

Содержание (сводка)

	Введение	29
1	Первые шаги. <i>С головой в пучину</i>	39
2	Построение интерактивных приложений. <i>Приложения, которые что-то делают</i>	75
3	Макеты. <i>Как работают макеты</i>	119
4	Макеты с ограничениями. <i>Построение эскиза</i>	159
5	Жизненный цикл активности. <i>Из жизни активностей</i>	207
6	Фрагменты и навигация. <i>Найди свой путь</i>	257
7	Плагин safe args. <i>Передача информации</i>	295
8	Интерфейс навигации. <i>Туда и обратно</i>	331
9	Представления material. <i>Материальный мир</i>	393
10	Связывание представлений. <i>Связанные одной целью</i>	441
11	Модели представлений. <i>Поведение модели</i>	473
12	Живые данные. <i>В самой гуще событий</i>	521
13	Связывание данных. <i>Построение умных макетов</i>	557
14	Базы данных Room. <i>Номер с видом</i>	607
15	Представления с переработкой. <i>Экономия и переработка</i>	659
16	Diffutil и связывание данных. <i>На полных оборотах</i>	709
17	Навигация в представлениях с переработкой. <i>Карточные фокусы</i>	743
18	Jetpack compose. <i>В мире Compose</i>	793
19	Интеграция с представлениями. <i>Полная гармония</i>	849
	Приложение. Остатки. <i>Десять важных тем, которые мы не рассмотрели</i>	899

Содержание (настоящее)

Введение

Ваш мозг и Android. Вы пытаетесь изучить что-то новое, а ваш мозг хочет оказать вам услугу и как можно быстрее забыть выученное. Он думает: «Лучше оставить место для чего-то поважнее, например, от каких диких животных стоит держаться подальше или почему на сноуборде не стоит кататься нагишом». Так как же заставить ваш мозг думать, что ваша жизнь зависит от умения разрабатывать приложения для Android?

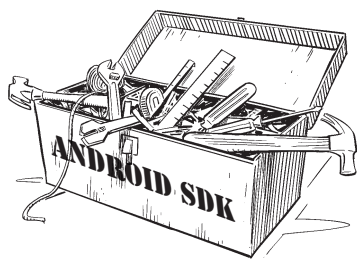
Для кого написана эта книга?	30
Мы знаем, о чем вы думаете	31
Метапознание: наука о мышлении	33
Вот что сделали мы	34
Что можете сделать вы	35
Примите к сведению	36

1

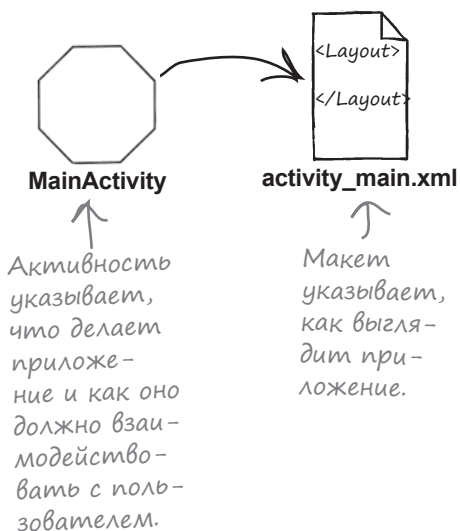
Первые Шаги

С головой в пучину

Android — самая популярная мобильная операционная система в мире. А по всему миру живут миллиарды пользователей Android, и все они мечтают загрузить вашу следующую замечательную разработку. В этой главе вы узнаете, как воплотить ваши идеи в жизнь для самой популярной мобильной операционной системы в мире, как **построить базовое приложение Android** и как обновить его. Также вы узнаете, как запустить его на физических и виртуальных устройствах. А попутно будут рассмотрены основные компоненты всех приложений Android: **активности** и **макеты**. Итак, за дело!



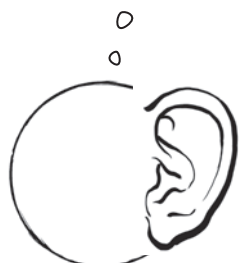
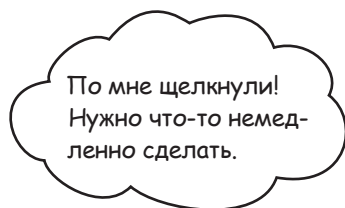
Добро пожаловать в мир Android	40
Активности и макеты образуют костяк вашего приложения	41
Вот что мы сейчас сделаем	42
Android Studio: среда разработки	43
Установка Android Studio	44
Построение простого приложения	45
Как построить приложение	46
Вы создали свой первый проект для Android	50
Структура нового проекта	51
Ключевые файлы вашего проекта	52
Редактирование кода в Android Studio	53
Чего мы добились	54
Запуск приложения на физическом устройстве	57
Как запустить приложение на виртуальном устройстве	58
Создание Android Virtual Device (AVD)	59
Компиляция, упаковка, развертывание, запуск	63
Что только что произошло?	65
Доработка приложения	66
Что содержит макет?	67
activity_main.xml состоит из двух элементов	68
Обновление текста в макете	71
Что делает код	72
Ваш инструментарий Android	74



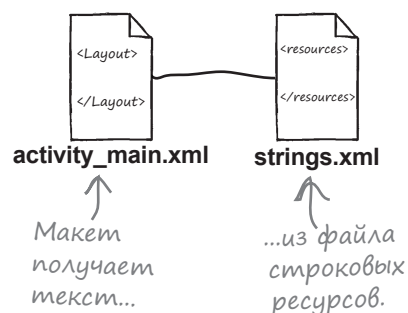
2 Построение интерактивных приложений

Приложения, которые что-то делают

Обычно приложение должно реагировать на действия пользователя. В этой главе вы узнаете, как существенно повысить **интерактивность** ваших приложений. Вы узнаете, как добавить в код активности метод **OnClickListener**, чтобы приложение могло **прослушивать** действия пользователя и соответствующим образом на них реагировать. Также вы научитесь **конструировать макеты** и поймете, как каждый UI-компонент, добавляемый в макет, происходит от общего **предка View**. Попутно вы узнаете, почему **строковые ресурсы** настолько важны для гибких, хорошо спроектированных приложений.



