.Activity;
import android.os.Bundle;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteException;
import android.database.sqlite.SQLiteOpenHelper;
```

Дополнительные
классы, используемые
в коде.

```
public class DrinkActivity extends Activity {
```

```
    public static final String EXTRA_DRINKNO = "drinkNo";
```

```
    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_drink);
```

```
        //Получение напитка из интента
```

```
        int drinkNo = (Integer) getIntent().getExtras().get(EXTRA_DRINKNO);
```

```
        //Создание курсора
```

```
        try {
```

```
            SQLiteOpenHelper starbuzzDatabaseHelper = new StarbuzzDatabaseHelper(this);
```

```
            SQLiteDatabase db = starbuzzDatabaseHelper.getReadableDatabase();
```

```
            Cursor cursor = db.query("DRINK",
```

```
                new String[] {"NAME", "DESCRIPTION", "IMAGE_RESOURCE_ID"},
```

```
                "_id = ?",
```

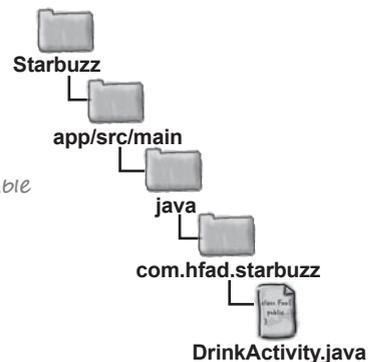
```
                new String[] {Integer.toString(drinkNo)},
```

```
                null, null, null);
```

Создать курсор для получения
из таблицы DRINK столбцов
NAME, DESCRIPTION и IMAGE_
RESOURCE_ID тех записей, у ко-
торых значение _id равно drinkNo.



DrinkActivity
DrinkCategoryActivity
Любимые напитки



Идентификатор
напитка, выбранного
пользователем.



Продолжение
на следующей
странице.

Kog DrinkActivity (продолжение)

```

//Переход к первой записи в курсоре
if (cursor.moveToFirst()) {
    //Получение данных напитка из курсора
    String nameText = cursor.getString(0);
    String descriptionText = cursor.getString(1);
    int photoId = cursor.getInt(2);

    //Заполнение названия напитка
    TextView name = (TextView) findViewById(R.id.name);
    name.setText(nameText);

    //Заполнение описания напитка
    TextView description = (TextView) findViewById(R.id.description);
    description.setText(descriptionText);

    //Заполнение изображения напитка
    ImageView photo = (ImageView) findViewById(R.id.photo);
    photo.setImageResource(photoId);
    photo.setContentDescription(nameText);
}
cursor.close();
db.close();
} catch (SQLException e) {
    Toast toast = Toast.makeText(this, "Database unavailable", Toast.LENGTH_SHORT);
    toast.show();
}

```

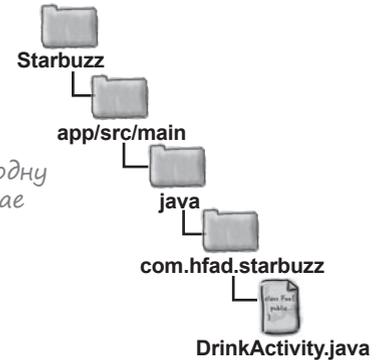
Название напитка хранится в первом столбце курсора, описание — во втором, а идентификатор ресурса изображения — в третьем. Вспомните, что столбцы NAME, DESCRIPTION и IMAGE_RESOURCE_ID базы данных были включены в курсор именно в таком порядке.

← Курсор содержит всего одну запись, но и в этом случае переход необходим.

← Использовать данные, полученные из курсора, для заполнения представлений.

← Закрыть курсор и базу данных.

↑ Если будет выдано исключение SQLException, значит, при работе с базой данных возникли проблемы. В этом случае объект Toast используется для выдачи сообщения для пользователя.



Итак, код DrinkActivity готов. Давайте посмотрим, что делать дальше.



РАССЛАБЬТЕСЬ

Подключение активностей к базе данных требует большего объема кода, чем использование класса Java.

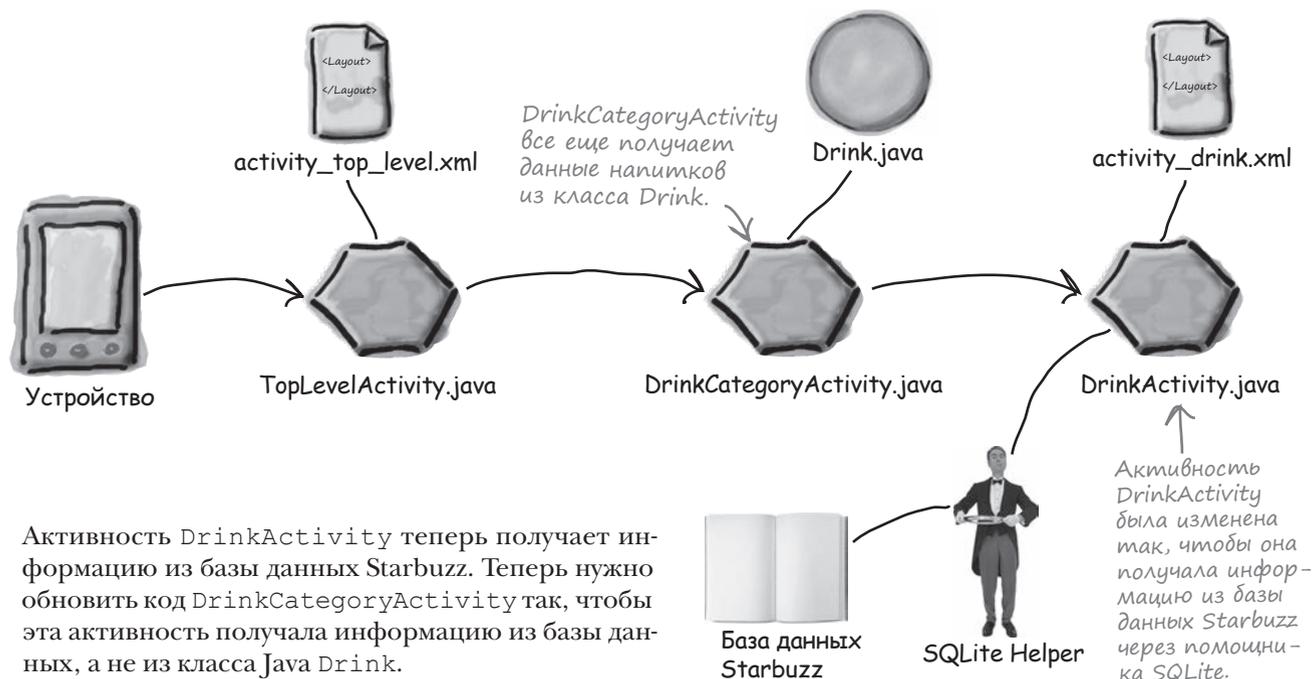
Но если вы не будете торопиться и тщательно проанализируете весь код, приведенный в этой главе, все будет хорошо.

Что мы сделали

Итак, мы успешно обновили код *DrinkActivity.java*. Теперь взглянем на диаграмму структуры приложения, посмотрим, что было сделано и что нужно сделать на следующем шаге.



DrinkActivity
DrinkCategoryActivity
 Любимые напитки



Активность *DrinkActivity* теперь получает информацию из базы данных *Starbuzz*. Теперь нужно обновить код *DrinkCategoryActivity* так, чтобы эта активность получала информацию из базы данных, а не из класса *Java Drink*.

Часто задаваемые вопросы

В: Насколько хорошо нужно знать SQL для создания курсоров?

О: Желательно понимать команды SQL *SELECT*, так как во внутренней реализации вызов метода *query()* преобразуется в такую команду. Вероятно, ваши запросы не будут особенно сложными, но понимать SQL все же полезно.

В: Вы сказали, что при неудачном обращении к базе данных выдается исключение *SQLiteException*. Что с ним делать?

О: Начните с проверки подробной информации об исключении. Возможно, исключение порождено ошибкой в синтаксисе SQL, которую можно исправить.

Способ обработки исключений зависит от того, как они влияют на работу приложения. Например, если к базе данных можно получить доступ для чтения, но не для записи, лучше предоставить ограниченный доступ пользователю и сообщить о невозможности сохранения изменений. В конечном итоге все зависит от приложения.

Текущий код DrinkCategoryActivity

Вспомним, как выглядит текущая версия кода *DrinkCategoryActivity.java*. Метод `onCreate()` заполняет списковое представление `ListView` данными напитков при помощи `ArrayAdapter`. Метод `onListItemClick()` добавляет напиток, выбранный пользователем, в интент, после чего запускает `DrinkActivity`:

```
package com.hfad.starbuzz;

import android.app.ListActivity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.view.View;
import android.content.Intent;

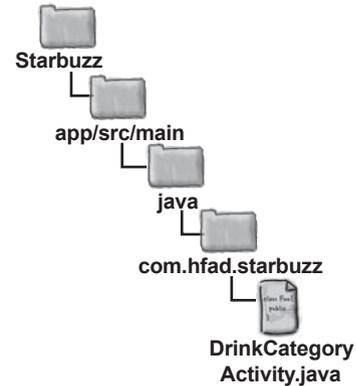
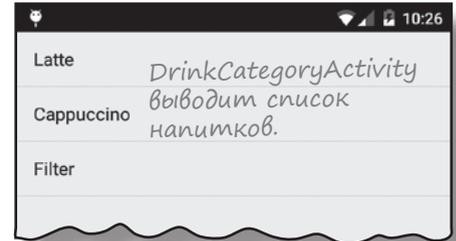
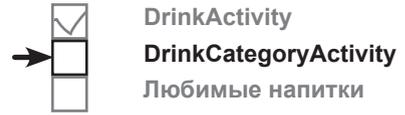
public class DrinkCategoryActivity extends ListActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ListView listDrinks = getListView();
        ArrayAdapter<Drink> listAdapter = new ArrayAdapter<Drink>(
            this,
            android.R.layout.simple_list_item_1,
            Drink.drinks);
        listDrinks.setAdapter(listAdapter);
    }

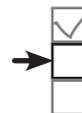
    @Override
    public void onListItemClick(ListView listView,
                               View itemView,
                               int position,
                               long id) {

        Intent intent = new Intent(DrinkCategoryActivity.this, DrinkActivity.class);
        intent.putExtra(DrinkActivity.EXTRA_DRINKNO, (int)id);
        startActivity(intent);
    }
}
```

курсоры и асинхронные задачи



В текущей версии мы используем объект `ArrayAdapter` для связывания массива с `ListView`. Код нужно изменить так, чтобы информация поступала из базы данных.



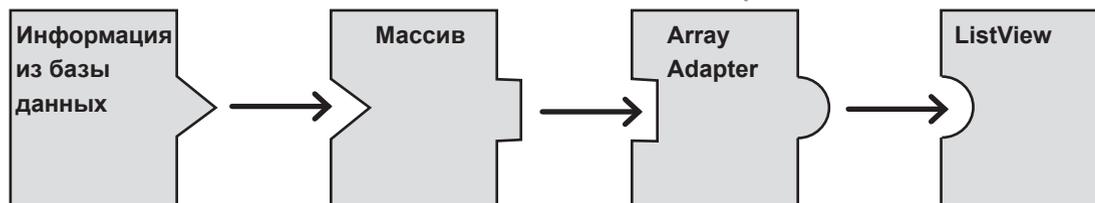
DrinkActivity
DrinkCategoryActivity
Любимые напитки

Как заменить данные массива в ListView?

Списковое представление `listDrinks` получает информацию из массива `Drink.drink`. Одно из *возможных* решений выглядит так: прочитать список напитков из базы данных и сохранить информацию в массиве, передаваемом адаптеру массива. Такое решение работает, но... Удается ли вам найти причину, по которой так действовать не стоит?

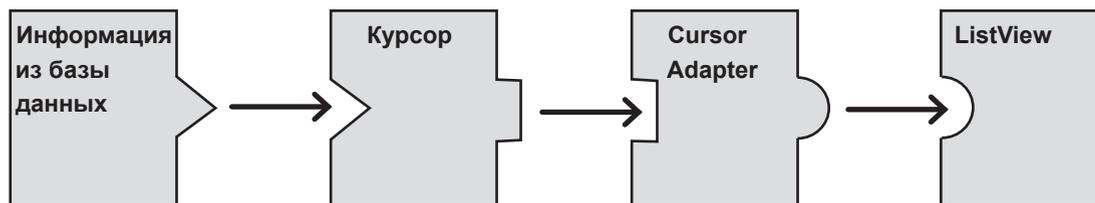
Информацию можно просто прочитать из базы данных и записать в массив.

Приложение работает со старым адаптером массива, остальной код остается неизменным. Но действительно ли это хорошее решение?



При такой маленькой базе данных, как у нас, чтение всей информации и хранение ее в массиве (то есть в памяти) не создает проблем. Но если приложение работает с очень большим объемом информации, ее чтение из базы данных займет некоторое время. Кроме того, для хранения массива потребуется много памяти.

Вместо этого мы перейдем с класса адаптера массива `ArrayAdapter` на адаптер курсора `CursorAdapter`.



Класс `CursorAdapter` очень похож на `ArrayAdapter`, но вместо того, чтобы читать данные из массива, он читает их из курсора.

Давайте посмотрим, как это делается.

Данные представлены в форме курсора, поэтому для их связывания с `ListView` можно воспользоваться классом `CursorAdapter`.

Адаптер курсора читает столько данных, сколько нужно

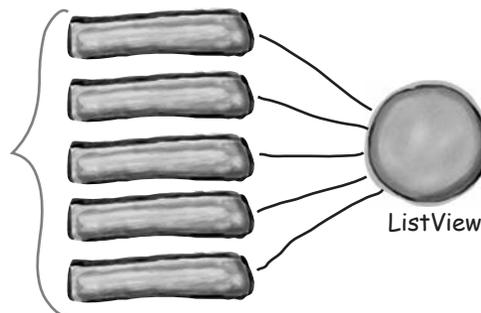
Допустим, наша база данных намного больше — например, сеть кофеен Starbuzz существенно расширила свой ассортимент для хипстеров. Может оказаться, что вместо *трех* типов кофе в базе данных придется хранить *300* напитков с разными сочетаниями крепости, молока и декоративной обсыпки. Однако в списке при этом будет отображаться лишь малая часть записей.

В представлении `ListView` одновременно может отображаться лишь небольшая часть записей. На устройствах с маленьким экраном изначально могут выводиться, скажем, первые 11 видов кофе. Если бы данные хранились в массиве, нам пришлось бы загрузить все 300 видов кофе из базы данных в массив и только потом вывести часть данных. С `CursorAdapter` все работает немного иначе.

1 Компонент `ListView` отображается на экране.

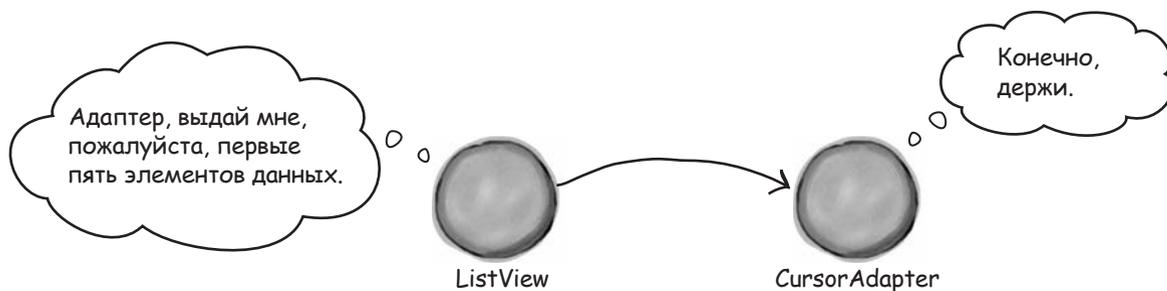
Когда списковое представление отображается впервые, оно масштабируется по размерам экрана. Предположим, в нем достаточно места для отображения пяти элементов.

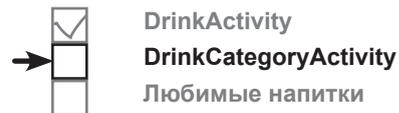
Данные, для которых хватает места в `ListView`. Для простоты будем считать, что таких элементов пять, но на практике их, вероятно, будет больше.



2 `ListView` запрашивает у своего адаптера первые пять элементов.

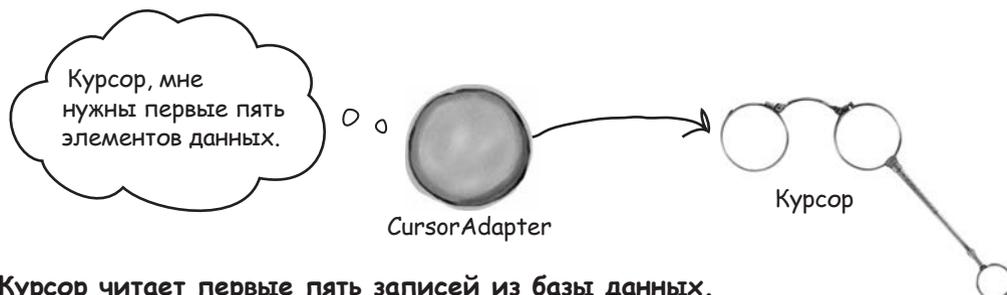
Компонент `ListView` не знает, где именно хранятся данные — в массиве или базе данных, — но *знает*, что он получит данные от адаптера. Соответственно, он обращается к адаптеру и запрашивает первые пять напитков.



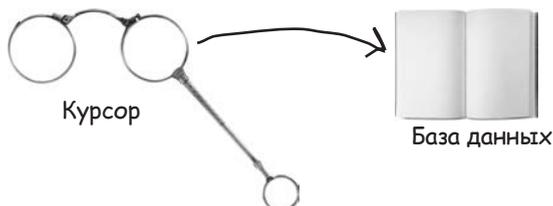


История продолжается

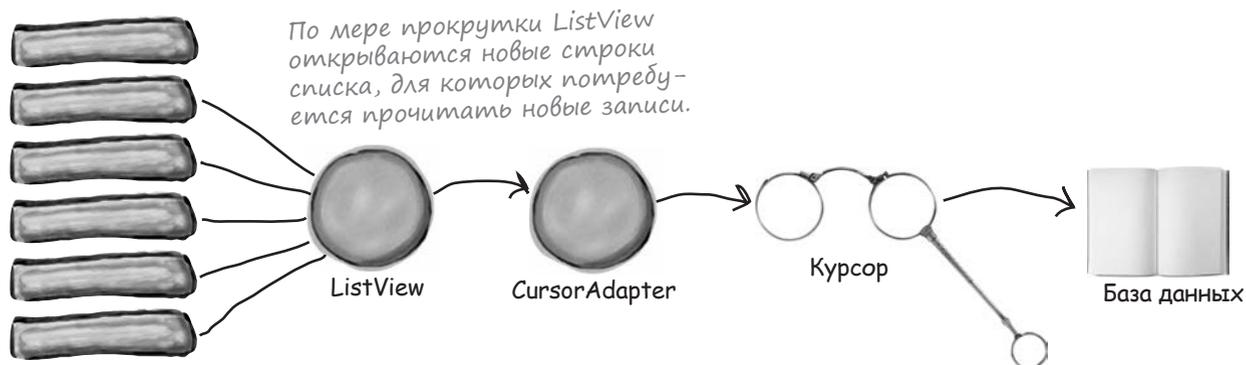
- 3** **CursorAdapter** приказывает курсору прочитать пять записей из базы данных. Объект `CursorAdapter` получает курсор при создании, но обращается к курсору за данными только тогда, когда потребуется.



- 4** **Курсор** читает первые пять записей из базы данных. Несмотря на то что таблица базы данных содержит 300 записей, курсор должен прочитать только первые пять. Такой подход намного эффективнее, и он означает, что данные появятся на экране намного быстрее.



- 5** **Пользователь прокручивает список.** Когда пользователь прокручивает список, `CursorAdapter` приказывает курсору прочитать другие записи из базы данных. Если пользователь ограничивается минимальной прокруткой и открывает только одну новую строку, курсор читает из базы данных одну запись.

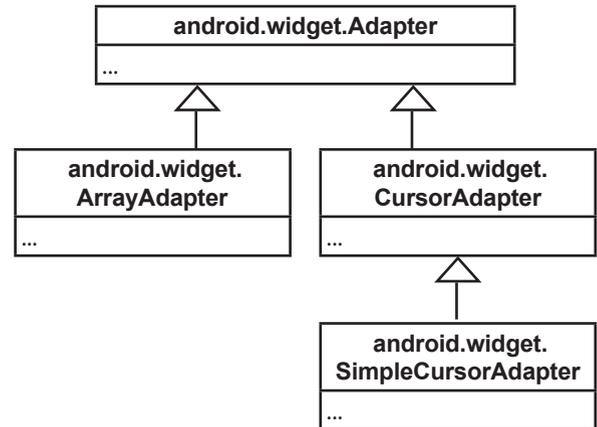
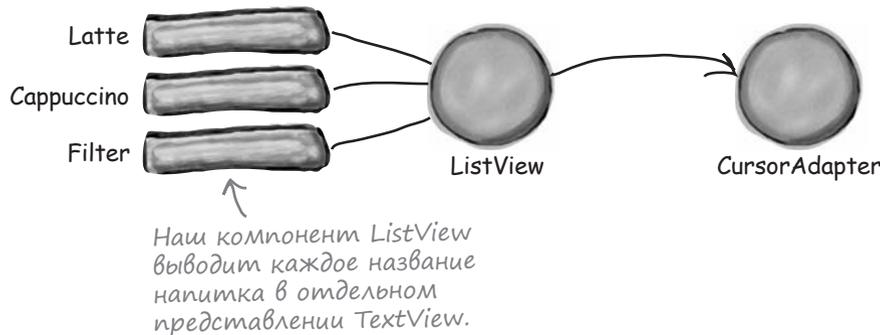


Итак, класс `CursorAdapter` в данном случае работает намного эффективнее `ArrayAdapter`: он читает данные только тогда, когда в них возникнет необходимость. Таким образом, он работает быстрее и занимает меньше памяти, а скорость и затраты памяти — важные факторы, о которых не следует забывать.

SimpleCursorAdapter связывает данные с представлениями

Мы создадим простой адаптер курсора для использования в приложении. Класс `SimpleCursorAdapter` — специализация `CursorAdapter`, которая может использоваться в большинстве ситуаций с выводом данных курсора в списковом представлении. Он получает столбцы из курсора и связывает их с компонентом `TextView` или `ImageView`.

В нашем случае список названий напитков выводится в списковом представлении активности `DrinkCategoryActivity`, поэтому мы используем простой адаптер курсора для связывания названия каждого напитка с текстовым представлением в `ListView`:



Как `ArrayAdapter`, так и `CursorAdapter` являются специализациями `Adapter`. `SimpleCursorAdapter` расширяет `CursorAdapter`.

Первый шаг — создание курсора

Первое, что необходимо учесть при создании курсора для адаптера, — какие столбцы должен содержать курсор. В курсор следует включить все столбцы, которые должны отображаться в списковом представлении, вместе со столбцом с именем `_id`. Столбец `_id` должен быть включен в курсор, иначе адаптер курсора работать не будет. Почему?

В главе 11 мы упоминали о том, что в Android принято присваивать столбцу первичного ключа таблицы имя `_id`. Это правило настолько закрепилось в Android, что *адаптер курсора предполагает, что этот столбец всегда присутствует*, и использует его для однозначной идентификации каждой строки в курсоре. При использовании адаптера курсора со списковым представлением этот столбец используется для определения строки, на которой щелкнул пользователь.

Так как адаптер курсора нужен нам для вывода названий напитков, курсор должен содержать столбцы `_id` и `NAME`:

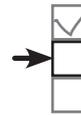
```

cursor = db.query("DRINK", new String[]{"_id", "NAME"},
    null, null, null, null, null);
  
```

← Столбец `_id` должен быть включен в массив, хотя его данные не выводятся.

На следующей странице курсор будет использован для создания адаптера курсора.

Создание объекта SimpleCursorAdapter



DrinkActivity
DrinkCategoryActivity
Любимые напитки

При создании простого адаптера необходимо указать, как должны выводиться данные, какой курсор следует использовать и какие столбцы должны связываться с теми или иными представлениями. Например, вот как создается простой адаптер курсора для вывода названий напитков:

```

cursor = db.query("DRINK", new String[]{"_id", "NAME"},
    null, null, null, null, null);
CursorAdapter listAdapter = new SimpleCursorAdapter(this,
    android.R.layout.simple_list_item_1,
    cursor,
    new String[]{"NAME"},
    new int[]{android.R.id.text1},
    0);
listDrinks.setAdapter(listAdapter);

```

← Курсор с предыдущей страницы.

← Тот же макет, который мы использовали ранее для адаптера массива. В этом макете в каждой строке спискового представления выводится одно значение.

← Вывести содержимое столбца NAME в текстовом представлении компонента ListView.

← Метод `setAdapter()` связывает адаптер со списковым представлением.

Это курсор. →

Как и в случае с адаптером массива, мы используем `android.R.layout.simple_list_item_1`, чтобы сообщить, что каждая строка в курсоре должна отображаться в виде одного текстового представления в списковом представлении. Этому текстовому представлению назначен идентификатор `android.R.id.text1`. Общая форма конструктора `SimpleCursorAdapter` выглядит так:

```

SimpleCursorAdapter adapter = new SimpleCursorAdapter(Context context,
    int layout,
    Cursor cursor,
    String[] fromColumns,
    int[] toViews,
    int flags)

```

← Созданный вами курсор. Он должен включать столбец `_id` и данные, которые должны выводиться в списковом представлении.

← Соответствие между столбцами курсора и представлениями.

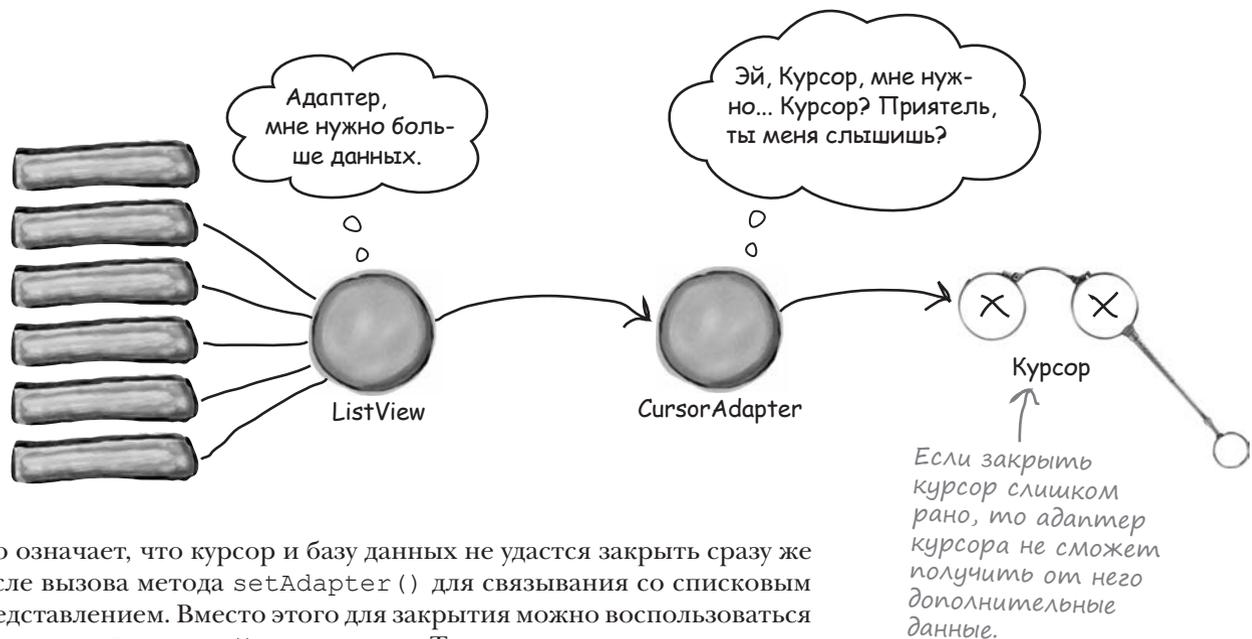
← Как должны отображаться данные. Здесь можно использовать тот же макет, который использовался с адаптером массива.

Параметры `context` и `layout` — те же, которые использовались при создании адаптера массива. В параметре `context` передается текущий контекст, а параметр `layout` сообщает, как должны отображаться данные. Вместо массива, из которого должны загружаться данные, нужно задать курсор с данными при помощи параметра `cursor`. Параметр `fromColumns` указывает, какие столбцы в курсоре должны использоваться, а параметр `toViews` — в каких представлениях они должны отображаться.

В параметре `flags` обычно передается 0 (значение по умолчанию). Также можно передать значение `FLAG_REGISTER_CONTENT_OBSERVER` для регистрации наблюдателя, который будет оповещаться об изменении данных. Здесь эта возможность не рассматривается, так как она может привести к утечке памяти. Позднее в этой главе вы узнаете, как обработать изменения в данных.

Заккрытие курсора и базы данных

Во время вашего знакомства с курсорами ранее в этой главе мы говорили, что курсор и базу данных после завершения работы с ними необходимо закрыть, чтобы освободить их ресурсы. В коде `DrinkActivity` мы использовали курсор для получения информации из базы данных. После заполнения представлений прочитанными данными мы немедленно закрывали курсор и базу данных. Схема с адаптером курсора работает немного иначе: курсор должен оставаться открытым на тот случай, если из него потребуется прочитать дополнительные данные. Например, это может произойти, если пользователь прокрутит списковое представление для просмотра новой порции данных.



Это означает, что курсор и базу данных не удастся закрыть сразу же после вызова метода `setAdapter()` для связывания со списковым представлением. Вместо этого для закрытия можно воспользоваться методом `onDestroy()` активности. Так как активность готовится к уничтожению, сохранять связь с курсором или базой данных не нужно, и их можно закрыть:

```
public void onDestroy() {
    super.onDestroy();
    cursor.close();
    db.close();
}
```

← Закрывать курсор и базу данных при уничтожении активности.

Посмотрим, удастся ли вам обновить код `DrinkCategoryActivity`.

У бассейна



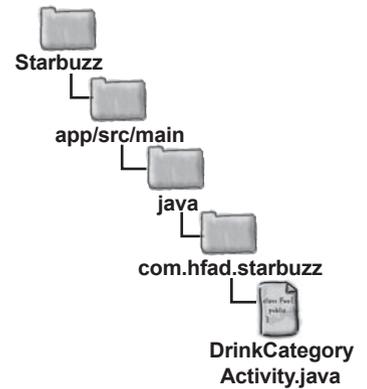
Выловите из бассейна сегменты кода и расставьте их в пропусках *DrinkCategoryActivity.java*. Каждый сегмент может использоваться **только один раз**; использовать все сегменты не обязательно. Ваша **задача**: заполнить компонент *ListView* данными напитков из базы данных.

```
...
public class DrinkCategoryActivity extends ListActivity {

    private SQLiteDatabase db;
    private Cursor cursor;

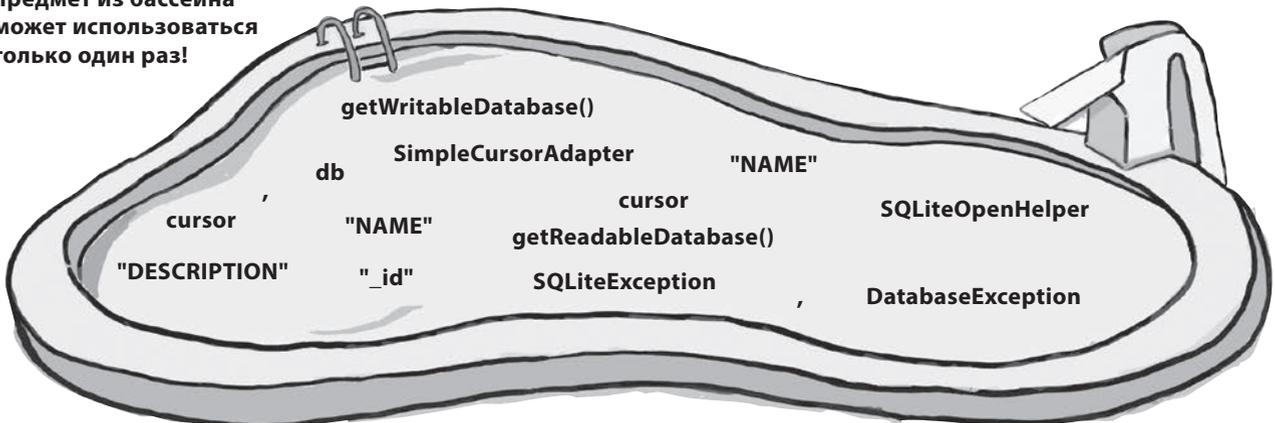
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ListView listDrinks = getListView();

        try {
            ..... starbuzzDatabaseHelper = new StarbuzzDatabaseHelper(this);
            db = starbuzzDatabaseHelper. .... ;
        }
    }
}
```



Внимание: каждый предмет из бассейна может использоваться только один раз!

Продолжение на следующей странице →



```

        cursor = db.query("DRINK",
            new String[]{.....},
            null, null, null, null, null);

        CursorAdapter listAdapter = new .....(this,
            android.R.layout.simple_list_item_1,
            .....',
            new String[]{.....},
            new int[]{android.R.id.text1},
            0);

        listDrinks.setAdapter(listAdapter);

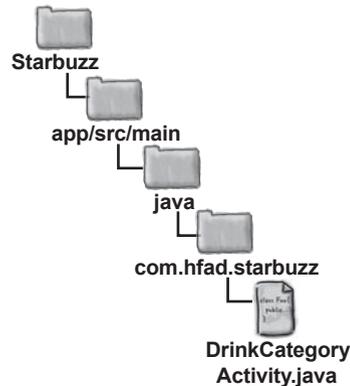
    } catch( .....e) {
        Toast toast = Toast.makeText(this, "Database unavailable", Toast.LENGTH_SHORT);
        toast.show();
    }
}

@Override
public void onDestroy(){
    super.onDestroy();
    ..... .close();
    ..... .close();
}

@Override
public void onItemClick(ListView listView,
    View itemView,
    int position,
    long id) {

    Intent intent = new Intent(DrinkCategoryActivity.this, DrinkActivity.class);
    intent.putExtra(DrinkActivity.EXTRA_DRINKNO, (int)id);
    startActivity(intent);
}
}

```



У бассейна. Решение



Выловите из бассейна сегменты кода и расставьте их в пропусках `DrinkCategoryActivity.java`. Каждый сегмент может использоваться **только один** раз; использовать все сегменты не обязательно. Ваша **задача**: заполнить компонент `ListView` данными напитков из базы данных.

...

```
public class DrinkCategoryActivity extends ListActivity {

    private SQLiteDatabase db;
    private Cursor cursor;

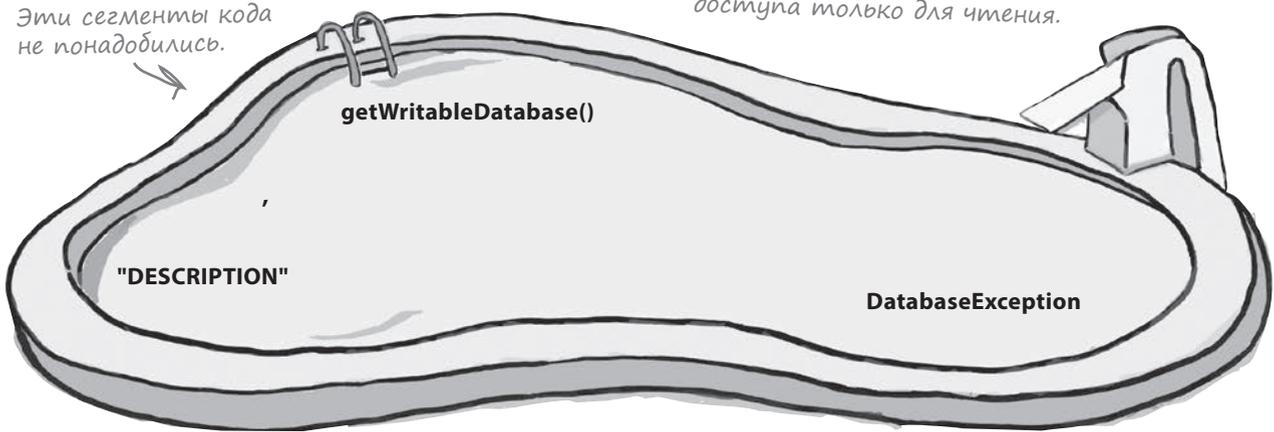
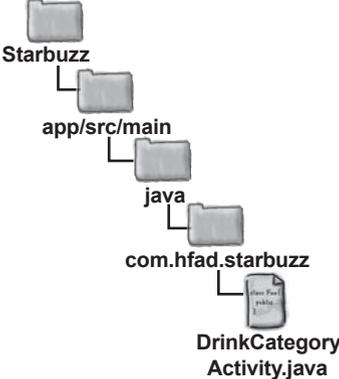
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ListView listDrinks = getListView();

        try {
            SQLiteOpenHelper starbuzzDatabaseHelper = new StarbuzzDatabaseHelper(this);
            db = starbuzzDatabaseHelper.getReadableDatabase();
        }
    }
}
```

Получить ссылку на базу данных с помощью `SQLiteOpenHelper`.

Информация читается из базы данных, поэтому хватит доступа только для чтения.

Эти сегменты кода не понадобились.



```

cursor = db.query("DRINK",
    new String[]{"_id", "NAME"},
    null, null, null, null, null);

CursorAdapter listAdapter = new SimpleCursorAdapter(this,
    android.R.layout.simple_list_item_1,
    cursor,
    new String[]{"NAME"},
    new int[]{android.R.id.text1},
    0);

listDrinks.setAdapter(listAdapter);

} catch (SQLiteException e) {
    Toast toast = Toast.makeText(this, "Database unavailable", Toast.LENGTH_SHORT);
    toast.show();
}

@Override
public void onDestroy() {
    super.onDestroy();
    cursor.close();
    db.close();
}

@Override
public void onItemClick(AdapterView listView,
    View itemView,
    int position,
    long id) {
    Intent intent = new Intent(DrinkCategoryActivity.this, DrinkActivity.class);
    intent.putExtra(DrinkActivity.EXTRA_DRINKNO, (int) id);
    startActivity(intent);
}
}

```

Курсор должен включать столбец `_id` — это необходимо для работы адаптера. Он также должен включать столбец `NAME` для вывода списка названий напитков.

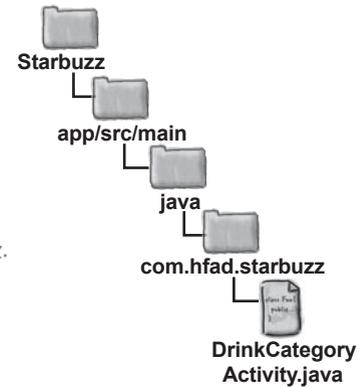
Используем класс `SimpleCursorAdapter`.

Использовать только что созданный курсор.

Вывести содержимое столбца `NAME`.

Если база данных недоступна, перехватим исключение `SQLiteException`.

Закрывать курсор перед закрытием базы данных.



Обновленный код DrinkCategoryActivity

Ниже приведен полный код `DrinkCategoryActivity.java`, в котором адаптер массива заменяется адаптером курсора (изменения выделены жирным шрифтом):

```
package com.hfad.starbuzz;

import android.app.ListActivity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.ListView;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteException;
import android.database.sqlite.SQLiteOpenHelper;
import android.widget.CursorAdapter;
import android.widget.SimpleCursorAdapter;
import android.widget.Toast;
import android.widget.SimpleCursorAdapter;
```

Чтобы использовать эти дополнительные классы, их необходимо импортировать.

```
public class DrinkCategoryActivity extends ListActivity {
    private SQLiteDatabase db;
    private Cursor cursor;

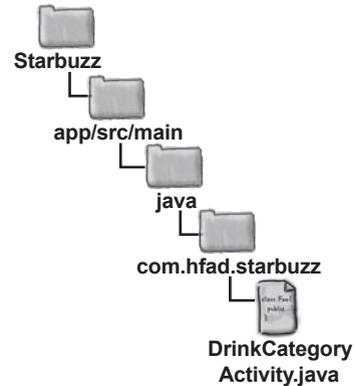
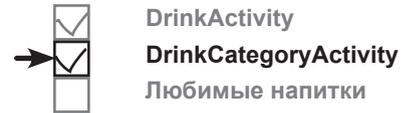
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ListView listDrinks = getListView();

        try {
            SQLiteOpenHelper starbuzzDatabaseHelper = new StarbuzzDatabaseHelper(this);
            db = starbuzzDatabaseHelper.getReadableDatabase();

            cursor = db.query("DRINK",
                new String[]{"_id", "NAME"},
                null, null, null, null, null);
        }
    }
}
```

↑
Создать курсор.

← Эти приватные переменные добавляются для того, чтобы базу данных и курсор можно было закрыть в методе `onDestroy()`.



↑ Получить ссылку на базу данных.

Продолжение на следующей странице.

Ког DrinkCategoryActivity (продолжение)

Создать адаптер курсора.

```
CursorAdapter listAdapter = new SimpleCursorAdapter(this,
    android.R.layout.simple_list_item_1,
    cursor,
    new String[]{"NAME"},
    new int[]{android.R.id.text1},
    0);
listDrinks.setAdapter(listAdapter);
} catch (SQLException e) {
    Toast toast = Toast.makeText(this, "Database unavailable", Toast.LENGTH_SHORT);
    toast.show();
}
```

Связать содержимое столбца NAME с текстом в ListView. →

В этой версии также используется адаптер — но на этот раз адаптер курсора.

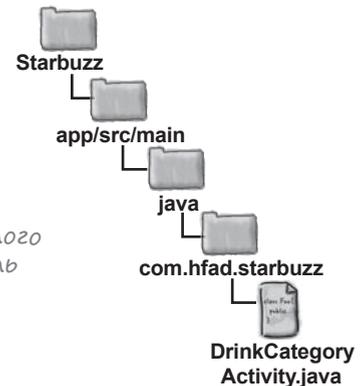
↑
Вывести сообщение для пользователя, если будет выдано исключение SQLException.

```
@Override
public void onDestroy() {
    super.onDestroy();
    cursor.close();
    db.close();
}
```

← База данных и курсор закрываются в методе onDestroy() активности. Курсор остается открытым до того момента, когда он перестает быть нужным адаптеру.

```
@Override
public void onItemClick (ListView listView,
    View itemView,
    int position,
    long id) {
    Intent intent = new Intent(DrinkCategoryActivity.this, DrinkActivity.class);
    intent.putExtra(DrinkActivity.EXTRA_DRINKNO, (int)id);
    startActivity(intent);
}
```

← Изменять этот метод не нужно.

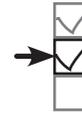


Попробуем запустить приложение.



Тест-драйв

Внесите изменения в код, постройте и заново запустите приложение. Вы увидите, что приложение выглядит точно так же, как и прежде.



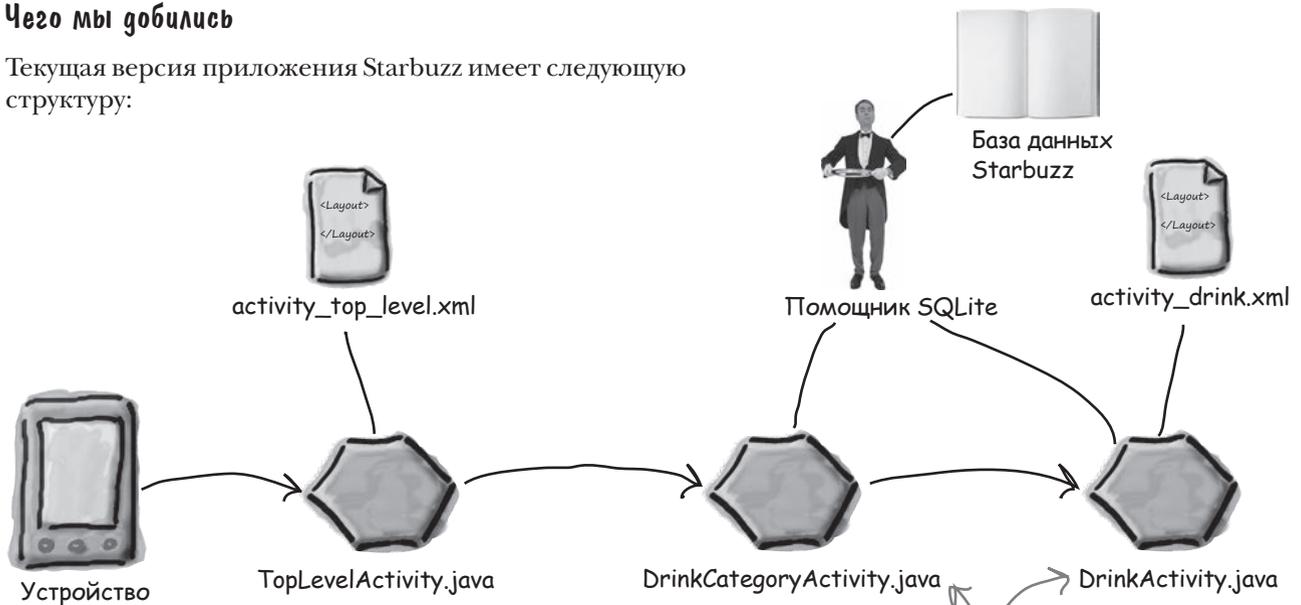
DrinkActivity
DrinkCategoryActivity
Любимые напитки



Но теперь все данные читаются из базы. Собственно, код *Drink.java* вообще можно удалить, потому что массив напитков нам уже не нужен. Вся необходимая информация берется из базы данных.

Чего мы добились

Текущая версия приложения Starbuzz имеет следующую структуру:



Осталось внести еще одно изменение: реализовать возможность обновления информации в базе данных.

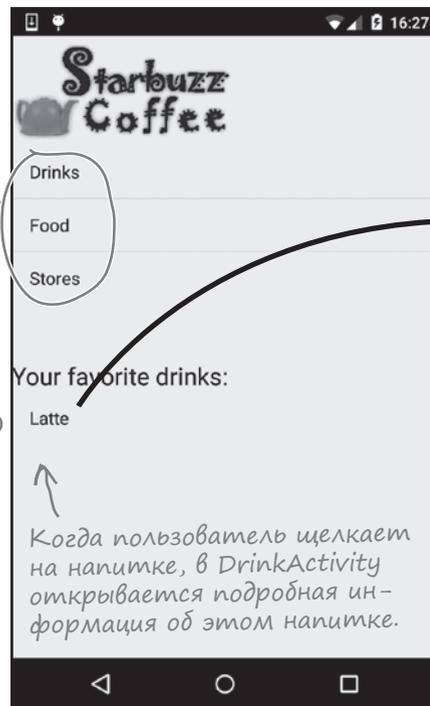
Мы изменили активности, работающие с классом *Drink*. Теперь они получают информацию из базы данных.

Размещение важной информации в активности верхнего уровня

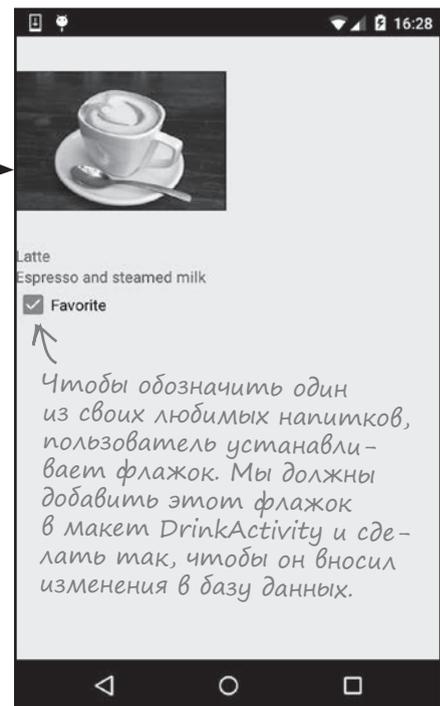
В самом начале работы над приложением Starbuzz мы намеренно сделали активность верхнего уровня очень простой. Активность верхнего уровня — первая активность, которую видит пользователь при запуске, и в приложении Starbuzz она содержит только изображение и три навигационные команды, которые были вынесены на выдвижную панель. Конечно, простота пользовательского интерфейса — дело хорошее, но не *слишком* ли он прост? Разработчик должен тщательно продумать макет активности верхнего уровня, потому что это первое, что видит пользователь. В идеале активность верхнего уровня должна содержать информацию, полезную как для новых, так и опытных пользователей. Как этого добиться? Один из возможных вариантов — подумать, что будет делать пользователь в вашем приложении, и предоставить ему соответствующие средства на начальном экране. Например, если разрабатываемое приложение предназначено для воспроизведения музыки, в активности верхнего уровня можно вывести список последних альбомов, выбранных пользователем. Мы изменим активность верхнего уровня приложения Starbuzz: в ней будет выводиться список любимых напитков пользователя, чтобы пользователь мог одним щелчком перейти к выбранному напитку.

В идеале эти команды следовало бы разместить на выдвижной панели. Мы не будем добавлять выдвижную панель в приложение Starbuzz, так как код реализации выдвижных панелей достаточно сложен, а мы сейчас хотим сосредоточиться на базах данных.

В `TopLevelActivity` добавляется списокое представление `ListView` с любимыми напитками пользователя.



Когда пользователь щелкает на напитке, в `DrinkActivity` открывается подробная информация об этом напитке.



Чтобы обозначить один из своих любимых напитков, пользователь устанавливает флажок. Мы должны добавить этот флажок в макет `DrinkActivity` и сделать так, чтобы он вносил изменения в базу данных.

Для этого необходимо предоставить пользователю средства для ввода информации о его любимых напитках.

Любимые напитки в DrinkActivity

В главе 11 мы добавили в таблицу DRINK базы данных Starbuzz столбец FAVORITE. С помощью этого столбца пользователь сможет пометить напиток как любимый, чтобы приложение знало, какие напитки должны отображаться в TopLevelActivity. Также пользователю будет предоставлена возможность редактировать данные в активности DrinkActivity, в которой выводится подробная информация о напитке.

Для этого нужно добавить в `activity_drink.xml` новое представление, которое будет использоваться для редактирования и вывода значения столбца FAVORITE. Тип представления, используемого в макете, зависит от типа данных, для которых оно предназначено. В нашем примере представление будет использоваться для выбора логических значений «да/нет», поэтому мы выберем флажок. Начните с добавления в `strings.xml` строкового ресурса с именем `favorite` (этот текст будет выводиться рядом с флажком):

```
<string name="favorite">Favorite</string>
```

Затем добавьте флажок в `activity_drink.xml`. Мы присваиваем ему идентификатор `favorite` и используем атрибут `android:text` для вывода подписи. Также атрибуту `android:onClick` присваивается значение `"onFavoriteClicked"`, чтобы при щелчке на флажке вызывался метод `onFavoriteClicked()` активности `DrinkActivity`.

```
<LinearLayout ...>
    <ImageView android:id="@+id/photo"
        ... />

    <TextView android:id="@+id/name"
        ... />

    <TextView android:id="@+id/description"
        ... />

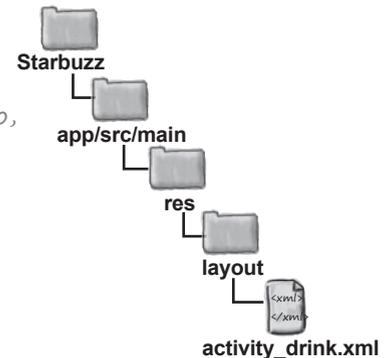
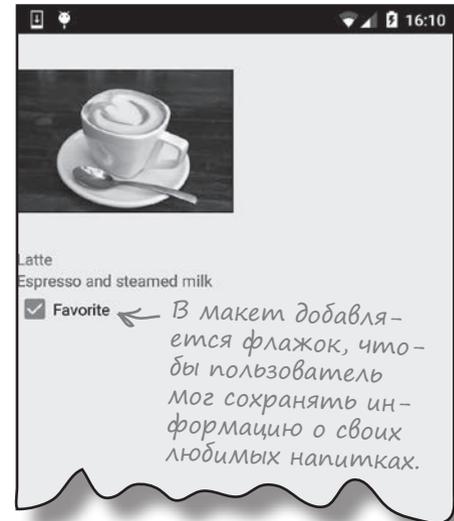
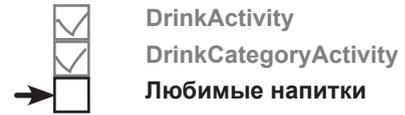
    <CheckBox android:id="@+id/favorite"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/favorite"
        android:onClick="onFavoriteClicked" />
</LinearLayout>
```

Представления `photo`, `name` и `description` были добавлены при создании исходной версии активности.

Флажку присваивается идентификатор `favorite`.

Текстовая подпись для флажка.

При щелчке на флажке вызывается метод `onFavoriteClicked()`.



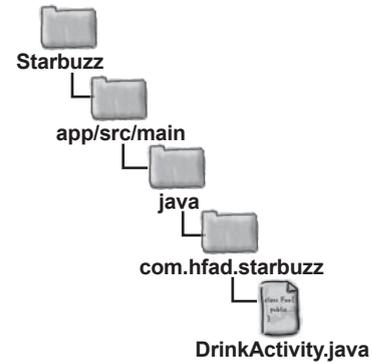
Добавление нового столбца в курсор

Дальше необходимо внести изменения в код `DrinkActivity`, чтобы флажок `favorite` соответствовал значению столбца `FAVORITE` из базы данных.

Значение столбца `FAVORITE` читается так же, как и данные других представлений активности, — в курсор добавляется столбец `FAVORITE`. Затем мы получаем значение столбца `FAVORITE` из курсора и устанавливаем или снимаем флажок. Соответствующая часть метода `onCreate()` выглядит так:

```
protected void onCreate(Bundle savedInstanceState) {
    ...
    SQLiteOpenHelper starbuzzDatabaseHelper = new StarbuzzDatabaseHelper(this);
    SQLiteDatabase db = starbuzzDatabaseHelper.getReadableDatabase();
    Cursor cursor = db.query("DRINK",
        new String[] {"NAME", "DESCRIPTION", "IMAGE_RESOURCE_ID", "FAVORITE"},
        "_id = ?",
        new String[] {Integer.toString(drinkNo)},
        null, null, null);

    //Переход к первой записи в курсоре
    if (cursor.moveToFirst()) {
        //Получение данных напитка из курсора
        String nameText = cursor.getString(0);
        String descriptionText = cursor.getString(1);
        int photoId = cursor.getInt(2);
        boolean isFavorite = (cursor.getInt(3) == 1);
        ...
        //Заполнение флажка любимого напитка
        CheckBox favorite = (CheckBox) findViewById(R.id.favorite);
        favorite.setChecked(isFavorite);
        ...
    }
}
```



↑
Добавить столбец
FAVORITE в курсор.

← Получить значение столбца FAVORITE. Оно хранится в базе данных в числовом виде: 1 — да, 0 — нет.

↑
Задать состояние флажка favorite.

Этого достаточно, чтобы состояние флажка в активности соответствовало значению столбца `FAVORITE`. Теперь нужно позаботиться о том, чтобы при щелчке на флажке в базу данных вносились изменения.

Обновление данных по щелчку на флажке



DrinkActivity
DrinkCategoryActivity
Любимые напитки

При добавлении флажка в файл `activity_drink.xml` с атрибутом `android:onClick` был связан метод `onFavoriteClicked()`. Это означает, что при каждом щелчке на флажке будет вызываться метод `onFavoriteClicked()` активности. Этот метод должен сохранять в базе данных текущее значение флажка. Когда пользователь устанавливает или снимает флажок, вызывается метод `onFavoriteClicked()`, и внесенные пользователем изменения сохраняются в базе данных. В главе 11 было показано, как использовать методы класса `SQLiteDatabase` для изменения данных, хранящихся в базе данных `SQLite`. В частности, мы показали, как метод `insert()` используется для вставки данных, метод `delete()` — для удаления данных, и метод `update()` — для обновления существующих записей.

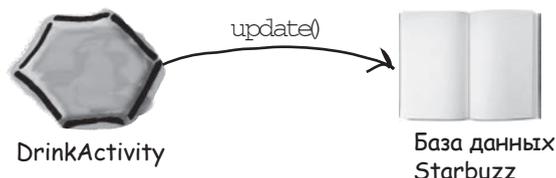
Эти методы также могут использоваться для модификации данных из кода активности. Например, мы можем использовать метод `insert()` для добавления новых записей в таблицу `DRINK` или метод `delete()` для их удаления. В нашем примере нужно обновить столбец `FAVORITE` таблицы `DRINK` состоянием флажка; эта задача решается при помощи метода `update()`.

Напомним, что вызов метода `update()` имеет следующую форму:

```
database.update(String table,
                ContentValues values,
                String whereClause,
                String[] whereArgs);
```

где `table` — имя таблицы, в которую вносятся изменения; `values` — объект `ContentValues` с парами «имя/значение» обновляемых столбцов и значений, которые им присваиваются. Параметры `whereClause` и `whereArgs` определяют записи, в которые вносятся изменения.

Вы уже знаете все, что необходимо знать для того, чтобы активность `DrinkActivity` обновляла столбец `FAVORITE` текущего напитка при щелчке на флажке. Проверьте свои силы в следующем упражнении.





Развлечения с Магнитами

В коде DrinkActivity столбец FAVORITE базы данных должен обновляться значением флажка favorite. Сможете ли вы построить метод onFavoriteClicked(), который будет решать эту задачу?

```
public class DrinkActivity extends Activity {
...
    //Обновление базы данных по щелчку на флажке

    public void onFavoriteClicked( .....){

        int drinkNo = (Integer) getIntent().getExtras().get(EXTRA_DRINKNO);
        CheckBox favorite = (CheckBox) findViewById(R.id.favorite);

        ..... drinkValues = new .....;.....

        drinkValues.put( ....., favorite.isChecked());

        SQLiteOpenHelper starbuzzDatabaseHelper =
            new StarbuzzDatabaseHelper(DrinkActivity.this);

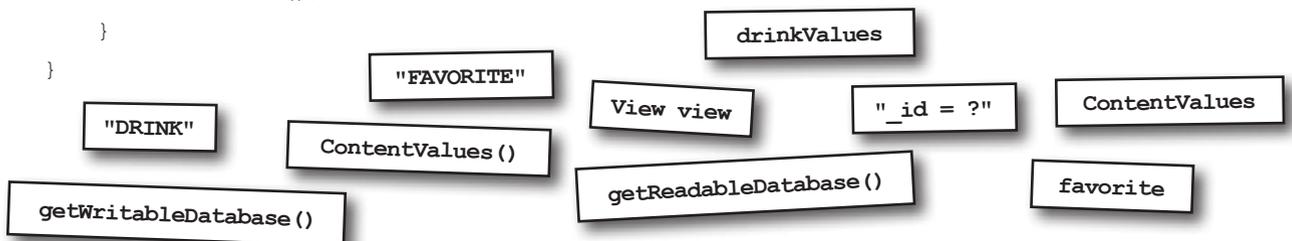
        try {

            SQLiteDatabase db = starbuzzDatabaseHelper. ....;.....

            db.update( .....;.....

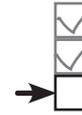
            ....., new String[] {Integer.toString(drinkNo)});

            db.close();
        } catch (SQLException e) {
            Toast toast = Toast.makeText(this, "Database unavailable", Toast.LENGTH_SHORT);
            toast.show();
        }
    }
}
```





Развлечения с магнитами. Решение



DrinkActivity
 DrinkCategoryActivity
 Любимые напитки

В коде DrinkActivity столбец FAVORITE базы данных должен обновляться значением флажка favorite. Сможете ли вы построить метод onFavoriteClicked(), который будет решать эту задачу?

```
public class DrinkActivity extends Activity {
...
    //Обновление базы данных по щелчку на флажке

    public void onFavoriteClicked(View view) {...

        int drinkNo = (Integer) getIntent().getExtras().get(EXTRA_DRINKNO);
        CheckBox favorite = (CheckBox) findViewById(R.id.favorite);

        ContentValues drinkValues = new ContentValues() ;.....

        drinkValues.put("FAVORITE", favorite.isChecked());

        SQLiteOpenHelper starbuzzDatabaseHelper =
            new StarbuzzDatabaseHelper(DrinkActivity.this);

        try {

            SQLiteDatabase db = starbuzzDatabaseHelper.getWritableDatabase() ;.....

            db.update("DRINK", drinkValues,
                "_id = ?", new String[] {Integer.toString(drinkNo)});

            db.close();
        } catch (SQLException e) {
            Toast toast = Toast.makeText(this, "Database unavailable", Toast.LENGTH_SHORT);
            toast.show();
        }
    }
}
```

↑
 Чтобы обновить базу данных,
 необходимо получить доступ
 к ней для чтения/записи.

Эти магниты не понадобились.

getReadableDatabase()

favorite

Kog DrinkActivity

Ниже приведен полный код *DrinkActivity.java* (изменения выделены жирным шрифтом):

```
package com.hfad.starbuzz;

import android.app.Activity;
import android.os.Bundle;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteException;
import android.database.sqlite.SQLiteOpenHelper;
import android.view.View;
import android.widget.CheckBox;
import android.content.ContentValues;
```

Эти классы используются в коде.

```
public class DrinkActivity extends Activity {

    public static final String EXTRA_DRINKNO = "drinkNo";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_drink);

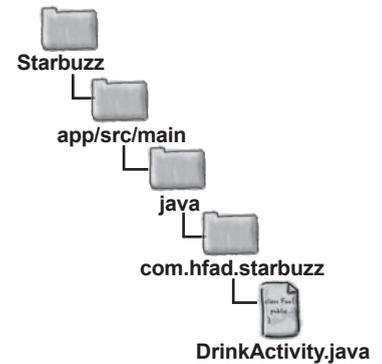
        //Получение напитка из интента
        int drinkNo = (Integer) getIntent().getExtras().get(EXTRA_DRINKNO);

        //Создание курсора
        try {
            SQLiteOpenHelper starbuzzDatabaseHelper = new StarbuzzDatabaseHelper(this);
            SQLiteDatabase db = starbuzzDatabaseHelper.getWritableDatabase();
            Cursor cursor = db.query("DRINK",
                new String[] {"NAME", "DESCRIPTION", "IMAGE_RESOURCE_ID", "FAVORITE"},
                "_id = ?",
                new String[] {Integer.toString(drinkNo)},
                null, null, null);

            //Переход к первой записи в курсоре
            if (cursor.moveToFirst()) {

                //Получение данных напитка из курсора
                String nameText = cursor.getString(0);
                String descriptionText = cursor.getString(1);
                int photoId = cursor.getInt(2);
                boolean isFavorite = (cursor.getInt(3) == 1);
            }
        }
    }
}
```

Получить значение столбца FAVORITE.

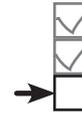


Чтобы обновить базу данных, необходимо получить доступ к ней для чтения/записи.

Добавить столбец FAVORITE в курсор.

Продолжение на следующей странице.

Код DrinkActivity (продолжение)



DrinkActivity
DrinkCategoryActivity
Любимые напитки

```

//Заполнение названия напитка
TextView name = (TextView) findViewById(R.id.name);
name.setText(nameText);

//Заполнение описания напитка
TextView description = (TextView) findViewById(R.id.description);
description.setText(descriptionText);

//Заполнение изображения напитка
ImageView photo = (ImageView) findViewById(R.id.photo);
photo.setImageResource(photoId);
photo.setContentDescription(nameText);

//Заполнение флажка любимого напитка
CheckBox favorite = (CheckBox) findViewById(R.id.favorite);
favorite.setChecked(isFavorite);
};
cursor.close();
db.close();
} catch(SQLiteException e) {
    Toast toast = Toast.makeText(this, "Database unavailable", Toast.LENGTH_SHORT);
    toast.show();
}
}

//Обновление базы данных по щелчку на флажке
public void onFavoriteClicked(View view) {
    int drinkNo = (Integer) getIntent().getExtras().get("drinkNo");
    CheckBox favorite = (CheckBox) findViewById(R.id.favorite);
    ContentValues drinkValues = new ContentValues();
    drinkValues.put("FAVORITE", favorite.isChecked());
    SQLiteOpenHelper starbuzzDatabaseHelper =
        new StarbuzzDatabaseHelper(DrinkActivity.this);

    try {
        SQLiteDatabase db = starbuzzDatabaseHelper.getWritableDatabase();
        Обновить столбец FAVORITE текущим значением флажка. → db.update("DRINK", drinkValues,
            "_id = ?", new String[] {Integer.toString(drinkNo)});
        db.close();
    } catch(SQLiteException e) {
        Toast toast = Toast.makeText(this, "Database unavailable", Toast.LENGTH_SHORT);
        toast.show();
    }
}
}
}

```

Задать состояние флажка.

Значение флажка добавляется в объект ContentValues с именем drinkValues.

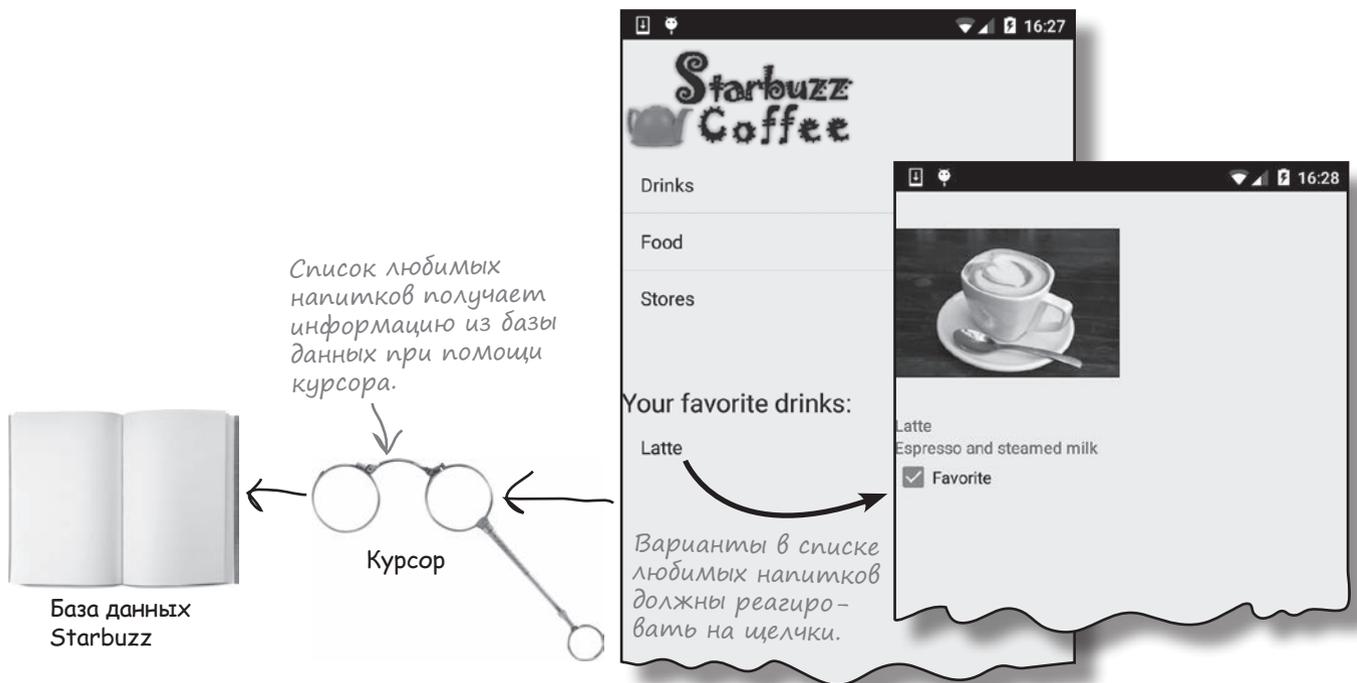
Вывести сообщение при возникновении проблемы с базой данных.

Вывод любимых напитков в TopLevelActivity

И последнее, что остается сделать, — вывести любимые напитки пользователя в TopLevelActivity.

- ★ **Добавление нового компонента ListView в макет.**
Компонент предназначен для вывода списка любимых напитков пользователя.
- ★ **Заполнение ListView.**
Список необходимо заполнить любимыми напитками пользователя из базы данных.
- ★ **Обработка щелчков на компоненте ListView.**
Если пользователь щелкает на одном из своих любимых напитков, информация об этом напитке выводится в DrinkActivity.

После внесения всех изменений в TopLevelActivity будет выводиться список любимых напитков пользователя.



На нескольких ближайших страницах мы подробно разберем код, который все это делает.

Вывод списка любимых напитков в activity_top_level.xml

Как было сказано на предыдущей странице, мы добавим в `activity_top_level.xml` списковое представление, которое будет использоваться для вывода любимых напитков пользователя. Также будет добавлено текстовое представление для вывода заголовка списка.

Начните с добавления в `strings.xml` следующего строкового ресурса (он будет использоваться для заполнения текстового представления):

```
<string name="favorites">Your favorite drinks:</string>
```

Затем внесите изменения в файл `activity_top_level.xml` и включите в него текстовое и списковое представление:

```
<LinearLayout ... >
  <ImageView
    android:layout_width="200dp"
    android:layout_height="100dp"
    android:src="@drawable/starbuzz_logo"
    android:contentDescription="@string/starbuzz_logo" />

  <ListView
    android:id="@+id/list_options"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:entries="@array/options" />

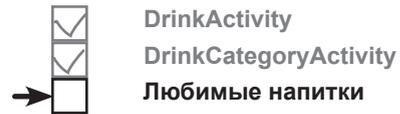
  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="50dp"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:text="@string/favorites" />

  <ListView
    android:id="@+id/list_favorites"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
</LinearLayout>
```

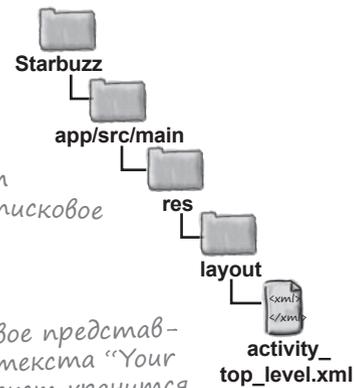
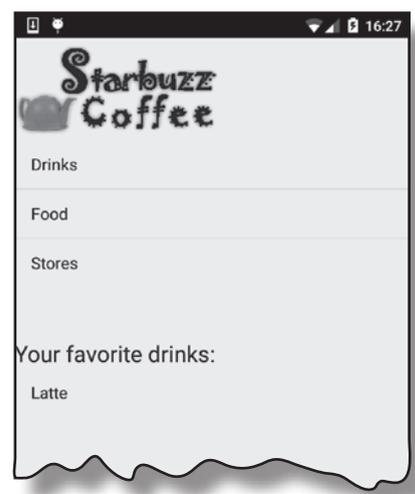
Макет уже содержит логотип Starbuzz и списковое представление.

Добавим текстовое представление для вывода текста "Your favorite drinks": Текст хранится в строковом ресурсе с именем favorites.

В списковом представлении `list_favorites` выводятся любимые напитки пользователя.



DrinkActivity
DrinkCategoryActivity
Любимые напитки



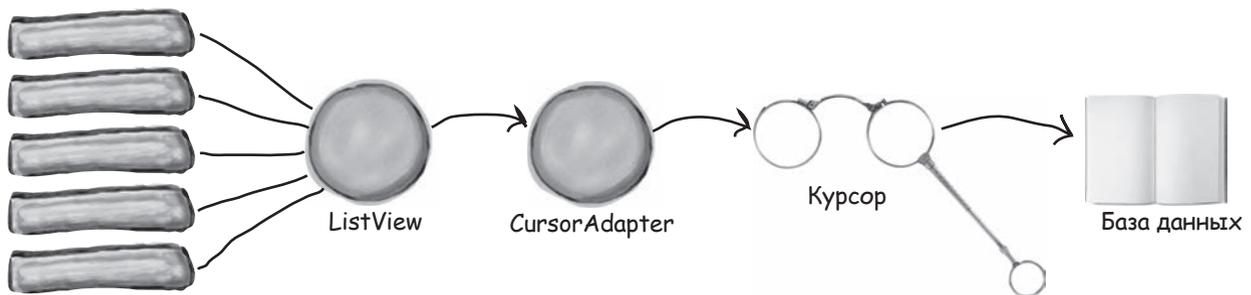
Вот и все изменения, которые необходимо внести в разметку `activity_top_level.xml`. Затем необходимо обновить `TopLevelActivity.java`.

Необходимые изменения в `TopLevelActivity.java`

Следующее, что нужно сделать, — вывести любимые напитки пользователя в только что добавленном списковом представлении и добиться того, чтобы списковое представление реагировало на выбор вариантов. Для этого необходимо:

1 Создать курсор для заполнения `ListView`.

Курсор будет возвращать все напитки, у которых столбец `FAVORITE` содержит 1, — то есть все напитки, помеченные пользователем как любимые. По аналогии с тем, как это было сделано в коде `DrinkCategoryActivity`, курсор связывается с `ListView` при помощи объекта `CursorAdapter`.



2 Создать объект `onItemClickListener`, чтобы компонент `ListView` мог реагировать на щелчки.

Если пользователь выбирает один из своих любимых напитков, мы создаем интент, который запускает `DrinkActivity`, и добавляем в него идентификатор выбранного напитка. Идентификатор будет использован для вывода подробной информации о только что выбранном напитке.



Вы уже видели, как это делается, поэтому на нескольких следующих страницах мы сразу приведем полный код `TopLevelActivity.java`.

Новый код активности верхнего уровня

Перед вами новый код, который необходимо добавить в `TopLevelActivity.java` (нового кода довольно много, так что не торопитесь и внимательно изучите его):

```
package com.hfad.starbuzz;

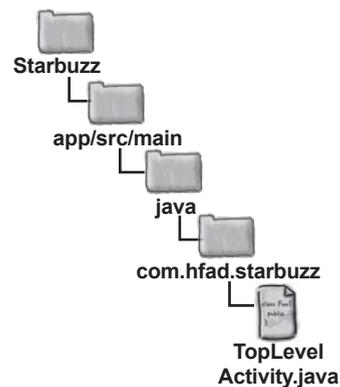
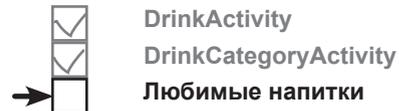
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.AdapterView.OnItemSelectedListener;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.AdapterView.OnItemSelectedListener;
import android.view.View;
import android.database.Cursor;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteException;
import android.database.sqlite.SQLiteDatabase;
import android.widget.SimpleCursorAdapter;
import android.widget.CursorAdapter;
import android.widget.Toast;

public class TopLevelActivity extends Activity {
```

```
private SQLiteDatabase db;
private Cursor favoritesCursor;
```

Эти приватные переменные добавлены для того, чтобы объекты были доступны в методе `onDestroy()`.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_top_level);
}
```

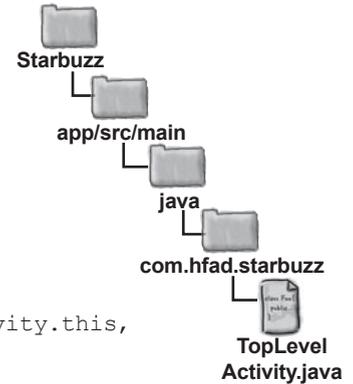


Дополнительные классы, используемые в коде.

Продолжение на следующей странице. →

Код TopLevelActivity (продолжение)

```
//Создание объекта onItemClickListener для списка команд
AdapterView.OnItemClickListener itemClickListener =
    new AdapterView.OnItemClickListener() {
        public void onItemClick(AdapterView<?> listView,
                                View v,
                                int position,
                                long id) {
            if (position == 0) {
                Intent intent = new Intent(TopLevelActivity.this,
                                           DrinkCategoryActivity.class);
                startActivity(intent);
            }
        }
    };
```



В исходной версии этот код содержался в методе onCreate(). Он заполняет списковое представление и обеспечивает обработку выбора команд. Этот код нам еще пригодится.

```
//Добавление слушателя в список команд
ListView listView = (ListView) findViewById(R.id.list_options);
listView.setOnItemClickListener(itemClickListener);
```

//Заполнение спискового представления list_favorites данными курсора

```
ListView listFavorites = (ListView) findViewById(R.id.list_favorites);
try {
    SQLiteOpenHelper starbuzzDatabaseHelper = new StarbuzzDatabaseHelper(this);
    db = starbuzzDatabaseHelper.getReadableDatabase();
    favoritesCursor = db.query("DRINK",
        new String[] { "_id", "NAME",
            "FAVORITE = 1",
            null, null, null, null});
```

Создать курсор, содержащий значения столбцов _id и NAME для записей, у которых FAVORITE=1.

Получить названия любимых напитков пользователя.

Получить списковое представление любимых напитков.

Продолжение на следующей странице.

Код TopLevelActivity (продолжение)



DrinkActivity
DrinkCategoryActivity
Любимые напитки

```

CursorAdapter favoriteAdapter =
    new SimpleCursorAdapter(TopLevelActivity.this,
        android.R.layout.simple_list_item_1,
        favoritesCursor,
        new String[]{"NAME"},
        new int[]{android.R.id.text1}, 0);
listFavorites.setAdapter(favoriteAdapter);
} catch(SQLiteException e) {
    Toast toast = Toast.makeText(this, "Database unavailable", Toast.LENGTH_SHORT);
    toast.show();
}

//Переход к DrinkActivity при выборе напитка
listFavorites.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> listView, View v, int position, long id)
    {
        Intent intent = new Intent(TopLevelActivity.this, DrinkActivity.class);
        intent.putExtra(DrinkActivity.EXTRA_DRINKNO, (int)id);
        startActivity(intent);
    }
});

//Закрытие курсора и базы данных в методе onDestroy()
@Override
public void onDestroy(){
    super.onDestroy();
    favoritesCursor.close();
    db.close();
}
}
    
```

Курсор используется в адаптере курсора.

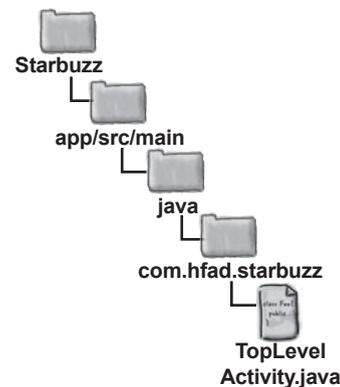
Вывести названия напитков в ListView.

Если с базой данных возникнут проблемы, вывести сообщение.

Метод вызывается при выборе варианта спискового представления.

Если пользователь выбирает один из вариантов спискового представления любимых напитков, создать интент для запуска DrinkActivity и передать идентификатор напитка.

Закрыть курсор и базу данных при уничтожении активности.



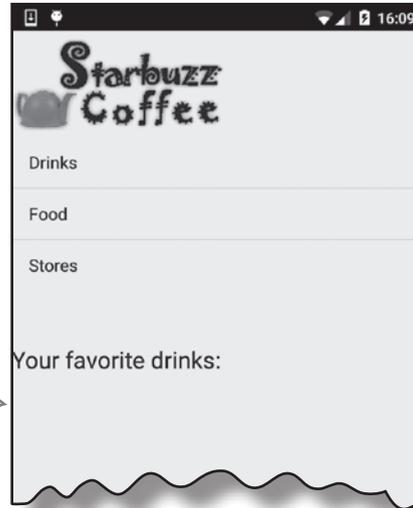
Этот код заполняет списковое представление данными любимых напитков пользователя. Когда пользователь выбирает один из этих напитков, интент запускает DrinkActivity и передает ей идентификатор напитка, и на экране появляется подробная информация об этом напитке. На следующей странице показано, как выглядит работающее приложение, — а также описана одна проблема, которую необходимо решить.



Тест-драйв

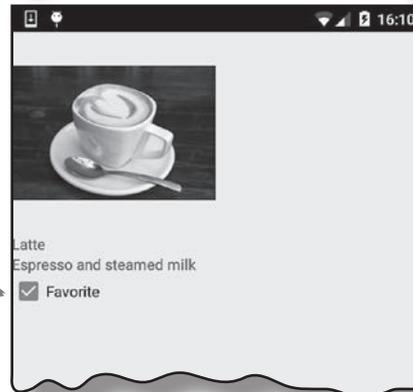
Когда вы открываете приложение, в `TopLevelActivity` отображается новое текстовое представление и новый список любимых напитков — как и было задумано. Список пока пуст, потому что пользователь еще не выбрал ни одного любимого напитка.

Списковое представление для любимых напитков находится на своем месте. Мы не видим его, потому что пока в нем нет ни одного напитка.



Если перейти к `DrinkActivity`, на экране виден новый флажок. Если щелкнуть на нем, напиток помечается как любимый.

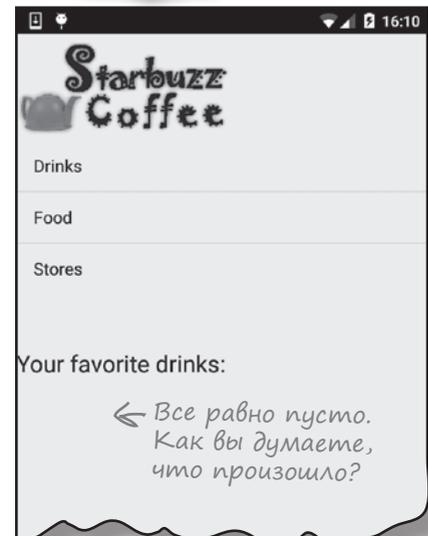
Флажок, который мы добавили. Щелчок на нем приводит к обновлению базы данных Starbuzz.

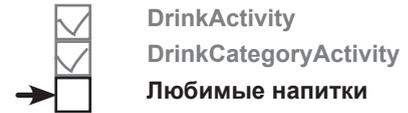


Если вернуться к `TopLevelActivity`, напиток, выбранный как любимый, не отображается в списке. Он появится только в том случае, если повернуть устройство.



Как вы думаете, почему напиток, помеченный как любимый, не появляется в списке до поворота экрана? Подумайте над этим, прежде чем переворачивать страницу.





Курсоры не обновляются автоматически

Если пользователь выбирает новый любимый напиток, переходя к DrinkActivity, новый любимый напиток не появляется автоматически в списке из TopLevelActivity. Дело в том, что *курсоры получают данные при создании*. В нашем случае курсор создается в методе onCreate () активности, поэтому он получает данные при создании активности. Когда пользователь переходит к другим активностям, активность TopLevelActivity останавливается, но не уничтожается и не создается повторно.



Если запустить вторую активность, она выводится поверх первой. Первая активность при этом не уничтожается. Вместо этого она сначала приостанавливается, а затем останавливается, когда теряет фокус и перестает быть видимой для пользователя.

Курсоры не отслеживают изменения информации в базе данных. Если информация изменится после создания курсора, то курсор обновлен не будет. Он по-прежнему содержит исходные записи без каких-либо изменений.

Если обновить информацию в базе данных...

_id	NAME	DESCRIPTION	IMAGE_RESOURCE_ID	FAVORITE
1	"Latte"	"Espresso and steamed milk"	54543543	1
2	"Cappuccino"	"Espresso, hot milk and steamed-milk foam"	654334453	0
3	"Filter"	"Our best drip coffee"	44324234	0

...курсор не увидит эти изменения, если он уже был создан.

Как же решить эту проблему?

Изменение курсора методом `changeCursor()`

Проблема решается заменой курсора, используемого списковым представлением, новой версией — это происходит при возвращении пользователя к `TopLevelActivity`. Если делать это в методе `onRestart()` активности, то данные в списковом представлении будут обновляться при возвращении пользователя к `TopLevelActivity`. Все новые любимые напитки, выбранные пользователем, появятся в списке, а все напитки, с которых пометка была снята, из списка исчезнут.

В этом нам поможет метод `changeCursor()` класса `CursorAdapter`. Метод `changeCursor()` заменяет курсор, в настоящее время используемый адаптером, новым курсором и закрывает старый курсор. Это выглядит так:

```
public void changeCursor(Cursor newCursor)
```

← Новый курсор, который должен использоваться адаптером курсора.

Метод `changeCursor()` получает один параметр — новый курсор.

Пример использования этого метода:

```
//Создание нового курсора
StarbuzzDatabaseHelper starbuzzDatabaseHelper = new StarbuzzDatabaseHelper(this);
SQLiteDatabase db = starbuzzDatabaseHelper.getReadableDatabase();
Cursor cursor = db.query("DRINK",
    new String[] { "_id", "NAME"},
    "FAVORITE = 1",
    null, null, null, null);
```

← Новый курсор создается точно так же, как это делалось прежде.

```
//Получение объекта CursorAdapter, используемого ListView
ListView listFavorites = (ListView) findViewById(R.id.list_favorites);
CursorAdapter adapter = (CursorAdapter) listFavorites.getAdapter();
```

← Для получения адаптера ListView используется метод `getAdapter()`.

```
//Замена курсора, используемого CursorAdapter, только что созданным
adapter.changeCursor(cursor);
```

← Курсор, используемый адаптером курсора, заменяется новым.

Переработанный код `TopLevelActivity.java` приводится на нескольких следующих страницах.

Обновленный код TopLevelActivity.java

Ниже приведен полный код `TopLevelActivity.java` (изменения выделены жирным шрифтом):

```
package com.hfad.starbuzz;
```

```
...
```

```
public class TopLevelActivity extends Activity {
```

```
...
```

```
@Override
```

```
protected void onCreate(Bundle savedInstanceState) {
```

```
...
```

```
}
```

Эти методы не изменились.

```
//Закрытие курсора и базы данных в методе onDestroy()
```

```
@Override
```

```
public void onDestroy(){
```

```
...
```

```
}
```

Метод вызывается при возвращении пользователя к TopLevelActivity.

```
public void onRestart() {
```

```
super.onRestart();
```

```
try{
```

```
StarbuzzDatabaseHelper starbuzzDatabaseHelper = new StarbuzzDatabaseHelper(this);
```

```
db = starbuzzDatabaseHelper.getReadableDatabase();
```

```
Cursor newCursor = db.query("DRINK", ← Новый курсор создается точно так же, как это делалось прежде.
```

```
new String[] { "_id", "NAME"},
```

```
"FAVORITE = 1",
```

```
null, null, null, null);
```

```
ListView listFavorites = (ListView) findViewById(R.id.list_favorites);
```

```
CursorAdapter adapter = (CursorAdapter) listFavorites.getAdapter(); ←
```

```
adapter.setCursor(newCursor); ← Заменить курсор, используемый адаптером курсора, новым.
```

```
favoritesCursor = newCursor;
```

Получить адаптер спискового представления.

```
} catch (SQLException e) {
```

```
Toast toast = Toast.makeText(this, "Database unavailable", Toast.LENGTH_SHORT);
```

```
toast.show();
```

Если с базой данных возникли проблемы, вывести сообщение.

```
}
```

```
}
```

```
}
```

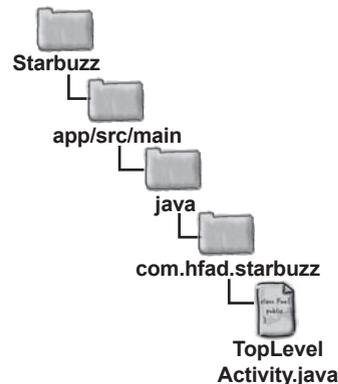
Вот и весь код, который нужно добавить в активность верхнего уровня. Проверим его в деле и посмотрим, как он работает.



DrinkActivity

DrinkCategoryActivity

Любимые напитки





Тест-драйв

На этот раз напиток, помеченный как любимый, появляется в `TopLevelActivity`. Если щелкнуть на нем, в приложении выводится подробная информация об этом напитке.



Я вот что думаю... Конечно, базы данных в приложениях обладают многими преимуществами, но разве открытие базы данных и чтение из нее не замедляет работу приложения?

Базы данных обладают широкими возможностями, но порой работают медленно.

А это означает, что даже если приложение успешно работает, не стоит забывать о быстродействии...

С базами данных приложение порой еле-еле движется...

Задумайтесь, что должно проделать приложение при открытии базы данных. Сначала оно должно пройтись по флэш-памяти и найти файл базы данных. Если файл базы данных не найден, то нужно создать пустую базу данных. Затем приложение должно выполнить все команды SQL для создания таблиц в базе данных и всех исходных данных, которые ему нужны. Наконец, приложение должно выдать запросы для извлечения данных из базы.

Все это требует времени. Для крошечной базы данных вроде той, что используется в приложении Starbuzz, затраты будут небольшими. Но с увеличением базы данных время также неуклонно возрастает. Не успеешь и глазом моргнуть, как приложение теряет весь задор и начинает работать медленнее, чем YouTube в Рождество. Со скоростью создания базы данных и чтения из нее в общем-то ничего не поделаешь, но предотвратить замедление работы интерфейса определенно *возможно*.

Совместная работа потоков

Основная проблема с обращениями к медленной базе данных заключается в том, что она может замедлить реакцию приложения на действия пользователя. Чтобы понять, почему это происходит, необходимо задуматься над тем, как работают программные потоки в Android. Начиная с версии Lollipop, существуют три вида потоков, которые необходимо учитывать:



Основной поток событий

Этот поток — настоящая «рабочая лошадка» Android: он прослушивает интенты, получает сообщения о касаниях от экрана и вызывает все методы внутри ваших активностей.



Поток визуализации

Этот поток, с которым вы обычно не взаимодействуете, читает список запросов на обновление экрана, а затем выдает команды низкоуровневому графическому оборудованию на перерисовку экрана. Благодаря ему ваше приложение обретает свою красоту.



Все остальные потоки, созданные вами

Если не принять специальных мер, приложение выполняет почти всю свою работу в основном потоке событий. Почему? Потому что основной поток событий выполняет методы событий приложения. Если просто включить код базы данных в метод `onCreate()` (как это было сделано в приложении Starbuzz), то основной поток событий будет занят работой с базой данных вместо того, чтобы заниматься обработкой событий от экрана или других приложений. Если выполнение кода базы данных занимает много времени, у пользователя может возникнуть ощущение, что приложение забыло о его существовании.

Итак, фокус заключается в том, чтобы **вынести код базы данных из основного потока событий и выполнить его в отдельном потоке в фоновом режиме**.

Возьми в руку карандаш



Мы хотим, чтобы код работы с базой данных из активности `DrinkActivity` был вынесен в фоновый поток. Но прежде чем поспешно хвататься за написание кода, стоит немного подумать над тем, что же нужно сделать.

Код, имеющийся в настоящее время, решает три разные задачи. Как вы думаете, в каком потоке должен выполняться каждый блок кода? Выберите тот тип потока, который вам кажется наиболее подходящим для каждого случая.

А Подготовка интерфейса.

```
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_drink);
int drinkNo = (Integer) getIntent().getExtras().get(EXTRA_DRINKNO);
```

Основной поток событий	Фоновый поток

← Выберите, в потоке какого типа, по вашему мнению, должен выполняться каждый блок кода: в основном потоке событий или в фоновом потоке.

В Взаимодействие с базой данных.

```
SQLiteOpenHelper starbuzzDatabaseHelper = new StarbuzzDatabaseHelper(this);
SQLiteDatabase db = starbuzzDatabaseHelper.getReadableDatabase();
Cursor cursor = db.query("Drink", ...
```

Основной поток событий	Фоновый поток

С Обновление представлений информацией из базы данных.

```
name.setText(...);
description.setText(...);
photo.setImageResource(...);
```

Основной поток событий	Фоновый поток



Возьми в руку карандаш Решение

Мы хотим, чтобы код работы с базой данных из активности `DrinkActivity` был вынесен в фоновый поток. Но прежде чем поспешно хвататься за написание кода, стоит немного подумать над тем, что же нужно сделать.

Код, имеющийся в настоящее время, решает три разные задачи. Как вы думаете, в каком потоке должен выполняться каждый блок кода? Выберите тот тип потока, который вам кажется наиболее подходящим для каждого случая.

A Подготовка интерфейса.

```
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_drink);
int drinkNo = (Integer) getIntent().getExtras().get(EXTRA_DRINKNO);
```

Основной поток событий	Фоновый поток
✓	

↑
Пользовательский интерфейс всегда создается в основном потоке.

B Взаимодействие с базой данных.

```
SQLiteOpenHelper starbuzzDatabaseHelper = new StarbuzzDatabaseHelper(this);
SQLiteDatabase db = starbuzzDatabaseHelper.getReadableDatabase();
Cursor cursor = db.query("Drink", ...
```

Основной поток событий	Фоновый поток
	✓

↑
Код базы данных должен выполняться в фоновом потоке, потому что он работает медленно.

C Обновление представлений информацией из базы данных.

```
name.setText(...);
description.setText(...);
photo.setImageResource(...);
```

← Код обновления представлений обязательно выполняется в главном потоке, иначе произойдет исключение.

Основной поток событий	Фоновый поток
✓	

Какой код для какого потока?

Если приложение работает с базой данных, этот код полезно вынести в фоновый поток, а обновлять представления данными из базы в основном потоке событий. Мы проанализируем код метода `onFavoritesClicked()` из кода `DrinkActivity`, чтобы вы поняли, как подходить к решению проблем такого рода.

Ниже приведен код метода (он состоит из нескольких частей, которые кратко описаны ниже):

```
//Обновление базы данных по щелчку на флажке
public void onFavoriteClicked(View view) {
```

1

```
    int drinkNo = (Integer) getIntent().getExtras().get(EXTRA_DRINKNO);
    CheckBox favorite = (CheckBox) findViewById(R.id.favorite);
    ContentValues drinkValues = new ContentValues();
    drinkValues.put("FAVORITE", favorite.isChecked());
```

2

```
    SQLiteOpenHelper starbuzzDatabaseHelper =
        new StarbuzzDatabaseHelper(DrinkActivity.this);
    try {
        SQLiteDatabase db = starbuzzDatabaseHelper.getWritableDatabase();
        db.update("DRINK", drinkValues,
            "_id = ?", new String[] {Integer.toString(drinkNo)});
        db.close();
    } catch (SQLException e) {
```

3

```
        Toast toast = Toast.makeText(this, "Database unavailable", Toast.LENGTH_SHORT);
        toast.show();
    }
}
```

1

Код, выполняемый до кода базы данных

В нескольких начальных строках метод получает текущее значение флажка и сохраняет его в объекте `ContentValues` с именем `drinkValues`. Выполнение этого кода должно предшествовать выполнению кода базы данных.

2

Код базы данных, который должен выполняться в фоновом потоке

Обновление таблицы `DRINK`.

3

Код, выполняемый после кода базы данных

Если база данных недоступна, следует вывести сообщение для пользователя. Этот код должен выполняться в основном потоке событий.

В реализации будет использоваться объект `AsyncTask`. Что же он собой представляет?

Класс AsyncTask выполняет асинхронные задачи

Класс AsyncTask предназначен для выполнения задач в фоновом режиме. Когда операции завершатся, он также позволяет обновлять представления в основном потоке событий. Если задача представляет собой серию повторяющихся операций, класс даже может использоваться для публикации информации о прогрессе во время выполнения задачи.