Кнопка

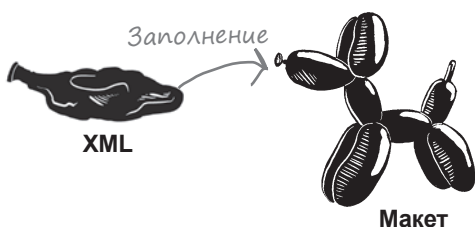


В этой главе мы построим приложение для выбора пива	76
Подробнее о визуальном редакторе	80
Добавление кнопки в визуальном редакторе	81
В activity_main.xml появилась новая кнопка	82
Подробнее о коде макета	83
Обновление разметки XML макета	85
Изменения XML отражаются в визуальном редакторе	86
Для макета выдаются предупреждения...	88
Размещение текста в файле строковых ресурсов	89
Извлечение строкового ресурса	90
activity_main.xml использует строковый ресурс	91
Добавление и использование нового строкового ресурса	92
Добавление значений в список	97
Добавление элемента string-array в файл strings.xml	98
Полный код файла activity_main.xml	99
Приложение должно быть интерактивным	101
Как выглядит код MainActivity	102
Передача лямбда-выражения методу setOnClickListener	105
Как редактировать текст надписи	106
Обновленный код MainActivity.kt	109
Добавление метода getBeers()	112
Полный код MainActivity.kt	115
Что происходит при запуске кода	116
Ваш инструментарий Android	118

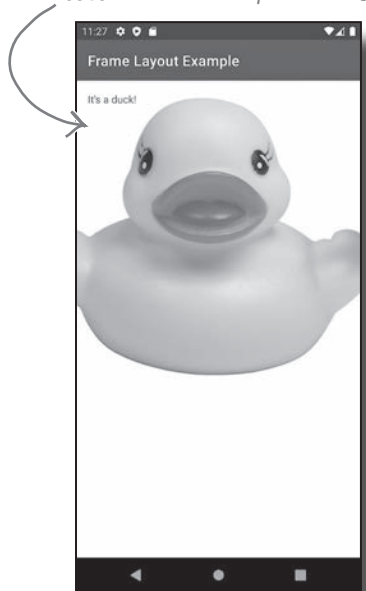
3 Макеты

Как работают макеты

Мы едва затронули тему использования макетов. К настоящему моменту вы видели, как разместить представления в простых линейных макетах, но это лишь малая часть того, на что способны макеты. В этой главе мы **рассмотрим вопрос глубже** и покажем, как на самом деле работают макеты. Вы узнаете, как **настроить линейные макеты**, и научитесь использовать **фреймовые макеты** и **представления с прокруткой**. А к концу главы вы узнаете, что, несмотря на внешние различия, все макеты — и добавленные к ним представления — имеют **больше общего**, чем кажется на первый взгляд.



Мы воспользуемся фреймовым макетом для наложения текстового поля на изображение утки.



Все начинается с макета	120
В Android существуют разные виды макетов	121
Построение линейного макета	122
Как определить линейный макет	123
Вертикальная и горизонтальная ориентация	124
Использование padding для добавления отступов	126
Текущий код макета	127
Добавление представлений в XML макета	129
Растяжение представлений	130
Назначение веса одному представлению	131
Назначение весов нескольким представлениям	132
Список значений атрибута android:gravity	134
Другие допустимые значения атрибута android:layout_gravity	139
Разделение представлений свободными промежутками	140
Полный код линейного макета	141
Код активности сообщает Android, какой макет он использует	145
Заполнение макета: пример	146
Фреймовый макет накладывает представления друг на друга	147
Добавление изображений в проект	148
Фреймовый макет накладывает изображения в порядке их перечисления в разметке XML макета	150
Все макеты являются специализациями ViewGroup...	151
Представление с прокруткой добавляет вертикальную полосу прокрутки	153
Как добавить представление с прокруткой	154
Ваш инструментарий Android	158

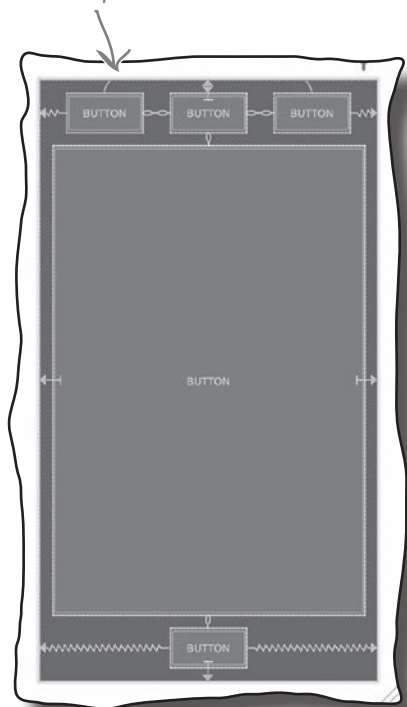
4

Макеты с ограничениями

Построение эскиза

Дом невозможно построить без плана. А некоторые макеты строятся по **эскизам**, чтобы они выглядели **в точности так, как вам нужно**. В этой главе представлены **ограничения в макетах Android: гибкий механизм проектирования более сложных графических интерфейсов**. Вы узнаете, как **ограничения и смещения** используются для позиционирования и определения размеров ваших представлений **независимо от размера и ориентации экрана**. Вы узнаете, как закрепить представления в положенных местах при помощи **направляющих и барьеров**. Наконец, вы научитесь группировать или распределять представления с использованием **цепочек и потоков**. За дело!

Эскиз макета с ограничениями. Он содержит всю информацию, необходимую макету с ограничениями для размещения представлений на экране.