Чтобы создать свою реализацию AsyncTask, вы расширяете класс AsyncTask и реализуете метод doInBackground(). Код этого метода выполняется в фоновом потоке, поэтому этот метод идеально подходит для размещения кода базы данных. Класс AsyncTask также содержит метод onPreExecute(), который выполняется до doInBackground(), и метод onPostExecute(), который выполняется после. Также имеется метод onProgressUpdate() на тот случай, если вы захотите передавать информацию о ходе выполнения задачи. Все это выглядит примерно так:

```
private class MyAsyncTask extends AsyncTask<Params, Progress, Result>

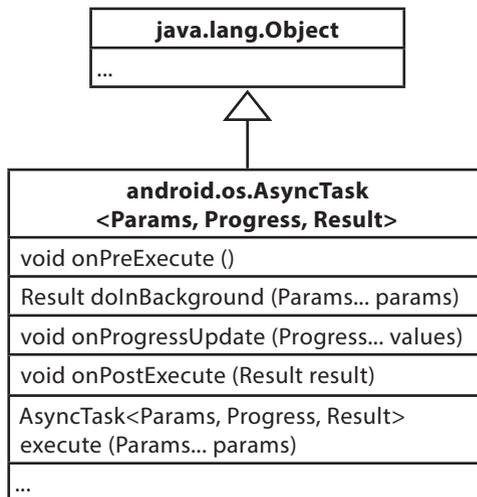
    protected void onPreExecute() {
        //Код, предшествующий выполнению задачи
    }

    protected Result doInBackground(Params... params) {
        //Код, выполняемый в фоновом потоке
    }

    protected void onProgressUpdate(Progress... values) {
        //Код, передающий информацию о ходе выполнения задачи
    }

    protected void onPostExecute(Result result) {
        //Код, выполняемый при завершении задачи
    }
}
```

AsyncTask определяется тремя обобщенными параметрами: Params, Progress и Results. Params — тип объекта, используемого для передачи произвольных параметров задачи методу doInBackground(), Progress — тип объекта, используемый для передачи информации о прогрессе задачи, и Result — тип результата задачи. Если любые из этих параметров не используются, в них можно передать Void. Сейчас мы создадим специализацию AsyncTask с именем UpdateDrinkTask, которая будет использоваться для фонового обновления информации о напитках. Позднее этот код будет добавлен в DrinkActivity.



Метод `onPreExecute()`



Начнем с метода `onPreExecute()`. Этот метод вызывается до начала фоновой задачи и используется для подготовки ее выполнения. Метод вызывается в основном потоке событий, поэтому для него доступны все представления в пользовательском интерфейсе. Метод `onPreExecute()` вызывается без параметров и возвращает `void`.

Мы используем метод `onPreExecute()` для получения значения флажка любимого напитка и включения его в объект `ContentValues` с именем `drinkValues`. Дело в том, что для выполнения этой операции нужен доступ к флажку, а сама операция должна быть выполнена до выполнения какого-либо кода базы данных. Для хранения объекта `ContentValues` с именем `drinkValues` используется внешний атрибут, чтобы к нему могли обращаться другие методы класса.

Код выглядит так:

```
private class UpdateDrinkTask extends AsyncTask<Params, Progress, Result> {

    ContentValues drinkValues;

    protected void onPreExecute() {
        CheckBox favorite = (CheckBox) findViewById(R.id.favorite);
        drinkValues = new ContentValues();
        drinkValues.put("FAVORITE", favorite.isChecked());
    }

    ...
}
```

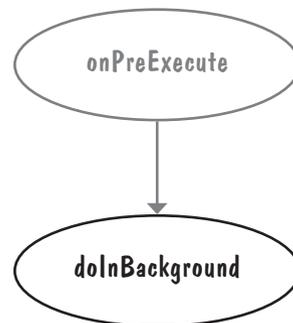
← Перед выполнением кода базы данных необходимо получить значение флажка любимого напитка.

Теперь рассмотрим метод `doInBackground()`.

Метод doInBackground()

Метод `doInBackground()` запускается в фоновом режиме сразу же после выполнения `onPreExecute()`. Вы определяете тип параметров, которые должны передаваться задаче, и тип возвращаемого значения.

В нашем приложении метод `doInBackground()` будет использоваться для кода работы с базой данных, чтобы он выполнялся в фоновом потоке. Метод получает идентификатор напитка, информацию о котором требуется обновить, а логическое возвращаемое значение позволит проверить, успешно ли была выполнена задача:



```
private class UpdateDrinkTask extends AsyncTask<Integer, Progress, Boolean> {
```

```
    ContentValues drinkValues;
```

↑
Заменяется на Integer в соответствии с параметром метода `doInBackground()`.

↑
Заменяется на Boolean в соответствии с возвращаемым типом метода `doInBackground()`.

```
    ...
```

↓
Этот код выполняется в фоновом потоке.

```
    protected Boolean doInBackground(Integer... drinks) {
```

```
        int drinkNo = drinks[0];
```

```
        SQLiteOpenHelper starbuzzDatabaseHelper =
```

```
            new StarbuzzDatabaseHelper(DrinkActivity.this);
```

```
        try {
```

```
            SQLiteDatabase db = starbuzzDatabaseHelper.getWritableDatabase();
```

```
            db.update("DRINK", drinkValues,
```

```
                "_id = ?", new String[] {Integer.toString(drinkNo)});
```

```
            db.close();
```

```
            return true;
```

```
        } catch (SQLiteException e) {
```

```
            return false;
```

```
        }
```

```
    }
```

```
    ...
```

```
}
```

←
Это массив целых чисел, но мы включаем всего один элемент — идентификатор напитка.

←
Метод `update()` использует объект `drinkValues`, созданный методом `onPreExecute()`.

Теперь перейдем к рассмотрению метода `onProgressUpdate()`.

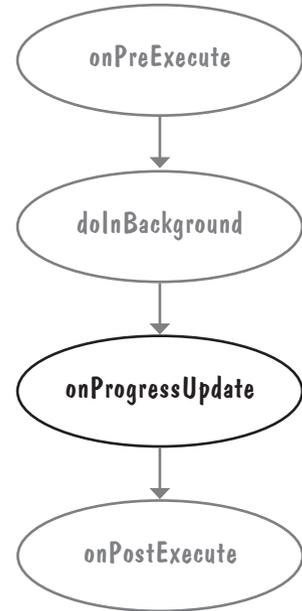
Метод `onProgressUpdate()`

Метод `onProgressUpdate()` вызывается в основном потоке событий, поэтому в нем доступны представления пользовательского интерфейса. Метод может использоваться для вывода сведений о ходе выполнения операции. Вы сами определяете тип параметров, которые должны передаваться методу. Метод `onProgressUpdate()` выполняется при вызове `publishProgress()` из метода `doInBackground()`:

```
protected Boolean doInBackground(Integer... count) {
    for (int i = 0; i < count; i++) {
        publishProgress(i);
    }
}

protected void onProgressUpdate(Integer... progress) {
    setProgress(progress[0]);
}
```

← Приводит к вызову метода `onProgressUpdate()` с передачей значения `i`.



В нашем примере информация о ходе выполнения задачи не публикуется, поэтому реализовать этот метод не нужно. Чтобы показать, что объекты для этой цели не используются, мы изменяем сигнатуру `UpdateDrinkTask`:

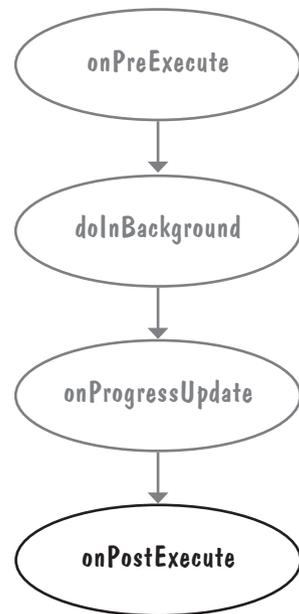
В нашем примере метод `onProgressUpdate()` не используется, поэтому передается `Void`.

```
private class UpdateDrinkTask extends AsyncTask<Integer, Void, Boolean> {
    ...
}
```

Остается рассмотреть метод `onPostExecute()`.

Мемог onPostExecute()

Метод `onPostExecute()` вызывается после завершения фоновой задачи. Он вызывается в основном потоке событий, а следовательно, для него доступны все представления в пользовательском интерфейсе. Метод может использоваться для отображения результатов задачи для пользователя. Методу `onPostExecute()` передаются результаты метода `doInBackground()`, поэтому его параметры должны соответствовать возвращаемому типу `doInBackground()`. Мы будем использовать метод `onPostExecute()` для проверки того, успешно ли был выполнен код базы данных в методе `doInBackground()`. Если при выполнении произошла ошибка, приложение выводит сообщение для пользователя. Это происходит в методе `onPostExecute()`, так как этот метод может обновлять пользовательский интерфейс; метод `doInBackground()` выполняется в фоновом потоке, и он обновлять представления не может. Код выглядит так:



```
private class UpdateDrinkTask extends AsyncTask<Integer, Void, Boolean> {
```

```
...
```

```
protected void onPostExecute(Boolean success) {
```

```
    if (!success) {
```

```
        Toast toast = Toast.makeText(DrinkActivity.this,
```

```
            "Database unavailable", Toast.LENGTH_SHORT);
```

```
        toast.show();
```

```
    }
```

```
}
```

```
}
```

Тип Boolean, так как наш метод `doInBackground()` возвращает Boolean.

Передать Toast контекст `DrinkActivity`.

Класс AsyncTask

При первом знакомстве с классом `AsyncTask` мы упомянули о том, что он определяется тремя обобщенными параметрами: `Params`, `Progress` и `Results`. Они определяются типами параметров, используемых методами `doInBackground()`, `onProgressUpdate()` и `onPostExecute()`. `Params` — тип параметров `doInBackground()`, `Progress` — тип параметров `onProgressUpdate()`, а `Result` — тип параметра `onPostExecute()`:

```
private class MyAsyncTask extends AsyncTask<Params, Progress, Result>

protected void onPreExecute() {
    //Код, предшествующий выполнению задачи
}

protected Result doInBackground(Params... params) {
    //Код, выполняемый в фоновом потоке
}

protected void onProgressUpdate(Progress... values) {
    //Код, передающий информацию о ходе выполнения задачи
}

protected void onPostExecute(Result result) {
    //Код, выполняемый при завершении задачи
}
}
```

В нашем примере `doInBackground()` получает параметры типа `Integer`, а `onPostExecute()` получает параметр `Boolean`. Метод `onProgressUpdate()` не используется. Это означает, что в нашем примере на место обобщенных параметров `Params`, `Progress` и `Result` подставляются `Integer`, `Void` и `Boolean` соответственно:

```
private class UpdateDrinkTask extends AsyncTask<Integer, Void, Boolean> {
    ...
    protected Boolean doInBackground(Integer... drinks) {
        ...
    }

    protected void onPostExecute(Boolean... success) {
        ...
    }
}
```

Теперь вы знаете все, что необходимо знать для создания задачи. Давайте посмотрим, как запустить созданную задачу.

Выполнение задачи

Чтобы запустить задачу на выполнение, вызовите метод `execute()` объекта `AsyncTask`. Если метод `doInBackground()` получает параметры, они добавляются в метод `execute()`. Например, в нашем приложении методу `doInBackground()` задачи `AsyncTask` должен передаваться напиток, выбранный пользователем, поэтому вызов выглядит так:

```
int drinkNo = (Integer) getIntent().getExtras().get(EXTRA_DRINKNO);
new UpdateDrinkTask().execute(drinkNo);
```

Тип параметра, передаваемого методу `execute()`, должен соответствовать типу параметра, который ожидает получить метод `doInBackground()` объекта `AsyncTask`. Наш метод `doInBackground()` получает параметры типа `Integer`, поэтому передавать нужно целые числа:

```
protected Boolean doInBackground(Integer... drinks) {
    ...
}
```

Задача `UpdateDrinkTask` будет запускаться из метода `onFavoritesClicked()` объекта `DrinkActivity`. Метод выглядит так:

```
//Обновление базы данных по щелчку на флажке
public void onFavoriteClicked(View view) {
    int drinkNo = (Integer) getIntent().getExtras().get(EXTRA_DRINKNO);
    CheckBox favorite = (CheckBox) findViewById(R.id.favorite);
    ContentValues drinkValues = new ContentValues();
    drinkValues.put("FAVORITE", favorite.isChecked());
    SQLiteOpenHelper starbuzzDatabaseHelper =
        new StarbuzzDatabaseHelper(DrinkActivity.this);
    try {
        SQLiteDatabase db = starbuzzDatabaseHelper.getWritableDatabase();
        db.update("DRINK", drinkValues, "_id = ?", new String[] {Integer.toString(drinkNo)});
        db.close();
    } catch (SQLException e) {
        Toast toast = Toast.makeText(this, "Database unavailable", Toast.LENGTH_SHORT);
        toast.show();
    }
    new UpdateDrinkTask().execute(drinkNo);
}
```

Весь этот код заменен кодом `AsyncTask`.

Выполнить задачу `AsyncTask` и передать ей идентификатор напитка.

Новая версия кода `DrinkActivity.java` приведена на следующей странице.

Код DrinkActivity.java

Задача `AsyncTask` добавляется как внутренний класс в активность, которая должна ее использовать. Мы добавим свой класс `UpdateDrinkTask` как внутренний класс `DrinkActivity.java`. Задача будет выполняться в методе `onFavoriteClicked()` класса `DrinkActivity`, чтобы база данных обновлялась в фоновом режиме, когда пользователь щелкает на флажке любимого напитка. Код выглядит так:

```
package com.hfad.starbuzz;
```

```
...
```

```
import android.os.AsyncTask; ← Импортировать класс AsyncTask.
```

```
public class DrinkActivity extends Activity {
```

```
... ← Метод onCreate() не изменился, поэтому он здесь не приводится.
```

```
//Обновление базы данных по щелчку на флажке
```

```
public void onFavoriteClicked(View view) {
```

```
    int drinkNo = (Integer) getIntent().getExtras().get(EXTRA_DRINKNO);
```

```
    new UpdateDrinkTask().execute(drinkNo); ← Выполнить задачу.
```

```
}
```

```
//Внутренний класс для обновления напитка. ←
```

AsyncTask добавляется в активность в виде внутреннего класса.

```
private class UpdateDrinkTask extends AsyncTask<Integer, Void, Boolean> {
```

```
    ContentValues drinkValues;
```

```
protected void onPreExecute() {
```

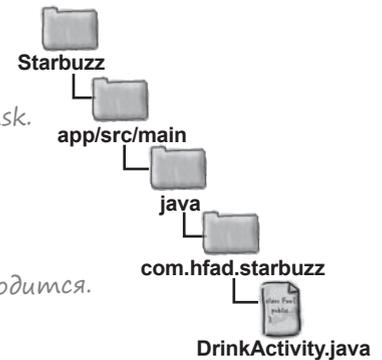
Прежде чем запускать код базы данных на выполнение, добавим значение флажка в объект ContentValues с именем drinkValues.

```
    CheckBox favorite = (CheckBox) findViewById(R.id.favorite);
```

```
    drinkValues = new ContentValues();
```

```
    drinkValues.put("FAVORITE", favorite.isChecked());
```

```
}
```



Продолжение на следующей странице. →

Код DrinkActivity.java (продолжение)

```

protected Boolean doInBackground(Integer... drinks) {
    int drinkNo = drinks[0];
    SQLiteOpenHelper starbuzzDatabaseHelper =
        new StarbuzzDatabaseHelper(DrinkActivity.this);

    try {
        SQLiteDatabase db = starbuzzDatabaseHelper.getWritableDatabase();
        db.update("DRINK", drinkValues,
            "_id = ?", new String[] {Integer.toString(drinkNo)});

        db.close();
        return true;
    } catch (SQLException e) {
        return false;
    }
}

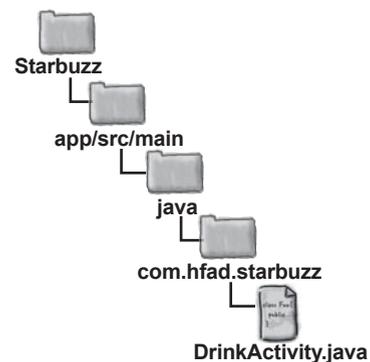
protected void onPostExecute(Boolean success) {
    if (!success) {
        Toast toast = Toast.makeText(DrinkActivity.this,
            "Database unavailable", Toast.LENGTH_SHORT);
        toast.show();
    }
}
}

```

← Код базы данных запускается на выполнение в фоновом потоке.

↑ Обновить значение столбца FAVORITE.

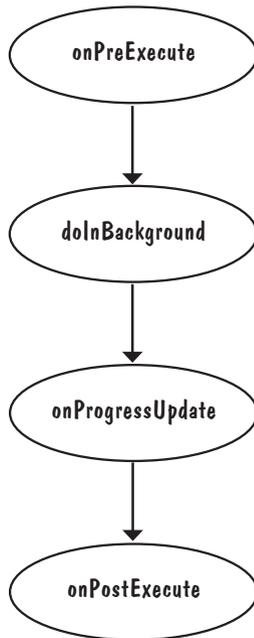
↑ Если при выполнении кода базы данных возникли проблемы, вывести сообщение для пользователя.



Вот и все, что нужно знать для создания AsyncTask. Когда пользователь щелкает на флажке любимого напитка в DrinkActivity, информация в базе данных обновляется в фоновом режиме.

В идеале весь код работы с базой данных должен выполняться в фоновом режиме. Мы не будем вносить изменения в другие активности Starbuzz, но почему бы вам не сделать это самостоятельно?

Схема работы с объектами AsyncTask



- 1 **Метод onPreExecute()** используется для подготовки задачи.
Он вызывается перед запуском фоновой задачи и выполняется в основном потоке событий.
- 2 **Метод doInBackground()** выполняется в фоновом потоке. Метод запускается сразу же после onPreExecute(). Вы можете указать типы его параметров и возвращаемого значения.
- 3 **Метод onProgressUpdate()** используется для вывода информации о прогрессе операции.
Метод выполняется в основном потоке событий, когда метод doInBackground() вызывает publishProgress().
- 4 **Метод onPostExecute()** используется для отображения результата операции при завершении doInBackground(). Метод выполняется в основном потоке событий. В его параметре передается возвращаемое значение doInBackground().

Часть Задаваемые Вопросы

В: Я уже писал приложения, в которых код работы базы данных просто выполнялся в основном потоке, и все было нормально. Так ли необходимо выполнять его в фоновом режиме?

О: С очень маленькими базами данных — как та, которая используется в приложении Starbuzz, — вы, вероятно, не заметите различий во времени работы с базой данных. Но это объясняется только малым размером базы данных. Если база данных достаточно велика или приложение работает на медленном устройстве, время обращения к базе данных будет значительным. Поэтому — да, код работы с базой данных *всегда* должен выполняться в фоновом режиме.

В: Напомните — почему нельзя обновлять представления из фонового потока?

О: В двух словах — такая попытка приведет к исключению. Если же говорить подробнее, многопоточные пользовательские интерфейсы обычно содержат множество ошибок. В Android эта проблема была решена простым запретом.

В: Какая часть кода баз данных работает медленнее? Открытие базы данных или чтение информации из нее?

О: В общем случае предсказать невозможно. Если в вашей базе данных используются сложные структуры данных, то первое открытие базы данных займет много времени, потому что при этом нужно будет создать все таблицы. Обработка сложных запросов тоже может занимать очень много времени. В общем случае следует избежать риска и выполнять все операции в фоновом режиме.

В: Если чтение информации из базы данных занимает несколько секунд, то что в это время видит пользователь?

О: Пользователь будет видеть пустые представления до тех пор, пока код базы данных не заполнит их.

В: Почему код работы с базой данных был вынесен в задачу AsyncTask только в одной активности?

О: Мы хотели продемонстрировать использование AsyncTasks в одной активности в качестве примера. В реальных приложениях это следует проделать с кодом базы данных во всех активностях.



Ваш инструментарий Android

Глава 12 осталась позади, а ваш инструментарий пополнился навыками работы с базами данных SQLite.

Весь код для этой главы можно загрузить по адресу <https://tinyurl.com/HeadFirstAndroid>.

КЛЮЧЕВЫЕ МОМЕНТЫ



- Курсоры используются для чтения и записи информации в базы данных.
- Курсор создается методом `query()` класса `SQLiteDatabase`. Во внутренней реализации при этом строится соответствующая команда SQL SELECT.
- Метод `getWritableDatabase()` возвращает объект `SQLiteDatabase`, который может использоваться для чтения и записи в базу данных.
- Метод `getReadableDatabase()` возвращает объект `SQLiteDatabase`, который предоставляет доступ к базе данных только для чтения. Он также может предоставить возможность выполнения операций чтения/записи, но это не гарантировано.
- Для перемещения по данным в курсоре используются методы `moveTo*()`.
- Для чтения данных из курсора используются методы `get*()`.
- Закрывайте курсоры и подключения к базе данных после завершения работы с ними.
- Класс `CursorAdapter` представляет адаптер для работы с курсорами. Используйте класс `SimpleCursorAdapter` для заполнения компонента `ListView` данными, возвращенными курсором.
- Проектируйте свои приложения так, чтобы в активности верхнего уровня размещался полезный контент.
- Метод `changeCursor()` класса `CursorAdapter` заменяет курсор, в настоящее время используемый адаптером, другим курсором, выбранным вами. Затем старый курсор закрывается.
- Выполняйте код работы с базами данных в фоновом потоке с использованием объектов `AsyncTask`.

К вашим услугам

Кстати, если что —
всегда можешь
рассчитывать на меня...



Существуют операции, которые должны выполняться постоянно. Например, если вы запустили воспроизведение музыкального файла в приложении-проигрывателе, вероятно, музыка не должна останавливаться при переключении на другое приложение. В этой главе вы узнаете, как использовать **службы** для подобных ситуаций, а заодно научитесь пользоваться некоторыми **встроенными службами Android**. **Служба уведомлений** поможет вам держать пользователей в курсе дел, а при помощи **службы позиционирования** пользователь сможет узнать, где он находится.

Службы работают незаметно для пользователя

Приложение Android состоит из активностей и других компонентов. Основная часть кода обеспечивает взаимодействие с пользователем, но иногда приложению приходится выполнять некоторые операции в фоновом режиме: например, загрузить большой файл, воспроизвести музыкальный файл или ожидать сообщения от сервера.

Активности для таких задач не приспособлены. В простых случаях можно создать вторичный поток, но при малейшей невнимательности код активности становится сложным и неудобочитаемым.

Для таких ситуаций были придуманы **службы**. Служба (service) представляет собой компонент приложения, похожий на активность, но не обладающий пользовательским интерфейсом. Службы имеют более простой жизненный цикл, чем активности, а их встроенная функциональность упрощает написание кода, выполняемого в фоновом режиме, пока пользователь занимается чем-то другим.

Два типа служб

Службы делятся на две разновидности:



Запускаемые службы

Запускаемые службы могут выполняться в фоновом режиме сколь угодно долго, даже после уничтожения запустившей их активности. Если вы хотите загрузить большой файл из Интернета, вероятно, эту операцию следует оформить в виде запускаемой службы. После завершения операции такая служба останавливается.



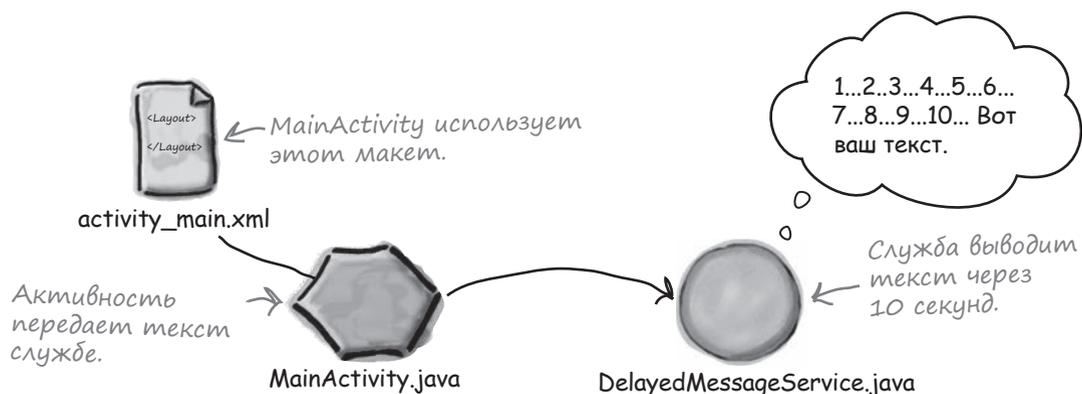
Связанные службы

Связанная служба привязывается к другому компоненту — например, активности. Активность может взаимодействовать со службой, отправлять ей запросы и получать результаты. Связанная служба работает, пока работают связанные с ней компоненты. Когда связь с компонентом прерывается, служба уничтожается. Например, если вы создаете одомер для измерения расстояния, пройденного машиной, вероятно, следует использовать связанную службу. В этом случае любые активности, связанные со службой, смогут обращаться к службе и запрашивать у нее пройденное расстояние.

В этой главе мы создадим две службы: запускаемую и связанную. Начнем с запускаемой службы.

Запускаемая служба

Наш следующий проект содержит активность с именем MainActivity и службу с именем DelayedMessageService. Каждый раз, когда MainActivity вызывает DelayedMessageService, служба ожидает 10 секунд, а затем выводит текст.



Работа будет проходить в три этапа:

- 1 Вывод сообщения в журнал.**
 Начнем с вывода сообщения в журнал, чтобы мы могли проверить работоспособность службы. Содержимое журнала можно просмотреть в Android Studio.
- 2 Вывод сообщения в уведомлении Toast.**
 Сообщение будет выводиться во временном уведомлении, чтобы проверка работоспособности приложения не требовала подключения к Android Studio.
- 3 Вывод сообщения через службу уведомлений.**
 Служба DelayedMessageService использует встроенную службу уведомлений Android для вывода сообщения. Это позволит пользователю просмотреть сообщение позднее.

Создание проекта

Начнем с создания проекта. Создайте новый проект Android для приложения с именем "Joke" и именем пакета `com.hfad.joke`. Минимальный уровень SDK должен быть равен API 16, чтобы приложение работало на большинстве устройств. Чтобы ваш код не отличался от нашего, присвойте пустой активности имя "MainActivity", а макету — имя "activity_main".

Затем необходимо создать службу.

Расширение класса IntentService

Новая служба создается расширением либо класса `Service`, либо класса `IntentService`.

Класс `Service` является базовым для всех служб. Он предоставляет основную функциональность служб; как правило, при создании связанных служб следует расширять именно этот класс.

Класс `IntentService` представляет собой subclass `Service`, предназначенный для работы с интентами. Обычно он расширяется для создания запускаемых служб.

Так как мы собираемся создать запускаемую службу, в проект следует добавить новую специализацию `IntentService`. Выполните команду `File→New...` и выберите вариант `Service`. Выберите создание новой специализации `IntentService`. Присвойте службе имя `DelayedMessageService`, снимите флажок включения вспомогательного метода запуска (так как мы собираемся заменить код, сгенерированный Android Studio).

Чтобы создать службу с поддержкой интентов, следует расширить класс `IntentService` и реализовать его метод `onHandleIntent()`. Этот метод должен содержать код, который должен выполняться при вызове службы:

```
package com.hfad.joke;

import android.app.IntentService;
import android.content.Intent;

public class DelayedMessageService extends IntentService {

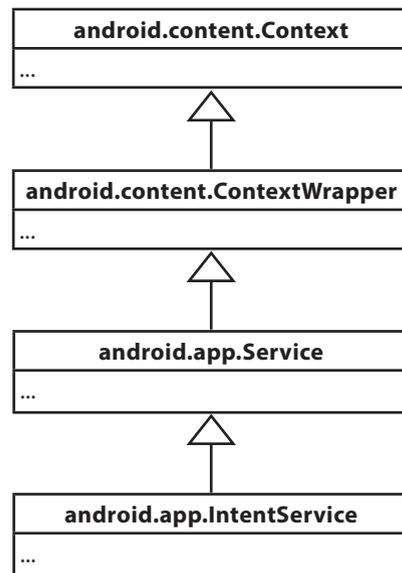
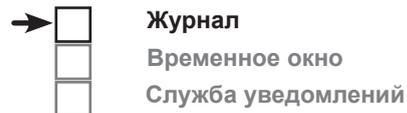
    public DelayedMessageService() {
        super("DelayedMessageService");
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        //...
    }
}
```

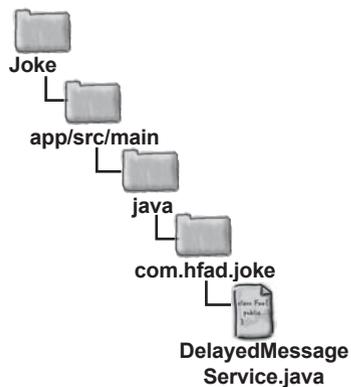
Расширим класс IntentService.

Поместите код, который должен выполняться службой, в метод onHandleIntent().

Общая схема работы запускаемых служб приведена на следующей странице.



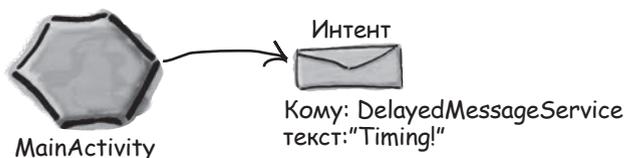
Иерархия классов служб.



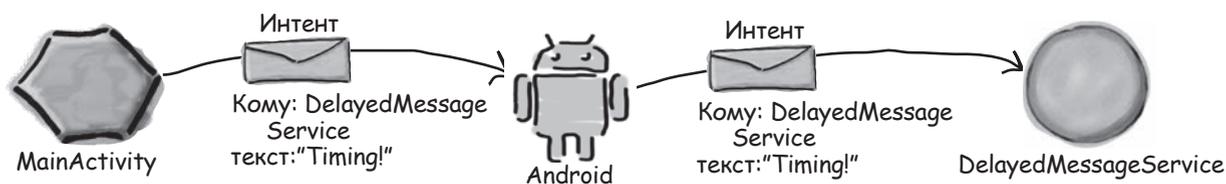
Класс IntentService — взгляд издалека

Мы собираемся использовать класс `IntentService` для создания запускаемой службы; давайте посмотрим, как он работает.

- 1 **Активность указывает, какая служба ей нужна, создавая явный интент.**
Интент определяет службу, для которой он предназначен.



- 2 **Интент передается службе.**



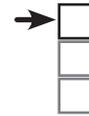
- 3 **Служба запускается и обрабатывает интент.**

Метод `onHandleIntent()` класса `IntentService` вызывается и выполняется в отдельном потоке. Если служба получает несколько интентов, она обрабатывает их последовательно, по одному за раз. После завершения своей работы служба останавливается.



Как видите, служба запускается тем же способом, которым запускаются активности: созданием интента. Отличие в том, что при запуске службы на экране ничего не изменяется, потому что у службы нет пользовательского интерфейса.

Служба `DelayedMessageService` должна вывести сообщение в журнале `Android`. Прежде чем браться за изменение кода, необходимо разобраться, как сохранять сообщения в журнале.



Запись сообщений в журнал

Запись сообщений в журнал часто помогает убедиться в том, что ваш код работает именно так, как нужно. Вы сообщаете Android, какую информацию нужно сохранить, в своем коде Java, а затем во время работы приложения просматриваете результаты в журнале Android.

Для сохранения сообщений в журнале используются следующие методы класса `Android.util.Log`:

<code>Log.v(String tag, String message)</code>	Сохраняет подробное сообщение.
<code>Log.d(String tag, String message)</code>	Сохраняет отладочное сообщение.
<code>Log.i(String tag, String message)</code>	Сохраняет информационное сообщение.
<code>Log.w(String tag, String message)</code>	Сохраняет предупреждение.
<code>Log.e(String tag, String message)</code>	Сохраняет сообщение об ошибке.

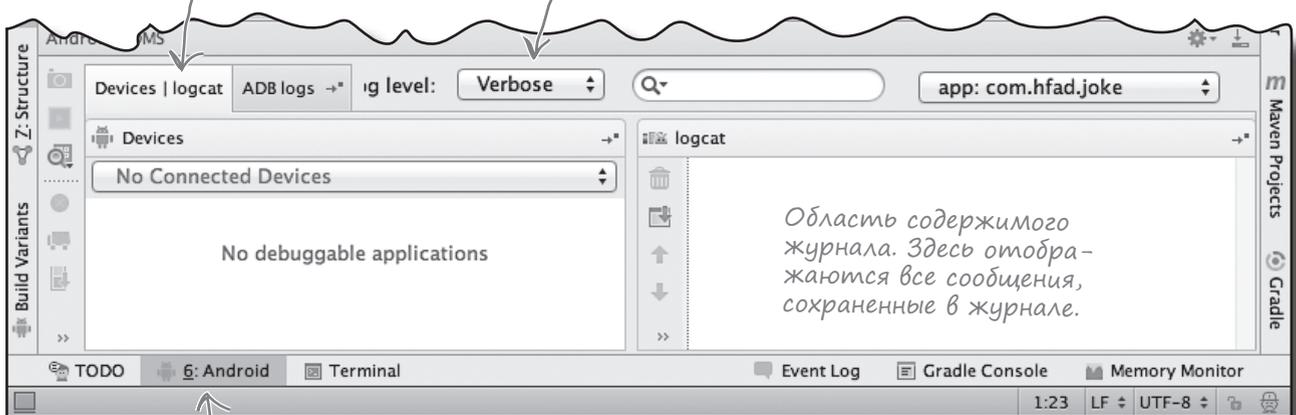
Также существует метод `Log.wtf()` для регистрации аварийных ситуаций, которых быть вообще не должно. Как уверяет документация Android, сокращение `wtf` означает "What a Terrible Failure", то есть «Какая ужасная ошибка».

Каждое сообщение состоит из строковой метки, которая может использоваться для идентификации источника сообщения, и самого сообщения. Например, для сохранения подробного сообщения, поступившего от службы `DelayedMessageService`, используется вызов `Log.v()` следующего вида:

```
Log.v("DelayedMessageService", "This is a message");
```

Android Studio предоставляет средства для просмотра журнала и фильтрации данных по типам сообщений. Чтобы просмотреть содержимое журнала, выберите режим Android в нижней части окна проекта в Android Studio, а затем перейдите на вкладку `Devices | logcat`:

Выберите вкладку `Devices | logcat`. Для фильтрации по типу сообщений.



Выберите режим Android.

Полный код DelayedMessageService

Наша служба должна получить текст из интента, подождать 10 секунд, а затем вывести текст в журнал. Мы создадим метод `showText()` для вывода текста в журнал, а затем вызовем его из метода `onHandleIntent()` после истечения задержки.

Ниже приведен полный код `DelayedMessageService.java` (замените им код, сгенерированный Android Studio):

```

package com.hfad.joke;

import android.app.IntentService;
import android.content.Intent;
import android.util.Log;

public class DelayedMessageService extends IntentService {

    public static final String EXTRA_MESSAGE = "message";

    public DelayedMessageService() {
        super("DelayedMessageService");
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        synchronized (this) {
            try {
                wait(10000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        String text = intent.getStringExtra(EXTRA_MESSAGE);
        showText(text);
    }

    private void showText(final String text) {
        Log.v("DelayedMessageService", "The message is: " + text);
    }
}

```

Расширить класс `IntentService`.

Использовать константу для передачи сообщения от активности службе.

Вызвать конструктор суперкласса.

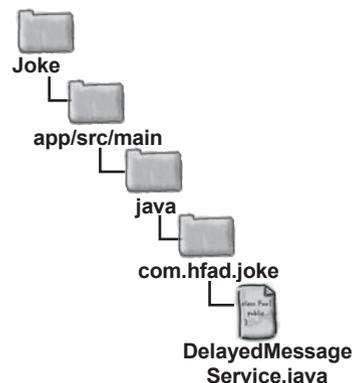
Метод содержит код, который должен выполняться при получении интента службой.

Подождать 10 секунд.

Получить текст из интента.

Вызвать метод `showText()`.

Вывести текст в журнал. В дальнейшем содержимое журнала можно просмотреть в Android Studio.





Объявление службы в AndroidManifest.xml

Службы, как и активности, должны объявляться в файле *AndroidManifest.xml* при помощи элемента `<service>`. Это необходимо для того, чтобы система Android могла вызвать службу; если служба не объявлена в *AndroidManifest.xml*, то Android ее вызвать не сможет.

Android Studio автоматически включает объявление в *AndroidManifest.xml* при создании службы. Вот как выглядит разметка:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.hfad.joke" >
    <application
        ... >

        <activity
            ...
        </activity>

        <service
            android:name=".DelayedMessageService"
            android:exported="false" >
        </service>
    </application>
</manifest>
```



Объявление службы в AndroidManifest.xml. Среда Android Studio должна создать его автоматически.

Имя службы начинается с префикса «.», чтобы его можно было объединить с именем пакета для получения полного имени класса.

Элемент `<service>` содержит два атрибута.

Атрибут `android:name` сообщает Android имя службы — в нашем примере `DelayedMessageService`.

Атрибут `android:exported` сообщает Android, должна ли служба использоваться другими приложениями. Если присвоить ему `false`, это означает, что служба будет использоваться только в текущем приложении.

Итак, мы успешно создали службу. Теперь можно переходить к следующему шагу — вызову этой службы из активности.

Добавление кнопки в activity_main.xml

В нашем приложении активность `MainActivity` будет запускать `DelayedMessageService` при нажатии кнопки. Начнем с добавления кнопки в макет `MainActivity`.

Добавьте в файл `strings.xml` следующие значения (они будут использоваться в коде активности и разметке макета):

```
<string name="button_response">Timing!</string>
<string name="button_text">What is the secret of comedy?</string>
```

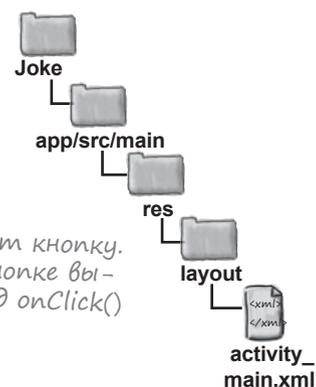
Эти строки используются в приложении.

Затем внесите изменения в `activity_main.xml`, чтобы добавить кнопку в `MainActivity`:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button_text"
        android:id="@+id/button"
        android:onClick="onClick"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true" />

</RelativeLayout>
```



Элемент создает кнопку. По щелчку на кнопке вызывается метод `onClick()` активности.



На следующей странице мы внесем изменения в код `MainActivity.java`, чтобы активность запускала службу.



Журнал
Временное окно
Служба уведомлений

Службы запускаются вызовом `startService()`

Запуск служб из активностей сильно напоминает запуск других активностей: нужно создать явный интент, предназначенный для запускаемой службы. После этого служба запускается вызовом метода `startService()`:

```
Intent intent = new Intent(this, DelayedMessageService.class);
startService(intent);
```

← Службы запускаются почти так же, как и активности, только вместо метода `startActivity()` используется метод `startService()`:

Этот метод будет использоваться в методе `onClick()` класса `MainActivity`, чтобы служба запускалась по щелчку на кнопке. Код выглядит так:

```
package com.hfad.joke;

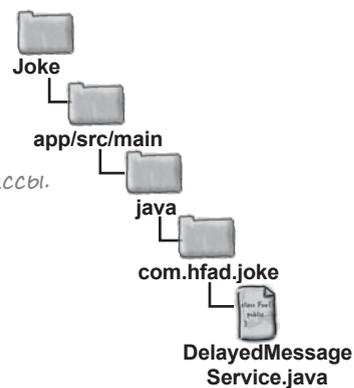
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public void onClick(View view) {
        Intent intent = new Intent(this, DelayedMessageService.class);
        intent.putExtra(DelayedMessageService.EXTRA_MESSAGE,
            getResources().getString(R.string.button_response));
        startService(intent);
    }
}
```

↔ Мы используем эти классы.



↙ Выполняется при щелчке на кнопке.

↙ Создать интент.

↑ Запустить службу.

↖ Добавить текст в интент.

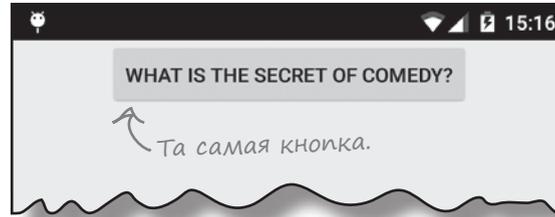
Вот и весь код, необходимый для запуска службы из активности. Посмотрим, что происходит при запуске приложения.



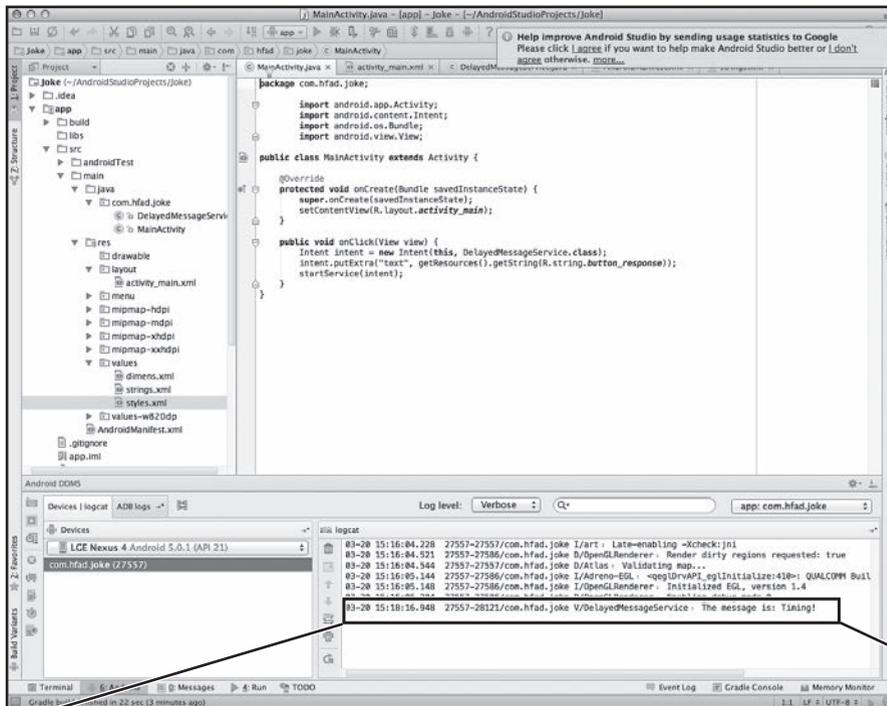
службы

Журнал
Временное окно
Служба уведомлений

При запуске приложения на экране появляется активность MainActivity. Она содержит одну кнопку:



Нажмите кнопку, переключитесь обратно в Android Studio и понаблюдайте за выводом в журнал в правом нижнем углу среды разработки. Через 10 секунд на панели журнала появится слово "Timing!".



Окно журнала.

03-20 15:18:16.948 27557-28121/com.hfad.joke V/DelayedMessageService: The message is: Timing!

Итак, вы представляете, как работают службы. Теперь давайте выведем сообщение на экран, чтобы для проверки устройство не приходилось держать подключенным к компьютеру.

Через 10 секунд сообщение появляется в журнале.

Вывод сообщения на экран

В отличие от активностей, службы не имеют пользовательского интерфейса, но это не означает, что им не нужно оповещать пользователя о происходящем. Например, пользователю было бы полезно узнать о завершении загрузки большого файла. В нашем примере было бы намного удобнее, если бы сообщение выводилось не в журнал, а на экран во временном окне. Вот только возникает одна проблема — код, обновляющий пользовательский интерфейс, должен выполняться только в главном потоке.

Для обновления интерфейса необходим основной поток

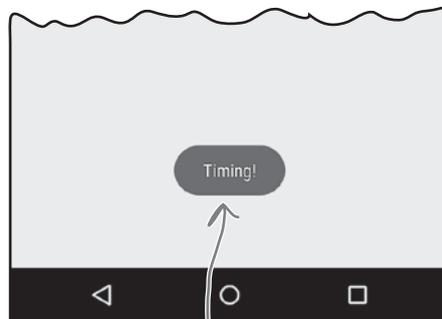
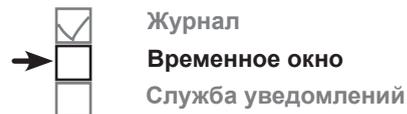
Как вы уже видели, при использовании службы с интентом выполняемый код помещается в метод `onHandleIntent()`. Этот код выполняется в фоновом режиме в отдельном потоке. Такой режим прекрасно подходит для кода, который должен выполняться в фоновом режиме, но создает проблемы, если код должен обновлять пользовательский интерфейс, — как говорилось ранее, обновление интерфейса может производиться только в основном потоке.

Чтобы обойти это препятствие, мы воспользуемся объектом `Handler`. В главе 4 уже упоминалось о том, что этот объект позволяет передавать код на выполнение в другом потоке. Метод `post()` будет использован для выполнения кода создания временного окна в основном потоке. Таким образом, код будет выполнен в основном потоке, а временное окно будет нормально выведено на экран.

Чтобы это решение работало, необходимо сделать следующее:

- ★ Создать объект `Handler` в основном потоке.
- ★ Использовать метод `post()` класса `Handler` в методе `onHandleIntent()` службы для отображения временного окна.

Первое, с чем необходимо разобраться, — как создать объект `Handler` в основном потоке?



В этой версии служба отображает сообщение во временном окне.

onStartCommand() выполняется в основном потоке

Чтобы выполнение проходило в основном потоке, объект Handler должен быть создан в методе, выполняемом в основном потоке. Метод onHandleIntent () не подходит, так как он выполняется в фоновом потоке. Вместо этого мы воспользуемся методом onStartCommand (). Метод onStartCommand () вызывается каждый раз при запуске службы интентом. Метод onStartCommand () выполняется в основном потоке и обрабатывает до метода onHandleIntent (). Если создать объект Handler в методе onStartCommand (), то его можно будет использовать для передачи кода в основной поток в методе onHandleIntent ():

...

```
public class DelayedMessageService extends IntentService {
```

```
    private Handler handler;
```

← Объект Handler сохраняется в приватной переменной, чтобы с ним могли работать другие методы.

...

```
@Override
```

```
public int onStartCommand(Intent intent, int flags, int startId) {
    handler = new Handler();
    return super.onStartCommand(intent, flags, startId);
}
```

↙ Метод выполняется в основном потоке, поэтому новый объект Handler будет создан в основном потоке.

↑ Вызвать метод onStartCommand() класса IntentService.

```
@Override
```

```
protected void onHandleIntent(Intent intent) {
    //Объект Handler используется для передачи кода
    // на выполнение в основном потоке
}
```

...

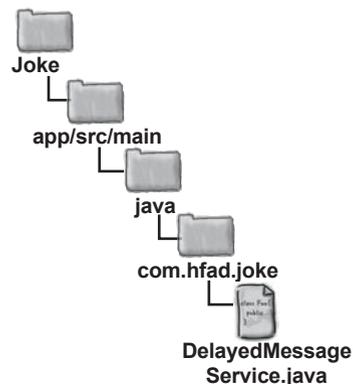
```
}
```

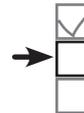
При использовании метода onStartCommand () необходимо вызвать реализацию из суперкласса:

```
super.onStartCommand(intent, flags, startId)
```

Это делается для того, чтобы служба могла нормально управлять жизненным циклом своего фонового потока.

На следующей странице приводится полный код DelayedMessageService.java, а затем будет представлен результат его выполнения.





Полный код DelayedMessageService.java

```

package com.hfad.joke;

import android.app.IntentService;
import android.content.Intent;
import android.os.Handler;
import android.widget.Toast;

public class DelayedMessageService extends IntentService {

    public static final String EXTRA_MESSAGE = "message";
    private Handler handler;

    public DelayedMessageService() {
        super("DelayedMessageService");
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        handler = new Handler();
        return super.onStartCommand(intent, flags, startId);
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        synchronized (this) {
            try {
                wait(10000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        String text = intent.getStringExtra(EXTRA_MESSAGE);
        showText(text);
    }

    private void showText(final String text) {
        handler.post(new Runnable() {
            @Override
            public void run() {
                Toast.makeText(getApplicationContext(), text, Toast.LENGTH_LONG).show();
            }
        });
    }
}

```

Мы используем эти дополнительные классы.

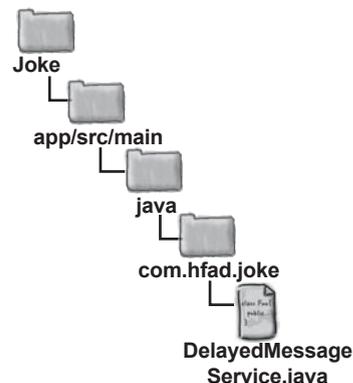
Для объекта Handler создается приватная переменная.

Создать объект Handler для основного потока.

Этот метод не изменяется.

Код с Toast передается основному потоку с использованием объекта Handler.

Контекст, в котором должно отображаться временное окно (подробности на следующей странице).





Журнал
Временное окно
Служба уведомлений

Контекст приложения

Присмотритесь повнимательнее к строке кода, в которой отображается окно уведомления:

```
Toast.makeText(getApplicationContext(), text, Toast.LENGTH_LONG).show();
```

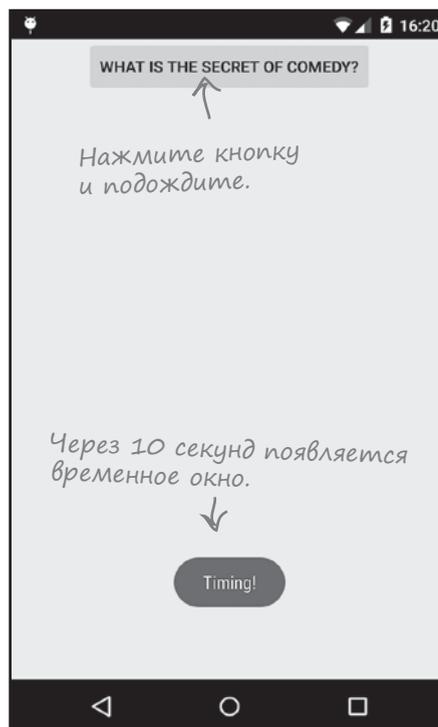
В первом параметре метода `Toast.makeText()` передается контекст, в котором должно отображаться временное окно. При создании временного окна в активности в этом параметре передается `this` для обозначения экземпляра текущей активности. Для служб такое решение не подходит, потому что контекст службы не имеет доступа к экрану. Если вам потребуется получить контекст в подобной ситуации, вызовите метод `getApplicationContext()`. Он предоставляет контекст для приложения, активного на момент выполнения кода. Это означает, что служба сможет вывести временное окно, даже если пользователь переключится на другое приложение.



Тест-драйв

Попробуем снова запустить приложение. При нажатии кнопки в `MainActivity` через 10 секунд появляется временное окно. Окно появляется независимо от того, какое приложение обладает фокусом.

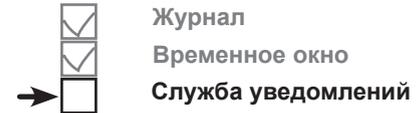
Если несколько раз быстро нажать кнопку, разные окна уведомлений появляются с интервалом приблизительно в 10 секунд. Служба последовательно обрабатывает все полученные интенты.



Временные окна не идеальны?

Вы уже знаете, как вывести фрагмент текста на экран при помощи временного окна. Например, оповещения такого рода могут быть полезны при загрузке очень большого файла. Но откровенно, уведомления не так уж сильно выделяются на экране, и если не смотреть на экран в нужный момент, вы их даже не увидите. Если вы хотите действительно донести до пользователя важную информацию, временное окно следует заменить **уведомлением**.

Уведомления (notifications) представляют собой сообщения, которые отображаются в списке в верхней части экрана. Если пользователь не увидит уведомление в момент его создания, неважно — он сможет просмотреть его позднее, проведя пальцем от верхнего края экрана для открытия выдвижной панели.



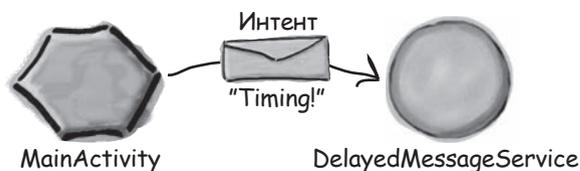
Чтобы отправить уведомление, следует воспользоваться одной из встроенных служб Android — **службой уведомлений**. Система Android включает несколько встроенных служб, которые могут использоваться в приложениях. В их число входят служба сигналов (для управления сигналами), служба загрузки (для запроса загрузок HTTP) и служба позиционирования (для получения данных местонахождения).

Служба уведомлений используется для управления уведомлениями. На следующей странице приведена общая схема ее использования в приложениях.

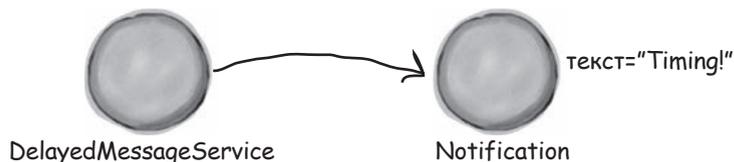
Использование службы уведомлений

Ниже представлена схема использования службы уведомлений Android в нашем приложении:

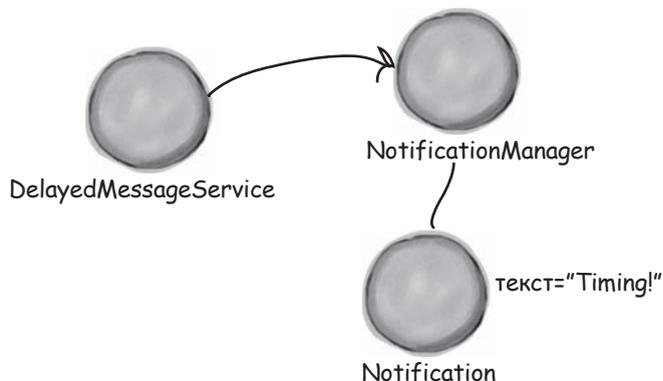
- 1 **MainActivity** запускает службу **DelayedMessageService**, передавая ей интент.



- 2 **DelayedMessageService** создает новый объект **Notification**. Объект **Notification** содержит сведения о конфигурации уведомления: текст, заголовок и значок.

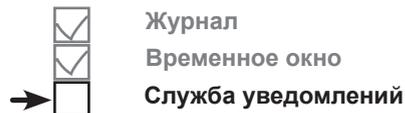


- 3 **DelayedMessageService** создает объект **NotificationManager** для обращения к службе уведомлений Android. **DelayedMessageService** передает объект **Notification** объекту **NotificationManager**, и уведомление появляется на экране.



Начнем с создания уведомления.

Построение объектов уведомлений



Для создания объектов `Notification` используется специальный объект — построитель уведомлений. Он позволяет создать уведомление с конкретными характеристиками без написания длинного кода. С каждым уведомлением должен быть связан маленький значок, заголовок и текст.

Ниже приведен пример кода создания уведомления. В нем используется высокоприоритетное уведомление, которое включает вибрацию при появлении и исчезает после щелчка:

```

Вывести ма- Notification notification = new Notification.Builder(this)
ленький значок → .setSmallIcon(R.mipmap.ic_launcher)
уведомления — .setContentTitle(getString(R.string.app_name))
в данном случае ре- ← Задать заголовок и текст.
сурс рѳртар ← .setContentText(text)
с именем ic_launcher. .setAutoCancel(true) ← Уведомление должно исчезать при щелчке.
. setPriority(Notification.PRIORITY_MAX) ← Назначить максимальный
. setDefaults(Notification.DEFAULT_VIBRATE) ← приоритет и включить
. build();                               вибрацию для привлечения
                                         внимания.

```

Это лишь некоторые из свойств уведомлений, которые вы можете использовать в приложениях. Также возможно управлять такими аспектами, как отображение уведомлений на экране блокировки, число, которое выводится рядом с уведомлением при отправке многих уведомлений от одного приложения, и звуковой сигнал для привлечения внимания пользователя. Более подробная информация доступна по адресу:

<https://developer.android.com/reference/android/app/Notification.Builder.html>

Также бывает полезно указать, какая активность должна отображаться при щелчке на уведомлении. Скажем, в нашем примере можно указать, что при щелчке на уведомлении должна отображаться активность `MainActivity`. На следующей странице мы покажем, как это делается.



Будьте
осторожны!

Некоторые свойства уведомления требуют API уровня 16 и выше.

Если вам необходима поддержка старых устройств, некоторые из свойств использовать не удастся.

Запуск активности из оповещения

Для запуска активности при щелчке на уведомлении используется **отложенный интент**. Приложение передает отложенный интент другим приложениям, чтобы те могли позднее отправить интент по поручению вашего приложения.

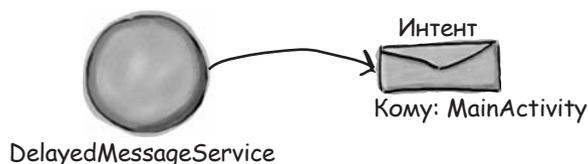
Процедура создания отложенного интента состоит из нескольких шагов:

1. Создание явного интента

Сначала вы создаете простой явный интент, предназначенный для активности, которая должна запускаться при щелчке на уведомлении. В нашем примере будет запускаться MainActivity:

```
Intent intent = new Intent(this, MainActivity.class);
```

Обычный интент
для запуска MainActivity.



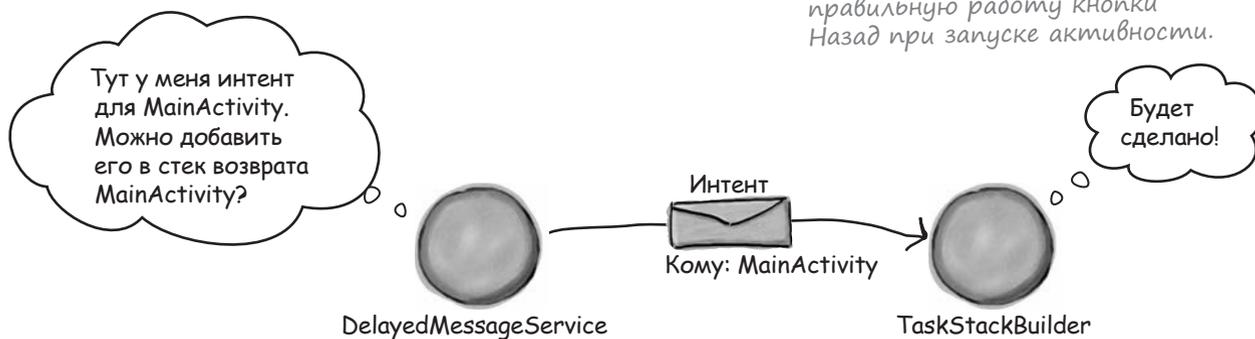
2. Передача интента TaskStackBuilder

Затем мы используем объект TaskStackBuilder, чтобы обеспечить правильную работу кнопки Назад при запуске активности. Объект TaskStackBuilder позволяет работать с историей активностей, используемой кнопкой Назад. Сначала мы получаем стек возврата, относящийся к активности, а затем добавляем в него только что созданный интент:

```
TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
stackBuilder.addParentStack(MainActivity.class);
stackBuilder.addNextIntent(intent);
```

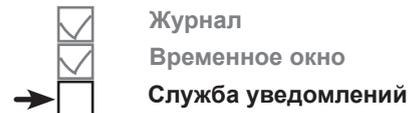
Создать TaskStackBuilder.

Эти строки обеспечивают правильную работу кнопки Назад при запуске активности.



История продолжается на следующей странице.

`getPendingIntent()`



3. Получение отложенного интента от TaskStackBuilder

Затем мы получаем отложенный интент от объекта `TaskStackBuilder`, используя его метод `getPendingIntent()`. Метод `getPendingIntent()` получает два параметра `int`: код запроса, используемый для идентификации интента, и флаг, определяющий поведение отложенного интента.

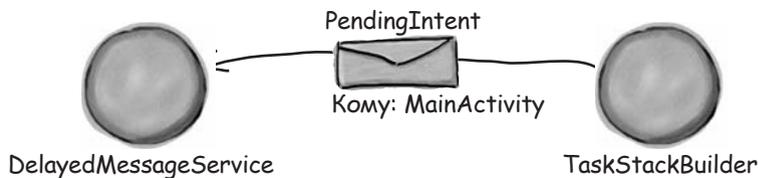
Несколько возможных значений флага:

<code>FLAG_CANCEL_CURRENT</code>	Если отложенный интент уже существует, отменить его перед созданием нового.
<code>FLAG_NO_CREATE</code>	Если подходящий отложенный интент не существует, не создавать его и вернуть <code>null</code> .
<code>FLAG_ONE_SHOT</code>	Отложенный интент может использоваться только один раз.
<code>FLAG_UPDATE_CURRENT</code>	Если подходящий отложенный интент уже существует, оставить его и заменить дополнительные данные данными из нового интента.

В нашем случае будет использоваться значение `FLAG_UPDATE_CURRENT` для модификации существующего отложенного интента. Код выглядит так:

```
PendingIntent pendingIntent =  
    stackBuilder.getPendingIntent(0, PendingIntent.FLAG_UPDATE_CURRENT);
```

Создает отложенный интент.

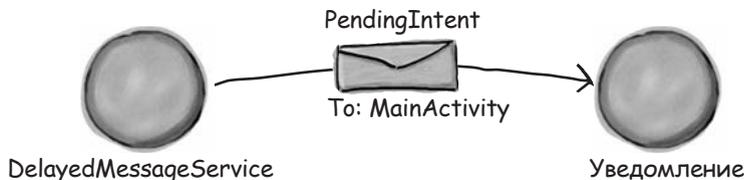


4. Добавление интента в уведомление

Наконец, отложенный интент добавляется в уведомление методом `setContentIntent()`:

```
notification.setContentIntent(pendingIntent);
```

Отложенный интент добавляется в уведомление, чтобы при щелчке на уведомлении запускалась активность MainActivity.



Когда уведомление будет связано с отложенным интентом, который определяет, какая активность должна запускаться при щелчке, остается только передать его Android.

Отправка уведомлений с использованием службы уведомлений

К настоящему моменту мы рассматривали процесс создания и настройки уведомлений. Следующий шаг – передача уведомления службе уведомлений Android, чтобы оно появилось на устройстве.

Для обращения к встроенным службам Android используется метод `getSystemService()`. Метод получает один аргумент – имя службы.

В нашем примере используется служба уведомлений, поэтому код выглядит так:

```
public static final int NOTIFICATION_ID = 5453;
...
NotificationManager notificationManager =
    (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
notificationManager.notify(NOTIFICATION_ID, notification);
```

Идентификатор, назначаемый уведомлению.

Так происходит обращение к службе уведомлений Android.

Вывести созданное нами уведомление через службу уведомлений.

Значение `NOTIFICATION_ID` идентифицирует уведомление. Если отправить другое уведомление с тем же идентификатором, оно заменит текущее уведомление. Такая возможность будет полезна, если вы хотите обновить существующее уведомление новой информацией.

Служба уведомлений берет на себя все хлопоты, связанные с обновлением экрана из фоновой службы. Это означает, что вам не нужно использовать объект `Handler` для обновления пользовательского интерфейса; служба уведомлений сделает все за вас. На следующей странице приведен обновленный код `DelayedMessageService`.

Часто задаваемые вопросы

В: Зачем включать значок в уведомление?

О: Значок используется системой уведомлений для вывода уведомления у верхнего края экрана.

В: Что произойдет, если не назначить приоритет и включить вибрацию?

О: Уведомление все равно будет отправлено, но оно не появится на экране. Вы сможете увидеть его на выдвинутой панели.

Полный код *DelayedMessageService.java*

Ниже приведен полный код *DelayedMessageService.java*. Вместо временного окна для вывода сообщений в нем используется служба уведомлений:

```
package com.hfad.joke;
```

```
import android.app.IntentService;
import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.app.TaskStackBuilder;
import android.content.Context;
import android.content.Intent;
import android.os.Handler;
import android.widget.Toast;
```

Мы используем эти дополнительные классы.

Временное окно `Toast` уже не выводится, поэтому эти директивы не нужны.

```
public class DelayedMessageService extends IntentService {
```

```
    public static final String EXTRA_MESSAGE = "message";
    private Handler handler;
    public static final int NOTIFICATION_ID = 5453;
```

Объект `Handler` тоже не нужен.

```
    public DelayedMessageService() {
        super("DelayedMessageService");
    }
```

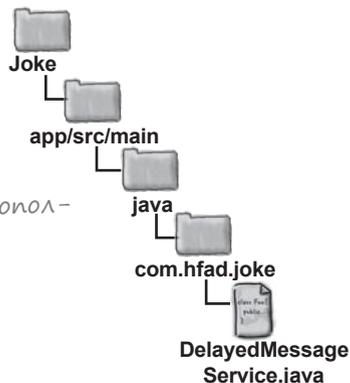
Используется для идентификации уведомлений. Число выбирается произвольно; мы выбрали 5453.

Этот метод не изменяется.

```
@Override
protected void onHandleIntent(Intent intent) {
    synchronized (this) {
        try {
            wait(10000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    String text = intent.getStringExtra(EXTRA_MESSAGE);
    showText(text);
}
```



Журнал
Временное окно
Служба уведомлений



Kog DelayedMessageService.java (продолжение)

```

@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    handler = new Handler();
    return super.onStartCommand(intent, flags, startId);
}

private void showText(final String text) {
    handler.post(new Runnable() {
        @Override
        public void run() {
            Toast.makeText(getApplicationContext(), text, Toast.LENGTH_LONG).show();
        }
    }

    Intent intent = new Intent(this, MainActivity.class);
    TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
    stackBuilder.addParentStack(MainActivity.class);
    stackBuilder.addNextIntent(intent);
    PendingIntent pendingIntent =
        stackBuilder.getPendingIntent(0, PendingIntent.FLAG_UPDATE_CURRENT
    );
    Notification notification = new Notification.Builder(this)
        .setSmallIcon(R.mipmap.ic_launcher)
        .setContentTitle(getString(R.string.app_name))
        .setAutoCancel(true)
        .setPriority(Notification.PRIORITY_MAX)
        .setDefaults(Notification.DEFAULT_VIBRATE)
        .setContentIntent(pendingIntent)
        .setContentText(text)
        .build();
    NotificationManager notificationManager =
        (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
    notificationManager.notify(NOTIFICATION_ID, notification);
}
}

```

Объект Handler в новой версии не используется, этот метод не нужен.

Объект Toast не используется, этот фрагмент можно удалить.

Создать интент.

Использовать объект TaskStackBuilder для того, чтобы обеспечить правильную работу кнопки Назад. Затем создается отложенный интент.

Построить объект уведомления.

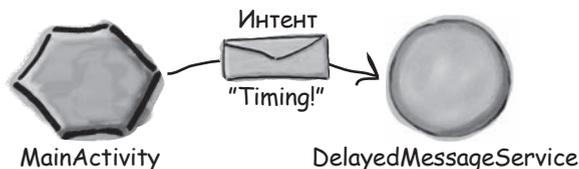
Вывести уведомление с использованием службы уведомлений Android.

Вот и весь код, необходимый для запускаемой службы. Посмотрим, что происходит при выполнении этого кода.

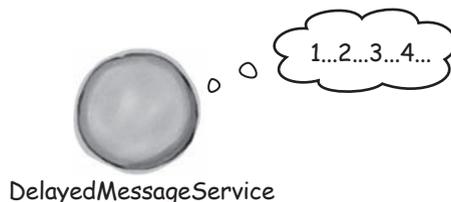
Что происходит при выполнении кода

Прежде чем вы увидите приложение в действии, взгляните, что происходит в процессе выполнения:

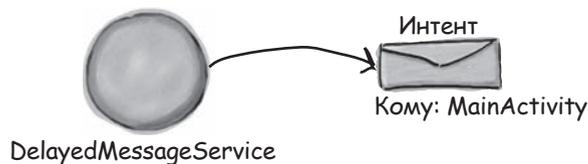
- 1 **MainActivity** запускает службу **DelayedMessageService**, передавая ей **интент**. Интент содержит сообщение, которое служба **DelayedMessageService** должна вывести по требованию **MainActivity**.



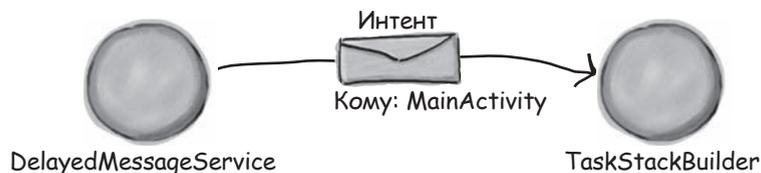
- 2 **DelayedMessageService** ожидает 10 секунд.



- 3 **DelayedMessageService** создает **интент** для **MainActivity**.

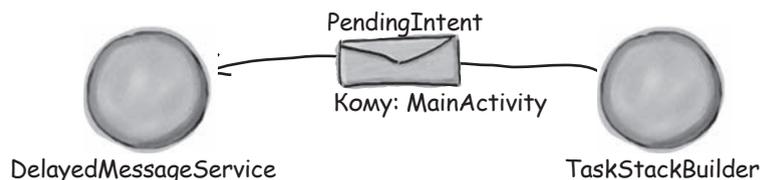


- 4 **DelayedMessageService** создает объект **TaskStackBuilder** и приказывает ему добавить **интент** в стек возврата **MainActivity**.

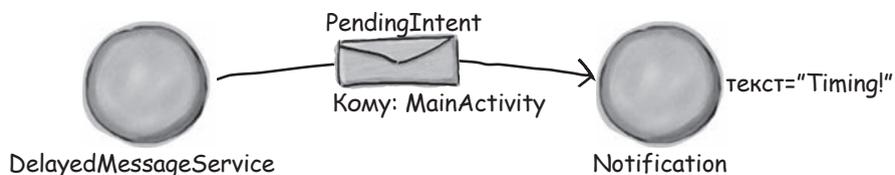


История продолжается

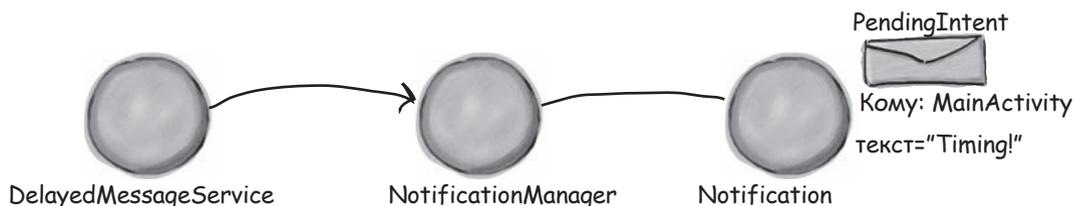
- 5 Объект `TaskStackBuilder` создает отложенный интент и передает его `DelayedMessageService`.



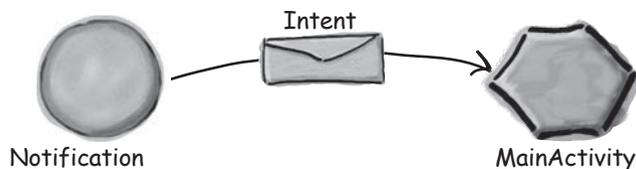
- 6 `DelayedMessageService` создает объект `Notification`, задает параметры его конфигурации и передает ему отложенный интент.



- 7 `DelayedMessageService` создает объект `NotificationManager` для обращения к службе уведомлений Android и передает ему объект `Notification`. Служба уведомлений выводит уведомление.



- 8 Когда пользователь щелкает на уведомлении, объект `Notification` использует свой отложенный интент для запуска `MainActivity`.



Посмотрим, как работает приложение.



Тест-драйв

При нажатии кнопки в MainActivity через 10 секунд появляется уведомление. Уведомление будет получено независимо от того, какое приложение является текущим.



Журнал
Временное окно
Служба уведомлений



После задержки на экране появляется уведомление. Возможно, на старых устройствах придется открыть выдвигающую панель уведомлений, чтобы увидеть его.

Если щелкнуть на уведомлении, Android возвращает пользователя к MainActivity.

Как и предполагалось, по щелчку на уведомлении открывается MainActivity.



Итак, вы узнали, как создать запускаемую службу для вывода уведомления с использованием службы уведомлений Android. После упражнения мы перейдем к другой теме и займемся созданием связанных служб.



Развлечения с магнитами

Ниже приведена большая часть кода создания запускаемой службы `WombleService`, которая воспроизводит файл `.mp3` в фоновом режиме, и активности, использующей эту службу. Попробуйте заполнить пропуски в коде.

← Это служба.

```
public class WombleService extends .....{

    public WombleService() {
        super("WombleService");
    }

    @Override
    protected void .....(Intent intent) {
        MediaPlayer mediaPlayer =
            MediaPlayer.create(getApplicationContext(), R.raw.wombling_song);
        mediaPlayer.start();
    }
}
```

← Использует класс `Android MediaPlayer` для воспроизведения файла с именем `wombling_song.mp3`. Файл находится в папке `res/raw`.

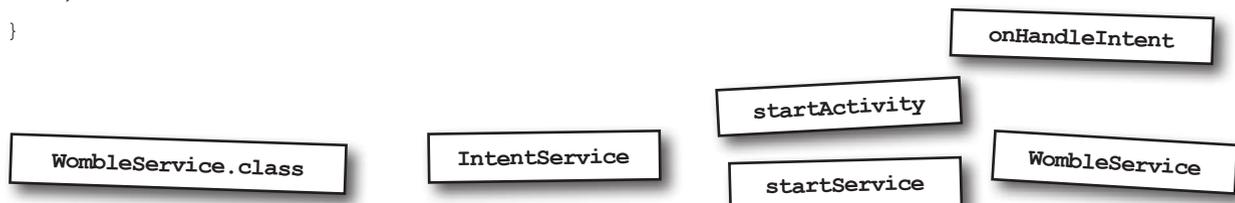
← Это активность.

```
public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public void onClick(View view) {
        Intent intent = new Intent(this, .....);

        ..... (intent);
    }
}
```





Развлечения с Магнитами. Решение

Ниже приведена большая часть кода создания запускаемой службы `WombleService`, которая воспроизводит файл `.mp3` в фоновом режиме, и активности, использующей эту службу. Попробуйте заполнить пропуски в коде.

```
public class WombleService extends IntentService {.....
    ↙ Служба расширяет
      класс IntentService.

    public WombleService() {
        super("WombleService");
    }

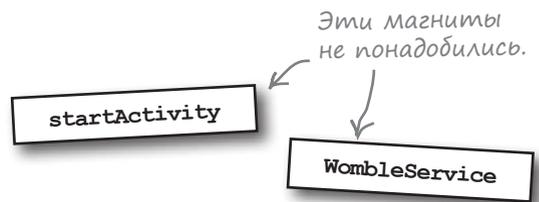
    @Override
    protected void onHandleIntent (Intent intent) {
        ↙ Код должен выполняться
          в методе onHandleIntent().
        MediaPlayer mediaPlayer =
            MediaPlayer.create(getApplicationContext(), R.raw.wombling_song);
        mediaPlayer.start();
    }
}
```

```
public class MainActivity extends Activity {
    ↙ Это активность.

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public void onClick(View view) {
        Intent intent = new Intent(this, WombleService.class);.....
        ↙ Создать явный интент,
          предназначенный для
          WombleService.class.

        startService (intent);
        ↙ Запустить службу.
    }
}
```



Связанные службы обладают большей интерактивностью

Как упоминалось ранее, запускаемая служба выполняется в фоновом режиме бесконечно долго, даже если запустившая ее активность будет уничтожена. После того как выполняемая операция будет завершена, служба останавливается сама.

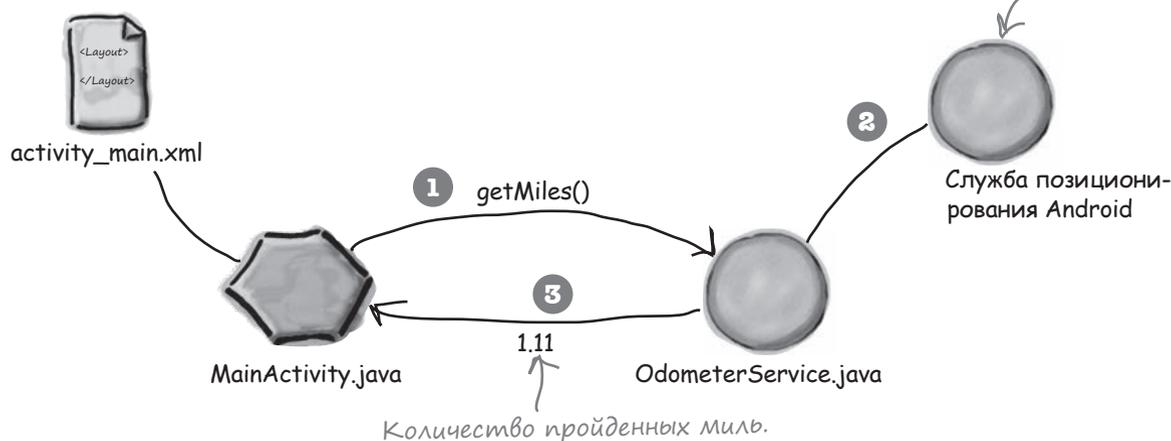
Связанная служба привязывается к другому компоненту – например, к активности. Активность может взаимодействовать со службой, отправлять запросы и получать результаты. Чтобы увидеть, как работают связанные службы, мы создадим новое приложение со связанной службой, которая, словно одометр, измеряет пройденное машиной расстояние.

Как работает приложение-одометр

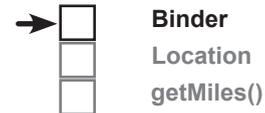
Мы создадим новый проект с активностью `MainActivity` и службой `OdometerService`. Активность `MainActivity` использует `OdometerService` для определения пройденного расстояния.

- 1 **MainActivity связывается с OdometerService.**
`MainActivity` использует метод `getMiles()` класса `OdometerService` для получения количества пройденных миль.
- 2 **OdometerService использует службу позиционирования Android для отслеживания перемещений устройства.**
 Полученная информация используется для вычисления расстояния, на которое переместилось устройство.
- 3 **OdometerService возвращает MainActivity пройденное расстояние.**
`MainActivity` выводит пройденное расстояние для пользователя.

Встроенная служба Android. Наша служба `OdometerService` будет использовать ее для прослушивания изменений текущего местонахождения.



Работа начнется с создания службы. Посмотрим, что для этого необходимо сделать.



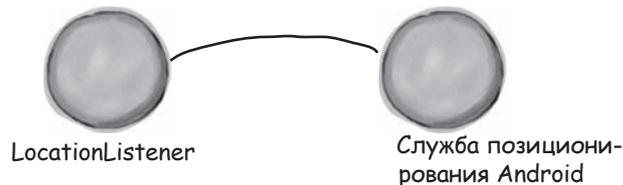
Основные этапы создания OdometerService

Чтобы создать службу OdometerService, необходимо проделать определенную последовательность действий:

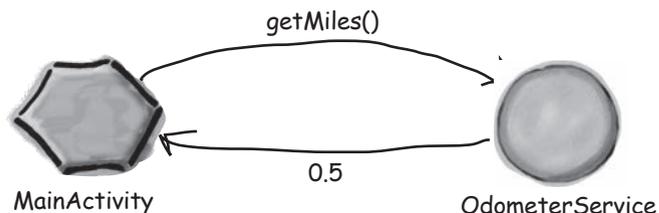
- 1 Определение класса OdometerBinder.**
 Класс Binder обеспечивает связывание активностей со службами. Мы определим subclass Binder с именем OdometerBinder, с помощью которого наша активность будет связываться с OdometerService.



- 2 Создание объекта LocationListener и его регистрация в службе позиционирования Android.**
 Это позволит OdometerService прослушивать изменения местонахождения устройства и вычислять пройденное расстояние в метрах.



- 3 Создание открытого метода getMiles().**
 Метод используется активностью для вычисления пройденного расстояния в милях.



Начнем с создания нового проекта для приложения.

Создание проекта Odometer

Создайте для приложения новый проект Android с именем “Odometer” и именем пакета `com.hfad.odometer`. Минимальный уровень SDK должен быть равен API 16, чтобы приложение работало на большинстве устройств. Чтобы ваш код не отличался от нашего, присвойте пустой активности имя “MainActivity”, а макету – имя “activity_main”. Затем в проект добавляется новая служба. На этот раз мы будем использовать службу, которая расширяет класс `Service`, а не класс `IntentService`. Это объясняется тем, что класс `IntentService` предназначен для служб, обрабатывающих интенты, как в нашем предыдущем примере. На этот раз служба запускается со связыванием, так что использование класса `IntentService` не дает никаких преимуществ.

Служба, расширяющая класс `Service`, добавляется так же, как и та служба, которую мы добавляли ранее. Выполните команду `File→New...` и выберите вариант `Service`. Выберите создание новой службы `Service` (а не `IntentService`) и присвойте службе имя “OdometerService”. Снимите флажок “exported” – он должен устанавливаться только для служб, к которым будут обращаться другие приложения. Проследите за тем, чтобы флажок “enabled” был установлен; если этого не сделать, активность не сможет запустить службу.

Вот как выглядит код создания связанной службы на базе класса `Service`:

```
package com.hfad.odometer;

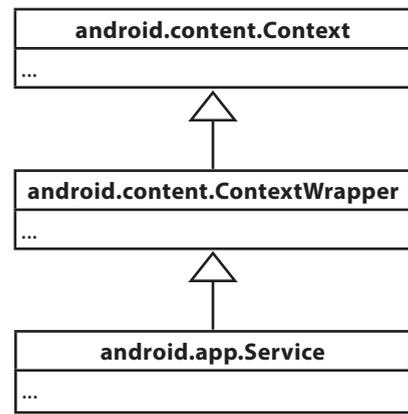
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;

public class OdometerService extends Service {

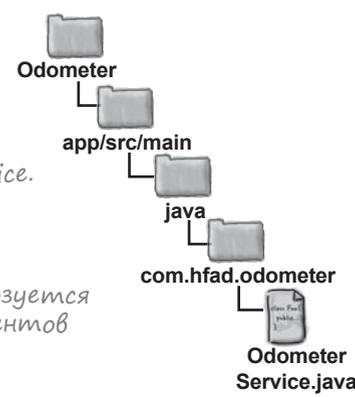
    @Override
    public IBinder onBind(Intent intent) {
        //Код связывания службы
    }
}
```

Класс расширяет класс `Service`.

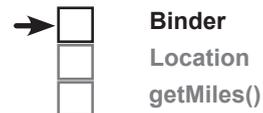
Метод `onBind()` используется для связывания компонентов со службами.



Иерархия классов службы.



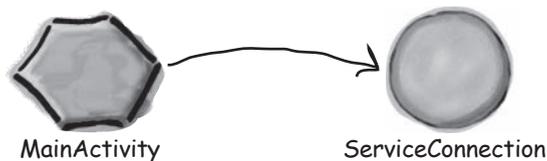
Метод `onBind()` используется для связывания службы с активностью. О том, как это делается, рассказано на следующей странице.



Как работает связывание

Вот как происходит связывание активности со службой:

- 1 Активность создает объект ServiceConnection.**
Объект ServiceConnection создает связь со службой.



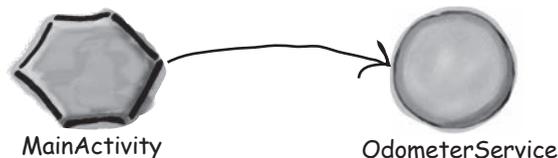
- 2 Активность передает интент службе.**
Интент содержит всю дополнительную информацию, которая должна передаваться службе активностью.



- 3 Связанная служба создает объект Binder.**
Объект Binder содержит ссылку на связанную службу. Служба возвращает объект Binder по созданному каналу связи.



- 4 Когда активность получает объект Binder, она извлекает объект Service и начинает использовать службу напрямую.**



Чтобы активность могла связаться со службой, служба должна создать объект Binder и передать его активности методом `onBind()`.

Определение Binder

Когда активность запрашивает связь со службой с использованием соединения `ServiceConnection`, последний вызывает метод `onBind()` службы. Метод `onBind()` возвращает объект `Binder` обратно по соединению, после чего объект передается активности.

При создании связанной службы вы должны определить реализацию `Binder` самостоятельно. Мы определим класс с именем `OdometerBinder` в форме внутреннего класса:

```
public class OdometerBinder extends Binder {
    OdometerService getOdometer() {
        return OdometerService.this;
    }
}
```

← При создании связанной службы необходимо предоставить реализацию `Binder`.

← Метод используется активностью для получения ссылки на `OdometerService`.

Затем экземпляр `OdometerBinder` возвращается методом `onBind()` службы:

```
...
import android.os.Binder;
import android.os.IBinder;

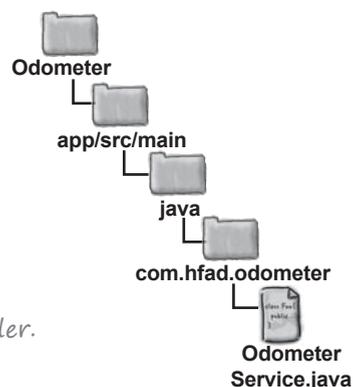
public class OdometerService extends Service {
    private final IBinder binder = new OdometerBinder();

    public class OdometerBinder extends Binder {
        OdometerService getOdometer() {
            return OdometerService.this;
        }
    }
}
...
@Override
public IBinder onBind(Intent intent) {
    return binder;
}
```

← Мы используем эти классы.

← Реализация `Binder`.

← Метод `onBind()` возвращает `IBinder` — интерфейс, реализуемый классом `Binder`.

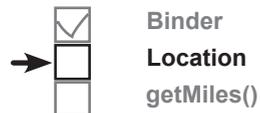


Когда активность связывается со службой через соединение, объект соединения вызывает метод `onBind()`, который возвращает объект `OdometerBinder`. Когда активность получает `OdometerBinder` от соединения, она использует метод `getOdometer()` для получения объекта `OdometerService`.

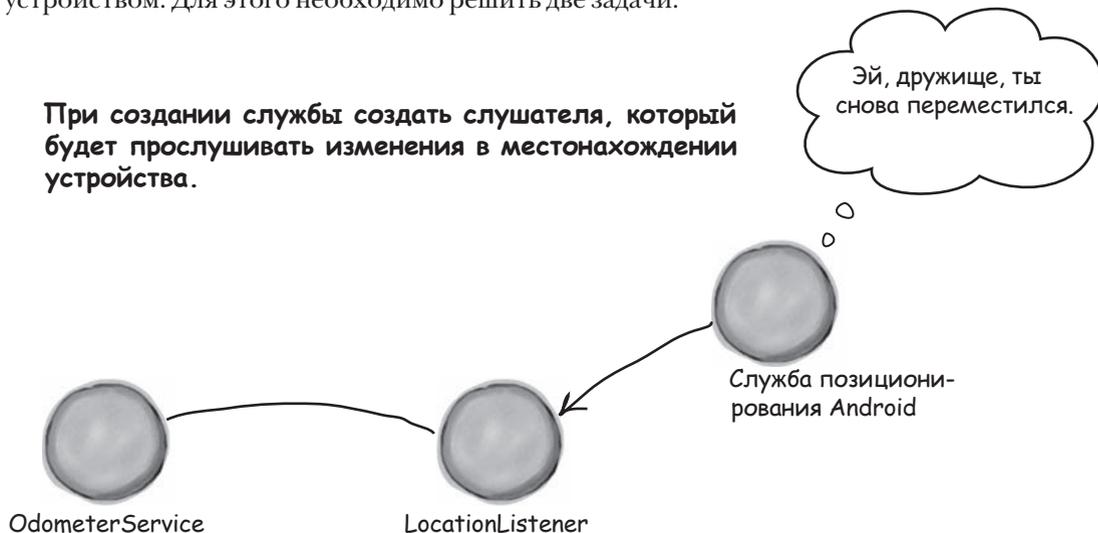
← Вы увидите, как работает эта схема, при создании активности, использующей службу.

Полезные функции службы

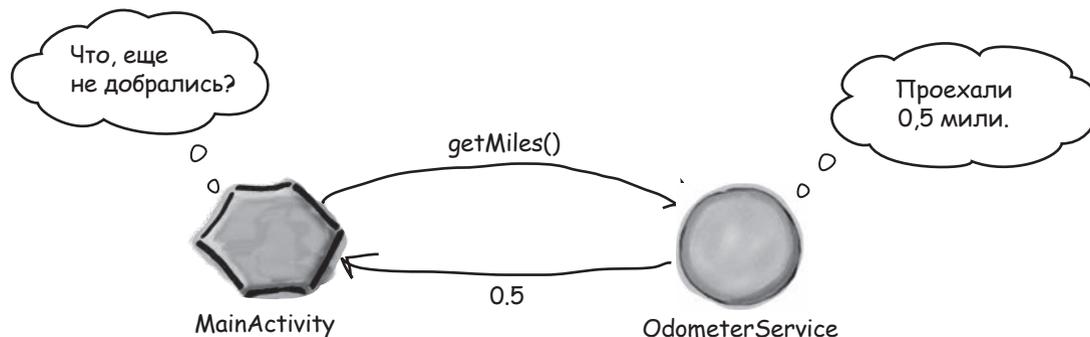
На следующем шаге нужно заставить службу делать что-то полезное. В нашем примере служба должна сообщать активности расстояние, пройденное устройством. Для этого необходимо решить две задачи:



- 1 При создании службы создать слушателя, который будет прослушивать изменения в местонахождении устройства.



- 2 Вернуть активности пройденное расстояние в милях, когда активность его запросит.



Для начала посмотрим, какие методы класса `Service` могут нам в этом помочь.

Четыре ключевых метода класса Service

Мы создаем связанную службу, которая расширяет класс Service. Класс Service содержит четыре ключевых метода, которые вам могут пригодиться:

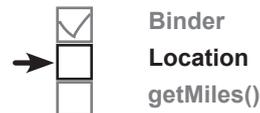
Метод	Когда вызывается	Для чего используется
onCreate()	При создании службы	Одноразовые подготовительные процедуры (например, создание экземпляра)
onStartCommand()	Когда активность запускает службу методом <code>startService()</code>	Не реализуйте этот метод, если служба не является запускаемой службой; он выполняется только в том случае, если служба запускается методом <code>startService()</code>
onBind()	Когда активность хочет связаться со службой	Реализация этого метода всегда должна возвращать объект <code>IBinder</code> ; если вы не хотите, чтобы активности связывались со службой, верните <code>null</code>
onDestroy()	Когда служба не используется и готовится к уничтожению	Метод используется для освобождения любых занятых ресурсов

В нашем случае обновления должны поступать с момента создания службы. Так как эта операция является одноразовой, она будет выполняться в методе `onCreate()`:

```
@Override
public void onCreate() {
    //Код настройки слушателя
}
```

← Так выглядит метод `onCreate()` класса `Service`.

На следующей странице вы увидите, как получать оповещения об изменении позиции.



Получение информации о местонахождении

Чтобы получить информацию о местонахождении своего устройства, воспользуйтесь службой позиционирования Android. Служба использует информацию, полученную от системы GPS, а также имена и уровни сигналов ближайших сетей WiFi для определения местонахождения устройства на поверхности земли. Начните с создания объекта `LocationListener`. Этот объект используется для получения оповещений об изменении позиции устройства. Объект `LocationListener` создается следующим образом:

```

LocationListener listener = new LocationListener() {
    @Override
    public void onLocationChanged(Location location) {
        //Код вычисления расстояния
    }
    @Override
    public void onProviderDisabled(String arg0) {}
    @Override
    public void onProviderEnabled(String arg0) {}
    @Override
    public void onStatusChanged(String arg0, int arg1, Bundle bundle) {}
};

```

Новый объект `LocationListener`.

Этот метод вызывается каждый раз, когда `LocationListener` получает информацию об изменении местонахождения устройства. Параметр `Location` описывает текущую позицию.

Эти методы тоже должны переопределяться, но их можно оставить пустыми. Они вызываются при включении/отключении модуля GPS или изменении его статуса. Нашему приложению не нужно реагировать на эти события.

Для получения информации о расстояниях следует переопределить метод `onLocationChanged()` класса `LocationListener`. Этот метод получает один параметр: объект `Location`, представляющий текущую позицию устройства. Расстояние между двумя позициями в метрах может быть вычислено методом `distanceTo()` класса `Location`. Например, если последняя позиция устройства хранится в объекте `Location` с именем `lastLocation`, то расстояние в метрах между позициями может быть вычислено следующим вызовом:

```
double distanceInMeters = location.distanceTo(lastLocation);
```

Полный код `LocationListener` приведен на следующей странице.

Добавление LocationListener в код службы

Ниже приведен обновленный код `OdometerService.java` (в метод `onCreate()` включен объект `LocationListener`, который отслеживает расстояние, пройденное устройством):

```

...
public class OdometerService extends Service {

    private static double distanceInMeters;
    private static Location lastLocation = null;
    ...

    @Override
    public void onCreate() {
        LocationListener listener = new LocationListener() {
            @Override
            public void onLocationChanged(Location location) {
                if (lastLocation == null) {
                    lastLocation = location;
                }
                distanceInMeters += location.distanceTo(lastLocation);
                lastLocation = location;
            }

            @Override
            public void onProviderDisabled(String arg0) {}

            @Override
            public void onProviderEnabled(String arg0) {}

            @Override
            public void onStatusChanged(String arg0, int arg1, Bundle bundle) {}
        };
    }
}

```

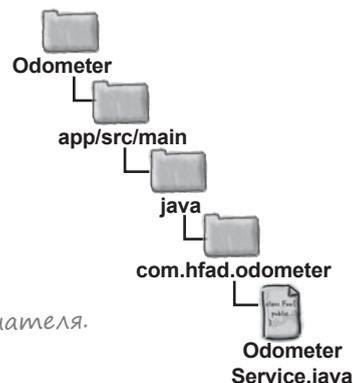
Пройденное расстояние в метрах и последнее местонахождение хранятся в статических частных переменных.

Создать слушателя.

Если позиционных данных еще не было, присвоить `lastLocation` текущую позицию.

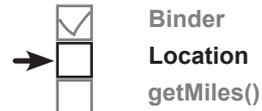
Прибавить расстояние между текущей и предыдущей позицией к переменной `distanceInMeters`, после чего присвоить `lastLocation` текущую позицию.

Эти методы необходимо переопределить, так как они входят в интерфейс `LocationListener`.



Затем созданного слушателя необходимо зарегистрировать в службе позиционирования.

Регистрация *LocationListener*



Объект *LocationListener* регистрируется в службе позиционирования Android при помощи объекта *LocationManager*. Этот объект предоставляет доступ к службе и создается следующим образом:

Так вы обращаетесь к службе позиционирования Android.

```
LocationManager locManager = (LocationManager) getSystemService (Context.LOCATION_SERVICE);
```

Метод *getSystemService()* возвращает ссылку на службу системного уровня. В данном случае нужно использовать службу позиционирования Android, поэтому используется вызов следующего вида:

Метод *getSystemService()* использовался ранее для получения доступа к службе уведомлений Android.

```
getSystemService (Context.LOCATION_SERVICE);
```

После того как объект *LocationManager* будет получен, его метод *requestLocationUpdates()* может использоваться для регистрации слушателя в службе позиционирования, и определения критериев частоты получения им данных. Методу *requestLocationUpdates()* передаются четыре параметра: провайдер GPS, минимальный интервал между обновлениями в миллисекундах, минимальное расстояние между обновлениями в метрах и объект *LocationListener*.