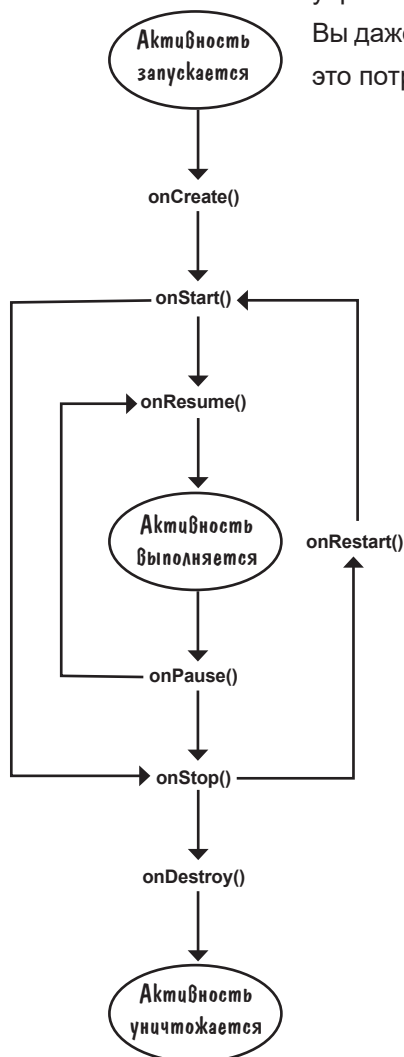
Снова о вложенных макетах	160
Макеты с ограничениями: первое знакомство	162
Макеты с ограничениями являются частью Android Jetpack	163
Использование Gradle для включения библиотек Jetpack	165
Добавление кнопки в макет	167
Размещение представлений с ограничениями	168
Добавление вертикального ограничения	169
Использование противоположных ограничений для выравнивания по центру	171
Удаление ограничений с использованием виджета	173
Для представлений может определяться смещение	175
Размеры представлений можно изменять	177
В макетах обычно используется несколько представлений	182
Представления можно связать с другими представлениями	183
Выравнивание представлений по направляющим	185
Направляющие имеют фиксированную позицию	186
Создание подвижного барьера	187
Добавление горизонтального барьера	188
Размещение кнопки под барьером	189
Использование цепочек для управления линейными группами представлений	192
Создание горизонтальной цепочки	194
Разные стили цепочек	197
Добавление потока	201
Управление внешним видом потока	202
Ваш инструментарий Android	206

5

Жизненный цикл активности

Из жизни активностей

Активности образуют основу любого Android-приложения. Ранее вы видели, как создавать активности и как использовать активности для взаимодействия с пользователем. Но если вы еще не знакомы с **жизненным циклом активностей**, некоторые аспекты их поведения могут вас заставить врасплох. В этой главе вы узнаете, что происходит при **создании** или **уничтожении** активностей и к каким **неожиданным последствиям** это может привести. Вы узнаете, как управлять их поведением, когда они становятся **видимыми** или **скрываются**. Вы даже научитесь **сохранять** и **восстанавливать состояние активности**, когда это потребуется. Просто продолжайте читать дальше...



Как на самом деле работают активности?	208
Создание нового проекта	209
Полный код activity_main.xml	210
Код активности управляет работой хронометра	211
Полный код MainActivity.kt	212
Что происходит при запуске приложения	214
История продолжается	215
Что происходит при запуске приложения	217
Поворот экрана изменяет конфигурацию устройства	218
Состояние выполнения	219
Жизненный цикл активности: от рождения до смерти	220
Активность наследует методы жизненного цикла	221
Сохранение текущего состояния в объекте Bundle	222
Сохранение состояния вызовом onSaveInstanceState()	223
Жизненный цикл активности: видимость	232
Перезапуск отсчета времени, когда приложение становится видимым	234
А если активность видна только частично?	242
Жизненный цикл переднего плана	243
Приостановка отсчета времени при приостановке активности	244
Полный код MainActivity.kt	247
Что происходит при запуске приложения	250
Краткая сводка методов жизненного цикла активностей	252
Ваш инструментарий Android	255

6 Фрагменты и навигация

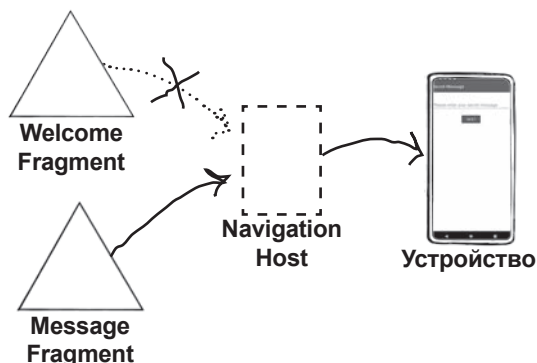
Найди свой путь

Для большинства приложений одного экрана недостаточно. До настоящего момента мы рассматривали приложения с одним экраном; для простых приложений этого хватает. А если в вашем приложении **более сложные требования**? В этой главе вы научитесь использовать **фрагменты** и компонент **Navigation** для построения приложений с несколькими экранами. Вы узнаете, что **фрагменты похожи на субактивности**, имеющие собственные методы. Вы научитесь **проектировать эффективные графы навигации**. В последней части главы представлен **навигационный хост** и **навигационный контроллер**; вы узнаете, как они используются для перемещения в приложениях.



Navigation Graph

Граф навигации — что-то вроде GPS-навигатора, который сообщает вам, как добраться до того или иного места.



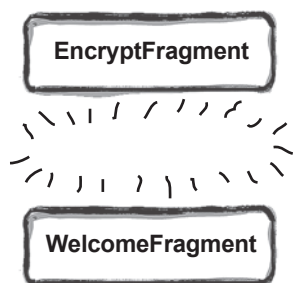
Многим приложениям недостаточно одного экрана	258
Переходы между экранами с использованием компонента Navigation	260
Создание нового проекта	262
Добавление фрагмента WelcomeFragment	263
Метод onCreateView() фрагмента	265
Код макета фрагмента похож на код макета активности	266
Отображение фрагмента в FragmentContainerView	267
Обновление кода activity_main.xml	268
Создание MessageFragment	273
Обновление макета MessageFragment	274
Обновление MessageFragment.kt	275
Применение компонента Navigation для перехода между фрагментами	276
Добавление компонента Navigation в проект с использованием Gradle	277
Создание графа навигации	278
Добавление фрагментов в граф навигации	279
Соединение фрагментов действиями	280
Графы навигации как ресурсы XML	281
Добавление хоста навигации в макет при помощи FragmentContainerView	282
Добавление NavHostFragment в activity_main.xml	283
Добавление OnClickListener для кнопки	284
Получение контроллера навигации	286

7

Плагин `safe args`

Передача информации

Иногда фрагментам для нормальной работы требуется дополнительная информация. Например, если во фрагменте выводятся контактные данные, этот фрагмент должен знать, какой контакт он должен выбрать. Но что, если эта информация **должна поступать из другого фрагмента**? В этой главе мы **воспользуемся вашими знаниями о навигации** и применим их **для передачи данных между фрагментами**. Вы узнаете, **как добавить аргументы** к целям навигации, чтобы они могли получать необходимую информацию. Вы познакомитесь с плагином **Safe Args** и научитесь использовать его **для написания кода, безопасного по отношению к типам**. Наконец, вы научитесь **работать со стеком возврата** и управлять поведением кнопки Back. Читайте дальше — отступать уже поздно.



Удаление `MessageFragment` из стека возврата означает, что при щелчке на кнопке Back из `EncryptFragment` вместо `MessageFragment` отображается `WelcomeFragment`.

Приложение Secret Message переходит между фрагментами	296
Фрагмент <code>MessageFragment</code> должен передать сообщение новому фрагменту	297
Что нам предстоит сделать	298
Создание фрагмента <code>EncryptFragment</code> ...	299
Обновление <code>EncryptFragment.kt</code>	300
Включение <code>EncryptFragment</code> в граф навигации	301
Обновленный код <code>nav_graph.xml</code>	302
Фрагмент <code>MessageFragment</code> должен переходить к <code>EncryptFragment</code>	303
Добавление Safe Args в файлы <code>build.gradle</code>	305
Фрагмент <code>EncryptFragment</code> должен получать строковый аргумент	306
Обновленный код <code>nav_graph.xml</code>	307
Фрагмент <code>MessageFragment</code> должен передать сообщение <code>EncryptFragment</code>	308
Safe Args генерирует классы <code>Directions</code>	309
Обновление кода <code>MessageFragment.kt</code>	310
Фрагмент <code>EncryptFragment</code> должен получить значение аргумента	311
Полный код <code>EncryptFragment.kt</code>	312
Знакомство со стеком возврата	322
Использование графа навигации для извлечения фрагментов из стека возврата	323
Обновленный код <code>nav_graph.xml</code>	324
Ваш инструментарий Android	330

8

Интерфейс навигации

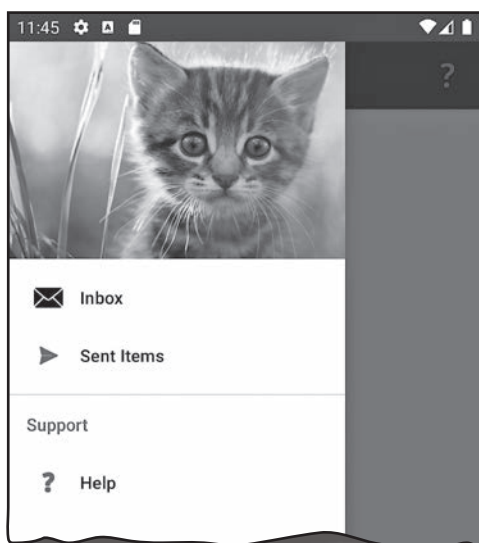
Туда и обратно

Во многих приложениях возникает необходимость в переходе между разными целями. Компонент Navigation сильно упрощает построение таких пользовательских интерфейсов. В этой главе вы научитесь пользоваться некоторыми навигационными UI-компонентами для Android, упрощающими перемещение пользователя по приложению. Вы узнаете, как пользоваться **темами** и как заменить панель приложения **панелью инструментов**. Вы научитесь добавлять **элементы меню**, которые могут использоваться для **навигации**, и освоите **реализацию навигации с нижней панели**. Наконец, мы создадим эффектную выдвигающую панель, которая появляется на экране сбоку от вашей активности.

Проследите за тем, чтобы идентификаторы совпадали, и я отведу вас туда, куда вам нужно..



Контроллер навигации



Разные приложения, разные структуры	332
В Android существуют навигационные UI-компоненты	333
Как работает приложение CatChat	334
Применение темы в AndroidManifest.xml	338
Определение стилей в файлах стилевых ресурсов	339
Замена панели приложения по умолчанию панелью инструментов	341
Создание HelpFragment	347
Определение элементов панели инструментов в файле ресурсов меню	353
onOptionsItemSelected() добавляет элементы меню на панель инструментов	355
Настройка панели инструментов с использованием AppBarConfiguration	358
Что происходит во время запуска приложения	360
Многие разновидности UI-навигации работают с компонентом Navigation	365
Создание SentItemsFragment	366
Нижней панели навигации потребуется новый файл ресурсов меню	368
Связывание нижней панели с контроллером навигации	371
Добавление категории Support...	378
Выделение текущего элемента в группах	379
Создание заголовка выдвигающей панели	381
Как создать выдвигающую панель	382
Настройка значка выдвигающей панели на панели инструментов...	385
Ваш инструментарий Android	391

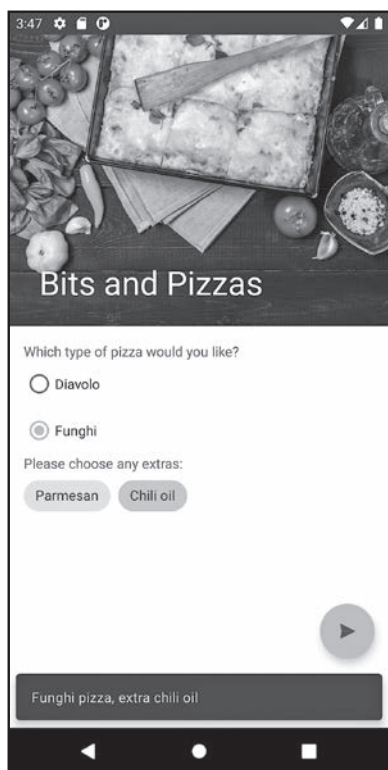
Если не связать выдвигающую панель с контроллером навигации, элементы будут отображаться на выдвигающей панели, но по щелчку на них ничего не произойдет.

9

Представления material

Материальный мир

Многим приложениям нужен эффектный пользовательский интерфейс, быстро реагирующий на действия пользователя. В предыдущих главах вы научились пользоваться разными представлениями: **текстовыми представлениями, кнопками и полями выбора**, а также применять **темы Material** для внесения масштабных изменений в оформление и поведение приложения. Впрочем, это далеко не все. В этой главе вы узнаете, как улучшить скорость реакции вашего приложения применением **координирующего макета**. Вы создадите **панели инструментов** с возможностью сворачивания или прокрутки содержимого по вашему усмотрению. Вы познакомитесь с **интересными новыми представлениями: флажками, переключателями, плашками и плавающими кнопками действий**. Наконец, вы узнаете, как отображать удобные всплывающие сообщения с использованием панелей **Toast** и **Snackbar**. Переверните страницу, чтобы узнать больше.