Например, для ежесекундного получения обновлений при перемещении устройства более чем на метр используется следующий вызов:

Провайдер GPS.

```
locManager.requestLocationUpdates (LocationManager.GPS_PROVIDER,
    1000, ← Время в миллисекундах.
    Расстояние в метрах. → 1,
    listener); ← Созданный объект LocationListener.
```

Этот метод можно использовать в методе *onCreate()* класса *Service*, чтобы зарегистрировать созданного слушателя в службе позиционирования и обеспечить регулярное получение обновленных данных. Код выглядит так:

```
@Override
public void onCreate() {
    LocationListener listener = new LocationListener() {...};
    LocationManager locManager = (LocationManager) getSystemService (Context.LOCATION_SERVICE);
    locManager.requestLocationUpdates (LocationManager.GPS_PROVIDER, 1000, 1, listener);
}
```

Слушатель создается и регистрируется в службе позиционирования при создании службы.

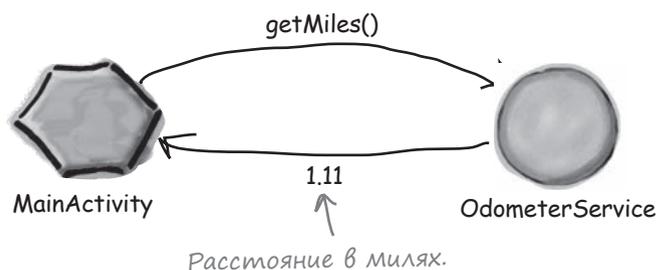
Вот и все, что необходимо для регистрации слушателя в службе позиционирования и использования его для отслеживания пройденного расстояния. Далее необходимо организовать передачу данных от слушателя к активности.

Передача пройденного расстояния активности

Если вы помните, служба должна выполнять две функции.

Во-первых, она должна отслеживать расстояние, на которое переместилось устройство. Мы решили эту задачу, создав объект `LocationListener` и зарегистрировав его в службе позиционирования.

Во-вторых, служба должна сообщить величину перемещения активности, чтобы активность могла вывести ее для пользователя. Для этого мы создадим в службе простой метод `getMiles()`, который переводит текущее пройденное расстояние в мили. Активность вызывает этот метод каждый раз, когда ей потребуется определить расстояние.



Метод `getMiles()` выглядит так:

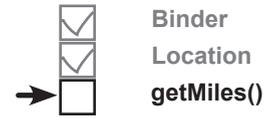
```
public double getMiles() {
    return this.distanceInMeters / 1609.344;
}
```

Метод переводит расстояние, заданное в метрах, в мили. При желании формулу можно было бы задать точнее, но для наших целей хватит и этого.

Метод получает текущее расстояние в метрах и делит его на 1609,344 для получения расстояния в милях.

На этом работа над кодом `OdometerService.java` подходит к концу. Полный код приведен на следующей странице.

Полный код OdometerService.java

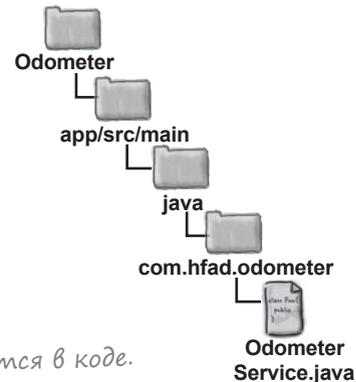


Ниже приведен полный код связанной службы *OdometerService.java*:

```
package com.hfad.odometer;
```

```
import android.app.Service;
import android.content.Context;
import android.content.Intent;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Binder;
import android.os.Bundle;
import android.os.IBinder;
```

↑
← Эти классы используются в коде.



```
public class OdometerService extends Service {
```

```
    private final IBinder binder = new OdometerBinder();
    private static double distanceInMeters;
    private static Location lastLocation = null;
```

← ↑ Используемые приватные переменные.

```
    public class OdometerBinder extends Binder {
        OdometerService getOdometer() {
            return OdometerService.this;
        }
    }
```

↑ При создании связанной службы необходимо определить объект Binder, который обеспечивает связывание активности со службой.

```
@Override
```

```
public IBinder onBind(Intent intent) {
    return binder;
}
```

↑ Вызывается при связывании активности со службой.

Код OdometerService.java (продолжение)

```

@Override
public void onCreate() {
    LocationListener listener = new LocationListener() {
        @Override
        public void onLocationChanged(Location location) {
            if (lastLocation == null) {
                lastLocation = location;
            }
            distanceInMeters += location.distanceTo(lastLocation);
            lastLocation = location;
        }

        @Override
        public void onProviderDisabled(String arg0) {}

        @Override
        public void onProviderEnabled(String arg0) {}

        @Override
        public void onStatusChanged(String arg0, int arg1, Bundle bundle) {}
    };
    locationManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
    locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 1000, 1, listener);
}

public double getMiles() {
    return this.distanceInMeters / 1609.344;
}
}

```

Настройка слушателя при создании службы.

Наша реализация LocationListener.

Перевести расстояние в мили и вернуть результат.

Слушатель регистрируется в службе позиционирования.

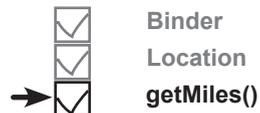
```

graph TD
    Odometer --> app_src_main[app/src/main]
    app_src_main --> java
    java --> com_hfad_odometer[com.hfad.odometer]
    com_hfad_odometer --> OdometerService_java[OdometerService.java]

```

Этот код предоставляет активности средства для связывания со службой, а затем по запросу сообщает активности, на какое расстояние переместилось устройство. Осталось сделать последний шаг: предоставить приложению разрешение на использование GPS.

Обновление AndroidManifest.xml



При создании приложения Android по умолчанию разрешает выполнение большинства действий. Однако выполнение некоторых действий требует явного разрешения со стороны пользователя. К их числу относится использование модуля GPS. Если приложение работает с оборудованием GPS, пользователь должен явно предоставить соответствующее разрешение при установке приложения.

Чтобы сообщить Android, что приложению необходимо разрешение на использование GPS, следует включить в манифест элемент `<uses-permission>`:

```
<manifest ... >
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    ...
</manifest>
```

↑
Добавляется, потому что в нашем приложении используется модуль GPS устройства.

Если это разрешение отсутствует в файле *AndroidManifest.xml*, при запуске произойдет сбой.

Также убедитесь в том, что среда Android Studio добавила вашу службу в *AndroidManifest.xml*:

```
<manifest ... >

    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

    <application
        ... >
        <activity
            ...
        </activity>

        <service
            android:name=".OdometerService"
            android:exported="false"
            android:enabled="true" >
        </service>
    </application>
</manifest>
```

Все службы должны быть объявлены в *AndroidManifest.xml*.

← Присваиваем false, так как служба будет использоваться только этим приложением.

↑ Атрибут `android:enabled` либо должен быть равен true, либо полностью опускается. Если присвоить ему false, ваше приложение не сможет использовать службу.



Мы вернемся к тому, что было сделано в приложении, после упражнения.



Развлечения с Магнитами

Попробуйте завершить приведенный ниже код и создать связанную службу с именем `NumberService`, которая возвращает случайное число при вызове метода `getNumber()`:

```
...
public class NumberService extends Service {

    private final IBinder binder = new NumberBinder();
    private final Random random = new Random();

    public class ..... extends Binder {

        ..... getNumberService() {
            return NumberService.this;
        }
    }

    @Override

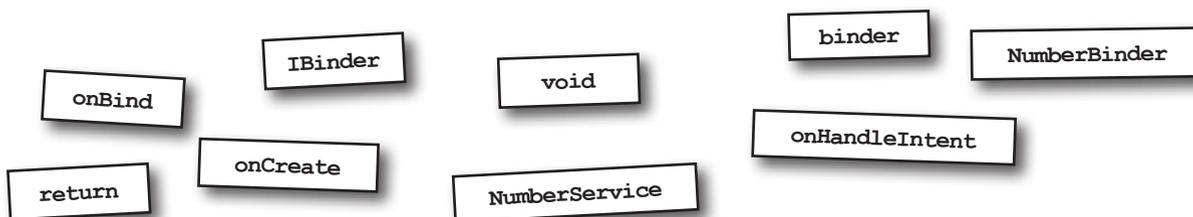
    public .....(Intent intent) {

        .....;
    }

    public int getNumber() {
        return random.nextInt(100);
    }
}

```

↑
Генерирует случайное число.





Развлечения с магнитами. Решение

Попробуйте завершить приведенный ниже код и создать связанную службу с именем `NumberService`, которая возвращает случайное число при вызове метода `getNumber()`:

...

```
public class NumberService extends Service {
```

```
    private final IBinder binder = new NumberBinder();
```

```
    private final Random random = new Random();
```

```
    public class NumberBinder extends Binder {
```

← Определить класс `NumberBinder`, расширяющий `Binder`.

```
        NumberService getService() {
```

← Активность должна получить от `Binder` ссылку на `NumberService`, поэтому возвращаться должен объект `NumberService`.

```
            return NumberService.this;
```

```
        }
```

Переопределить метод `onBind()`, чтобы активность установила связь со службой.

```
    @Override
```

```
    public IBinder onBind(Intent intent) {
```

```
        return binder;
```

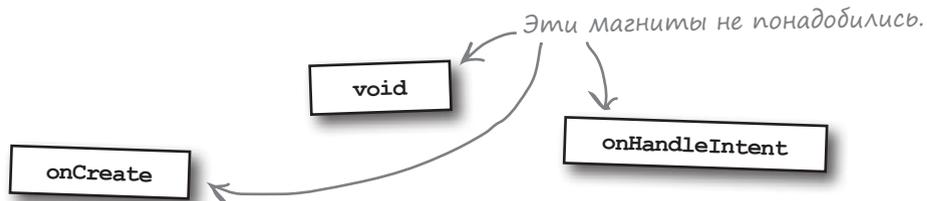
← Метод `onBind()` должен возвращать `Binder`.

```
    public int getNumber() {
```

```
        return random.nextInt(100);
```

```
    }
```

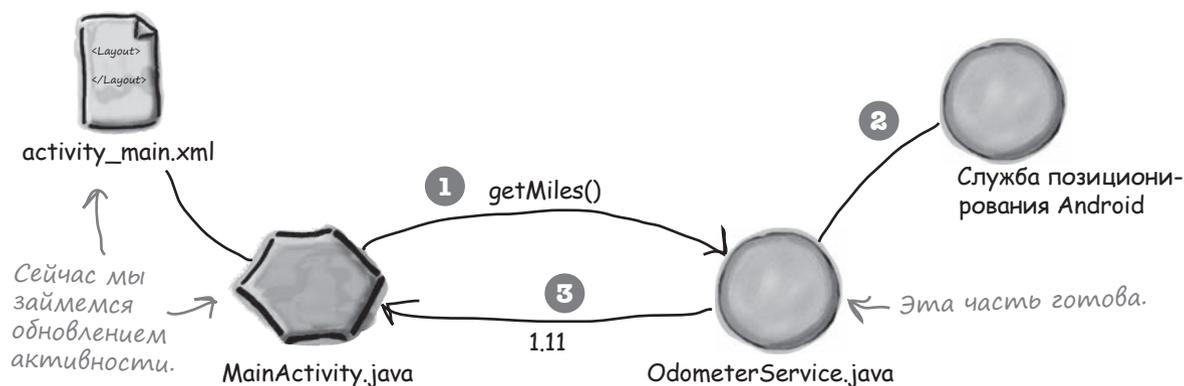
```
}
```



Что было сделано

Давайте еще раз посмотрим, что должно делать приложение, — это поможет нам понять, что еще остается сделать:

- 1 **MainActivity связывается с OdometerService.**
 MainActivity использует метод `getMiles()` класса `OdometerService` для получения количества пройденных миль.
- 2 **OdometerService использует службу позиционирования Android для отслеживания перемещений устройства.**
 Полученная информация используется для вычисления расстояния, на которое переместилось устройство.
- 3 **OdometerService возвращает MainActivity пройденное расстояние.**
 MainActivity выводит пройденное расстояние для пользователя.



Итак, мы создали службу `OdometerService`. Она обращается к службе позиционирования Android, и по полученным данным вычисляет пройденное расстояние.

Следующее, что необходимо сделать, — создать активность `MainActivity`. Мы должны связать ее с `OdometerService`, а затем использовать метод `getMiles()` класса `OdometerService` для вывода пройденного расстояния.

Обновление макета MainActivity

Активность MainActivity должна использовать службу для вывода расстояния в милях, поэтому мы начнем с обновления файла макета `activity_main.xml`. В макет добавляется текстовое представление, в котором будет отображаться расстояние; оно будет ежесекундно обновляться в коде Java.

Разметка `activity_main.xml` выглядит так:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

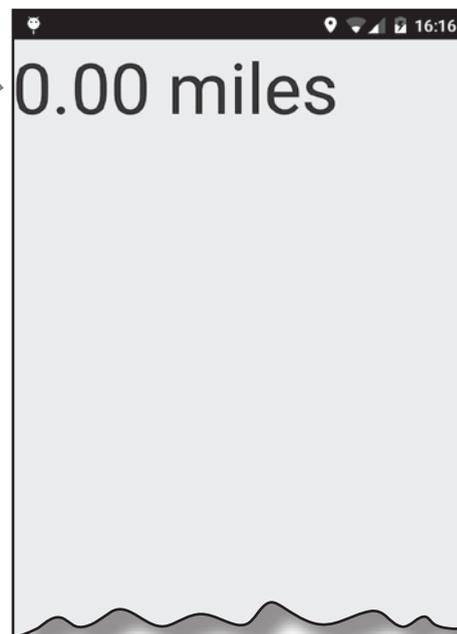
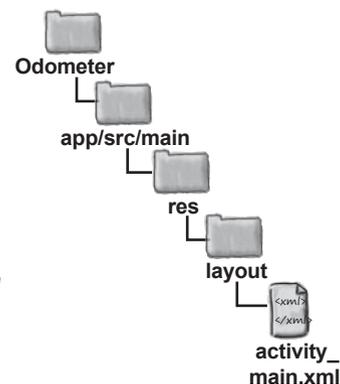
    <TextView android:text=""
        android:id="@+id/distance"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_centerHorizontal="true"
        android:singleLine="false"
        android:textSize="60dp"/>

</RelativeLayout>
```

Для вывода расстояния используется компонент TextView.



Это нужно сделать в MainActivity.



Затем следует изменить код активности, чтобы он связывался со службой и обновлял текстовое представление. Обновлять представления мы уже умеем, но как осуществить связывание со службой? Давайте посмотрим, как это делается.

Создание объекта ServiceConnection

Ранее в этой главе мы упоминали о том, что активность связывается со службой при помощи объекта ServiceConnection. Интерфейс ServiceConnection содержит два метода: onServiceConnected() и onServiceDisconnected().

Метод onServiceConnected() вызывается при создании соединения со службой и получении от службы объекта Binder. Полученный объект используется для получения ссылки на службу. Метод onServiceDisconnected() используется при потере связи со службой.

Если активность должна связываться со службой, вы должны создать собственную реализацию ServiceConnection. Наша реализация выглядит так:

```

...
public class MainActivity extends Activity {

    private OdometerService odometer;
    private boolean bound = false;
    ...

    private ServiceConnection connection = new ServiceConnection() {
        @Override
        public void onServiceConnected(ComponentName componentName, IBinder binder) {
            OdometerService.OdometerBinder odometerBinder =
                (OdometerService.OdometerBinder) binder;
            odometer = odometerBinder.getOdometer();
            bound = true;
        }
        @Override
        public void onServiceDisconnected(ComponentName componentName) {
            bound = false;
        }
    };
}

```

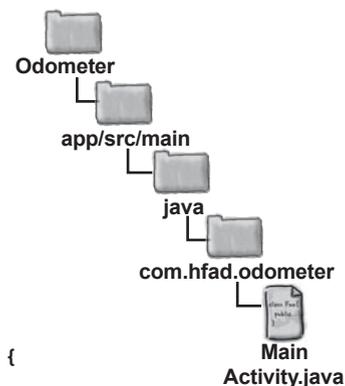
Будет использоваться для OdometerService.

Флаг связывания активности со службой.

Преобразовать Binder в OdometerBinder, после чего использовать для получения ссылки на OdometerService.

Если активность связана со службой, bound присваивается true.

При разрыве связи со службой bound присваивается false.



При установлении связи со службой метод onServiceConnected() использует объект Binder для получения ссылки на службу. При помощи методов onServiceConnected() и onServiceDisconnected() мы следим за тем, существует ли подключение к службе в настоящий момент.

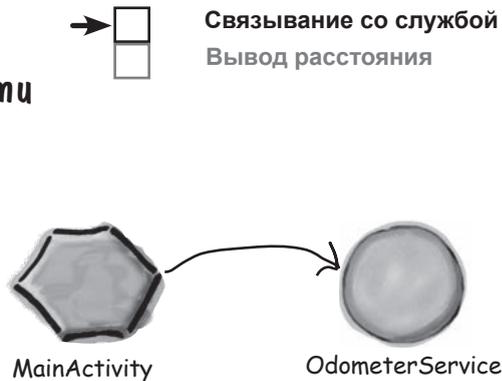
Связывание происходит при запуске активности

Объект `ServiceConnection` будет использоваться для связывания активности со службой, когда активность становится видимой. Напомним, что когда активность становится видимой, вызывается ее метод `onStart()`.

Чтобы связать активность со службой, сначала следует создать явный интент, предназначенный для нужной службы. Затем метод `bindService()` активности осуществляет связывание:

```
@Override
protected void onStart() {
    super.onStart();
    Intent intent = new Intent(this, OdometerService.class);
    bindService(intent, connection, Context.BIND_AUTO_CREATE);
}
```

Значение `Context.BIND_AUTO_CREATE` приказывает Android создать службу, если она еще не существует.



Интент, предназначенный для OdometerService.

Интент и объект ServiceConnection используются для связывания активности со службой.

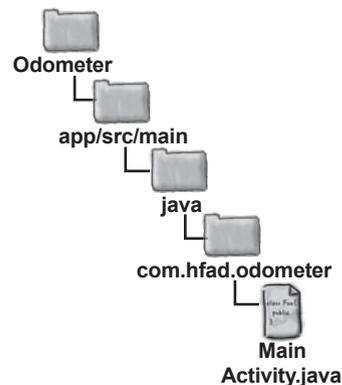
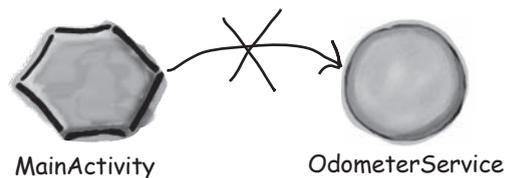
Разрыв связи со службой при остановке активности

Когда активность перестает быть видимой, мы будем прерывать связь со службой. При потере видимости активностью вызывается ее метод `onStop()`.

Разрыв связи осуществляется методом `unbindService()`. Метод получает один параметр — объект `ServiceConnection()`. При потере видимости мы проверяем, связана ли активность со службой, и если связана — разрываем эту связь:

```
@Override
protected void onStop() {
    super.onStop();
    if (bound) {
        unbindService(connection);
        bound = false;
    }
}
```

Объект ServiceConnection используется для разрыва связи активности со службой.



На данный момент наша активность связывается со службой при запуске и разрывает связь при остановке. Остается лишь позаботиться о том, чтобы активность запрашивала у службы пройденное расстояние.

Вывод расстояния

После того как соединение со службой будет установлено, вы можете вызывать ее методы. Мы будем вызывать метод `getMiles()` класса `OdometerService` каждую секунду для получения пройденного расстояния, после чего использовать полученные данные для обновления текстового представления в макете.

Для этого будет написан новый метод с именем `watchMileage()`. Он работает точно так же, как и метод `runTimer()` из главы 4. Единственное различие заключается в том, что вместо прошедшего времени отображается расстояние в милях.

Метод `watchMileage()` выглядит так:

```
private void watchMileage() {
    final TextView distanceView = (TextView)findViewById(R.id.distance);
    final Handler handler = new Handler();
    handler.post(new Runnable() {
        @Override
        public void run() {
            double distance = 0.0;
            if (odometer != null) {
                distance = odometer.getMiles();
            }
            String distanceStr = String.format("%.1f miles", distance);
            distanceView.setText(distanceStr);
            handler.postDelayed(this, 1000);
        }
    });
}
```

Получить текстовое представление.

Создать новый объект Handler.

Вызвать метод `post()` с передачей нового объекта `Runnable`.

Если ссылка на `OdometerService` получена успешно, вызвать метод `getMiles()`.

Отформатировать расстояния.

Передать код в объекте `Runnable` для повторного выполнения с задержкой 1000 миллисекунд (то есть 1 секунда). Так как эта строка кода включена в метод `run()` объекта `Runnable`, она будет выполняться каждую секунду (возможно, с небольшой задержкой).

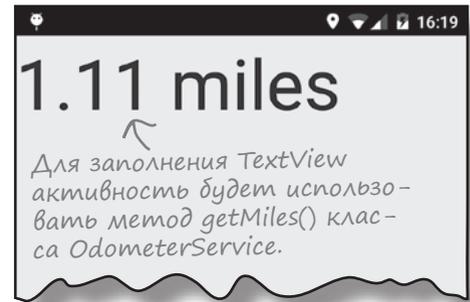
Затем этот метод вызывается в методе `onCreate()` активности, чтобы он запускался при создании активности:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    ...
    watchMileage();
}
```

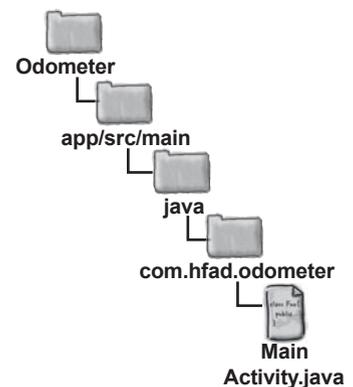
Полный код `MainActivity` приведен на следующей странице.



Связывание со службой
Вывод расстояния



Получить текстовое представление.



Полный код MainActivity.java

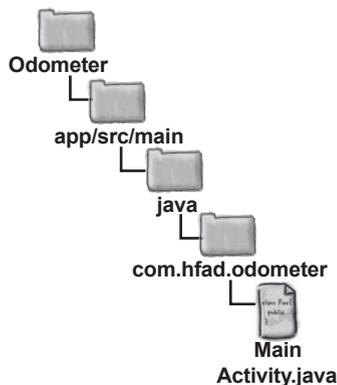


Связывание со службой
Вывод расстояния

Ниже приведен полный код *MainActivity.java*:

```
package com.hfad.odometer;

import android.app.Activity;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.Bundle;
import android.os.Handler;
import android.os.IBinder;
import android.widget.TextView;
```



```
public class MainActivity extends Activity {
```

```

private OdometerService odometer;
private boolean bound = false;
private ServiceConnection connection = new ServiceConnection() {
    @Override
    public void onServiceConnected(ComponentName componentName, IBinder binder) {
        OdometerService.OdometerBinder odometerBinder =
            (OdometerService.OdometerBinder) binder;
        odometer = odometerBinder.getOdometer();
        bound = true;
    }
    @Override
    public void onServiceDisconnected(ComponentName componentName) {
        bound = false;
    }
};

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    watchMileage();
}

```

← Используется для *OdometerService*.

← Флаг связывания активности со службой.

← Получить ссылку на *OdometerService* при создании связи со службой.

← Необходимо определить объект *ServiceConnection*.

← Вызвать функцию *watchMileage()* при создании активности.

Код MainActivity.java (продолжение)

```

@Override
protected void onStart() {
    super.onStart();
    Intent intent = new Intent(this, OdometerService.class);
    bindService(intent, connection, Context.BIND_AUTO_CREATE);
}

@Override
protected void onStop() {
    super.onStop();
    if (bound) {
        unbindService(connection);
        bound = false;
    }
}

private void watchMileage() {
    final TextView distanceView = (TextView)findViewById(R.id.distance);
    final Handler handler = new Handler();
    handler.post(new Runnable() {
        @Override
        public void run() {
            double distance = 0.0;
            if (odometer != null) {
                distance = odometer.getMiles();
            }
            String distanceStr = String.format("%1$, .2f miles", distance);
            distanceView.setText(distanceStr);
            handler.postDelayed(this, 1000);
        }
    });
}
}

```

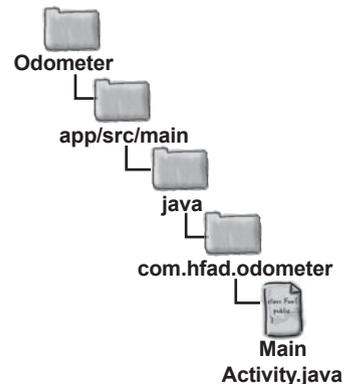
Установить связь со службой при запуске активности.

Разорвать связь со службой при остановке активности.

Метод обновляет выводимое расстояние.

Если ссылка на OdometerService получена успешно, вызвать метод getMiles() службы.

Обновлять расстояние каждую секунду.



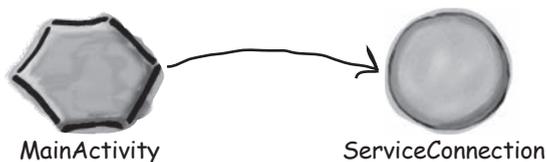
Вот и весь код, необходимый для использования OdometerService активностью MainActivity. Посмотрим, что происходит при выполнении кода.



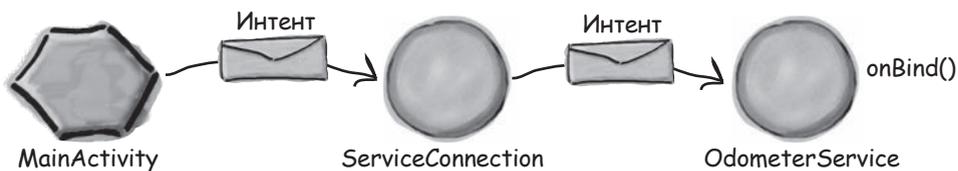
Что происходит при выполнении кода

Прежде чем опробовать приложение на практике, посмотрим, что происходит при выполнении кода:

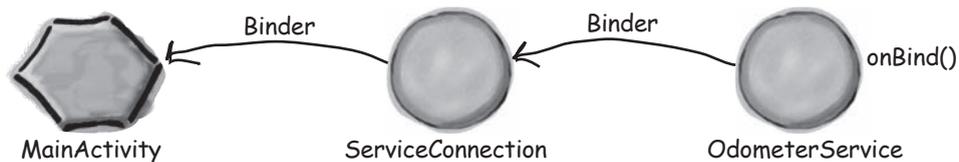
- 1 При запуске `MainActivity` метод `onStart()` создает объект `ServiceConnection`. Объект запрашивает установление связи с `OdometerService`.



- 2 Запускается служба `OdometerService`, ее метод `onBind()` вызывается с копией интента из `MainActivity`.

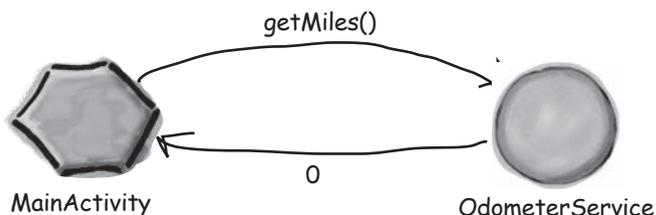


- 3 Метод `onBind()` возвращает объект `Binder`.

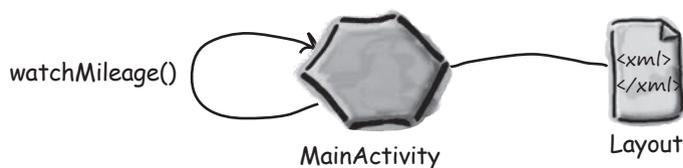


История продолжается

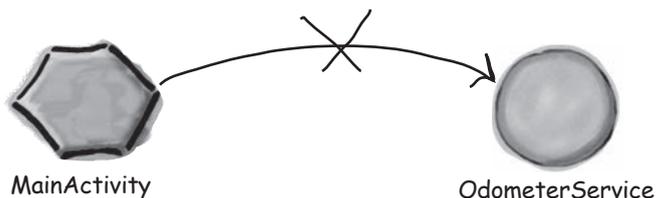
- 4 MainActivity получает от Binder ссылку на OdometerService и начинает использовать службу напрямую.



- 5 Во время работы MainActivity метод watchMileage() каждую секунду вызывает метод getMiles() службы OdometerService и обновляет расстояние на экране.



- 6 При остановке MainActivity активность разрывает связь с OdometerService вызовом метода unbindService().



Теперь запустим приложение и посмотрим, как оно работает.



Тест-драйв

Чтобы увидеть приложение в действии, необходимо запустить его на устройстве с модулем GPS. В противном случае приложение работать не будет.

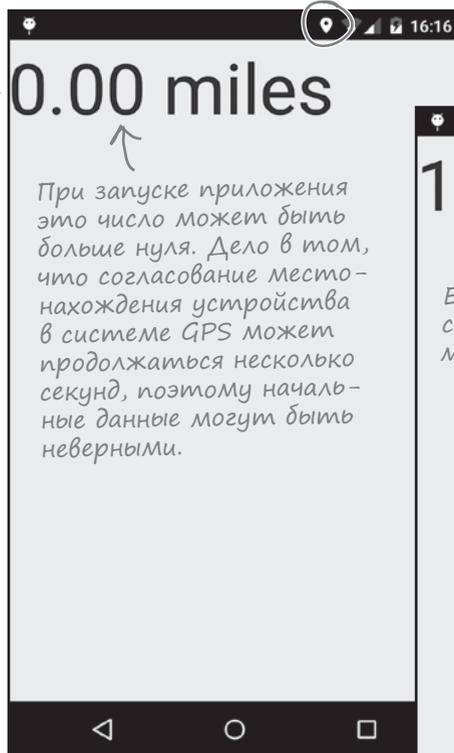
При запуске приложения пройденное расстояние равно 0. У верхнего края экрана появляется значок, сообщающий о включении службы геопозиционирования:



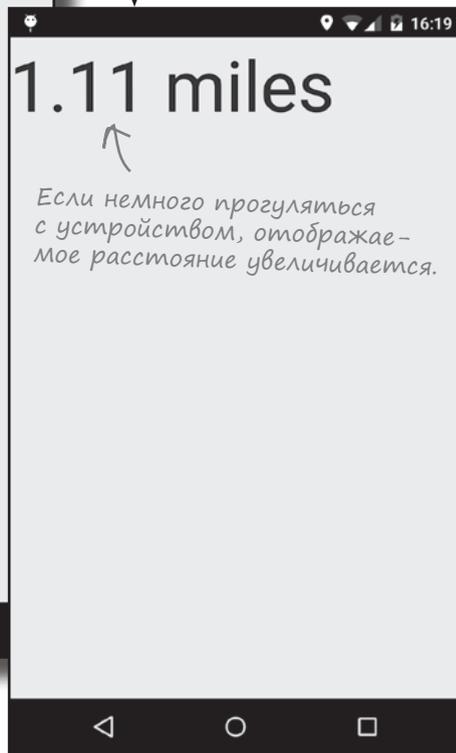
Связывание со службой
Вывод расстояния

Служба позиционирования работает нормально.

В начале работы приложения отображается расстояние 0,00 мили.



При запуске приложения это число может быть больше нуля. Дело в том, что согласование местонахождения устройства в системе GPS может продолжаться несколько секунд, поэтому начальные данные могут быть неверными.



Если немного прогуляться с устройством, отображаемое расстояние увеличивается.

Если взять устройство с собой в поездку, пройденное расстояние увеличивается.

Конечно, приложение Odometer можно усовершенствовать, и у вас наверняка найдется немало замечательных идей — так почему бы не опробовать их? Например, почему бы не добавить кнопки Пуск, Стоп и Сброс?



Ваш инструментарий Android

Глава 13 осталась позади, а ваш инструментарий пополнился навыками работы со службами.

Весь код для этой главы можно загрузить по адресу <https://tinyurl.com/HeadFirstAndroid>.

КЛЮЧЕВЫЕ МОМЕНТЫ



- Служба представляет собой компонент, выполняющий операции в фоновом режиме. Служба не обладает пользовательским интерфейсом.
- Запускаемая служба выполняется в фоновом режиме неопределенно долго, даже при уничтожении запустившей ее активности. После завершения операции такая служба останавливается.
- Службы объявляются в файле *AndroidManifest.xml* элементом `<service>`.
- Простая запускаемая служба создается расширением класса `IntentService` и переопределением его метода `onHandleIntent()`. Класс `IntentService` рассчитан на обработку интенгов.
- Запускаемая служба запускается методом `startService()`.
- Переопределяя метод `onStartCommand()` класса `IntentService`, вы должны вызвать реализацию суперкласса.
- Уведомления строятся при помощи объекта `Notification.Builder`. Чтобы уведомление запустило активность, используйте отложенный интенд. Для отображения уведомлений может использоваться служба уведомлений Android.
- Связанная служба привязывается к другому компоненту — например, к активности. Активность может взаимодействовать с этим компонентом и получать результаты.
- Связанные службы обычно создаются расширением класса `Service`. Вы должны определить свой объект `Binder` и переопределить метод `onBind()`. Этот метод вызывается тогда, когда компонент хочет установить связь со службой.
- Метод `onCreate()` класса `Service` вызывается при создании службы. Используйте его при создании экземпляра.
- Метод `onDestroy()` класса `Service` вызывается непосредственно перед уничтожением службы.
- Служба позиционирования Android используется для получения информации о текущем местонахождении устройства. Вы создаете объект `LocationListener`, после чего регистрируете его в службе позиционирования. При этом можно добавить критерии частоты оповещений об изменениях. Если вы используете модуль GPS устройства, соответствующее разрешение необходимо добавить в *AndroidManifest.xml*.
- Чтобы соединить активность со службой, создайте объект `ServiceConnection`. Переопределите метод `onServiceConnected()` для получения ссылки на службу.
- Связывание со службой осуществляется методом `bindService()`. Связь разрывается методом `unbindService()`.

Жизнь в материальном мире

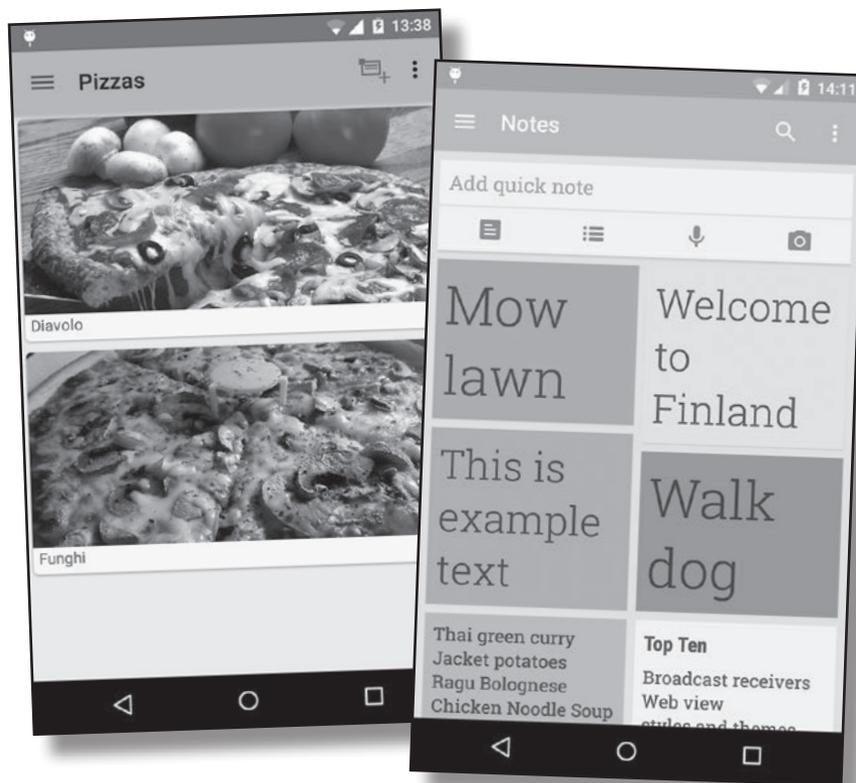


Один игрок уходит на скамью запасных, другой вступает в игру. Прямо как в RecyclerView.

В API уровня 21 компания Google представила концепцию **материального оформления**. В этой главе вы узнаете, что такое **материальное оформление** и как реализовать его принципы в ваших приложениях. Мы начнем с **карточек**, которые могут повторно использоваться в приложениях для обеспечения **целостности оформления**. Затем будет рассмотрен компонент **RecyclerView** — хороший друг спискового представления. Попутно вы узнаете, как **создавать адаптеры** и как полностью изменить внешний вид RecyclerView *всего двумя строками кода*.

Знакомство с материальным оформлением

Концепция материального оформления (Material Design) появилась в API уровня 21 для обеспечения целостности оформления всех Android-приложений. Идея заключается в том, что пользователь может переключиться с приложения, разработанного Google (например, Play Store), на приложение, созданное независимым разработчиком; при этом он будет чувствовать себя уверенно и будет знать, что делать. «Материальность» в названии происходит от визуального стиля материального оформления, в котором части интерфейса выглядят как куски материала или бумаги:



Материальное оформление использует анимацию и трехмерные эффекты (например, тени) для того, чтобы подсказать пользователю, как он может взаимодействовать с приложением. Для этого в Android включается набор библиотек поддержки с различными виджетами и темами, предназначенными для приложений с поддержкой материального оформления. В этой главе мы рассмотрим некоторые из этих виджетов и воспользуемся ими для переработки приложения Pizza, созданного в главах 9 и 10, в стиле материального оформления.

Карточки и RecyclerView

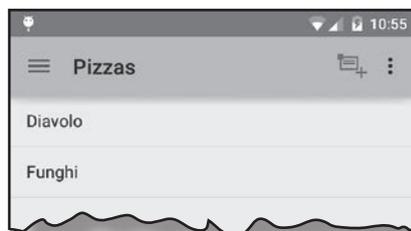
Два самых важных виджета материального оформления — RecyclerView и карточки.

Карточка (или карточное представление) выполняет функции контейнера для других представлений. Карточки имеют закругленные углы, а эффект тени создает иллюзию «парения» над фоном. К карточке можно применить анимацию, которая будет имитировать движение при нажатии.

RecyclerView напоминает новую разновидность спискового представления. Выбор названия объясняется тем, что это представление может эффективно перерабатывать (или повторно использовать) представления для вывода списка на экране. Компонент RecyclerView может использоваться для отображения карточек. В этой главе приложение Pizza будет изменено так, чтобы в нем использовались карточки и RecyclerView. Список видов пиццы, который в настоящее время

выглядит так:

Обычное
списковое
представление.



будет выглядеть так:



RecyclerView с двумя карточками. Каждая карточка содержит название и изображение пиццы.



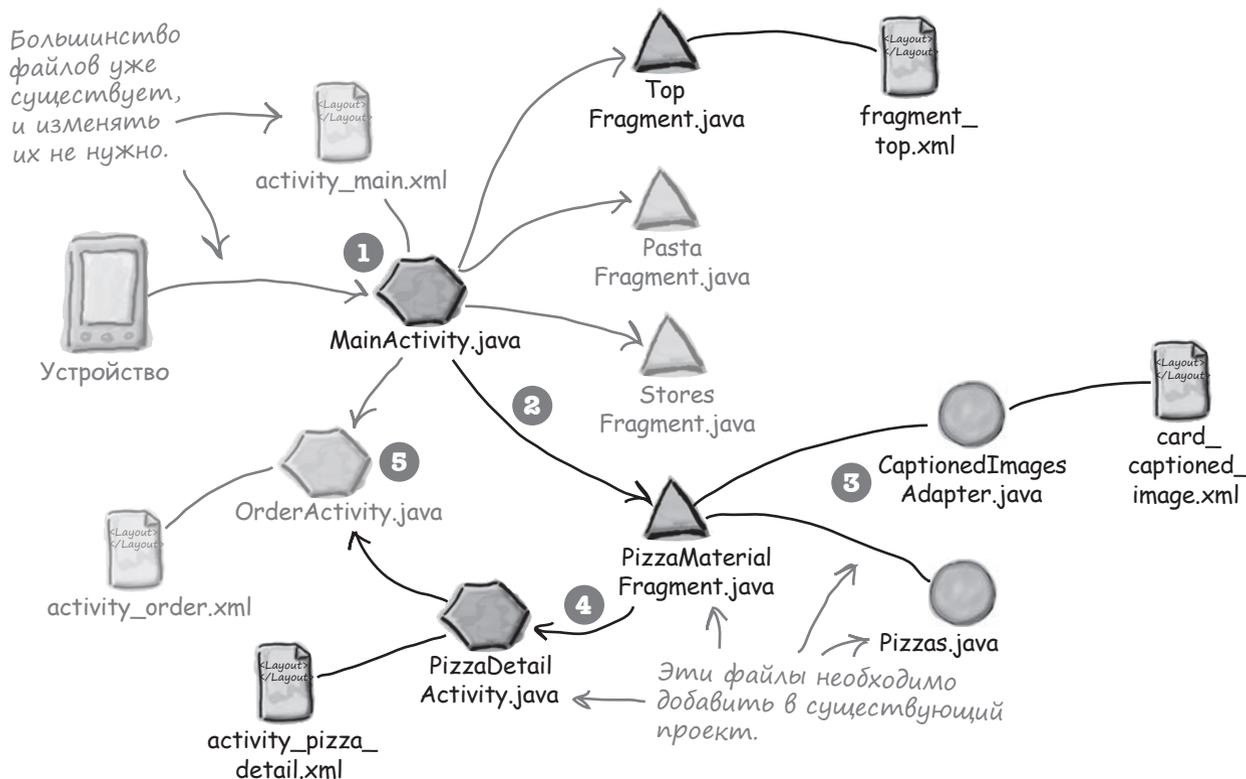
Для любознательных

В материальном оформлении широко применяются трехмерные эффекты. Но не замедляет ли это работу приложения? Для большинства устройств — нет, не замедляет. Представления стараются использовать возможности графического оборудования для отображения теней точно так же, как они бы делали в игре. В результате сгенерированные тени не только выглядят более естественно, но и не требуют лишнего времени на прорисовку. На старых устройствах за каждым представлением вставлялась имитация тени; это требовало дополнительных вычислительных ресурсов и дополнительной памяти. Если вы хотите, чтобы ваше приложение работало на очень старых устройствах, лучше протестировать его на физическом устройстве перед выпуском.

Структура приложения Pizza

Мы собираемся изменить приложение так, чтобы для вывода списка видов пиццы использовались карточки и RecyclerView. Ниже приведена краткая схема структуры приложения и описание его работы:

- 1 При запуске приложения запускается активность **MainActivity**. Активность использует макет *activity_main.xml* и выдвижную панель навигации. Когда пользователь выбирает один из вариантов на панели навигации, приложение отображает соответствующий фрагмент.
- 2 Когда пользователь выбирает вариант **Pizzas**, открывается фрагмент **PizzasMaterialFragment**. **PizzasMaterialFragment** содержит **RecyclerView**.
- 3 **PizzaMaterialFragment** использует адаптер **CaptionedImagesAdapter** для отображения карточек, в которых выводится название и изображение каждого вида пиццы. Карточки определяются в файле *card_captioned_image.xml*. Информация о пицце хранится в файле *Pizzas.java*.
- 4 Когда пользователь выбирает пиццу, подробная информация о ней выводится в **PizzaDetailActivity**.
- 5 Когда пользователь выбирает действие **Create Order** на панели действий **MainActivity** или **PizzaDetailActivity**, открывается активность **OrderActivity**.



Добавление информации о пицце

Начнем с добавления изображений в проект Bits and Pizzas. Загрузите файлы *diavolo.jpg* и *funghi.jpg* по адресу <https://tinyurl.com/HeadFirstAndroid>. Перетащите их в папку *app/src/main/res/drawable-nodpi*. Если среда Android Studio не создала папку за вас, создайте ее самостоятельно. Мы поместим изображения в папку *drawable-nodpi*, потому что будем использовать одни и те же изображения независимо от плотности пикселей экрана. Если хотите, создайте разные версии для разных разрешений и поместите их в подходящую папку *drawable**.



Так как в этой главе мы изменим приложение Pizza, откройте исходный проект Bits and Pizzas в Android Studio.

Добавление класса Pizza

Мы добавим в приложение класс *Pizza*, из которого *RecyclerView* будет получать информацию о пицце. Класс определяет массив с двумя элементами, каждый из которых содержит название и идентификатор ресурса изображения. Добавьте новый класс в пакет *com.hfad.bitsandpizzas* из папки *app/src/main/java* вашего проекта, присвойте ему имя *Pizza* и сохраните изменения:

← Вероятно, в реальном приложении для этого использовалась бы база данных. Здесь для простоты используется класс *Java*.

```
package com.hfad.bitsandpizzas;
```

```
public class Pizza {
    private String name;
    private int imageResourceId;

    public static final Pizza[] pizzas = {
        new Pizza("Diavolo", R.drawable.diavolo),
        new Pizza("Funghi", R.drawable.funghi)
    };

    private Pizza(String name, int imageResourceId) {
        this.name = name;
        this.imageResourceId = imageResourceId;
    }

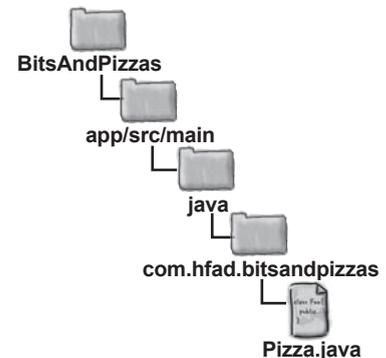
    public String getName() {
        return name;
    }

    public int getImageResourceId() {
        return imageResourceId;
    }
}
```

Каждый объект *Pizza* содержит название и идентификатор ресурса изображения.

Конструктор *Pizza*.

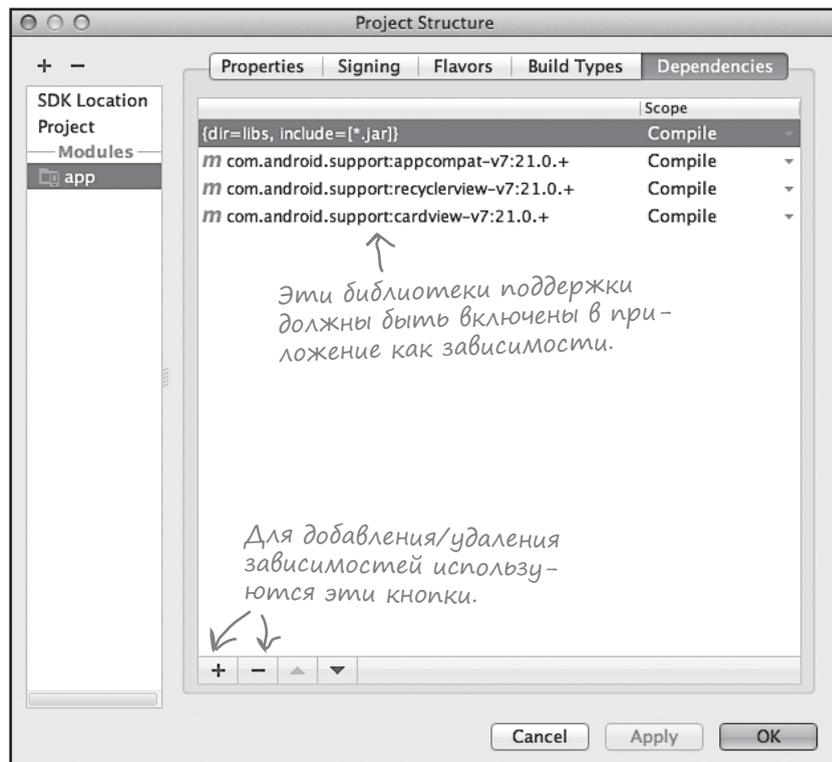
Get-методы для частных переменных.