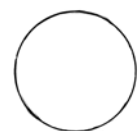


Material широко используется в мире Androidville	394
Приложение Bits and Pizzas	395
Создание OrderFragment	398
У фрагментов нет метода <code>setSupportActionBar()</code>	401
Координирующий макет координирует анимации между представлениями	403
Макет панели приложения включает анимацию панели инструментов	404
Создание сворачиваемой панели инструментов	411
Создание простой сворачиваемой панели инструментов	412
Добавление изображения	414
Построение основного контента OrderFragment	418
Выбор вида пиццы при помощи переключателя	419
Переключатель – разновидность композитной кнопки	420
Плашка – разновидность композитной кнопки	421
Объединение нескольких плашек в группу	422
FAB – плавающая кнопка действия	423
Построение макета OrderFragment	425
Добавление <code>OnClickListener</code> к FAB	431
Toast – простое всплывающее сообщение	432
Вывод заказа в сообщении Snackbar	433
Код Snackbar для заказа пиццы	434
Полный код OrderFragment.kt	435
Ваш инструментарий Android	440

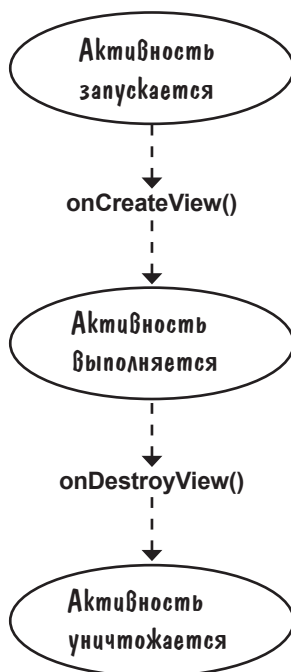
10 (Вязывание представлений

Связанные одной целью

Пришло время попрощаться с `findViewById()`. Как вы, вероятно, уже заметили, чем больше у вас представлений и чем более интерактивным становится ваше приложение, тем больше вызовов **`findViewById()`** приходится использовать. И если вам надоело вызывать этот метод каждый раз, когда вам потребуется поработать с представлением, знайте: *вы не одиноки*. В этой главе вы узнаете, **как связывание представлений делает `findViewById()` пережитком прошлого**. Вы научитесь применять этот механизм в коде **активностей и фрагментов** и узнаете, почему этот подход предоставляет более безопасный и эффективный способ обращения к представлениям вашего макета. Итак, за дело!



RadioGroup



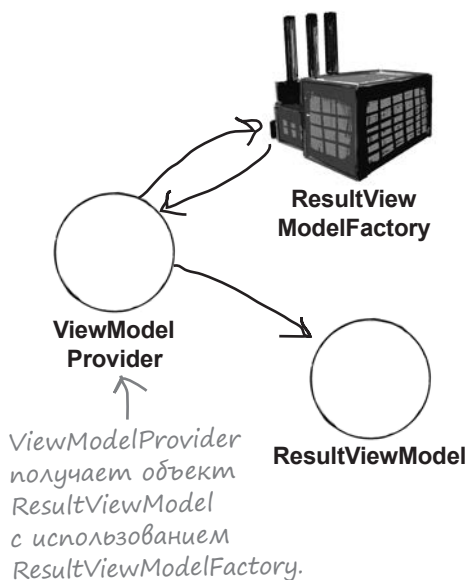
Под капотом <code>findViewById()</code>	442
История продолжается	443
У <code>findViewById()</code> есть оборотная сторона	444
На помощь приходит связывание представлений	445
Как использовать связывание представлений	446
Снова к приложению Stopwatch	447
Включение связывания представлений в файле <code>build.gradle</code> приложения	448
Как добавить связывание представлений в активность	449
Использование свойства <code>binding</code> для взаимодействия с представлениями	450
Полный код <code>MainActivity.kt</code>	451
Что делает код	454
Фрагменты тоже могут использовать связывание представлений, но код выглядит немного иначе	457
Включение связывания представлений для <code>Bits and Pizzas</code>	458
Фрагменты могут обращаться к представлениям от <code>onCreateView()</code> до <code>onDestroyView()</code>	460
Как выглядит код связывания представлений для фрагмента	462
<code>_binding</code> хранит ссылку на объект связывания...	463
Полный код <code>OrderFragment.kt</code>	465
Ваш инструментарий Android	471

11

Модели представлений

Поведение модели

С ростом сложности приложений фрагментам приходится жонглировать все большим количеством объектов. И если не проявить осторожность, это может привести к **разбуханию** кода, который пытается делать все сразу. Бизнес-логика, навигация, управление пользовательским интерфейсом, обработка изменений в конфигурации... что ни возьми, оно здесь. В этой главе вы узнаете, как действовать в подобных ситуациях с использованием **моделей представлений**. Вы узнаете, как они упрощают код активности и фрагментов и как они справляются с изменениями конфигурации, поддерживая целостное состояние приложения. Наконец, мы покажем, как построить **фабрику моделей представлений** и когда может возникнуть такая необходимость.



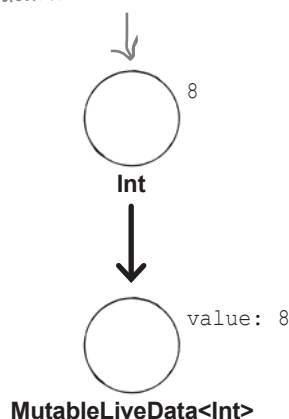
Снова об изменениях конфигурации	474
Знакомство с моделями представлений	475
Как будет работать игра	476
Структура приложения	477
Обновление файла build.gradle проекта...	479
В приложении Guessing Game используются два фрагмента	480
Как должна работать навигация	481
Обновление графа навигации	482
Что происходит при выполнении приложения	491
При повороте экрана состояние теряется	493
Модель представления содержит бизнес-логику	494
Включение зависимости для модели представления в файл build.gradle приложения...	495
Создание объекта GameViewModel	498
Что происходит при выполнении приложения	501
ResultViewModel необходимо хранить результат	508
Фабрика модели представления создает модели представлений	509
Создание класса ResultViewModelFactory	510
Использование фабрики для создания модели представления	511
Что происходит при выполнении приложения	514
Ваш инструментарий Android	520

12 Живые данные

В самой гуще событий

Вашему коду часто приходится реагировать на изменения значений свойств. Например, если свойство модели представления меняет значение, **фрагмент может отреагировать** обновлением своих представлений или переходом. Но **как фрагмент узнает о том, что свойство было обновлено?** В этой главе вы узнаете о **Live Data**: механизме оповещения заинтересованных сторон о каких-либо изменениях. Вы узнаете о **MutableLiveData** и о том, как заставить ваш фрагмент наблюдать за изменениями свойств этого типа. Мы покажем, как тип **LiveData** помогает обеспечить целостность данных вашего приложения. Вскоре вы будете писать приложения, которые реагируют на происходящее быстрее, чем когда-либо.

Свойство `livesLeft`
преобразуется из `Int`
в `MutableLiveData<Int>`.
Свойство `livesLeft`
преобразуется из `Int`
в `MutableLiveData<Int>`.



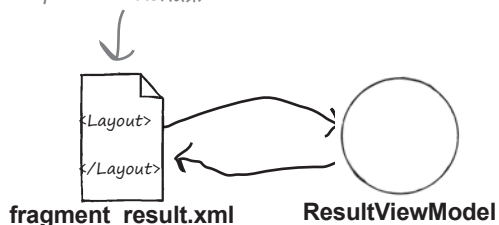
Снова о приложении Guessing Game	522
Фрагменты решают, когда обновлять представления	523
Что нам предстоит сделать	524
GameViewModel и GameFragment должны использовать Live Data	525
Объекты Live Data используют свойство value	526
Полный код GameViewModel.kt	528
Фрагмент наблюдает за свойствами модели представления и реагирует на изменения	530
Полный код GameFragment.kt	531
Что происходит при выполнении приложения	534
Фрагменты могут обновлять свойства GameViewModel	539
Полный код GameViewModel.kt	540
Что происходит при выполнении приложения	542
GameFragment содержит игровую логику	544
Полный код GameViewModel.kt	547
Отслеживание нового свойства в GameFragment	549
Что происходит при выполнении приложения	551
Ваш инструментарий Android	556

13 (Вязывание данных)

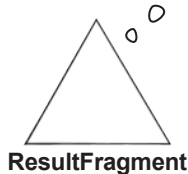
Построение умных макетов

Возможности макетов не ограничиваются управлением внешним видом вашего приложения. У всех макетов, написанных нами ранее, поведение должно было определяться кодом активности или фрагмента. Но только представьте, что макеты могли бы думать самостоятельно и принимать собственные решения. В этой главе представлен механизм **связывания данных**: способ наращивания интеллекта ваших макетов. Вы узнаете, **как заставить представления получать данные** непосредственно от модели представления. Мы воспользуемся **связыванием слушателей**, чтобы кнопки вызывали свои методы. Вы даже увидите, как **одна простая строка кода позволяет реагировать на обновления Live Data**. Скоро ваши макеты станут более мощными, чем когда-либо.

Текстовое представление макета может получить свой текст напрямую от модели представления.



Короче, сами разбирайтесь.



Снова к приложению Guessing Game	558
Фрагменты обновляют представления в своих макетах	559
Включение связывания данных в файле build.gradle приложения	561
ResultFragment обновляет текст в своем макете	562
1. Добавление элементов <layout> и <data>	563
2. Присваивание значения переменной связывания данных в макете	564
3. Использование переменной связывания данных макета для обращения к модели представления	565
Что происходит во время выполнения приложения	568
GameFragment тоже может использовать связывание данных	573
Добавление элементов <layout> и <data> в fragment_game.xml	574
Использование переменной связывания данных для задания текста в макете	575
Снова о строковых ресурсах	576
Макет может передавать параметры строковым ресурсам	577
Полный код fragment_game.xml	578
Необходимо задать значение переменной gameViewModel	580
Что происходит при выполнении приложения	583
Связывание данных может использоваться для вызова методов	587
Добавление метода finishGame() в GameViewModel.kt	588
Использование связывания данных для вызова метода по щелчку на кнопке	590
Что происходит при выполнении приложения	593
Связывание представлений можно отключить	598
Ваш инструментарий Android	605

14 Базы данных Room

Номер с видом

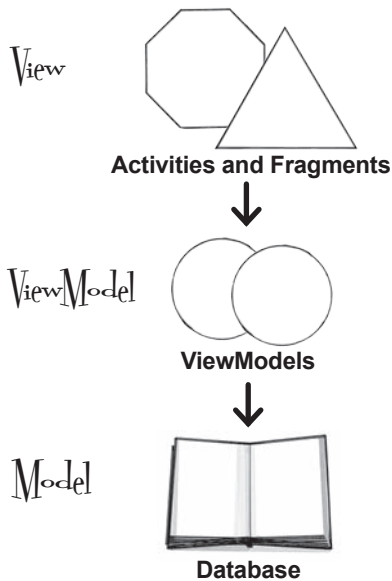
В большинстве приложений используются данные. Но если вы не позаботитесь о том, чтобы эти данные были где-то сохранены, **данные будут навсегда потеряны** при закрытии приложения. В мире приложений Android информация обычно **хранится в базах данных**, поэтому в этой главе мы представим **библиотеку хранения данных Room**. Вы научитесь **строить базы данных, создавать таблицы и определять методы доступа к данным** с использованием аннотаций классов и интерфейсов. Вы узнаете, как **использовать сопрограммы** для выполнения кода баз данных в фоновом режиме. Заодно вы научитесь **преобразовывать данные Live Data при их изменении с помощью Transformations.map()**.

Не угодно ли чаю к данным?

Интерфейс DAO обеспечивает все ваши потребности в доступе к данным. Просто укажите, что вам нужно, а DAO сделает это за вас.



TaskDao

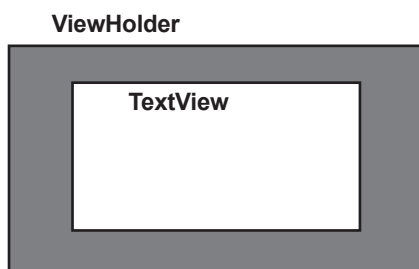


Многим приложениям требуется хранить данные	608
Room — библиотека баз данных, работающая на SQLite	610
Создание TaskFragment	613
Как создаются базы данных Room	616
Данные задач будут храниться в таблице	617
Определение имени таблицы аннотацией @Entity	618
Использование интерфейса для определения операций с данными	620
Использование аннотации @Insert для вставки записи	621
Использование @Delete для удаления записей	622
Создание абстрактного класса TaskDatabase	624
Добавление свойств для интерфейсов DAO	625
Снова о MVVM	629
Операции баз данных могут работать медленнеееенноооооо...	631
1. Пометка методов TaskDao ключевым словом suspend	632
2. Метод insert() запускается в фоновом режиме	633
TaskViewModel необходима фабрика модели представления	634
TaskFragment может использовать связывание представлений	637
Для вставки данных будет использоваться связывание данных	638
Что происходит при выполнении кода	641
В TaskFragment должны выводиться записи	645
Чтение всех задач из базы данных методом getAll()	646
LiveData<List<Task>> — более сложный тип	647
Свойство tasksString будет связано с текстовым представлением в макете	649
Ваш инструментарий Android	658