В приложении будут использоваться *RecyclerView* и карточки, а для них нужны библиотеки поддержки. Сейчас мы их добавим.

Добавление библиотек поддержки

Работа карточек и RecyclerView обеспечивается библиотеками CardView и RecyclerView v7, поэтому эти библиотеки необходимо добавить как зависимости. Для этого выполните команду File→Project Structure, в окне Project Structure выберите вариант app и перейдите на вкладку Dependencies. Добавьте зависимости для библиотек recyclerview-v7 и cardview-v7.



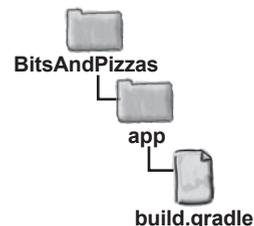
Android Studio сохраняет добавленные зависимости в файле `app/build.gradle`:

```
...
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support.appcompat-v7:21.0.+
    compile 'com.android.support.recyclerview-v7:21.0.+
    compile 'com.android.support.cardview-v7:21.0.+
}
```

Добавление библиотек поддержки в окне приводит к автоматическому обновлению файла `build.gradle`.

При желании вы можете управлять зависимостями приложения, редактируя этот файл вручную. Результат будет таким же, как и при добавлении зависимостей в окне Project Structure.

После добавления библиотек поддержки можно переходить к созданию карточного представления.



Создание представлений CardView

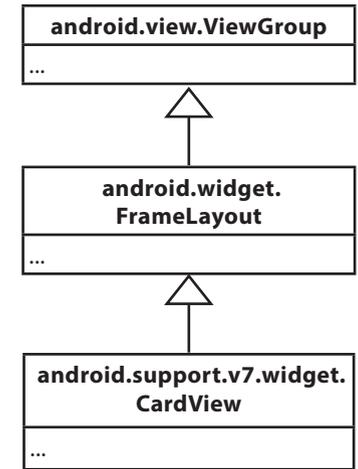
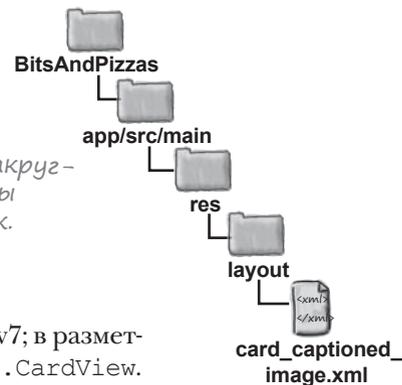
Карточки предоставляют узнаваемый, понятный способ представления основных данных в приложении. В нашем примере это данные о пицце, пасте и магазинах, поэтому мы создадим карточное представление, которое может использоваться для отображения этой информации. Карточные представления включаются в макет — либо уже существующий, либо созданный специально для карточки. Создание нового файла макета позволит использовать карточки внутри RecyclerView.

Так как мы планируем отображать карточки в RecyclerView, в нашем примере будет использоваться отдельный макет. Добавьте в папку `app/src/main/res/layout` новый файл макета с именем `card_captioned_image.xml`. Карточка определяется разметкой следующего вида:

```
<android.support.v7.widget.CardView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    android:id="@+id/card_view"
    android:layout_width="match_parent"
    android:layout_height="200dp"
    android:layout_margin="5dp"
    card_view:cardCornerRadius="4dp">
    ...
</android.support.v7.widget.CardView>
```

Добавляет поддержку CardView.

Создает закругленные углы у карточек.



CardView расширяет класс FrameLayout.

Класс CardView реализуется библиотекой поддержки CardView v7; в разметке указывается полное имя класса `android.support.v7.widget.CardView`. Чтобы карточки имели закругленные углы, следует добавить в элемент пространство имен

```
xmlns:card_view="http://schemas.android.com/apk/res-auto"
```

и присвоить атрибуту `card_view:cardCornerRadius` радиус закругления. Например, атрибут

```
card_view:cardCornerRadius="4dp"
```

задает радиус углового закругления равным 4dp.

Внешний вид карточки определяется добавлением в нее других представлений. В нашем примере карточка содержит изображение и текст. Полная разметка приводится на следующей странице.

Полная разметка card_captioned_image.xml

Ниже приводится полная разметка `card_captioned_image.xml` (в представлении добавлен линейный макет, в котором позиционируется графическое представление и надпись; мы выбрали такой способ, потому что класс `CardView` расширяет `FrameLayout`, а фреймы могут содержать только одно вложенное представление — в данном случае это представление линейного макета):

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v7.widget.CardView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    android:id="@+id/card_view"
    android:layout_width="match_parent"
    android:layout_height="200dp"
    android:layout_margin="5dp"
    card_view:cardCornerRadius="4dp">
```

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical">
```

```
<ImageView android:id="@+id/info_image"
    android:layout_height="0dp"
    android:layout_width="match_parent"
    android:layout_weight="1.0"
    android:scaleType="centerCrop"/>
```

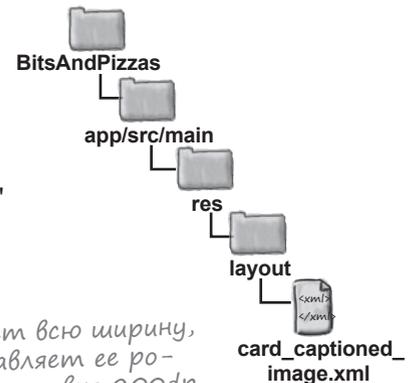
```
<TextView
    android:id="@+id/info_text"
    android:layout_marginLeft="5dp"
    android:layout_marginBottom="5dp"
    android:layout_height="wrap_content"
    android:layout_width="match_parent"/>
```

```
</LinearLayout>
```

```
</android.support.v7.widget.CardView>
```

Карточка занимает всю ширину, которую предоставляет ее родитель, а ее высота равна 200dp.

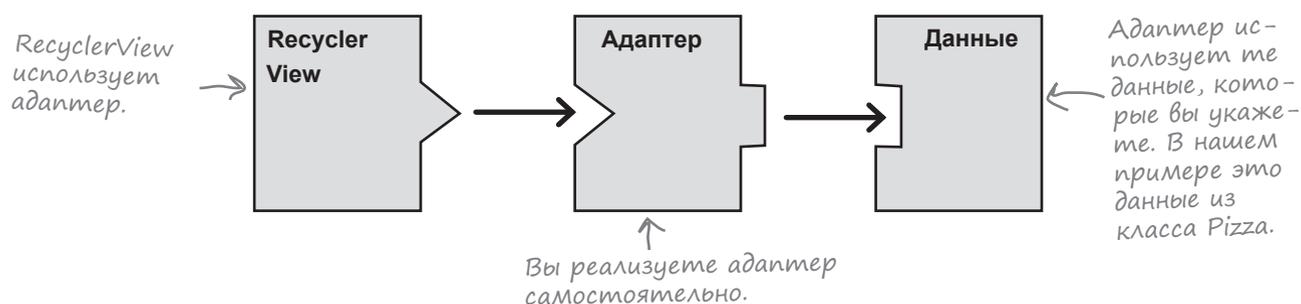
Изображение занимает всю ширину, которую предоставляет ее родитель. Режим `centerCrop` обеспечивает равномерное масштабирование изображения.



Макет карточки может использоваться для любых данных, состоящих из текста и изображения, — таких, как описания пиццы в нашем приложении. Следующий шаг — создание компонента `RecyclerView` для списка карточек.

RecyclerView используем RecyclerView.Adapter

RecyclerView представляет собой усовершенствованную версию спискового представления. Как и списковое представление, RecyclerView выполняет функции контейнера с прокруткой для вывода наборов данных. При этом RecyclerView эффективнее работает для больших наборов данных: этот компонент повторно использует (перерабатывает) представления, которые уходят с экрана, тогда как списковое представление создает новое представление для каждого варианта списка. Как и в случае со списковым представлением, для добавления данных в RecyclerView используется адаптер. К сожалению, RecyclerView не работает со встроенными адаптерами — такими, как адаптеры массивов или курсоров. Вместо этого вам придется создать собственный адаптер, расширяющий класс RecyclerView.Adapter.



Адаптер выполняет две основные функции: он создает все представления, видимые в RecyclerView, и связывает каждое представление с определенным блоком данных. В нашем примере в RecyclerView должен отображаться список карточек, каждая из которых содержит графическое представление и надпись. Это означает, что адаптер должен создать представления для этих данных и заменять их содержимое каждый раз, когда вариант в наборе данных перестает быть видимым. На нескольких ближайших страницах мы займемся созданием адаптера RecyclerView. Адаптер должен выполнять три основные функции:

- 1 **Указать тип данных, с которыми должен работать адаптер.**
В нашем примере адаптер должен использовать карточки, каждая из которых содержит изображение и текст.
- 2 **Создать представления.**
Адаптер должен создать все представления, отображаемые на экране.
- 3 **Связать данные с представлениями.**
Адаптер должен заполнить данными каждое представление, когда оно становится видимым.

Начнем с добавления класса RecyclerView.Adapter в проект.

Создание базового адаптера

Наш адаптер RecyclerView будет называться CaptionedImagesAdapter. Создайте новый класс с именем CaptionedImagesAdapter и приведите его код к следующему виду:

```
package com.hfad.bitsandpizzas;

import android.support.v7.widget.RecyclerView;
import android.view.ViewGroup;

class CaptionedImagesAdapter extends RecyclerView.Adapter<CaptionedImagesAdapter.ViewHolder>{
    //Предоставляет ссылку на представления, используемые в RecyclerView
    public static class ViewHolder extends RecyclerView.ViewHolder {
        //Определение класса ViewHolder
    }

    @Override
    public CaptionedImagesAdapter.ViewHolder onCreateViewHolder(
        ViewGroup parent, int viewType){
        //Создание нового представления
    }

    @Override
    public void onBindViewHolder(ViewHolder holder, int position){
        //Заполнение заданного представления данными
    }

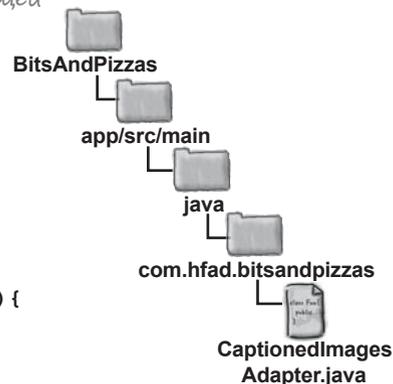
    @Override
    public int getItemCount(){
        //Возвращает количество вариантов в наборе данных
    }
}
```

← Класс RecyclerView реализован в библиотеке поддержки.

← О том, зачем это нужно, рассказано на следующей странице.

↑ Необходимо реализовать эти методы.

↑ И этот метод тоже необходим.



Как видите, CaptionedImagesAdapter расширяет класс RecyclerView.Adapter и реализует его методы getItemCount(), onCreateViewHolder() и onBindViewHolder(). Метод getItemCount() возвращает количество вариантов в наборе данных, метод onCreateViewHolder() используется для заполнения представлений, а метод onBindViewHolder() заполняет представления данными. Эти методы должны переопределяться каждый раз, когда вы создаете собственный адаптер RecyclerView. Также в CaptionedImagesAdapter определяется класс ViewHolder, который сообщает, с какими данными должен работать адаптер.

Определение класса ViewHolder

ViewHolder предоставляет ссылку на представление (или представления) каждого варианта данных в RecyclerView; это своего рода «ячейка» для размещения отображаемых данных. При создании адаптера для RecyclerView необходимо создать ViewHolder внутри адаптера. Для этого вы расширяете класс RecyclerView.ViewHolder и указываете, данные какого типа он должен содержать.

Каждый вариант в RecyclerView представляет собой карточку; следовательно, класс ViewHolder должен содержать CardView:

```
package com.hfad.bitsandpizzas;
```

...

```
import android.support.v7.widget.CardView;
```

```
class CaptionedImagesAdapter extends RecyclerView.Adapter<CaptionedImagesAdapter.ViewHolder>{
```

```
    //Предоставляет ссылку на представления, используемые в RecyclerView
```

```
    public static class ViewHolder extends RecyclerView.ViewHolder {
```

```
        private CardView cardView;
```

```
        public ViewHolder(CardView v) {
```

```
            super(v);
```

```
            cardView = v;
```

```
        }
```

```
    }
```

...

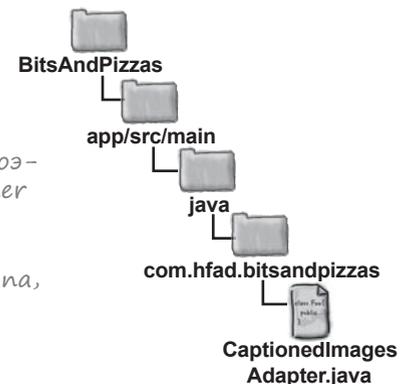
```
}
```

В нашем компоненте RecyclerView должны отображаться карточки, поэтому мы указываем, что ViewHolder содержит представления CardView. Если вы захотите отображать в RecyclerView данные другого типа, определите их здесь.

Класс ViewHolder содержит одно или несколько представлений.



Каждый объект ViewHolder содержит CardView.



При создании реализации ViewHolder необходимо вызвать конструктор суперкласса:

```
super(v);
```

Эта необходимость объясняется тем, что суперкласс ViewHolder включает метаданные, необходимые для правильной работы адаптера (например, позицию варианта в RecyclerView). Итак, мы создали реализацию ViewHolder для хранения карточек. Теперь можно переходить к отображению карточек в RecyclerView.

Создание ViewHolder

Компонент `RecyclerView` поддерживает фиксированный набор объектов `ViewHolder` — «ячеек» для представлений, отображаемых в списке на экране. Количество таких ячеек зависит от размера экрана и от того, сколько места занимает каждый вариант. Чтобы компонент `RecyclerView` мог определить, сколько объектов `ViewHolder` ему потребуется, ему необходимо сообщить, какой макет должен использоваться для каждой ячейки, — эта информация передается в методе `onCreateViewHolder()` адаптера.

В момент создания компонент `RecyclerView` строит свой набор объектов `ViewHolder`, многократно вызывая метод `onCreateViewHolder()` адаптера, пока не будут созданы все необходимые объекты. Метод `onCreateViewHolder()` получает два параметра: родительский объект `ViewGroup` (сам компонент `RecyclerView`) и параметр типа `int` с именем `viewType`. Он используется в тех случаях, когда вы хотите отображать разные представления для разных вариантов в списке.

В нашем примере нужно создать объекты `ViewHolder`, содержащие представления карточек на базе макета `card_captioned_image.xml`. Эта задача решается следующим кодом:

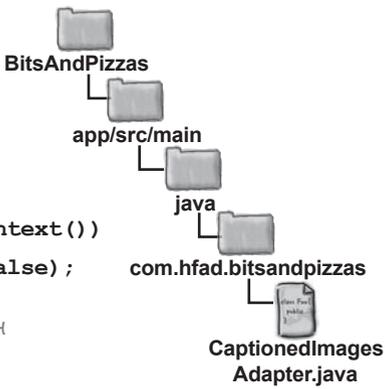
```
...
import android.view.LayoutInflater;

class CaptionedImagesAdapter extends RecyclerView.Adapter<CaptionedImagesAdapter.ViewHolder>{

    ...

    @Override
    public CaptionedImagesAdapter.ViewHolder onCreateViewHolder(
        ViewGroup parent, int viewType){
        CardView cv = (CardView) LayoutInflater.from(parent.getContext())
        .inflate(R.layout.card_captioned_image, parent, false);
        return new ViewHolder(cv);
    }
}
```

↑
Указать, какой макет должен использоваться для содержимого ViewHolder.



The diagram illustrates the project structure. It starts with a folder named 'BitsAndPizzas'. Inside it is a folder 'app/src/main'. Within 'app/src/main' is a folder 'java'. Inside 'java' is a folder 'com.hfad.bitsandpizzas'. Finally, inside 'com.hfad.bitsandpizzas' is a file named 'CaptionedImagesAdapter.java'.

Итак, наш адаптер может создавать объекты `ViewHolder` в `RecyclerView`; теперь нужно научить его заполнять карточки данными.

Каждая карточка содержит изображение и текст

Каждый раз, когда при прокрутке RecyclerView появляется новый элемент, компонент RecyclerView берет из резерва один из объектов ViewHolder и вызывает метод `onBindViewHolder()` для связывания данных с его содержимым. Код метода `onBindViewHolder()` должен задать содержимое представлений в ViewHolder, чтобы они соответствовали данным.

В нашем случае ViewHolder содержит представления карточек, которые необходимо заполнить изображениями и текстом. Для этого в адаптер добавляется конструктор, через который будут передаваться данные. После этого метод `onBindViewHolder()` сможет связать данные с карточками.

Создание конструктора

Компонент RecyclerView должен передать адаптеру тексты названий и идентификаторы изображений, поэтому мы добавим в класс адаптера конструктор, получающий эти данные в параметрах. Массивы сохраняются в переменных экземпляра. Также количество названий, переданных адаптеру, будет использоваться для определения количества объектов в наборе данных:

...

```
class CaptionedImagesAdapter extends RecyclerView.Adapter<CaptionedImagesAdapter.ViewHolder>{
```

```
    private String[] captions;
```

```
    private int[] imageIds;
```

← Переменные для хранения информации о пицце.

...

```
    public CaptionedImagesAdapter(String[] captions, int[] imageIds){
```

```
        this.captions = captions;
```

```
        this.imageIds = imageIds;
```

```
    }
```

↑ ↑ Данные передаются адаптеру через конструктор.

```
    @Override
```

```
    public int getItemCount(){
```

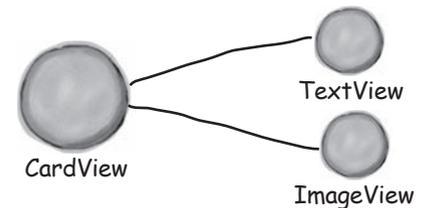
```
        return captions.length;
```

← Длина массива captions равна количеству элементов данных в RecyclerView.

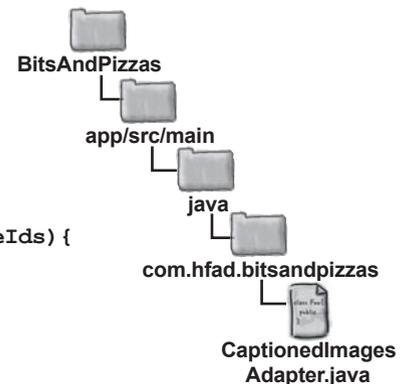
```
    }
```

}

Итак, теперь адаптер получает данные. Чтобы он смог отобразить их в RecyclerView, мы напишем метод `onBindViewHolder()`.



Каждый компонент CardView содержит компоненты TextView и ImageView, которые необходимо заполнить названием и изображением каждого вида пиццы.



Добавление данных в карточки

Метод `onBindViewHolder()` вызывается каждый раз, когда компоненту `RecyclerView` потребуется вывести данные в `ViewHolder`. Метод получает два параметра: объект `ViewHolder`, с которым должны быть связаны данные, и позиция связываемых данных в наборе. Итак, карточку требуется заполнить данными. Карточка содержит два представления: графическое представление с идентификатором `info_image` и надпись с идентификатором `info_text`. Мы заполним их данными из массивов `captions` и `imageIds`.

Следующий код решает эту задачу:

...

```
import android.widget.ImageView;
import android.widget.TextView;
import android.graphics.drawable.Drawable;
```

Эти дополнительные классы используются в коде.

```
class CaptionedImagesAdapter extends RecyclerView.Adapter<CaptionedImagesAdapter.ViewHolder>{
```

```
private String[] captions;
private int[] imageIds;
```

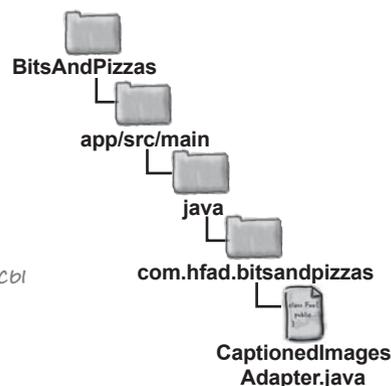
Переменные для названий и идентификаторов ресурсов изображений разных видов пиццы.

...

```
public void onBindViewHolder(ViewHolder holder, int position){
    CardView cardView = holder.cardView;
    ImageView imageView = (ImageView)cardView.findViewById(R.id.info_image);
    Drawable drawable = cardView.getResources().getDrawable(imageIds[position]);
    imageView.setImageDrawable(drawable);
    imageView.setContentDescription(captions[position]);
    TextView textView = (TextView)cardView.findViewById(R.id.info_text);
    textView.setText(captions[position]);
}
```

*↑
Название выводится в компоненте TextView.*

Изображение выводится в графическом представлении ImageView.



Вот и весь код, обеспечивающий работу адаптера. Полный код приведен на следующей странице.

Полный код CaptionedImagesAdapter.java

```
package com.hfad.bitsandpizzas;
```

```
import android.support.v7.widget.RecyclerView;
import android.view.LayoutInflater;
import android.view.ViewGroup;
import android.support.v7.widget.CardView;
import android.widget.ImageView;
import android.widget.TextView;
import android.graphics.drawable.Drawable;
```

Используемые классы.

```
class CaptionedImagesAdapter extends RecyclerView.Adapter<CaptionedImagesAdapter.ViewHolder>{
```

```
    private String[] captions;
    private int[] imageIds;
```

```
    public static class ViewHolder extends RecyclerView.ViewHolder{
        private CardView cardView;
        public ViewHolder(CardView v) {
            super(v);
            cardView = v;
        }
    }
```

Каждый объект ViewHolder отображает CardView.

```
    public CaptionedImagesAdapter(String[] captions, int[] imageIds){
        this.captions = captions;
        this.imageIds = imageIds;
    }
```

Передать данные адаптеру в конструкторе.

```
    @Override
    public CaptionedImagesAdapter.ViewHolder onCreateViewHolder(ViewGroup parent, int viewType){
        CardView cv = (CardView) LayoutInflater.from(parent.getContext())
            .inflate(R.layout.card_captioned_image, parent, false);
        return new ViewHolder(cv);
    }
```

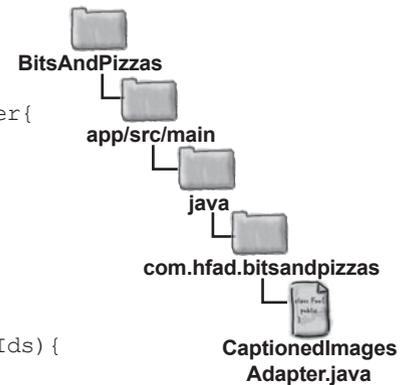
Использовать макет для представлений CardView.

```
    public void onBindViewHolder(ViewHolder holder, int position){
        CardView cardView = holder.cardView;
        ImageView imageView = (ImageView) cardView.findViewById(R.id.info_image);
        Drawable drawable = cardView.getResources().getDrawable(imageIds[position]);
        imageView.setImageDrawable(drawable);
        imageView.setContentDescription(captions[position]);
        TextView textView = (TextView) cardView.findViewById(R.id.info_text);
        textView.setText(captions[position]);
    }
```

Заполнить данными компоненты ImageView и TextView внутри CardView.

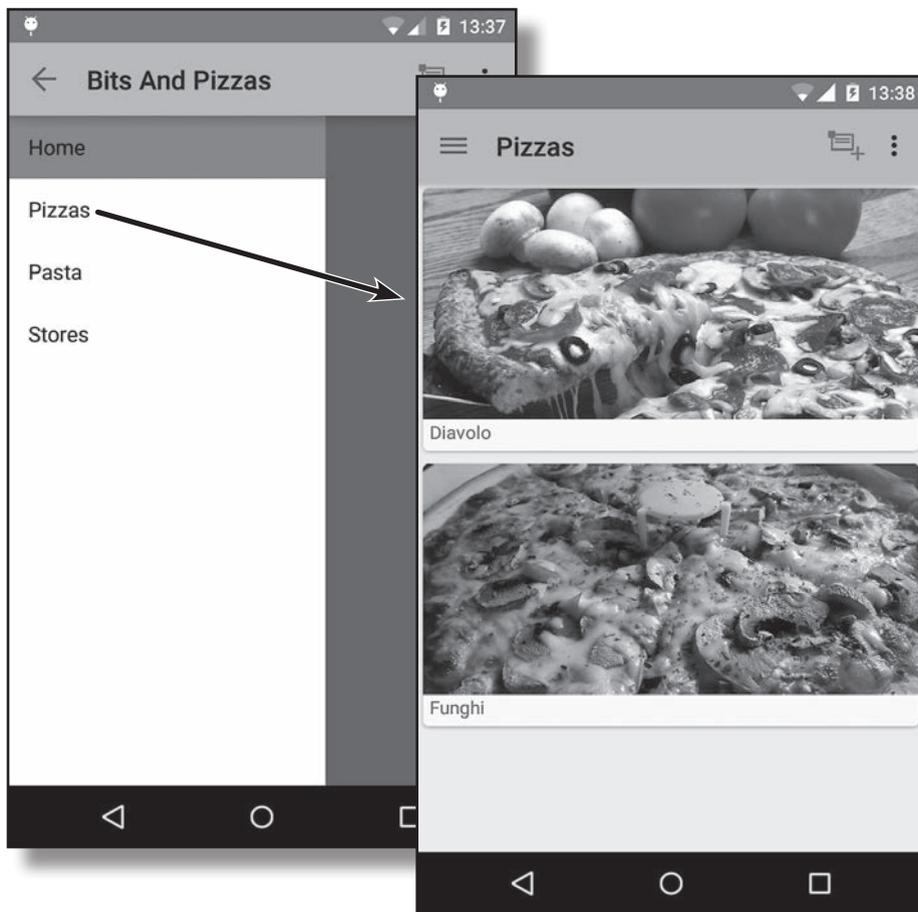
```
    @Override
    public int getItemCount(){
        return captions.length;
    }
}
```

Количество элементов данных.



Создание RecyclerView

Итак, мы создали карточное представление и адаптер. Следующее, что нужно сделать, — создать компонент `RecyclerView`, который будет передавать адаптеру информацию о пицце, чтобы адаптер мог заполнить представления карточек. Мы поместим компонент `RecyclerView` в новый фрагмент, чтобы он отображался в `MainActivity` при выборе пользователем команды `Pizzas` на выдвжной панели:



Начнем с создания фрагмента. Добавьте в проект новый пустой фрагмент, присвойте ему имя `PizzaMaterialFragment`, а макету — имя `fragment_pizza_material`. На следующей странице мы добавим в макет компонент `RecyclerView`.

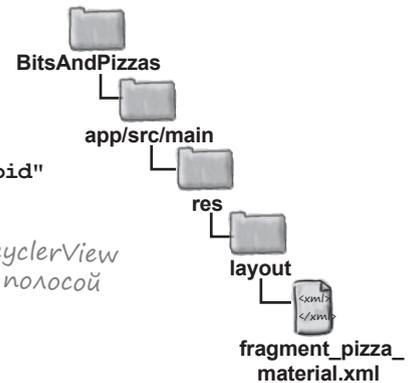
Добавление RecyclerView в макет

Для включения в макет компонента RecyclerView используется элемент `<android.support.v7.widget.RecyclerView>`.

В приведенной ниже разметке `fragment_pizza_material.xml` определяется RecyclerView с идентификатором `pizza_recycler`:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v7.widget.RecyclerView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/pizza_recycler"
    android:scrollbars="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

↑ Определяет RecyclerView с вертикальной полосой прокрутки.



Атрибут `android:scrollbars` добавляет в RecyclerView полосы прокрутки. Мы присвоили атрибуту значение "vertical", потому что компонент RecyclerView будет использоваться для вывода вертикального списка с вертикальной прокруткой. После добавления компонента RecyclerView в файл `fragment_pizza_material.xml` необходимо добавить в `PizzaMaterialFragment.java` код, управляющий его поведением.

Использование адаптера

В коде `PizzaMaterialFragment.java` необходимо настроить RecyclerView для использования адаптера. Мы должны сообщить адаптеру, какие данные следует использовать, при помощи конструктора адаптера, а затем использовать метод `setAdapter()` класса RecyclerView для назначения адаптера:



Код `PizzaMaterialFragment.java` приведен на следующей странице.

Код `PizzaMaterialFragment.java`

Ниже приведен код `PizzaMaterialFragment.java` (он создает экземпляр `CaptionedImagesAdapter`, приказывает использовать названия и изображения пиццы в качестве данных, после чего назначает адаптер компоненту `RecyclerView`):

```
package com.hfad.bitsandpizzas;

import android.app.Fragment;
import android.os.Bundle;
import android.support.v7.widget.RecyclerView;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

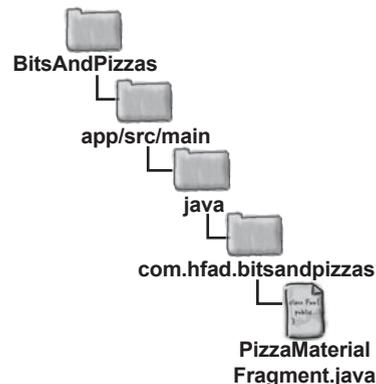
public class PizzaMaterialFragment extends Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        RecyclerView pizzaRecycler = (RecyclerView)inflater.inflate(
            R.layout.fragment_pizza_material, container, false);

        String[] pizzaNames = new String[Pizza.pizzas.length];
        for (int i = 0; i < pizzaNames.length; i++) {
            pizzaNames[i] = Pizza.pizzas[i].getName();
        }

        int[] pizzaImages = new int[Pizza.pizzas.length];
        for (int i = 0; i < pizzaImages.length; i++) {
            pizzaImages[i] = Pizza.pizzas[i].getImageResourceId();
        }

        CaptionedImagesAdapter adapter = new CaptionedImagesAdapter(pizzaNames, pizzaImages);
        pizzaRecycler.setAdapter(adapter);
        return pizzaRecycler;
    }
}
```



Используемые классы.

Использовать макет с предыдущей страницы.

Названия пиццы добавляются в массив строк, а изображения — в массив с элементами `int`.

Передать массивы адаптеру.

Осталось сделать еще один шаг: указать, как должны размещаться представления в `RecyclerView`.

Размещение представлений в RecyclerView

Одним из преимуществ RecyclerView по сравнению с традиционными списковыми представлениями является большая гибкость в размещении содержимого. Списковое представление размещает свои варианты в виде одного вертикального списка, а RecyclerView предоставляет больше возможностей: представления можно разместить в виде линейного списка, таблицы или неравномерной таблицы. Для размещения представлений используется объект `LayoutManager`. Конкретный способ размещения определяется выбранной разновидностью `LayoutManager`:



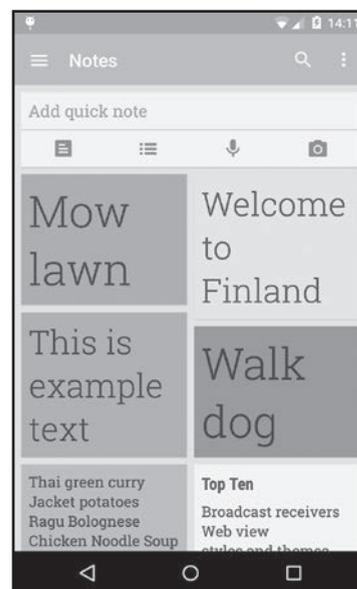
LinearLayoutManager

Варианты образуют вертикальный или горизонтальный список.



GridLayoutManager

Варианты образуют таблицу.



StaggeredGridLayoutManager

Варианты имеют разные размеры и образуют неравномерную таблицу.

На следующей странице мы покажем, как выбрать используемую разновидность `LayoutManager`.

Выбор способа размещения

Способ размещения задается следующей парой строк кода:

```
LinearLayoutManager layoutManager = new LinearLayoutManager(getActivity());  
pizzaRecycler.setLayoutManager(layoutManager);
```

Этот код приказывает компоненту RecyclerView использовать объект LinearLayoutManager, чтобы все представления в нем отображались в виде списка:

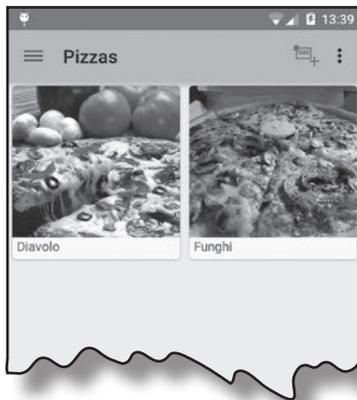


← При использовании объекта LinearLayoutManager варианты образуют линейный список.

↑
Здесь должен передаваться объект Context. Если вы используете этот код в активности, вместо getActivity() передавайте this.

Объекты LayoutManager позволяют легко изменить внешний вид RecyclerView. Например, чтобы переключиться с линейного списка на таблицу, достаточно перевести код на использование объекта GridLayoutManager:

```
GridLayoutManager layoutManager = new GridLayoutManager(getActivity(), 2);  
pizzaRecycler.setLayoutManager(layoutManager);
```



← При использовании объекта GridLayoutManager варианты образуют таблицу.

↑
Это означает, что таблица GridLayoutManager состоит из двух столбцов.

Полный код PizzaMaterialFragment.java

Полный код `PizzaMaterialFragment.java` выглядит так:

```

package com.hfad.bitsandpizzas;

import android.app.Fragment;
import android.os.Bundle;
import android.support.v7.widget.LinearLayoutManager;
import android.support.v7.widget.RecyclerView;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class PizzaMaterialFragment extends Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        RecyclerView pizzaRecycler = (RecyclerView)inflater.inflate(
            R.layout.fragment_pizza_material, container, false);

        String[] pizzaNames = new String[Pizza.pizzas.length];
        for (int i = 0; i < pizzaNames.length; i++) {
            pizzaNames[i] = Pizza.pizzas[i].getName();
        }

        int[] pizzaImages = new int[Pizza.pizzas.length];
        for (int i = 0; i < pizzaImages.length; i++) {
            pizzaImages[i] = Pizza.pizzas[i].getImageResourceId();
        }

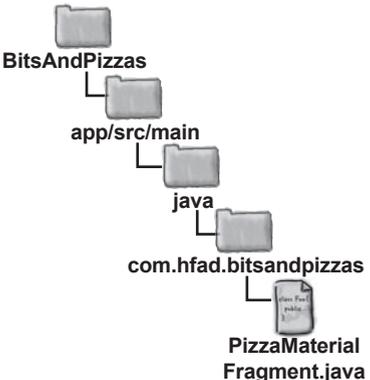
        CaptionedImagesAdapter adapter = new CaptionedImagesAdapter(pizzaNames, pizzaImages);
        pizzaRecycler.setAdapter(adapter);
LinearLayoutManager layoutManager = new LinearLayoutManager(getActivity());
pizzaRecycler.setLayoutManager(layoutManager);
        return pizzaRecycler;
    }
}

```

Мы используем этот класс, его необходимо импортировать.

Весь этот код остается неизменным.

Чтобы карточки отображались в линейном списке, используем объект `LinearLayoutManager`.



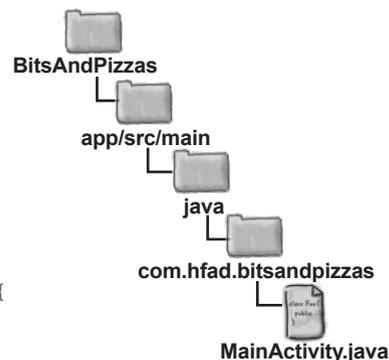
Работа над кодом `RecyclerView` закончена; изменим `MainActivity` так, чтобы эта активность отображалась при выборе пользователем варианта `Pizzas` на выдвигной панели.

Использование нового фрагмента *PizzaMaterialFragment* в *MainActivity*

Когда пользователь выбирает вариант *Pizzas*, отображается реализация *ListFragment* с именем *PizzaFragment*. Чтобы вместо нее выводился фрагмент *PizzaMaterialFragment*, необходимо заменить все упоминания *PizzaFragment* в коде *MainActivity* на *PizzaMaterialFragment*. *PizzaFragment* используется в *MainActivity.java* два раза: в методах *onCreate()* и *selectItem()*. Отредактируйте эти строки, чтобы в них использовался фрагмент *PizzaMaterialFragment*:

```
package com.hfad.bitsandpizzas;
...
public class MainActivity extends Activity {
...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        getFragmentManager().addOnBackStackChangedListener(
            new FragmentManager.OnBackStackChangedListener() {
                public void onBackStackChanged() {
                    ...
                    if (fragment instanceof PizzaFragment) {
                    if (fragment instanceof PizzaMaterialFragment) {
                        currentPosition = 1;
                    }
                    ...
                }
            }
        );
    }

    private void selectItem(int position) {
        ...
        switch(position) {
            case 1:
                fragment = new PizzaFragment();
                fragment = new PizzaMaterialFragment();
                break;
            ...
        }
    }
}
```

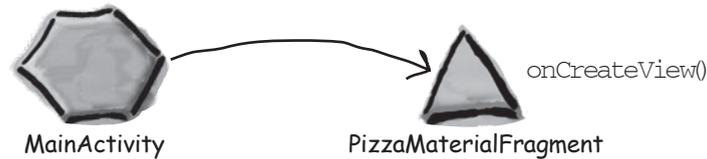


Вместо *PizzaFragment* здесь будет использоваться новый фрагмент *PizzaMaterialFragment*. В результате когда пользователь выбирает вариант *Pizzas* на выдвигной панели, на экране появляется новый компонент *RecyclerView*.

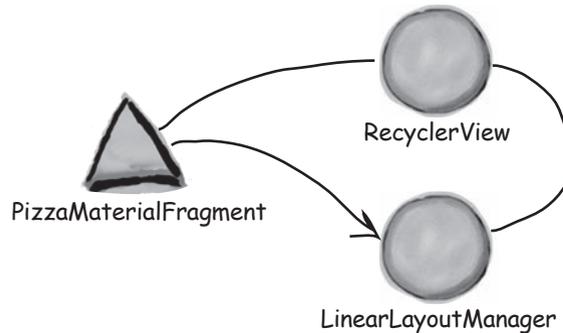
Прежде чем запускать приложение, давайте проанализируем, как работает только что написанный нами код.

Что происходит при выполнении кода

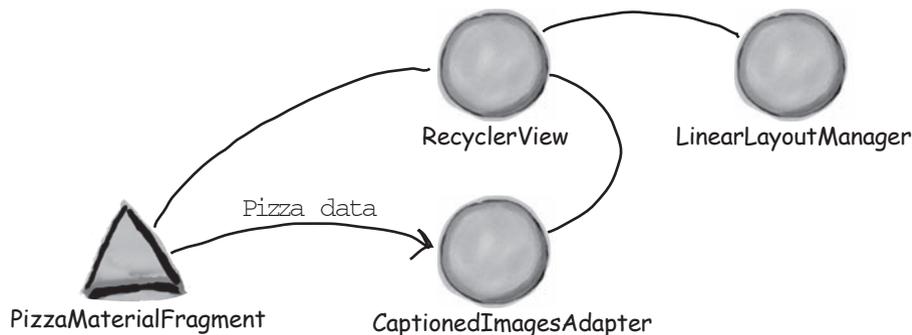
- 1 **Пользователь выбирает вариант Pizzas на выдвигной панели.**
Выполняется код MainActivity, отображающий PizzaMaterialFragment, и выполняется метод onCreateView() фрагмента PizzaMaterialFragment.



- 2 **Метод onCreateView() класса PizzaMaterialFragment создает объект LinearLayoutManager и назначает его RecyclerView.**
Выбор LinearLayoutManager означает, что представления будут отображаться в виде списка. Так как RecyclerView имеет вертикальную полосу прокрутки, элементы списка будут отображаться по вертикали.

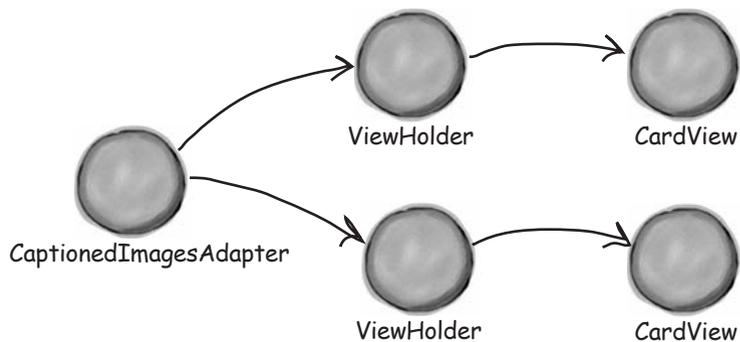


- 3 **Метод onCreateView() класса PizzaMaterialFragment создает новый объект CaptionedImagesAdapter.**
Он передает названия и изображения пиццы адаптеру через конструктор и назначает адаптер компоненту RecyclerView.

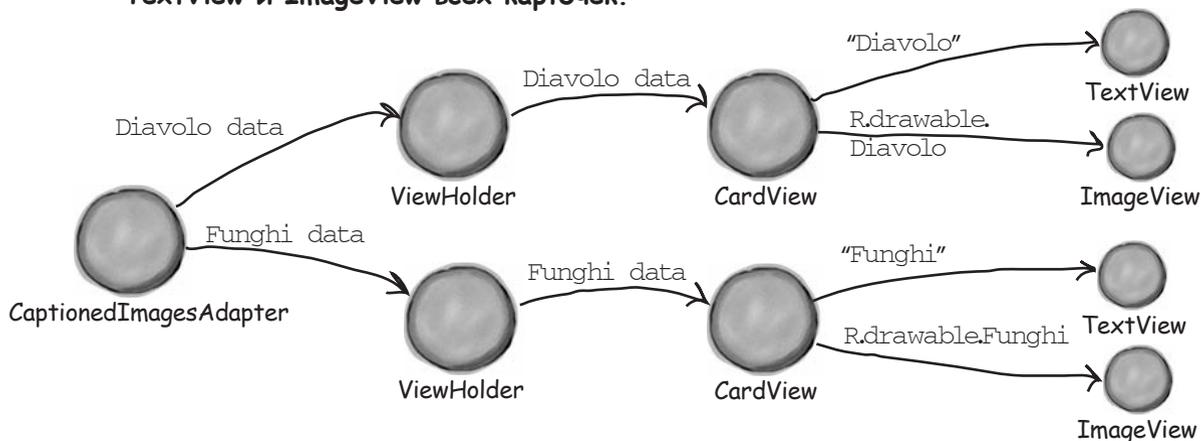


История продолжается

- 4 **Адаптер создает объект ViewHolder для каждого представления CardView, которое должно отображаться в списке RecyclerView.**



- 5 **Адаптер связывает названия и изображения пиццы с компонентами TextView и ImageView всех карточек.**

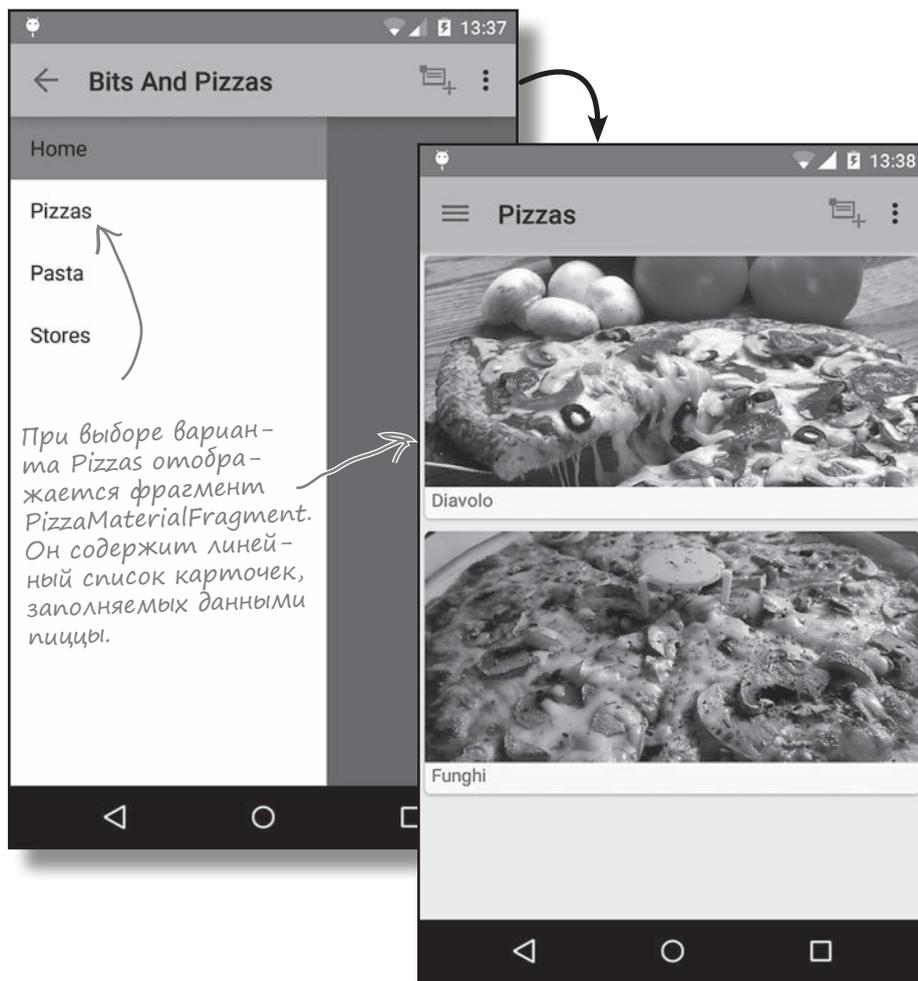


А теперь запустим приложение и посмотрим, как оно выглядит.



Тест-драйв

Запустите приложение, откройте выдвижную панель и выберите вариант Pizzas.



`RecyclerView` отображает линейный список карточных представлений. Каждое такое представление содержит данные определенного вида пиццы.

упражнение



Развлечения с Магнитами

При помощи магнитов на этой и следующей странице создайте новый компонент RecyclerView для разных видов пасты. Компонент RecyclerView должен содержать линейный список карточных представлений, каждое из которых содержит название и изображение определенного вида пасты.

package com.hfad.bitsandpizzas; ← Код класса Pasta.

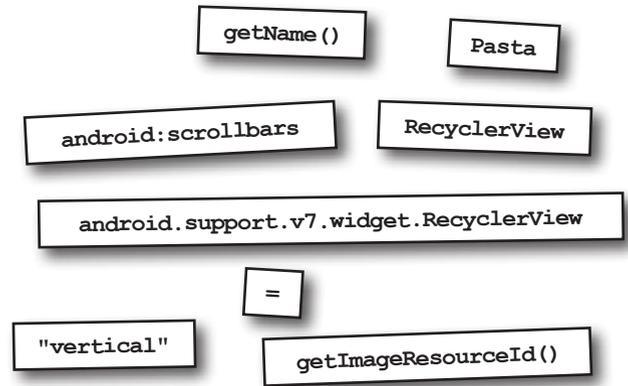
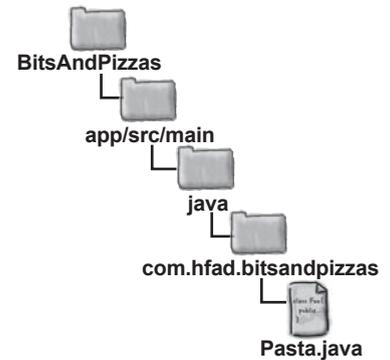
```
public class Pasta {
    private String name;
    private int resourceId;

    public static final .....[] pastas = {
        new Pasta("Spaghetti Bolognese", R.drawable.spag_bol),
        new Pasta("Lasagne", R.drawable.lasagne)
    };

    private Pasta(String name, int resourceId) {
        this.name = name;
        this.imageResourceId = resourceId;
    }

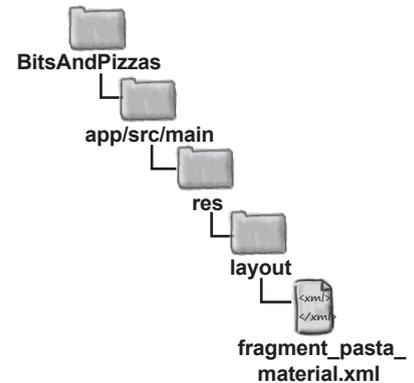
    public String ..... {
        return name;
    }

    public int ..... {
        return resourceId;
    }
}
```



Разметка макета. ↘

```
< .....
xmlns:android="http://schemas.android.com/apk/res/android"
android:id="@+id/pasta_recycler"
.....
android:layout_width="match_parent"
android:layout_height="match_parent"/>
```





Развлечения с магнитами. Решение

При помощи магнитов на этой и следующей странице создайте новый компонент RecyclerView для разных видов пасты. Компонент RecyclerView должен содержать линейный список карточных представлений, каждое из которых содержит название и изображение определенного вида пасты.

```
package com.hfad.bitsandpizzas;

public class Pasta {
    private String name;
    private int resourceId;

    public static final Pasta [] pastas = {
        new Pasta("Spaghetti Bolognese", R.drawable.spag_bol),
        new Pasta("Lasagne", R.drawable.lasagne)
    };

    private Pasta(String name, int resourceId) {
        this.name = name;
        this.resourceId = resourceId;
    }

    public String getName() {
        return name;
    }

    public int getImageResourceId() {
        return resourceId;
    }
}
```

Массив объектов Pasta.