15 Представления с переработкой

Экономия и переработка

Списки данных являются важнейшей частью многих приложений. В этой главе мы покажем, как создать такой список с использованием **представлений с переработкой**: **невероятно гибкого** способа построения **прокручиваемых списков**. Вы научитесь создавать **гибкие макеты** для ваших списков, включая текстовые представления, флажки и многое другое. Вы узнаете, **как создавать адаптеры**, которые **отображают ваши данные** в представлениях с переработкой так, как вам нужно. Вы научитесь использовать **карточные представления** для оформления данных **с имитацией объема**. Наконец, мы покажем, как **менеджеры макетов могут полностью изменить внешний вид вашего списка всего одной или двумя строками кода**. Давайте займемся переработкой.




Каждый держатель представления содержит корневое представление макета для каждого элемента. В данном случае корневым является текстовое представление.

Как в настоящее время выглядит приложение Tasks	660
Список можно преобразовать в представление с переработкой	661
Для чего нужны представления с переработкой?	662
Чтобы сообщить представлению с переработкой, как отображать каждый элемент...	664
Адаптер добавляет данные в представление с переработкой	665
Сообщаем адаптеру, с какими данными он должен работать	666
Определение держателя представления для адаптера	667
Переопределение метода onCreateView()	668
Добавление данных в представление макета	669
Код адаптера готов	671
В приложении должно отображаться представление с переработкой	672
Представление с переработкой добавляется в макет TasksFragment	676
Обновление кода TasksViewModel.kt	677
Фрагмент TasksFragment должен обновлять свойство данных TaskItemAdapter	678
Что происходит при выполнении кода	681
Представления с переработкой чрезвычайно гибки	688
Создание карточного представления	690
Держатель представления адаптера должен работать с новым кодом макета	692
Галерея менеджеров макетов	696
Обновление fragment_tasks.xml для размещения элементов в сетке	697
Ваш инструментарий Android	707

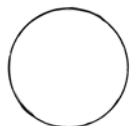
16 DiffUtil и связывание данных

На полных оборотах

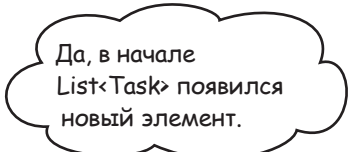
Приложение должно быть настолько плавным и быстрым, насколько это возможно. Но если действовать неосторожно, большие и сложные наборы данных могут привести к сбоям в вашем представлении с переработкой. В этой главе мы представим *DiffUtil*: вспомогательный класс, который **расширяет возможности представлений с переработкой**. Вы научитесь использовать его для **эффективного обновления** представлений с переработкой. Вы узнаете, как объекты *ListAdapter* упрощают работу с *DiffUtil*. А заодно будет показано, **как полностью избавиться от *findViewById()*** реализацией **связывания данных в коде представления с переработкой**.



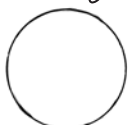
Видишь
какие-нибудь
различия?



TaskItemAdapter



Да, в начале
List<Task> появился
новый элемент.

TaskDiff
ItemCallback

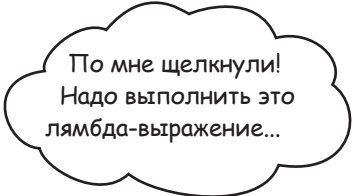
Снова о приложении Tasks	711
Как представление с переработкой получает свои данные	712
Метод записи свойства data вызывает notifyDataSetChanged()	713
Передача представлению с переработкой информации о том, что должно измениться	714
Что мы собираемся сделать	715
Реализация DiffUtil.ItemCallback	716
ListAdapter получает аргумент DiffUtil.ItemCallback	717
Обновленный код TaskItemAdapter.kt	718
Заполнение списка ListAdapter...	719
Обновленный код TasksFragment.kt	720
Что происходит при выполнении кода	721
Представления с переработкой могут использовать связывание данных	725
Включение переменной связывания данных в task_item.xml	726
Макет заполняется в коде держателя представления адаптера	727
Использование класса связывания для заполнения макета	728
Полный код TaskItemAdapter.kt	729
TaskItemAdapter.kt (продолжение)	730
Полный код task_item.xml	731
Что происходит при выполнении кода	733
Ваш инструментарий Android	741

17

Навигация в представлениях с переработкой

Карточные фокусы

Некоторые приложения требуют, чтобы пользователь выбрал элемент из списка. В этой главе вы узнаете, как сделать представления с переработкой центральной частью структуры вашего приложения, для чего следует организовать обработку щелчков на их элементах. Вы увидите, как реализовать навигацию в представлениях с переработкой, чтобы приложение переходило к новому экрану каждый раз, когда пользователь щелкает на записи. Мы покажем, как вывести дополнительную информацию о выбранной записи и обновить ее в базе данных. К концу этой главы у вас будут все средства, необходимые для того, чтобы преобразовать ваши великолепные замыслы в приложение вашей мечты.

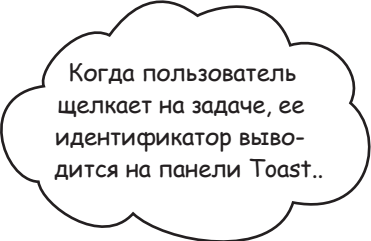


По мне щелкнули!
Надо выполнить это
лямбда-выражение...

{ ... }



CardView



Когда пользователь
щелкает на задаче, ее
идентификатор выво-
дится на панели Toast..

TasksFragment

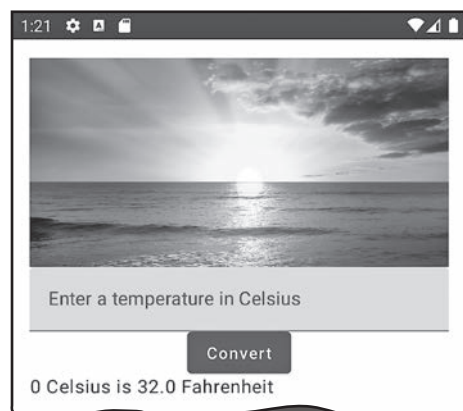
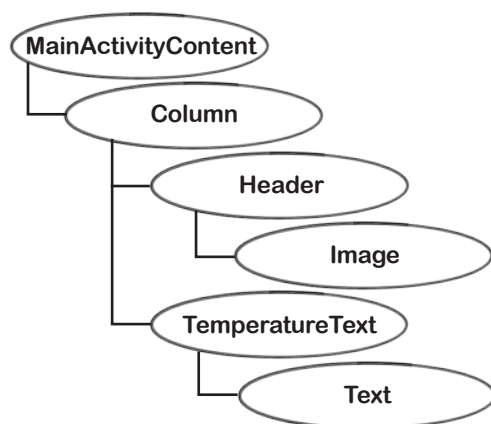
Навигация в представлениях с переработкой	744
Реакция на щелчки	748
Где создать Toast?	749
Передача лямбда-выражения TaskItemAdapter	752
Что происходит при выполнении кода	755
Представление с переработкой должно использоваться для перехода к новому фрагменту	762
Создание EditTaskFragment...	764
Обновление графа навигации	765
Добавление NavHostFragment в макет MainActivity	766
Переход от TasksFragment к EditTaskFragment	767
Добавление нового свойства в TasksViewModel	768
Вывод идентификатора задачи в EditTaskFragment	773
Что происходит при выполнении кода	775
Использование EditTaskFragment для обновления записей	778
Использование TaskDao для взаимодействия с записями базы данных	779
Создание EditTaskViewModel	780
EditTaskViewModel сообщает EditTaskFragment, когда выполнить переход	781
EditTaskViewModel необходима фабрика модели представления	783
В fragment_edit_task.xml должны отображаться данные задачи	784
Что происходит при выполнении кода	788
Ваш инструментарий Android	792

18 Jetpack Compose

В мире Compose

Во всех пользовательских интерфейсах, которые мы строили ранее, использовались представления и файлы макетов. Но благодаря инструментарию **Jetpack Compose** это не единственный вариант. В этой главе перед вами развернется мир **Compose**. Вы научитесь строить пользовательские интерфейсы из компонентов Compose вместо представлений. Вы узнаете, как пользоваться такими компонентами Compose, как **Text**, **Image**, **TextField** и **Button**, размещать их по строкам и столбцам и применять стилевое оформление при помощи **тем**. Вы напишете собственные **компонентные функции** и даже научитесь управлять **состоянием** компонентов Compose с использованием объектов **MutableState**. Переверните страницу.

Compose строит дерево компонентов, которое выглядит так:

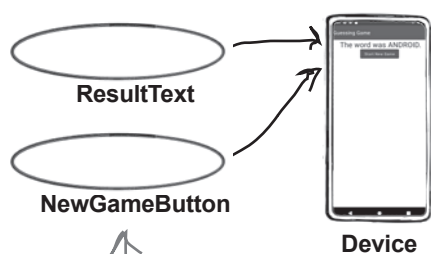


UI-компоненты не обязаны быть представлениями	794
Создание нового проекта Compose	798
У проектов Compose нет файлов макетов	800
Как выглядит код активности Compose	801
Использование компонента Text для вывода текста	802
Использование компонентов Compose в компонентных функциях	803
Предварительный просмотр компонентов в режимах Design и Split	808
Преобразование температур в приложении	811
Добавление компонентной функции MainActivityContent	812
Добавление Image в MainActivity.kt	814
Вывод температуры в виде текста	815
Использование компонента Button для добавления кнопки	816
ConvertButton должно передаваться лямбда-выражение	817
Необходимо изменить значение аргумента TemperatureText	819
Ввод температуры	827
Добавление TextField в компонентную функцию	828
Полный код MainActivity.kt	829
Что происходит при выполнении приложения	831
Настройка внешнего вида приложения	834
Android Studio включает дополнительный код темы	839
Полный код MainActivity.kt	841
Ваш инструментарий Android	847

19 Интеграция с представлениями

Полная гармония

Лучшие результаты достигаются при совместной работе. К настоящему моменту вы научились строить пользовательские интерфейсы из представлений и компонентов Compose. А если вы хотите использовать их **одновременно**? В этой главе вы узнаете, как пользоваться преимуществами обеих технологий, добавляя компоненты Compose в пользовательские интерфейсы на базе представлений. Вы освоите средства, позволяющие компонентам **работать с моделями представлений**. Вы даже узнаете, как заставить их **реагировать на обновления Live Data**. К концу главы у вас появится все необходимое для **использования компонентов Compose с представлениями** и даже для **перехода на пользовательские интерфейсы, построенные исключительно на базе Compose**.



Компоненты ComposeView отображаются на устройстве

Компоненты Compose могут включаться в интерфейсы на базе View	850
Структура приложения Guessing Game	851
Обновление файла build.gradle проекта...	853
Замена представлений в ResultFragment компонентами Compose	854
Добавление компонентов Compose в код Kotlin	856
Добавление компонентной функции для содержимого фрагмента	857
Замена кнопки Start New Game	858
Замена TextView в ResultFragment	859
onCreateView() возвращает корневое представление	864
Полный код ResultFragment.kt	865
Что происходит при выполнении приложения	868
GameFragment тоже переводится на использование компонентов Compose	872
Добавление ComposeView в fragment_game.xml	873
Добавление компонентной функции для содержимого GameFragment	874
Замена кнопки Finish Game	875
Замена EditText на TextField	876
Замена кнопки Guess	877
Вывод ошибочных предположений в компоненте Text	881
Создание компонентной функции IncorrectGuessesText	882
Ваш инструментарий Android	896

Приложение. Остатки

Десять важных тем, которые мы не рассмотрели

Даже после всего сказанного еще кое-что осталось. Нам хотелось бы рассмотреть еще ряд вопросов. Было бы неправильно делать вид, что их не существует; с другой стороны, нам не хотелось, чтобы нашу книгу можно было поднять только после интенсивной тренировки в спортзале. Прежде чем ставить книгу на полку, **ознакомьтесь с приложением.**



1. Совместное использование данных с другими приложениями	900
2. WorkManager	902
3. Диалоговые окна и уведомления	903
4. Автоматизированное тестирование	904
5. Поддержка разных размеров экрана	906
6. Другие возможности Compose	908
7. Retrofit	911
8. Android Game Development Kit	911
9. CameraX	911
10. Публикация приложения	912

Диалоговые окна используются для сообщений, предлагающих пользователю принять некоторое решение.