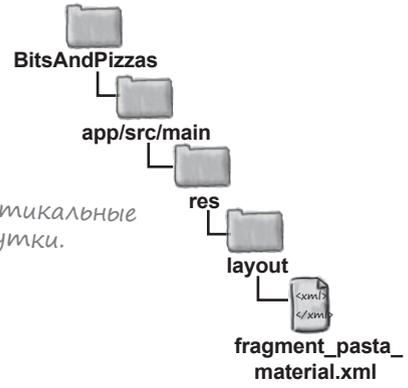
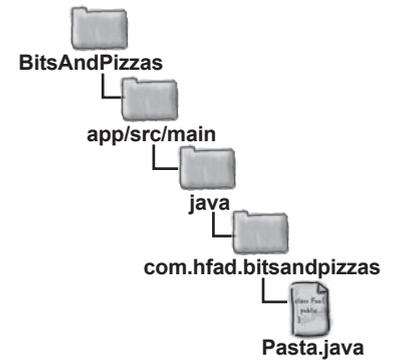
Эти методы используются в коде PastaMaterialFragment.java.

RecyclerView
Лишний магнит.

В макет добавляется компонент RecyclerView.

```
< android.support.v7.widget.RecyclerView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/pasta_recycler"
    android:scrollbars = "vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

Добавить вертикальные полосы прокрутки.



```
public class PastaMaterialFragment extends Fragment {
```

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
    RecyclerView pastaRecycler = (RecyclerView)inflater.inflate(
```

Использовать этот макет.

R.layout.fragment_pasta_material

```
....., container, false);
```

```
String[] pastaNames = new String[Pasta.pastas.length];
for (int i = 0; i < pastaNames.length; i++) {
    pastaNames[i] = Pasta.pastas[i].getName();
}
```

```
int[] pastaImages = new int[Pasta.pastas.length];
for (int i = 0; i < pastaImages.length; i++) {
    pastaImages[i] = Pasta.pastas[i].getImageResourceId();
}
```

```
.....adapter =
new ..... (pastaNames, .....);
pastaRecycler.setAdapter(adapter);
```

```
.....layoutManager = new ..... (getActivity());
pastaRecycler.setLayoutManager(layoutManager);
return pastaRecycler;
```

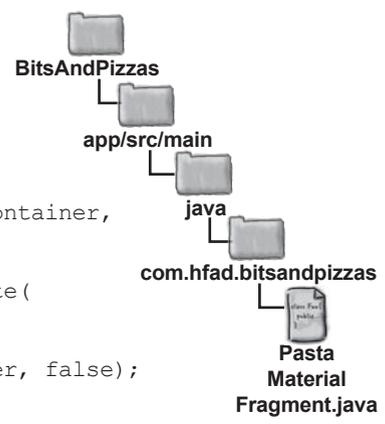
```
}
```

Используем класс CaptionedImagesAdapter, написанный ранее.

Названия и изображения пасты передаются адаптеру.

Использовать LinearLayoutManager для вывода карточек в линейном списке.

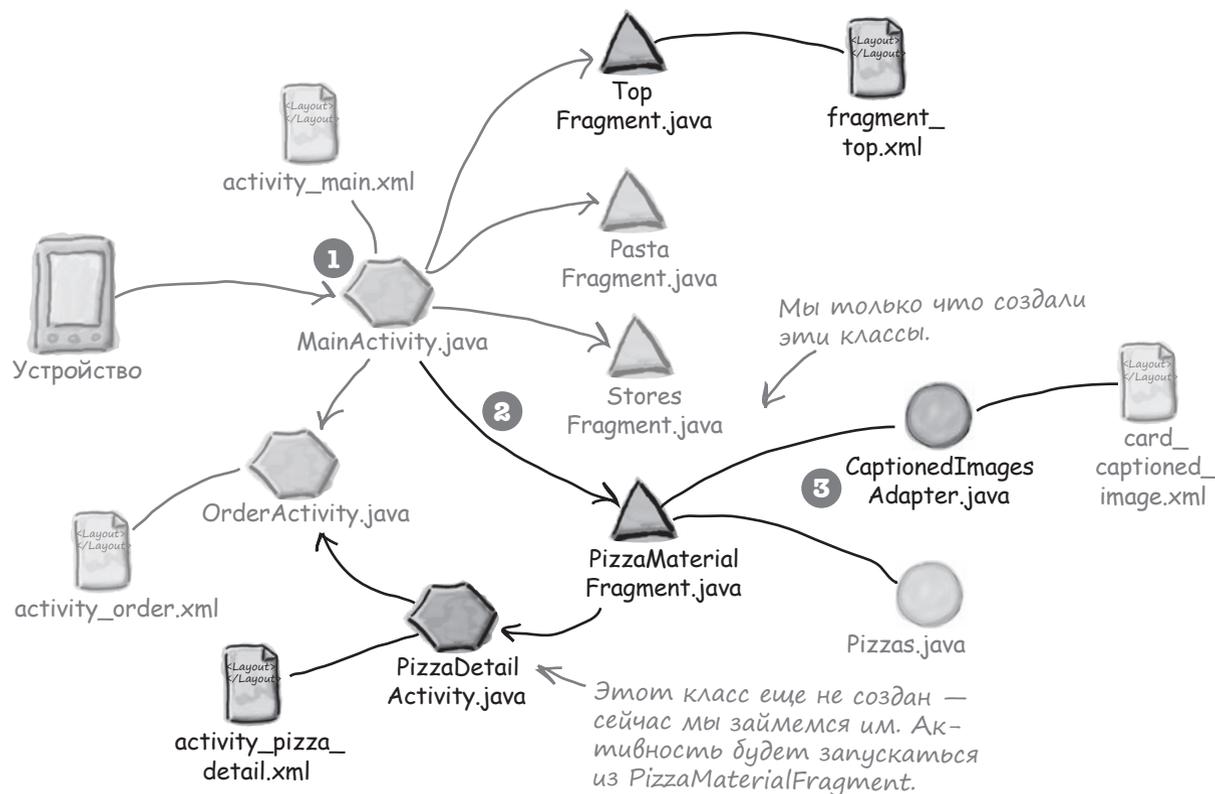
Эти магниты не понадобились.



Что было сделано

Вспомним, что было сделано к настоящему моменту:

- 1 При запуске приложения открывается активность MainActivity.**
Активность использует макет `activity_main.xml` и содержит выдвижную панель. Когда пользователь выбирает один из вариантов на выдвижной панели, приложение отображает соответствующий фрагмент.
- 2 Когда пользователь выбирает вариант Pizzas, отображается фрагмент PizzasMaterialFragment.**
`PizzasMaterialFragment` содержит компонент `RecyclerView`.
- 3 PizzaMaterialFragment использует адаптер CaptionedImagesAdapter для отображения карточек с названиями и изображениями разных видов пиццы.**
Карточки определяются в макете `card_captioned_image.xml`.



Следующее, что нужно сделать, — научить `RecyclerView` реагировать на щелчки, чтобы при выборе одного из вариантов открывалась активность `PizzaDetailActivity`. В `PizzaDetailActivity` выводится подробная информация о пицце, выбранной пользователем. Сейчас мы займемся созданием `PizzaDetailActivity`.

Создание PizzaDetailActivity

Активность `PizzaDetailActivity` выводит название пиццы, выбранной пользователем, вместе с ее изображением. Создайте пустую активность с именем “`PizzaDetailActivity`”, макетом “`activity_pizza_detail`” и заголовком “`Pizza Detail`”. Затем включите в файл `activity_pizza_detail.xml` следующую разметку, которая добавляет в макет надпись и графическое представление для вывода подробной информации о пицце:

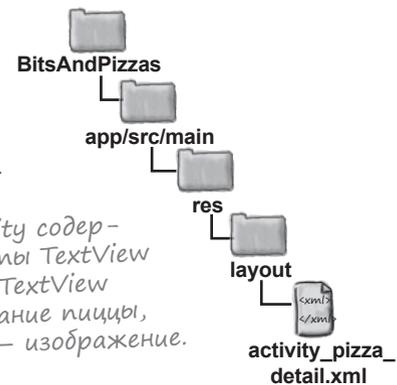
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.hfad.bitsandpizzas.PizzaDetailActivity">

    <TextView
        android:id="@+id/pizza_text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge" />

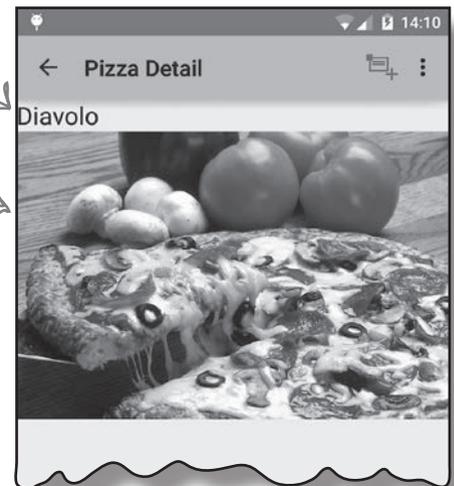
    <ImageView
        android:id="@+id/pizza_image"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:adjustViewBounds="true" />

</LinearLayout>
```

PizzaDetailActivity содержит компоненты `TextView` и `ImageView`. В `TextView` выводится название пиццы, а в `ImageView` — изображение.



В списке на следующей странице перечислены основные функции, которые должны выполняться в коде `PizzaDetailActivity.java`.



Что должен делать код `PizzaDetailActivity.java`

Активность `PizzaDetailActivity.java` должна решать несколько задач:

- ★ `PizzaDetailActivity` создается прежде всего для вывода названия и изображения пиццы, выбранной пользователем. Для этого мы извлекаем идентификатор выбранной пиццы из интента, запустившего активность, и передаем его `PizzaDetailActivity` из `PizzaMaterialFragment`, когда пользователь выбирает один из видов пиццы в `RecyclerView`.
- ★ В главе 9 мы создали файл ресурсов меню с описанием элементов, которые должны отображаться на панели действий. Для добавления этих элементов на панель действий `PizzaDetailActivity` будет использоваться метод `onCreateOptionsMenu()`.
- ★ Файл ресурсов меню описывает действие передачи информации `Share`. Мы добавим для действия `Share` интент, который будет передавать название пиццы, выбранной пользователем.
- ★ Файл ресурсов меню также описывает действие `Create Order`. При выборе этого действия пользователем запускается активность `OrderActivity`.
- ★ Кнопка `Вверх` активности `PizzaDetailActivity` должна быть настроена так, чтобы при нажатии пользователь возвращался к `MainActivity`.

Обновление `AndroidManifest.xml`

Начнем с обновления файла `AndroidManifest.xml`: необходимо указать, что `MainActivity` является родителем `PizzaDetailActivity`. Это означает, что при нажатии кнопки `Вверх` на панели действий `PizzaDetailActivity` будет отображаться `MainActivity`:

```
<activity
    android:name=".PizzaDetailActivity"
    android:label="@string/title_activity_pizza_detail"
    android:parentActivityName=".MainActivity">
</activity>
```

MainActivity является родителем PizzaDetailActivity.



Когда это будет сделано, можно переходить к программированию реакции `RecyclerView` на щелчки.

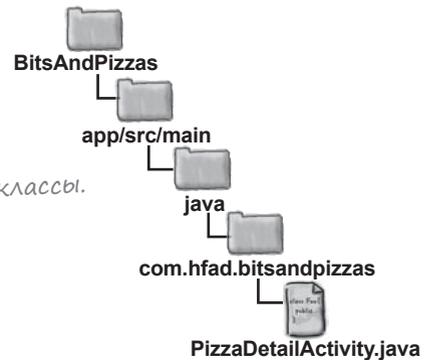
Код PizzaDetailActivity.java

Ниже приведен полный код *PizzaDetailActivity.java* (если он покажется слишком большим, не беспокойтесь — весь этот код вы уже видели ранее):

```
package com.hfad.bitsandpizzas;
```

```
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.ImageView;
import android.widget.ShareActionProvider;
import android.widget.TextView;
```

Используемые классы.



```
public class PizzaDetailActivity extends Activity {
```

```
    private ShareActionProvider shareActionProvider;
```

```
    public static final String EXTRA_PIZZANO = "pizzaNo";
```

```
    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_pizza_detail);
```

```
        //Включение кнопки Вверх
```

```
        getActionBar().setDisplayHomeAsUpEnabled(true);
```

```
        //Вывод подробной информации о пицце
```

```
        int pizzaNo = (Integer) getIntent().getExtras().get(EXTRA_PIZZANO);
```

```
        String pizzaName = Pizza.pizzas[pizzaNo].getName();
```

```
        TextView textView = (TextView) findViewById(R.id.pizza_text);
```

```
        textView.setText(pizzaName);
```

```
        int pizzaImage = Pizza.pizzas[pizzaNo].getImageResourceId();
```

```
        ImageView imageView = (ImageView) findViewById(R.id.pizza_image);
```

```
        imageView.setImageDrawable(getResources().getDrawable(pizzaImage));
```

```
        imageView.setContentDescription(pizzaName);
```

```
    }
```

← Константа будет использоваться для передачи идентификатора пиццы в дополнительной информации интента.

← Добавить кнопку Вверх.

↑ Получить пиццу, выбранную пользователем, из интента.

↑ Идентификатор используется для заполнения TextView и ImageView.

Код *PizzaDetailActivity* (продолжение)

```
@Override
```

```
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.menu_main, menu);
```

← Добавить элементы из файла ресурсов меню на панель действий.

```
//Использование названия пиццы в действии Share
```

```
TextView textView = (TextView) findViewById(R.id.pizza_text);
```

```
CharSequence pizzaName = textView.getText();
```

```
MenuItem menuItem = menu.findItem(R.id.action_share);
```

```
shareActionProvider = (ShareActionProvider) menuItem.getActionProvider();
```

```
Intent intent = new Intent(Intent.ACTION_SEND);
```

```
intent.setType("text/plain");
```

```
intent.putExtra(Intent.EXTRA_TEXT, pizzaName);
```

```
shareActionProvider.setShareIntent(intent);
```

```
return true;
```

```
}
```

↑
Назначить текст по умолчанию для действия Share.

```
@Override
```

```
public boolean onOptionsItemSelected(MenuItem item) {
```

```
    switch (item.getItemId()) {
```

```
        case R.id.action_create_order:
```

```
            Intent intent = new Intent(this, OrderActivity.class);
```

```
            startActivity(intent);
```

```
            return true;
```

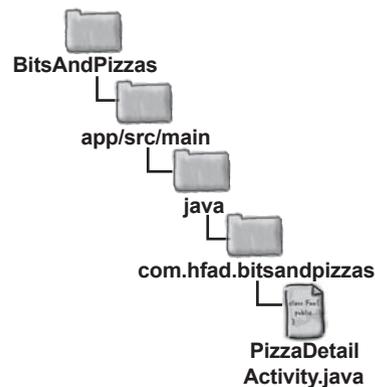
```
        default:
```

```
            return super.onOptionsItemSelected(item);
```

```
    }
```

```
}
```

```
}
```



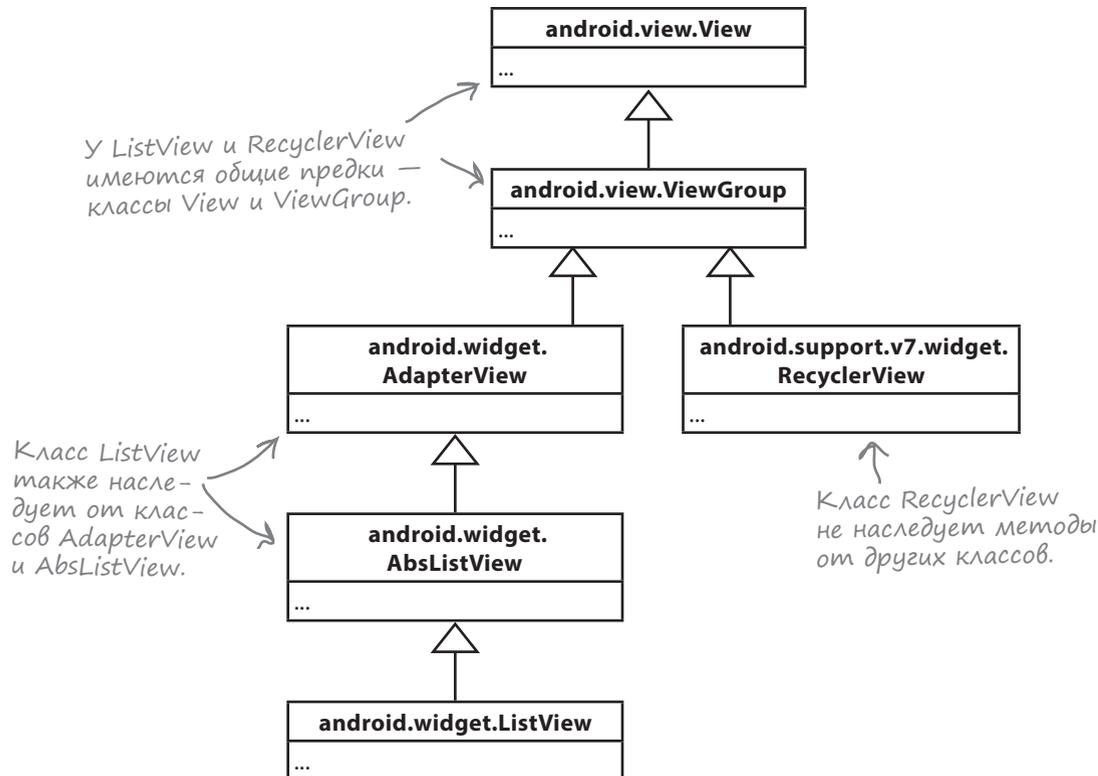
↑
Запустить *OrderActivity*, когда пользователь выбирает элемент на панели действий.

После обновления кода *PizzaDetailActivity.java* нужно запрограммировать реакцию *RecyclerView* на щелчки.

Выбор вариантов в RecyclerView

Чтобы при выборе определенного вида пиццы запускалась активность `PizzaDetailActivity`, необходимо реализовать в `RecyclerView` обработку щелчков на вариантах списка.

При создании навигационного списка на базе спискового представления для обработки щелчков достаточно передать списковому представлению объект `OnItemClickListener`. Списковое представление прослушивает события всех содержащихся в нем представлений, и при щелчке на любом из них списковое представление вызывает своего слушателя `OnItemClickListener`. Это позволяет отреагировать на выбор элементов с минимальным объемом кода. Такой способ работает для списковых представлений, потому что они наследуют значительную функциональность из очень глубокой иерархии суперклассов. С другой стороны, компоненты `RecyclerView` не имеют столь богатого набора встроенных методов, так как они не наследуют от тех же суперклассов:



Хотя такая схема наследования обеспечивает большую гибкость, она также означает, что при использовании `RecyclerView` вам придется проделывать намного больше работы вручную. Итак, как же обеспечить реакцию на выбор вариантов компонента `RecyclerView`?

Прослушивание событий представлений в адаптере

Если вы хотите, чтобы компоненты RecyclerView реагировали на щелчки, код придется писать самостоятельно. Чтобы написать код обработки события, необходимо иметь доступ к представлениям, отображаемым внутри RecyclerView. Где должен находиться этот код?

Все представления создаются в классе CaptionedImagesAdapter. При появлении представления на экране RecyclerView вызывает код onBindViewHolder(), который приводит содержимое карточки в соответствие с информацией элемента списка. Допустим, вы хотите, чтобы при щелчке на карточке пиццы открывалась активность с информацией об этой пицце. Теоретически *возможно* разместить код запуска активности в адаптере — например, так:

```
class CaptionedImagesAdapter extends RecyclerView.Adapter<CaptionedImagesAdapter.ViewHolder>{
...
    public void onBindViewHolder(ViewHolder holder, int position){
        CardView cardView = holder.cardView;
        ImageView imageView = (ImageView) cardView.findViewById(R.id.info_image);
        Drawable drawable = cardView.getResources().getDrawable(imageIds[position]);
        imageView.setImageDrawable(drawable);
        TextView textView = (TextView) cardView.findViewById(R.id.info_text);
        textView.setText(captions[position]);
        cardView.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(container.getContext(), PizzaDetailActivity.class);
                intent.putExtra(PizzaDetailActivity.EXTRA_PIZZANO, position);
                container.getContext().startActivity(intent);
            }
        });
    }
}
```

↑
Если добавит этот код в CaptionedImagesAdapter, то при щелчке на CardView будет запускаться PizzaDetailActivity.

Однако из того, что этот код *можно* написать, вовсе не следует, что это *нужно* делать.



События щелчка *можно* обрабатывать, добавляя код в класс адаптера. Но нет ли каких-либо причин, по которым так поступать *не следует*?

Повторное использование адаптеров

Выполняя обработку щелчков в классе `CaptionedImagesAdapter`, вы *ограничиваете возможности использования адаптера*. Подумайте, какое приложение мы строим: в нем должны выводиться списки пиццы, пасты и магазинов. Вероятно, во всех трех списках будет выводиться краткий текст с изображением. Если мы изменим класс `CaptionedImagesAdapter` так, чтобы щелчки всегда переводили пользователя к активности, выводящей подробную информацию об одном виде пиццы, класс `CaptionedImagesAdapter` не удастся использовать для списков пасты и магазинов. Для каждого списка придется создавать отдельный адаптер.

Использование интерфейса для ослабления связей

Вместо этого код, запускающий активность, будет располагаться вне адаптера. Когда пользователь щелкает на варианте в списке, адаптер должен вызвать фрагмент, содержащий список, а код фрагмента запускает интент для следующей активности. Это позволит нам повторно использовать `CaptionedImagesAdapter` для списков пиццы, пасты и магазинов, а фрагмент в каждом отдельном случае будет сам решать, что должно происходить по щелчку.

Мы будем действовать по тому же принципу, который применялся при отделении фрагмента от активности. В `CaptionedImagesAdapter` создается интерфейс `Listener`:

```
public static interface Listener {
    public void onClick(int position);
}
```

При щелчке на любой из карточек в `RecyclerView` будет вызываться метод `onClick()` интерфейса `Listener`. Затем в `PizzaMaterialFragment` добавляется код реализации интерфейса; это позволит фрагменту отреагировать на щелчки и запустить активность.

Вот что будет происходить во время выполнения:

- ❶ Пользователь щелкает на карточке в `RecyclerView`.
- ❷ Вызывается метод `onClick()` интерфейса `Listener`.
- ❸ Метод `onClick()` реализован в `PizzaMaterialFragment`. Код фрагмента запускает `PizzaDetailActivity`.

Начнем с добавления кода в `CaptionedImagesAdapter.java`.

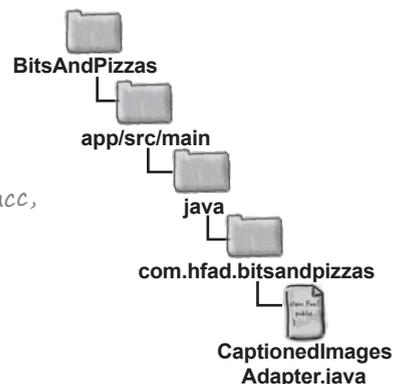
Добавление интерфейса в адаптер

Ниже приведен обновленный код *CaptionedImagesAdapter.java*: в него добавлен интерфейс *Listener*, а при щелчке на одной из карточек вызывается метод *onClick()* (внесите изменения в свой код и сохраните их):

```
package com.hfad.bitsandpizzas;
```

```
import android.graphics.drawable.Drawable;
import android.support.v7.widget.RecyclerView;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.support.v7.widget.CardView;
import android.widget.ImageView;
import android.widget.TextView;
```

В коде используется внешний класс, его необходимо импортировать.



```
class CaptionedImagesAdapter extends RecyclerView.Adapter<CaptionedImagesAdapter.ViewHolder>{
```

```
    private String[] captions;
```

```
    private int[] imageIds;
```

```
    private Listener listener;
```

Добавить объект *Listener* как приватную переменную.

```
    public static interface Listener {
        public void onClick(int position);
    }
```

Интерфейс.

```
    public static class ViewHolder extends RecyclerView.ViewHolder{
```

```
        private CardView cardView;
```

```
        public ViewHolder(CardView v) {
```

```
            super(v);
```

```
            cardView = v;
```

```
        }
```

```
    }
```

Эти методы не изменяются.

```
    public CaptionedImagesAdapter(String[] captions, int[] imageIds){
```

```
        this.captions = captions;
```

```
        this.imageIds = imageIds;
```

```
    }
```

Код CaptionedImagesAdapter.java (продолжение)

```

public void setListener(Listener listener) {
    this.listener = listener; ← Активности и фрагменты используют этот метод для регистрации себя в качестве слушателя.
}

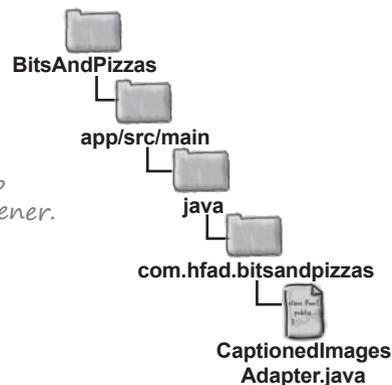
@Override
public CaptionedImagesAdapter.ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
    CardView cv = (CardView) LayoutInflater.from(parent.getContext())
        .inflate(R.layout.card_captioned_image, parent, false);
    return new ViewHolder(cv);
}

public void onBindViewHolder(ViewHolder holder, final int position) {
    CardView cardView = holder.cardView;
    ImageView imageView = (ImageView) cardView.findViewById(R.id.info_image);
    Drawable drawable = cardView.getResources().getDrawable(imageIds[position]);
    imageView.setImageDrawable(drawable);
    imageView.setContentDescription(captions[position]);
    TextView textView = (TextView) cardView.findViewById(R.id.info_text);
    textView.setText(captions[position]);
    cardView.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            if (listener != null) {
                listener.onClick(position);
            }
        }
    });
}

@Override
public int getItemCount() {
    return captions.length;
}
}

```

При щелчке на CardView вызвать метод onClick() интерфейса Listener.



Итак, мы добавили интерфейс Listener в адаптер. Теперь реализуем его в *PizzaMaterialFragment.java*.

Реализация слушателя в `PizzaMaterialFragment.java`

Мы реализуем интерфейс `Listener` из `CaptionedImagesAdapter` в классе `PizzaMaterialFragment` так, что при щелчке на карточке в `RecyclerView` будет запускаться `PizzaDetailActivity`. Код выглядит так:

```
package com.hfad.bitsandpizzas;

import android.app.Fragment;
import android.content.Intent;
import android.os.Bundle;
import android.support.v7.widget.LinearLayoutManager;
import android.support.v7.widget.RecyclerView;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

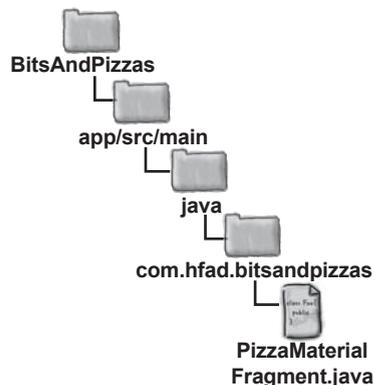
public class PizzaMaterialFragment extends Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        RecyclerView pizzaRecyclerView = (RecyclerView)inflater.inflate(
            R.layout.fragment_pizza_material, container, false);

        String[] pizzaNames = new String[Pizza.pizzas.length];
        for (int i = 0; i < pizzaNames.length; i++) {
            pizzaNames[i] = Pizza.pizzas[i].getName();
        }

        int[] pizzaImages = new int[Pizza.pizzas.length];
        for (int i = 0; i < pizzaImages.length; i++) {
            pizzaImages[i] = Pizza.pizzas[i].getImageResourceId();
        }
    }
}
```

Для запуска активности используется интент, поэтому класс необходимо импортировать.



Этот код не изменился.

Код PizzaMaterialFragment.java (продолжение)

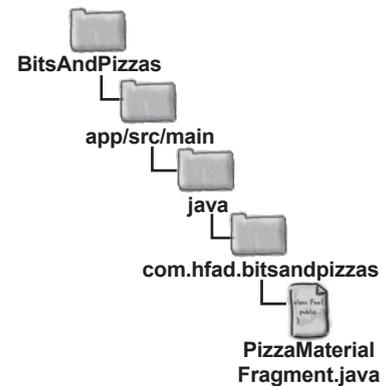
```

CaptionedImagesAdapter adapter = new CaptionedImagesAdapter(pizzaNames, pizzaImages);
pizzaRecycler.setAdapter(adapter);
LinearLayoutManager layoutManager = new LinearLayoutManager(getActivity());
pizzaRecycler.setLayoutManager(layoutManager);

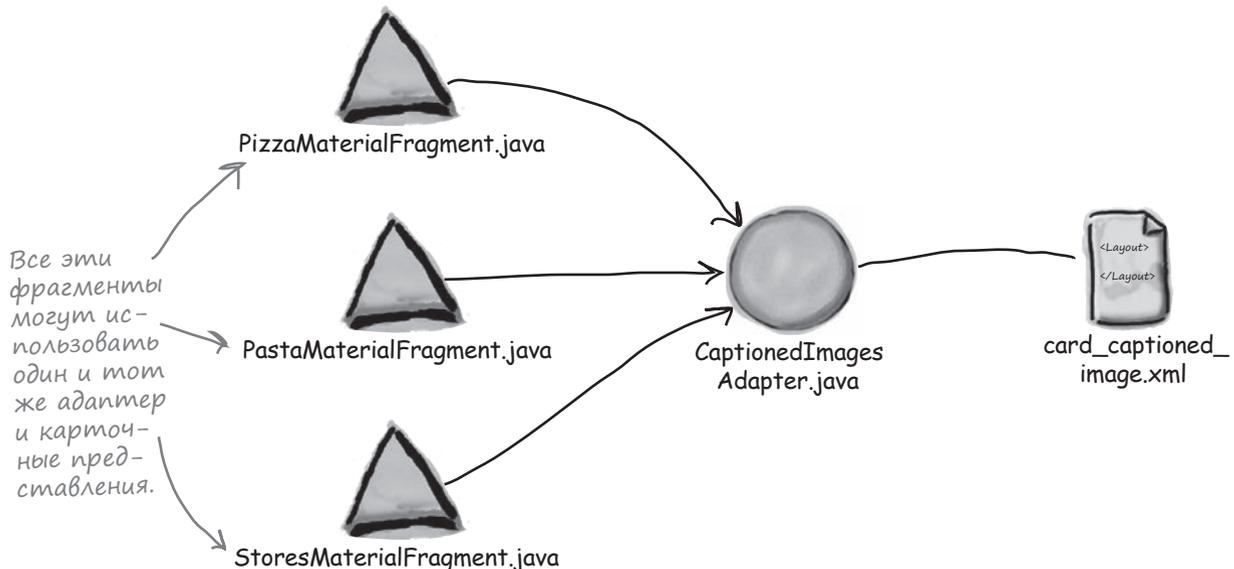
adapter.setOnClickListener(new CaptionedImagesAdapter.Listener() {
    public void onClick(int position) {
        Intent intent = new Intent(getActivity(), PizzaDetailActivity.class);
        intent.putExtra(PizzaDetailActivity.EXTRA_PIZZANO, position);
        getActivity().startActivity(intent);
    }
});
return pizzaRecycler;
}

```

Реализация метода `onClick()` интерфейса `Listener` запускает активность `PizzaDetailActivity`, передавая ей идентификатор пиццы, выбранной пользователем.



Мы рассмотрели весь код, необходимый для того, чтобы представления в `RecyclerView` реагировали на щелчки. Это решение позволяет использовать один и тот же адаптер и карточные представления для разных типов данных, состоящих из изображения и надписи.

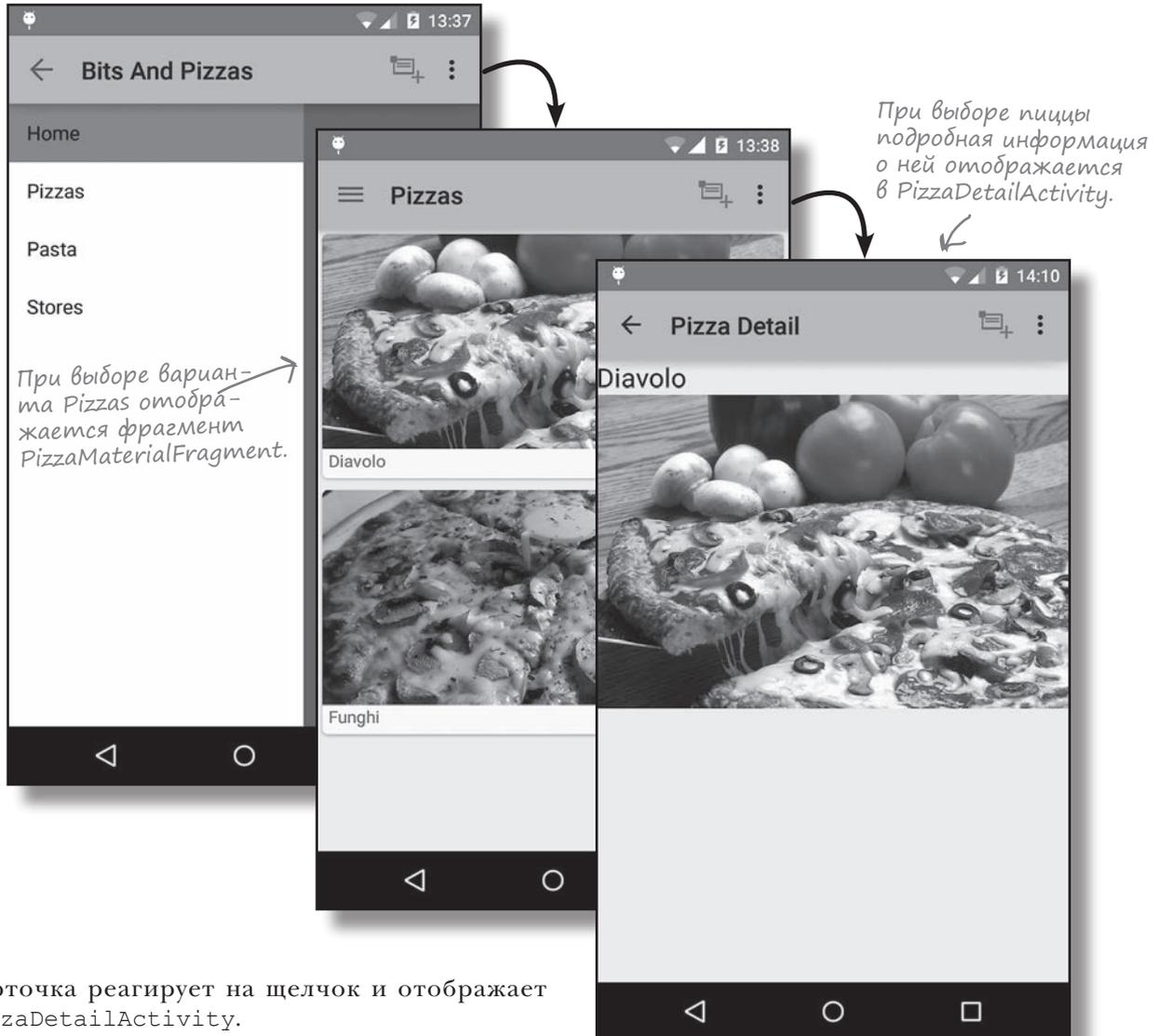


Посмотрим, что происходит при выполнении кода.



Тест-драйв

Запустите приложение, откройте выдвижную панель и выберите вариант Pizzas. На экране появляется список карточек с разными видами пиццы, как и прежде. Посмотрим, что произойдет, если щелкнуть на какой-либо карточке:



Карточка реагирует на щелчок и отображает `PizzaDetailActivity`.

Осталось решить еще один вопрос: выбрать информацию, которая должна размещаться в `TopFragment`.

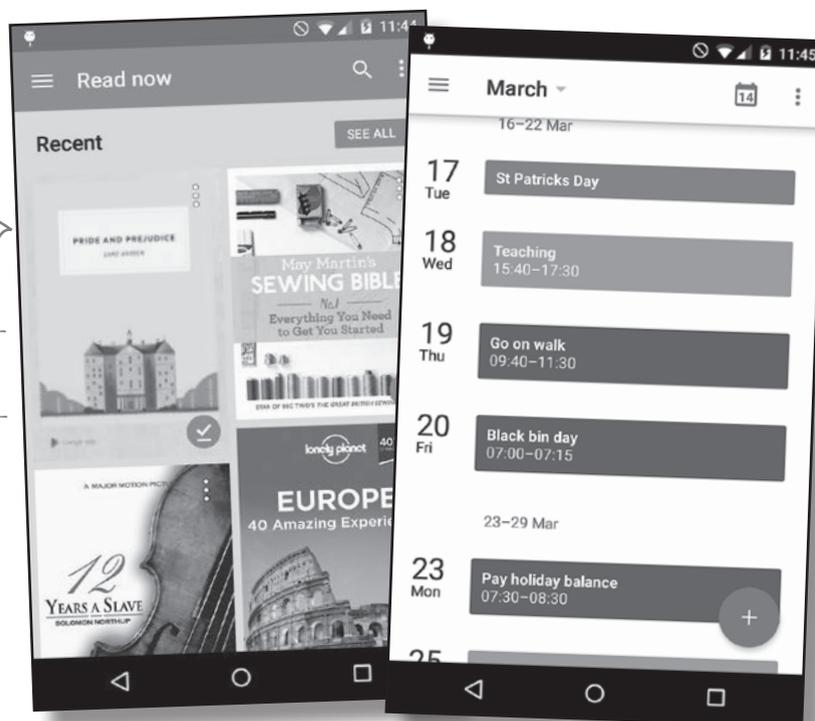
Информацию — на передний план

Когда мы разрабатывали первый вариант приложения Pizza, фрагмент TopFragment содержал список навигационных команд. Затем эти команды были вынесены из TopFragment на панель действий и выдвигающую панель, а фрагмент TopFragment остался пустым. Какую же информацию он должен содержать?

TopFragment — экран верхнего уровня; это первое, что видит пользователь при запуске приложения. Экран верхнего уровня должен быть полезным как для новых, так и для опытных пользователей, а этого можно добиться только одним способом: вывести информацию на передний план.

Присмотревшись к приложениям Google на вашем устройстве, вы найдете в них кое-что общее: все они позволяют быстро перейти к основному контенту за счет вынесения части этого контента на экран верхнего уровня. В приложении Calendar отображаются предстоящие события. В таких приложениях, как Play Books и Play Music, отображаются ваши последние действия и рекомендации. Эта информация занимает центральное место на экране верхнего уровня.

В приложении Play Books на экране верхнего уровня отображаются недавно загруженные книги, а также рекомендации по другим книгам, которые вам могут понравиться.



← Приложение Calendar отображает предстоящие события, чтобы вам было проще добраться до них.

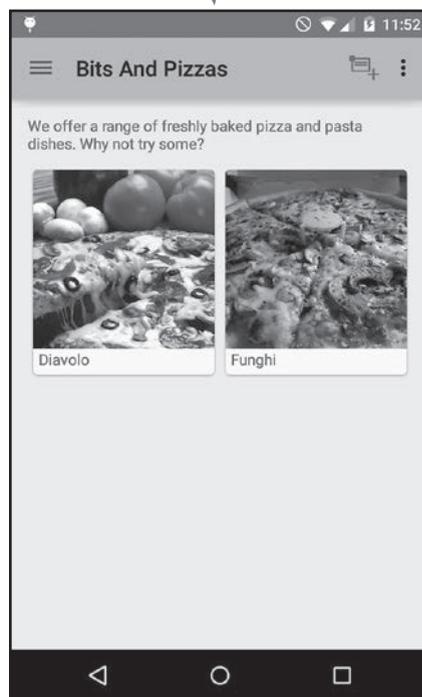
Чтобы вывести на первый план полезную информацию в приложении Pizza, мы можем разместить в TopFragment описания некоторых блюд из меню. К счастью, то, что вы узнали в этой главе, поможет решить эту задачу с минимальными усилиями.



Упражнение

Ваша задача — изменить фрагмент `TopFragment` так, чтобы в нем отображался короткий вводный текст и `RecyclerView` с информацией о пицце. Для начала напишите разметку макета для файла `fragment_top.xml`. Фрагмент `TopFragment` должен выглядеть примерно так, как показано ниже.

Так должен выглядеть фрагмент `TopFragment`.



Возьми в руку карандаш



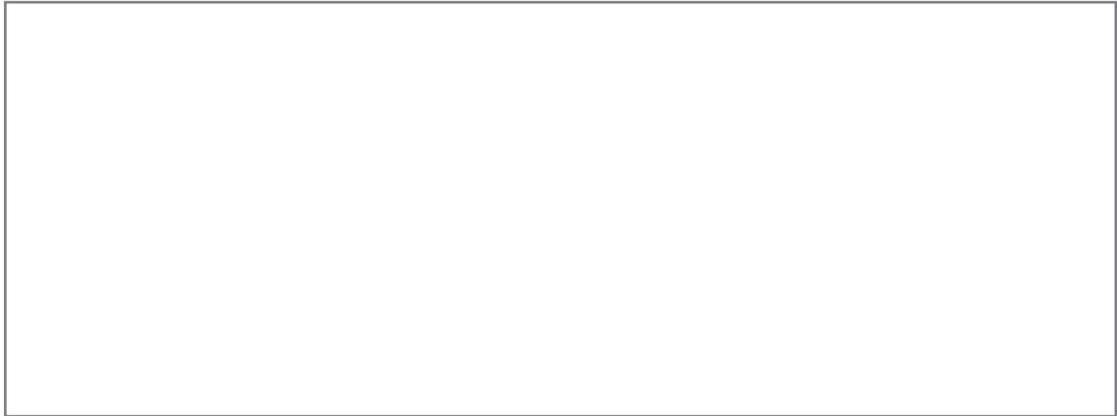
Теперь запишите внизу код *TopFragment.java*, который заполняет RecyclerView данными двух видов пиццы в табличном макете. Если пользователь выбирает один из этих вариантов, информация о нем должна выводиться в *PizzaDetailActivity.java*.

...

```
public class TopFragment extends Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        RelativeLayout layout = (RelativeLayout)
            inflater.inflate(R.layout.fragment_top, container, false);
```

Запи-
шите
свой
код
здесь.



```
        CaptionedImagesAdapter adapter =
            new CaptionedImagesAdapter(pizzaNames, pizzaImages);
        pizzaRecycler.setAdapter(adapter);
        adapter.setListener(new CaptionedImagesAdapter.Listener() {
            public void onClick(int position) {
                Intent intent = new Intent(getActivity(), PizzaDetailActivity.class);
                intent.putExtra(PizzaDetailActivity.EXTRA_PIZZANO, position);
                getActivity().startActivity(intent);
            }
        });
        return layout;
    }
}
```



Большая часть кода не отличается
от кода *PizzaMaterialFragment.java*.



Упражнение
Решение

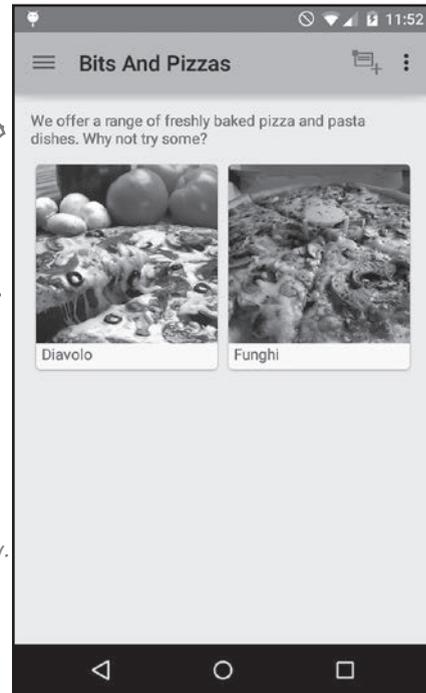
Ваша задача — изменить фрагмент TopFragment так, чтобы в нем отображался короткий вводный текст и RecyclerView с информацией о пицце. Для начала напишите разметку макета для файла *fragment_top.xml*. Фрагмент TopFragment должен выглядеть примерно так, как показано ниже.

Не беспокойтесь, если ваша разметка выглядит иначе. Такой макет можно создать разными способами.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    android:paddingRight="16dp"
    android:paddingLeft="16dp"
    tools:context=".MainActivity">
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/welcome_text"
        android:id="@+id/welcome_text" />
    <android.support.v7.widget.RecyclerView
        android:id="@+id/pizza_recycler"
        android:scrollbars="vertical"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/welcome_text"
        android:layout_marginTop="10dp"/>
</RelativeLayout>
```

Для вводного текста.

Для RecyclerView.



Возьми в руку карандаш

Решение



Теперь запишите внизу код *TopFragment.java*, который заполняет *RecyclerView* данными двух видов пиццы в табличном макете. Если пользователь выбирает один из этих вариантов, информация о нем должна выводиться в *PizzaDetailActivity.java*.

...

```
public class TopFragment extends Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        RelativeLayout layout = (RelativeLayout)
            inflater.inflate(R.layout.fragment_top, container, false);

        RecyclerView pizzaRecycler = (RecyclerView)layout.findViewById(R.id.pizza_recycler);
        String[] pizzaNames = new String[2];
        for (int i = 0; i < 2; i++) {
            pizzaNames[i] = Pizza.pizzas[i].getName();
        }
        int[] pizzaImages = new int[2];
        for (int i = 0; i < 2; i++) {
            pizzaImages[i] = Pizza.pizzas[i].getImageResourceId();
        }
        GridLayoutManager layoutManager = new GridLayoutManager(getActivity(),2);
        pizzaRecycler.setLayoutManager(layoutManager);

        CaptionedImagesAdapter adapter =
            new CaptionedImagesAdapter(pizzaNames, pizzaImages);
        pizzaRecycler.setAdapter(adapter);
        adapter.setListener(new CaptionedImagesAdapter.Listener() {
            public void onClick(int position) {
                Intent intent = new Intent(getActivity(), PizzaDetailActivity.class);
                intent.putExtra(PizzaDetailActivity.EXTRA_PIZZANO, position);
                getActivity().startActivity(intent);
            }
        });
        return layout;
    }
}
```

Ваш код может выглядеть иначе.

Выводится информация о двух видах пиццы.

Информация выводится в виде таблицы из двух столбцов.

Большая часть кода не отличается от кода *PizzaMaterialFragment.java*.

Полная разметка fragment_top.xml

Мы изменили фрагмент TopFragment так, чтобы в нем выводился вводный текст и два вида пиццы. Полная разметка приводится на следующих двух страницах.

Сначала добавьте в *strings.xml* следующую строку:

```
<string name="welcome_text">We offer a range of freshly baked pizza and pasta dishes. Why not try some?</string>
```

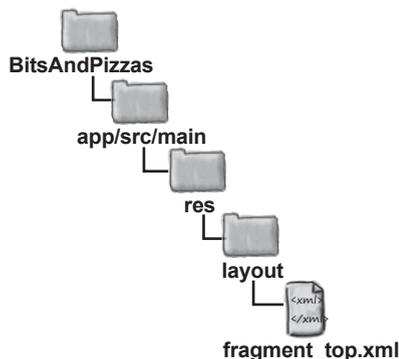
Затем обновите *fragment_top.xml* следующим кодом:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    android:paddingRight="16dp"
    android:paddingLeft="16dp"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/welcome_text"
        android:id="@+id/welcome_text" />

    <android.support.v7.widget.RecyclerView
        android:id="@+id/pizza_recycler"
        android:scrollbars="vertical"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/welcome_text"
        android:layout_marginTop="10dp"/>

</RelativeLayout>
```



В макет добавляются компоненты *TextView* и *RecyclerView*.

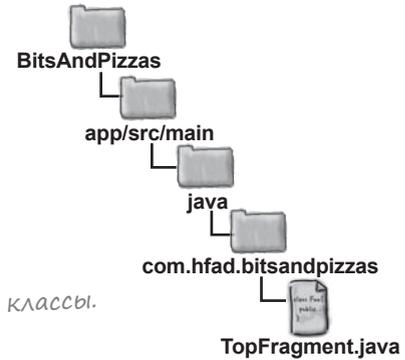
На следующей странице приведен код *TopFragment.java*.

Полный код TopFragment.java

```
package com.hfad.bitsandpizzas;
```

```
import android.content.Intent;
import android.os.Bundle;
import android.app.Fragment;
import android.support.v7.widget.GridLayoutManager;
import android.support.v7.widget.RecyclerView;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.RelativeLayout;
```

Используемые классы.



```
public class TopFragment extends Fragment {
```

```
    @Override
```

```
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
```

```
        RelativeLayout layout = (RelativeLayout)
```

```
            inflater.inflate(R.layout.fragment_top, container, false);
```

```
        RecyclerView pizzaRecycler = (RecyclerView) layout.findViewById(R.id.pizza_recycler);
```

```
        String[] pizzaNames = new String[2];
```

```
        for (int i = 0; i < 2; i++) {
```

```
            pizzaNames[i] = Pizza.pizzas[i].getName();
```

```
        }
```

```
        int[] pizzaImages = new int[2];
```

```
        for (int i = 0; i < 2; i++) {
```

```
            pizzaImages[i] = Pizza.pizzas[i].getImageResourceId();
```

```
        }
```

```
        GridLayoutManager layoutManager = new GridLayoutManager(getActivity(), 2);
```

```
        pizzaRecycler.setLayoutManager(layoutManager);
```

```
        CaptionedImagesAdapter adapter = new CaptionedImagesAdapter(pizzaNames, pizzaImages);
```

```
        pizzaRecycler.setAdapter(adapter);
```

```
        adapter.setOnClickListener(new CaptionedImagesAdapter.Listener() {
```

```
            public void onClick(int position) {
```

```
                Intent intent = new Intent(getActivity(), PizzaDetailActivity.class);
```

```
                intent.putExtra(PizzaDetailActivity.EXTRA_PIZZANO, position);
```

```
                getActivity().startActivity(intent);
```

```
            }
```

```
        });
```

```
        return layout;
```

```
    }
```

```
}
```

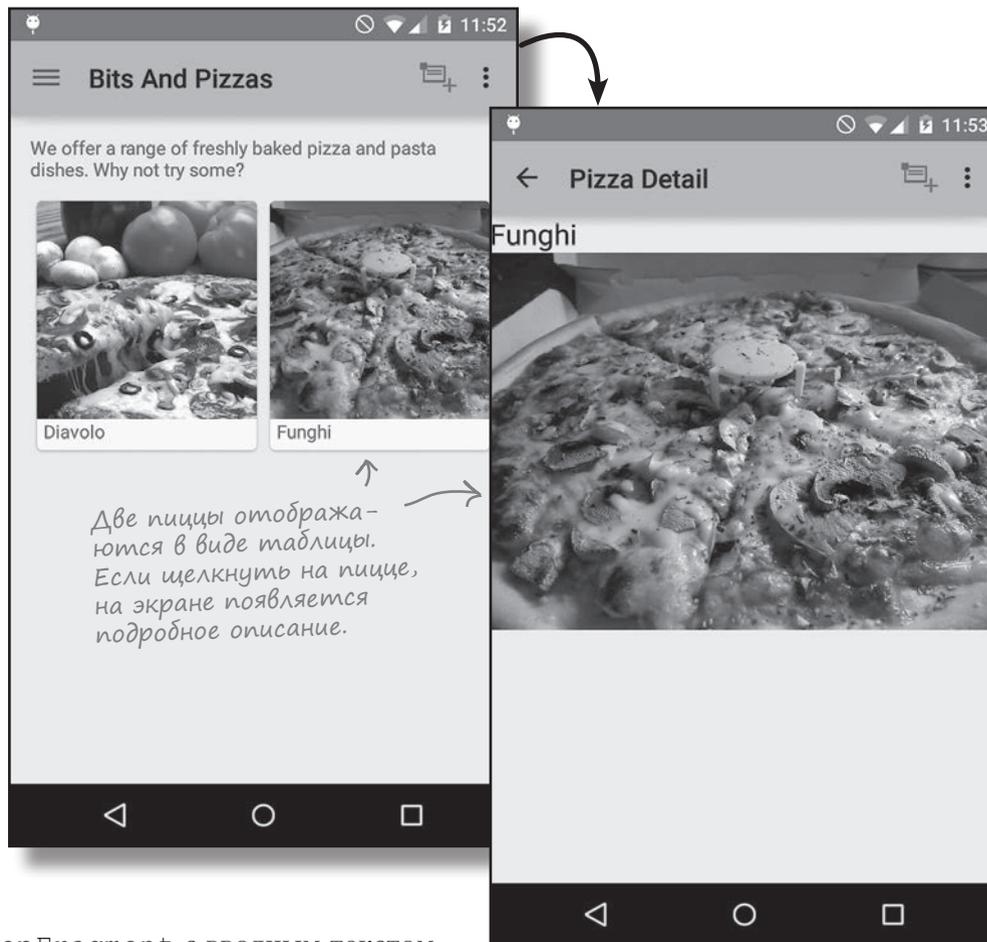
↑ Создать массивы с названиями и изображениями пиццы.

← Информация выводится в виде таблицы.

↑ Использовать адаптер для вывода информации.

↑ Когда пользователь выбирает пиццу, запустить PizzaDetailActivity и передать позицию выбранной пиццы.

Посмотрим, что произойдет при запуске приложения.



Открывается `TopFragment` с вводным текстом и изображениями двух видов пиццы. Если щелкнуть на пицце, информация о ней выводится в `PizzaDetailActivity`.



Ваш инструментарий Android

Глава 14 осталась позади, а ваш инструментарий пополнился навыками построения интерфейсов.

Весь код для этой главы можно загрузить по адресу <https://tinyurl.com/HeadFirstAndroid>.

КЛЮЧЕВЫЕ МОМЕНТЫ



- Компоненты `CardView` и `RecyclerView` имеют собственные библиотеки поддержки.
- Для добавления компонентов `CardView` в макет используется элемент `<android.support.v7.widget.CardView>`.
- Чтобы карточки отображались с закругленными углами, используйте атрибут `cardCornerRadius`. Для этого необходимо пространство имен `"http://schemas.android.com/apk/res-auto"`.
- Компоненты `RecyclerView` работают с адаптерами, расширяющими класс `RecyclerView.Adapter`.
- При создании собственной реализации `RecyclerView.Adapter` необходимо определить объекты `ViewHolder` и реализовать методы `onCreateViewHolder()`, `onBindViewHolder()` и `getItemCount()`.
- Для добавления компонентов `RecyclerView` в макет используется элемент `<android.support.v7.widget.RecyclerView>`. Полоса прокрутки назначается при помощи атрибута `android:scrollbars`.
- Способ размещения элементов в `RecyclerView` задается объектом `LayoutManager`. `LinearLayoutManager` размещает элементы в виде линейного списка, `GridLayoutManager` размещает их в виде таблицы, а `StaggeredGridLayoutManager` использует неравномерную таблицу.

всего хорошего!

Пара слов на прощанье...



Надеемся, вы хорошо провели время в мире Android.

Конечно, жаль расставаться, но новые знания заслуживают того, чтобы применить их на практике. В конце книги еще осталось несколько приложений, просмотрите их — и переходите к самостоятельной работе. Приятного путешествия!

Приложение 1. Исполнительная среда

Исполнительная среда Android



Приложения Android должны работать на устройствах с маломощными процессорами и ограниченной памятью.

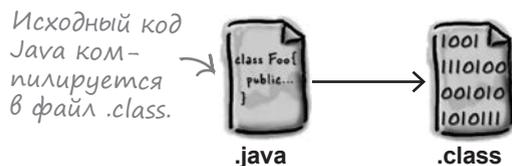
Java-приложения могут расходовать много памяти. Кроме того, если бы приложения выполнялись на виртуальной машине Java (JVM, Java Virtual Machine), на маломощных устройствах запуск приложения мог бы занимать много времени. Чтобы избежать этих проблем, система Android вместо JVM использует для запуска своих приложений другую виртуальную машину, которая называется **ART (Android runtime)**. В этом приложении вы узнаете, как ART удастся обеспечить нормальное выполнение Java-приложений на компактном маломощном устройстве.

Что такое «ART»?

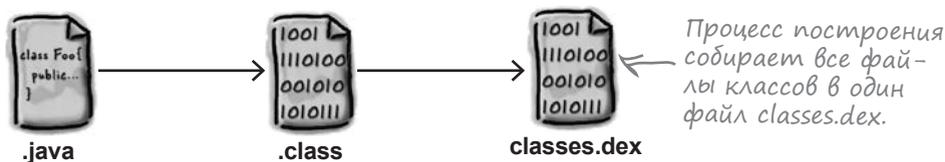
Исполнительная среда Android, или ART (Android Runtime), — система, выполняющая ваш откомпилированный код на устройствах Android. Впервые она появилась в версии KitKat, а в версии Lollipop стала стандартным механизмом выполнения кода. Среда ART предназначена для быстрого и эффективного выполнения откомпилированных Android-приложений на компактных, маломощных устройствах.

ART сильно отличается от JVM

Язык Java появился уже давно, а откомпилированные Java-программы почти всегда выполнялись на виртуальной машине JVM компании Oracle. JVM моделирует работу центрального процессора и читает откомпилированный файл `.class`, содержащий инструкции машинного кода JVM, называемого байт-кодом. Традиционно исходные файлы `.java` компилировались в файлы `.class`, после чего запускались с использованием интерпретатора JVM.



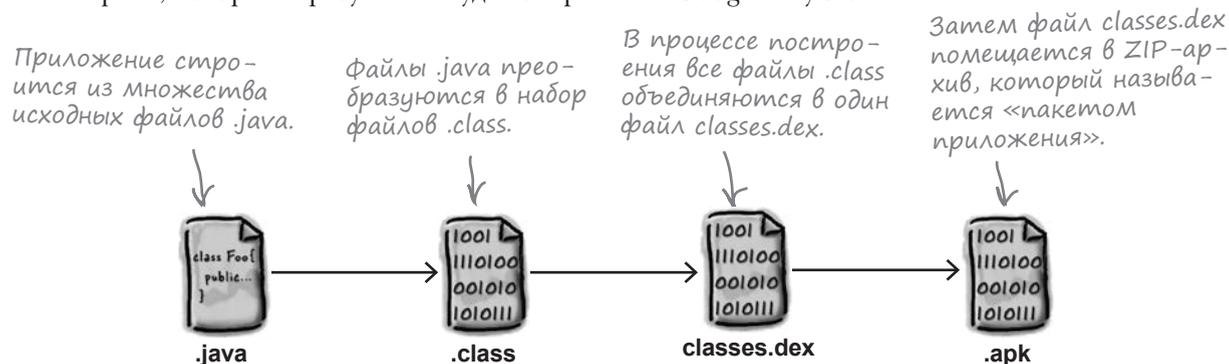
ART работает по другим правилам. При компиляции Android-приложения все начинается так же: вы пишете исходный код в файлах `.java` и компилируете их в файлы `.class`, но затем программа `dx` преобразует набор файлов `.class` (или архивов `.jar`) в один файл с именем `classes.dex`.



Файл `classes.dex` также содержит байт-код, но этот байт-код сильно отличается от байт-кода в файлах `.class`. Байт-код `.dex` предназначен для совершенно другого виртуального процессора, который называется **Dalvik**. Собственно, сокращение `dex` означает «Dalvik Executable», то есть «исполняемый файл Dalvik».

Процессор Dalvik отчасти похож на JVM: и JVM, и Dalvik являются виртуальными процессорами. Однако работа процессора Oracle JVM основана на операциях со стеком, а процессор Dalvik основан на операциях с регистрами. Некоторые специалисты полагают, что код для регистровых процессоров можно оптимизировать с сокращением объема и ускорением работы. Преобразование целого набора файлов в один файл `classes.dex` позволяет существенно уменьшить размер откомпилированного приложения, потому что при преобразовании удаляются многочисленные повторяющиеся символические имена, встречающиеся в разных файлах `.class`.

Файл `classes.dex` вместе с другими файлами ресурсов и данных объединяется в файл, сжатый по алгоритму ZIP, — пакет приложения (application package), или файл APK. Файл `.apk` представляет собой итоговое откомпилированное приложение, которое может устанавливаться на устройство Android. Это тот самый файл, который в результате будет отправлен в Google Play Store.

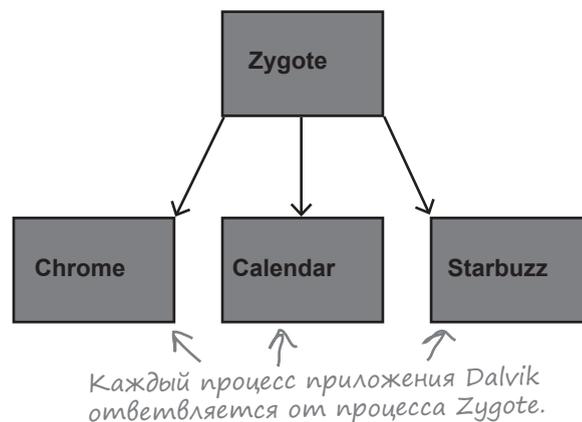


Как Android выполняет файлы APK

Файл APK представляет собой обычный архив в формате ZIP. Когда этот файл передается на устройство Android, он сохраняется в каталоге с именем `/data/app/<имя пакета>`, после чего из него извлекается файл `classes.dex`.

Файл `classes.dex`, извлеченный из архива APK, преобразуется в платформенную библиотеку. Байт-код Dalvik превращается в машинный код, который может выполняться непосредственно на процессоре устройства. Откомпилированная библиотека сохраняется в каталоге `/data/dalvik-cache`. Компиляция выполняется системой Android только при первом запуске приложения. В дальнейшем устройство на базе Android может просто загрузить и запустить платформенную библиотеку.

Система Android — всего лишь разновидность операционной системы Linux, а в системе Linux возможность запуска приложений Android обычно отсутствует. Вот почему на каждом устройстве Android выполняется процесс **Zygote**. Когда вы приказываете Android запустить новое приложение Android, процесс **Zygote** создает ответвленную версию самого себя. Ответвленный процесс представляет собой обычную копию процесса в памяти. В Linux ответвление процессов выполняется очень быстро, поэтому ответвление процесса **Zygote** с последующей загрузкой платформенной библиотеки позволяет очень быстро загружать приложения Android.



Быстродействие и размер

Устройства Android обычно располагают существенно меньшими вычислительными ресурсами и памятью, чем машины, на которых обычно выполняется код Java. Среда ART использует файлы *.dex*, которые обычно меньше эквивалентных файлов *.class*. Виртуальная машина Oracle JVM позволяет компилировать некоторые части обрабатываемого кода с использованием механизма JIT (Just-In-Time), при котором байт-код Java преобразуется в машинный код непосредственно в процессе выполнения. Такой подход хорошо подходит для приложений, выполняющихся в течение очень долгого времени (например, серверов приложений), но приложения Android регулярно запускаются и останавливаются. Предварительная компиляция всех байт-кодов Dalvik в платформенную библиотеку гарантирует, что код достаточно будет откомпилировать всего один раз.

Наконец, запуск исполнительной среды Oracle Java на маломощных устройствах может занимать значительное время. Использование процесса *Zygote* позволяет Android существенно ускорить запуск приложений. Процесс *Zygote* также может использовать общие библиотеки для безопасного выполнения кода, общего для всех процессов Dalvik.

Безопасность

На устройствах Android может выполняться код, написанный многими разработчиками, поэтому очень важно, чтобы каждое приложение было полностью изолировано от всех остальных приложений. Без такой изоляции одно приложение может нарушить защиту другого приложения на устройстве. Чтобы приложения существовали изолированно друг от друга, Android выполняет каждое приложение в отдельном процессе, от имени автоматически сгенерированной учетной записи пользователя. Эта схема обеспечивает изоляцию приложений в условиях механизма безопасности операционной системы, предоставленного Linux. Если бы вместо этого использовалась исполнительная среда Oracle Java, то каждому процессу потребовалась бы отдельная копия процесса Java, что привело бы к существенному росту затрат памяти при запуске нескольких приложений.

Приложение II. adb

Android Debug Bridge

Что еще подарить
девушке, у которой
есть все, — разве что
новую программу
командной строки?

Милый, ты такой
заботливый.



В этой книге для всех задач Android-программирования использовались средства среды разработки. Однако в некоторых случаях инструменты командной строки попросту более эффективны — например, если Android Studio упорно отказывается видеть ваше устройство Android, а вы *знаете*, что оно есть. В этой главе мы познакомимся с **Android Debug Bridge (сокращенно adb)** — утилитой командной строки, предназначенной для взаимодействия с эмулятором или устройствами Android.

adb: Ваш груз из режима командной строки

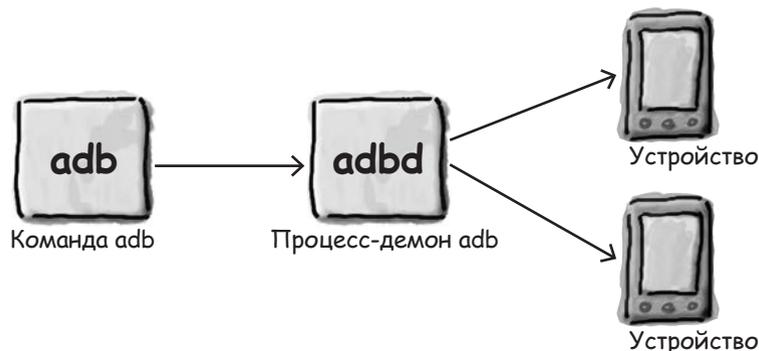
Каждый раз, когда компьютеру, на котором ведется разработка, требуется выполнить операцию с устройством на базе Android — будь то реальное устройство, подключенное кабелем USB, или виртуальное устройство, выполняемое в эмуляторе, — он использует **Android Debug Bridge (adb)**. Процесс adb находится под управлением команды, которая также называется adb.

Команда adb хранится в каталоге `platform-tools` пакета Android System Developer's Kit. Вероятно, на Mac вы найдете ее в каталоге `/Users/<имя пользователя>/Library/Android/sdk/platform-tools`. Если каталог `platform-tools` включен в переменную среды `PATH`, то вы сможете запустить adb прямо из командной строки.

В окне терминала или командной строки программа используется примерно так:

```
Interactive Session
$ adb devices
List of devices attached
emulator-5554device
$
```

Команда `adb devices` означает: «Вывести список подключенных устройств Android». Работа adb основана на взаимодействии с серверным процессом adb, выполняемым в фоновом режиме. Сервер adb иногда называется *демоном adb*, или *adb*. Когда вы вводите команду `adb command` в окне терминала, на сетевой порт 5037 вашей машины отправляется запрос. Процесс adb прослушивает команды, поступающие на этот порт. Когда среда Android Studio хочет запустить приложение, проверить содержимое журнала или выполнить любую другую операцию, требующую взаимодействия с устройством Android, она делает это через порт 5037.



При получении команды adb передает ее отдельному процессу, выполняемому на соответствующем устройстве Android. Этот процесс вносит изменения на устройстве или возвращает запрошенную информацию.

Если сервер adb почему-либо не работает, команда adb запускает его:

```
Interactive Session
$ adb devices
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
List of devices attached
emulator-5554device
$
```

Если вы подключаете устройство Android, а среда Android Studio его не видит, вы можете вручную остановить сервер adb и перезапустить его:

```
Interactive Session
$ adb devices
List of devices attached
$ adb kill-server
$ adb start-server
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
$ adb devices
List of devices attached
emulator-5554device
$
```

Остановка и перезапуск сервера заставляют adb заново связаться с подключенными устройствами Android.

Запуск командного процессора

Обычно вам не придется работать с adb напрямую; всю работу за вас выполняет среда разработки — такая, как Android Studio. Тем не менее в отдельных случаях бывает полезно перейти в режим командной строки и взаимодействовать с устройствами напрямую. Допустим, вам потребовалось запустить на устройстве командный процессор (shell):

```
Interactive Session
$ adb shell
root@generic_x86:/ #
```

Команда `adb shell` открывает интерактивный командный процессор прямо на устройстве Android. Команда `adb shell` работает только в том случае, если к компьютеру подключено одно устройство Android; в противном случае она не будет знать, с каким устройством Android вы собираетесь взаимодействовать.

После того как на устройстве будет запущен командный процессор, вы сможете выполнить многие стандартные команды Linux:

```
Interactive Session
$ adb shell
root@generic_x86:/ # ls
acct
cache
charger
config
d
data
default.prop
dev
etc
file_contexts
....
|root@generic_x86:/ # df
Filesystem      Size      Used      Free      Blksize
/dev            439.8M    60.0K    439.8M    4096
/mnt/asec       439.8M    0.0K    439.8M    4096
/mnt/obb        439.8M    0.0K    439.8M    4096
/system         738.2M    533.0M    205.2M    4096
/data           541.3M    237.8M    303.5M    4096
/cache          65.0M     4.3M     60.6M    4096
/mnt/media_rw/sdcard 196.9M    4.5K    196.9M    512
/storage/sdcard 196.9M    4.5K    196.9M    512
root@generic_x86:/ #
```

Получение вывода от logcat

Все приложения, работающие на устройствах Android, направляют свой вывод в центральный поток logcat. Чтобы просмотреть вывод logcat, выполните команду `adb logcat`:

```
Interactive Session
$ adb logcat
----- beginning of system
I/Vold ( 936): Vold 2.1 (the revenge) firing up
D/Vold ( 936): Volume sdcard state changing -1
(Initializing) -> 0 (No-Media)
W/DirectVolume( 936): Deprecated implied prefix pattern
detected, please use '/devices/platform/goldfish_mmc.0*'
instead
...

```

Вывод logcat будет поступать до тех пор, пока вы его не остановите. Команда `adb logcat` обычно используется для сохранения выходных данных в файл; она же используется Android Studio для получения данных, которые выводятся на панели Devices/logcat.

Копирование файлов на устройство и с него

Команды `adb pull` и `adb push` могут использоваться для передачи файлов. Например, в следующем примере файл свойств `/default.prop/` копируется в локальный файл с именем `1.txt`:

```
Interactive Session
$ adb pull /default.prop 1.txt
28 KB/s (281 bytes in 0.009s)
$ cat 1.txt
#
# ADDITIONAL_DEFAULT_PROPERTIES
#
ro.secure=0
ro.allow.mock.location=1
ro.debuggable=1
ro.zygote=zygote32
dalvik.vm.dex2oat-Xms=64m
dalvik.vm.dex2oat-Xmx=512m
dalvik.vm.image-dex2oat-Xms=64m
dalvik.vm.image-dex2oat-Xmx=64m
ro.dalvik.vm.native.bridge=0
persist.sys.usb.config=adb
$

```

И многое, многое другое...

Существует еще очень много команд, которые можно выполнить при помощи программы adb: резервное копирование и восстановление баз данных (очень полезно, если вам понадобится отладить ошибки в приложении, работающем с базой данных), запуск сервера adb на другом порте, перезагрузка машины или получение полезной информации о работающих устройствах. Чтобы получить список всех команд, введите в командной строке adb без параметров:

```
Interactive Session
$ adb
Android Debug Bridge version 1.0.32
  -a          - directs adb to listen on all
interfaces for a connection
  -d          - directs command to the only
connected USB device
              returns an error if more than one
USB device is present.
  -e          - directs command to the only
running emulator.
              returns an error if more than one emulator is ...
```

Приложение III. Эмулятор

Эмулятор *Android*

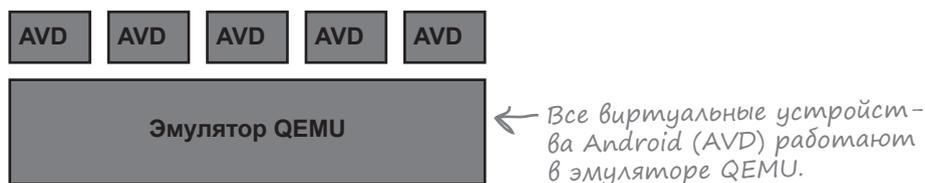


У вас никогда не возникало ощущение, что эмулятора приходится ждать целую вечность? Безусловно, эмулятор Android полезен. С его помощью можно увидеть, как приложение будет работать на других устройствах — помимо физических устройств, имеющихся у вас. Но иногда он кажется слегка... заторможенным. В этом приложении мы объясним, почему эмулятор кажется медленным. Более того, мы приведем несколько полезных советов для **ускорения** его работы.

Почему эмулятор так медленно работает

В процессе работы над Android-приложениями много времени уходит на ожидание, пока эмулятор Android запускает или устанавливает ваш код. Почему это происходит? Почему эмулятор Android работает так м-е-е-е-дле-е-н-н-н-но-о-о? Если вы писали код для iPhone, то знаете, как быстро работает имитатор iPhone. Если это возможно для iPhone, то почему не для Android? Подсказка кроется в самом названии: *имитатор iPhone* и *эмулятор Android*. Имитатор iPhone моделирует устройство, на котором работает операционная система iOS. Весь код для iOS компилируется для выполнения в родном формате Mac, и имитатор iPhone выполняется на скорости, с которой работает Mac. Это означает, что загрузка iPhone может быть смоделирована буквально за несколько секунд.

Эмулятор Android работает по совершенно иному принципу. Он использует приложение с открытым кодом QEMU (Quick Emulator) для эмуляции всего физического устройства Android. Эмулятор выполняет код, который интерпретирует машинный код, предназначенный для выполнения на процессоре устройства. В нем присутствует код, эмулирующий систему памяти, экран и практически все физическое оборудование устройства на базе Android.



Такой эмулятор, как QEMU, создает намного более реалистичное представление виртуального устройства, но у этого реализма есть и обратная сторона: ему приходится проделывать намного больше работы даже для таких простых операций, как чтение с диска или вывод данных на экран. Вот почему эмулятору требуется столько времени для загрузки: ему приходится моделировать все до единого аппаратные компоненты устройства, а также интерпретировать все возможные инструкции.

Как ускорить разработку

1. Используйте реальное устройство

Самый простой способ ускорения процесса разработки — использование реального устройства. Реальное устройство загружается намного быстрее эмулируемого; вероятно, установка и запуск приложений тоже будут происходить намного быстрее. Если вы хотите заниматься разработкой на реальном устройстве, загляните в раздел “Developer options” и установите флажок Stay Awake — это предотвратит блокировку экрана, что может быть полезно при многократной установке приложения.

2. Используйте снимок состояния эмулятора

Загрузка — один из самых медленных процессов, выполняемых эмулятором. Если вы сохраните «моментальный снимок» устройства во время его работы, эмулятор сможет восстановиться в этом состоянии, не проходя через процесс загрузки. Чтобы создать снимок устройства, выполните в меню Android Studio команду Tools→Android→AVD Manager, измените AVD, щелкнув на значке Edit, и установите флажок “Store a snapshot for faster startup”.

Этот флажок сохранит снимок памяти устройства во время его работы. В этом случае эмулятор сможет восстановить память в сохраненном состоянии без моделирования загрузки устройства.

3. Используйте аппаратное ускорение

По умолчанию эмулятору QEMU приходится интерпретировать каждую инструкцию машинного кода на виртуальном устройстве. Такой подход означает, что эмулятор чрезвычайно гибок, потому что он может «изображать» многие разные процессоры, но по этой же причине эмулятор работает очень медленно. К счастью, существует возможность непосредственного выполнения инструкций машинного кода на машине разработки. Существуют два основных типа AVD: машины ARM и машины x86. Если вы создаете устройство Android на базе x86, а на вашей машине установлена специальная разновидность процессора Intel x86, то вы сможете настроить эмулятор для выполнения инструкций машинного кода Android прямо на процессоре Intel.

Для этого вам придется установить программу Intel Hardware Accelerated Execution Manager (HAXM). На момент написания книги HAXM можно было найти по следующему адресу:

<https://software.intel.com/en-us/android/articles/intel-hardware-accelerated-execution-manager>

HAXM представляет собой гипервизор. Это означает, что программа может переключить процессор в специальный режим для непосредственного выполнения инструкций виртуальной машины. HAXM работает только на процессорах Intel с поддержкой технологии Intel Virtualization. Если ваша машина разработки совместима, использование HAXM существенно ускорит работу AVD.

Если проект переместился в другое место, вы быстро найдете его в поисковой системе.



Приложение IV. Остатки

Десять важнейших тем (которые мы не рассмотрели)

Только посмотрите,
сколько еще всего
вкусного осталось...



Но и это еще не все. Осталось еще несколько тем, о которых, как нам кажется, вам следует знать. Делать вид, что их не существует, было бы неправильно — как, впрочем, и выпускать книгу, которую поднимет разве что культурист. Прежде чем откладывать книгу, ознакомьтесь с этими **лакомыми кусочками**, которые мы оставили напоследок.

1. Распространение приложений

Когда разработка приложения будет завершена, вероятно, вы захотите сделать его доступным для других пользователей. Обычно для этого приложение публикуется в магазине приложений — таком, как Google Play.

Публикация состоит из двух этапов: подготовки приложения к выпуску и собственно выпуска.

Подготовка приложения к выпуску

Прежде чем выпускать приложение, необходимо подготовить, построить и протестировать окончательную версию приложения. В частности, при этом решаются такие задачи, как выбор значка приложения и изменение файла *AndroidManifest.xml*, чтобы он мог загружаться только на тех устройствах, на которых может выполняться ваше приложение.

Прежде чем выпускать приложение, обязательно протестируйте его хотя бы на одном планшете и хотя бы на одном телефоне. Убедитесь в том, что оно выглядит именно так, как задумано, и работает с приемлемой скоростью. Дополнительную информацию о подготовке приложения к выпуску можно найти по адресу:

<http://developer.android.com/tools/publishing/preparing.html>

Выпуск приложения

На этой стадии приложение становится доступным для публики, продается и распространяется.

Чтобы опубликовать приложение в Play Store, необходимо зарегистрировать учетную запись для публикации и воспользоваться Developer Console для публикации приложения. Дополнительная информация доступна по адресу:

<http://developer.android.com/distribute/googleplay/start.html>

Если вас интересует, как лучше донести информацию о приложении до пользователей и как организовать его продвижение, мы рекомендуем ознакомиться с документами по следующей ссылке:

<http://developer.android.com/distribute/index.html>

2. Провайдеры контента

Вы видели, как использовать интенды для запуска активностей из других приложений. Например, вы можете запустить приложение Сообщения, которое отправит переданный ему текст. Но что, если вы хотите использовать в своем приложении данные из другого приложения? Что, если на основании данных приложения Контакты ваше приложение выполняет некоторую операцию или вставляет новое событие в Календарь? Ваше приложение не может работать с данными другого приложения, обратившись к его базе данных. Вместо этого следует использовать **провайдера контента** — интерфейс, обеспечивающий управляемое совместное использование данных. Этот интерфейс позволяет выполнять запросы для чтения данных, вставлять новые записи, обновлять или удалять существующие записи.



Чтобы другие приложения могли пользоваться вашими данными, создайте своего провайдера контента.

Концепция провайдеров контента более подробно рассматривается по адресу:
<http://developer.android.com/guide/topics/providers/content-providers.html>

Руководство по использованию данных приложения Контакты в ваших приложениях:

<http://developer.android.com/guide/topics/providers/contacts-provider.html>

Руководство по использованию данных Календаря:

<http://developer.android.com/guide/topics/providers/calendar-provider.html>

3. Класс *WebView*

Если ваше приложение должно предоставлять пользователям доступ к веб-страницам, у вас есть два варианта. Первый вариант — разработка веб-приложения, с которым пользователь сможет работать в браузере на своем устройстве. Второй вариант — использование класса *WebView*.

Класс *WebView* позволяет вывести содержимое веб-страницы внутри макета активности. Он может использоваться как для оформления всего веб-приложения в виде клиентского приложения, так и для поставки отдельных веб-страниц. Этот способ может быть полезен при наличии в приложении периодически обновляемого контента — например, соглашения конечного пользователя или руководства пользователя.

Чтобы добавить класс *WebView* в приложение, включите его в макет:

```
<WebView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/webview"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

Загружаемая веб-страница определяется при вызове метода `loadUrl()` в коде Java:

```
WebView webView = (WebView) findViewById(R.id.webview);
webView.loadUrl("http://www.oreilly.com/");
```

Также необходимо указать, что приложению следует предоставить доступ к Интернету, включив разрешение `INTERNET` в файл *AndroidManifest.xml*:

```
<manifest ... >
    <uses-permission android:name="android.permission.INTERNET" />
    ...
</manifest>
```

За дополнительной информацией об использовании веб-контента в приложениях обращайтесь по адресу:

<http://developer.android.com/guide/webapps/index.html>

4. Анимация

Устройства Android начинают более эффективно использовать возможности своего встроеного графического оборудования, и анимация все чаще применяется для улучшения опыта взаимодействия пользователя с приложениями.

В Android поддерживаются несколько разновидностей анимации.

Анимация свойств

Анимация свойств основана на том факте, что внешний вид визуальных компонентов Android описывается набором числовых свойств. При изменении значения свойства (например, ширины или высоты приложения) создается эффект анимации. Анимация свойств представляет собой именно это: плавное изменение свойств визуальных компонентов с течением времени.

Декларативные анимации

Многие анимации могут создаваться на декларативном уровне в виде ресурсов XML. Таким образом, файл XML может использовать стандартный набор анимаций (масштабирование, смещение и повороты) для создания эффектов, которые вы можете вызывать из своего кода. Главное преимущество декларативных анимаций заключается в том, что они не привязаны к коду Java; это позволяет элементарно портировать их из одного проекта в другой.

Переходы активностей

Допустим, вы пишете приложение для вывода списка объектов с названиями и изображениями. Пользователь щелкает на одном объекте и переходит к его детализированному представлению. Активность для вывода подробной информации, вероятно, будет использовать то же изображение, которое отображалось в предыдущей активности.

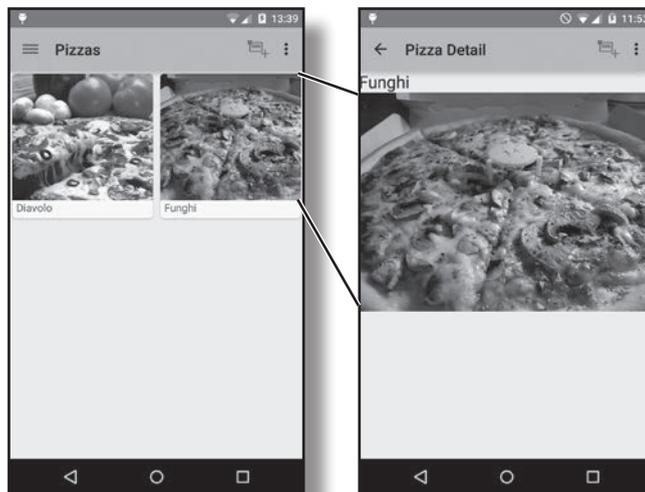
Переходы активностей позволяют организовать анимацию представления из одной активности, которое также должно присутствовать в другой активности. Например, изображение из списка может плавно переместиться по экрану в позицию, которую оно будет занимать в следующей активности. Такое приложение выглядит более профессионально.

За дополнительной информацией об анимации в Android обращайтесь по адресу:

<https://developer.android.com/guide/topics/graphics/index.html>

За дополнительной информацией о переходах и материальном оформлении обращайтесь по адресу:

<https://developer.android.com/training/material/animations.html>



5. Карты

Устройство Android может путешествовать вместе с вами где угодно, поэтому поддержка геопозиционирования и карт стала важным аспектом многих приложений Google.

Установка библиотеки Google Play Services позволит вам вставлять карты Google прямо в приложение. В поддержке карт доступны все возможности исходного приложения; кроме того, вы можете провести разнообразную настройку для интеграции карт в приложение.

Карта вставляется в макет в виде фрагмента:

```
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/map"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:name="com.google.android.gms.maps.MapFragment"/>
```

Далее вы работаете с картой на программном уровне через объект `GoogleMap`:

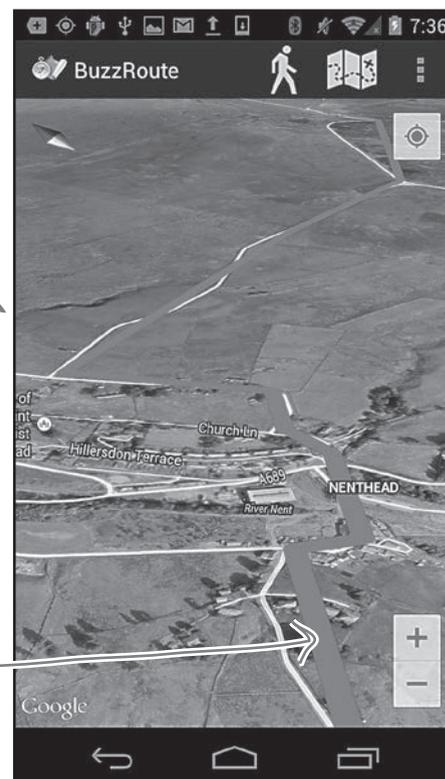
```
GoogleMap map = getMap();
```

Наконец, карту можно дополнять собственной информацией. Например, ломаная добавляется на карту следующим образом:

```
routeLine = map.addPolyline(new PolylineOptions()
    .width(ROUTE_THICKNESS_PIXELS)
    .color(Color.RED));
```

Приложение со встроенной поддержкой карт Google.

Ломаная.



5. Карты (продолжение)

Ваше приложение также может прослушивать события в приложении. Слушатель `OnCameraChangeListener` передает информацию о перемещениях устройства, а при помощи слушателя `OnMapClickListener` можно определить широту и долготу точки на карте, на которой был сделан щелчок:

```
map.setOnCameraChangeListener(new OnCameraChangeListener() {
    @Override
    public void onCameraChange(CameraPosition cameraPosition) {
        // Перемещение в новое место на карте
    }
});

map.setOnMapClickListener(new OnMapClickListener() {
    @Override
    public void onMapClick(LatLng latLng) {
        // Щелчок сделан в точке с широтой/долготой latLng
    }
});
```

За дополнительной информацией о картах Google и об их интеграции с приложениями Android обращайтесь по адресу:

<https://developer.android.com/google/play-services/maps.html>

6. Загрузчики курсоров

Если вы интенсивно работаете с базами данных или провайдерами контента, рано или поздно вы столкнетесь с загрузчиками курсоров (`cursor loaders`). Загрузчик курсора выполняет асинхронный запрос в фоновом режиме и возвращает результаты активности или фрагменту, из которых он был вызван. Загрузчик автоматически управляет курсором, чтобы вам не приходилось этим заниматься, а также оповещает об изменении данных, чтобы вы могли обработать это изменение в интерфейсе.

За дополнительной информацией о загрузчиках курсоров обращайтесь по адресу:

<https://developer.android.com/training/load-data-background/setup-loader.html>

7. Широковещательные приемники

Допустим, вы хотите, чтобы ваше приложение определенным образом реагировало на системные события. Например, воспроизведение музыки в вашем приложении должно прерываться при отсоединении наушников. Как сообщить приложению о наступлении таких событий?

К числу системных событий относятся такие ситуации, как низкий заряд аккумулятора, входящий телефонный звонок или загрузка системы. Android рассылает эти системные события при их возникновении; их можно прослушивать при помощи широковещательного приемника (broadcast receiver). Широковещательные приемники позволяют подписаться на конкретные широковещательные сообщения. Таким образом, ваше приложение может реагировать на события системного уровня.

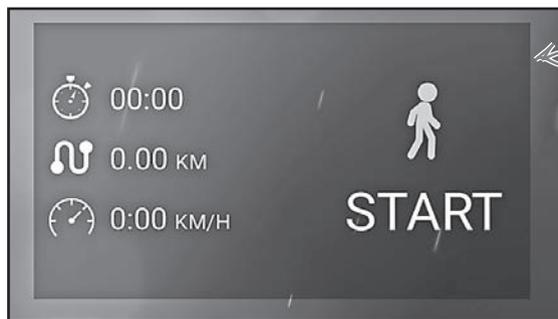


Дополнительная информация о широковещательных приемниках доступна по адресу:

<http://developer.android.com/reference/android/content/BroadcastReceiver.html>

8. Виджеты приложений

Виджет приложения представляет собой миниатюрное представление, которое можно добавлять в другие приложения или на домашний экран. Виджет предоставляет доступ к контенту или функциональности приложения прямо с домашнего экрана, так что вам не приходится запускать приложение. Пример виджета приложения:



Это виджет приложения. Он предоставляет прямой доступ к основной функциональности приложения.

Для создания виджета приложения вам понадобится объект `AppWidgetProviderInfo`, реализация класса `AppWidgetProvider` и макет `View`. Объект `AppWidgetProviderInfo` содержит метаданные для виджета (например, описания класса `AppWidgetProvider` и макета). Он определяется в разметке XML. Реализация класса `AppWidgetProvider` содержит методы, с которыми вам придется взаимодействовать в виджете приложения. Макет `View` представляет собой макет XML с описанием виджета.

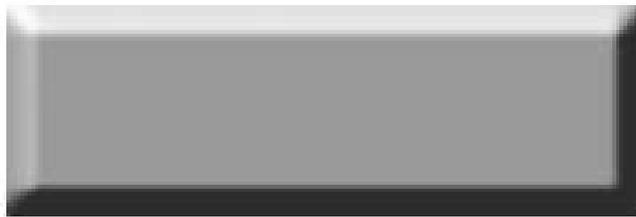
Если вас интересует тема создания виджетов приложений, необходимую информацию можно найти по адресу:

<http://developer.android.com/guide/topics/appwidgets/index.html>

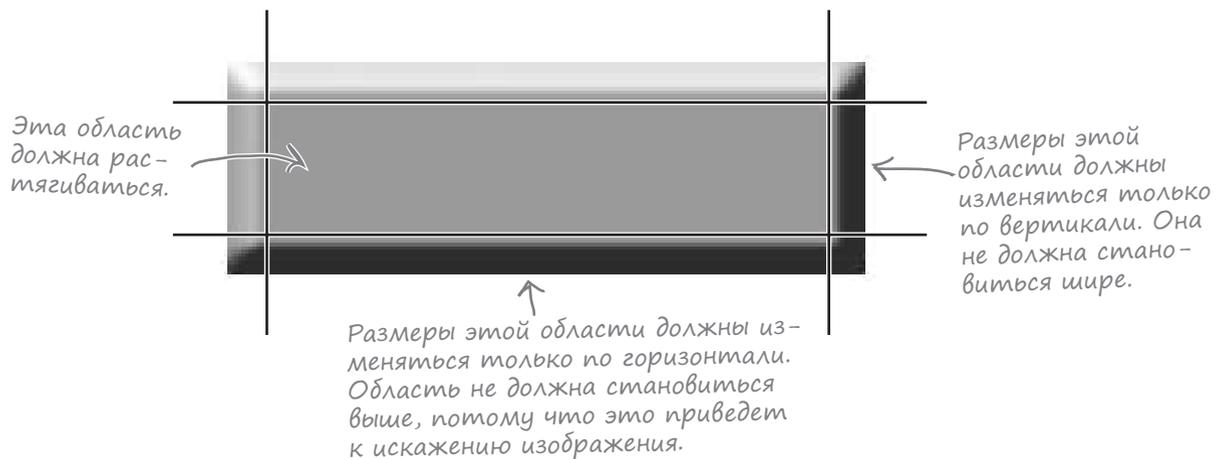
9. Графика NinePatch

NinePatch – растровое изображение, которое может использоваться в качестве фона представления. Изображение автоматически изменяет размеры в зависимости от содержимого представления и размера экрана. Здесь интересно то, что вы сами определяете, какие части могут растягиваться, а какие должны сохранять свои размеры.

Допустим, вы хотите использовать следующее изображение в качестве фона кнопки:



Это изображение должно растягиваться под разную длину текста, но при этом края не должны искажаться при растяжении:



Если преобразовать изображение в графику NinePatch, вы сможете растягивать его в точности так, как требуется.

В Android имеется программа Draw 9-patch, упрощающая создание изображений NinePatch. Дополнительная информация о программе Draw 9-patch и графике NinePatch доступна по ссылке:

<http://developer.android.com/guide/topics/graphics/2d-graphics.html#nine-patch>

10. Тестирование

Все современные методы разработки сильно зависят от тестирования, поэтому в Android реализована обширная встроенная поддержка тестирования. Так как основным языком программирования Android является Java, вы можете использовать стандартные инфраструктуры тестирования Java, но Android идет еще дальше – инфраструктура тестирования включается прямо в SDK. Более того, Android Studio автоматически генерирует иерархию файлов для тестирования каждый раз, когда вы создаете проект.

Поддержка тестирования в Android базируется на JUnit с расширениями, построенными специально для Android. Для простейшего тестирования компонентов используются объекты `AndroidTestCase`. Инфраструктура включает модели для таких объектов, как `Intent` и `Context`, упрощающие тестирование отдельных компонентов. Также имеется специальная разновидность `ApplicationTestCase` для тестирования конфигурации таких файлов, как `AndroidManifest.xml`. Самой впечатляющей особенностью базовой инфраструктуры тестирования является *инструментальное тестирование*. Приложения Android могут быть настроены таким образом, что разработчик может отслеживать взаимодействия между компонентом и операционной системой и вносить в них изменения. Это означает, что вы можете запускать прямо на устройстве тесты, которые вызывают методы жизненного цикла активности и отправляют интенды операционной системе. За дополнительной информацией о средствах тестирования Android обращайтесь по адресу:

http://d.android.com/tools/testing/testing_android.html

Если вам потребуются более мощные средства тестирования сценариев, обратите внимание на инфраструктуру тестирования **Robotium**. Robotium расширяет возможности инструментального тестирования базовой инфраструктуры Android и выводит их на принципиально новый уровень. При использовании Robotium вы можете писать тестовый код, который почти не отличается от тестовых сценариев, выполняемых при ручном тестировании.

За дополнительной информацией о Robotium обращайтесь по адресу:
<https://code.google.com/p/robotium/>