## Содержание (сводка)



## Содержание (настоящее)

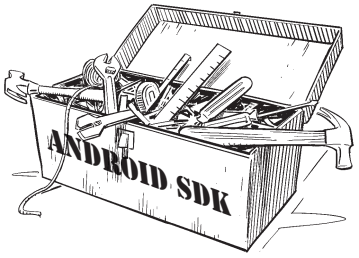
# Введение

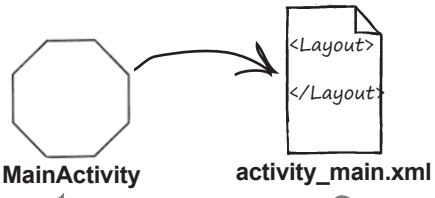
Ваш мозг и Android. Вы пытаетесь изучить что-то новое, а ваш мозг хочет оказать вам услугу и как можно быстрее забыть выученное. Он думает: «Лучше оставить место для чего-то поважнее, например, от каких диких животных стоит держаться подальше или почему на сноуборде не стоит кататься нагишом». Так как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

# 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!



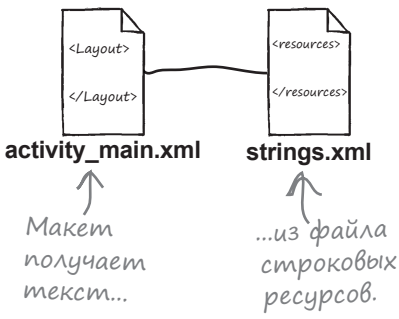

Активность указывает, что делает приложение и как оно должно взаимодействовать с пользователем.

Макет указывает, как выглядит приложение.

# 2 Построение интерактивных приложений

## Приложения, которые что-то делают

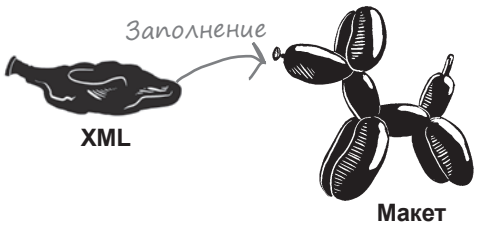
Обычно приложение должно реагировать на действия пользователя. В этой главе вы узнаете, как существенно повысить **интерактивность** ваших приложений. Вы узнаете, как добавить в код активности метод ***OnClickListener***, чтобы приложение могло **прослушивать** действия пользователя и соответствующим 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.

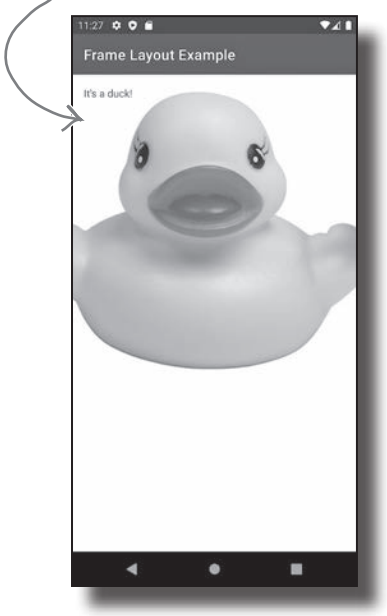
# 3 Макеты

## 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.*



4

Макеты с ограничениями

# Построение эскиза

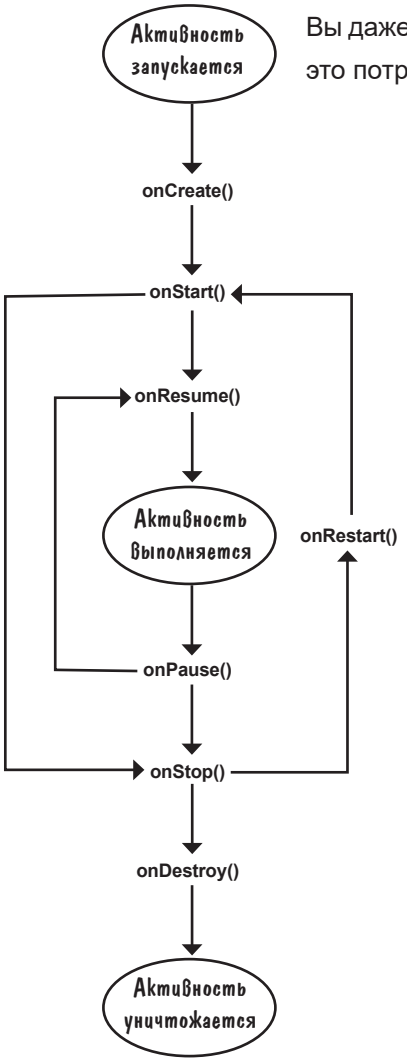
Дом невозможно построить без плана. А некоторые макеты строятся по **эскизам,** чтобы они выглядели **в точности так, как вам нужно**. В этой главе представлены **ограничения в макетах** Android: **гибкий механизм проектирования более сложных графических интерфейсов.** Вы узнаете, как **ограничения** 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.*

# 5 Жизненный цикл активности

## Из жизни активностей

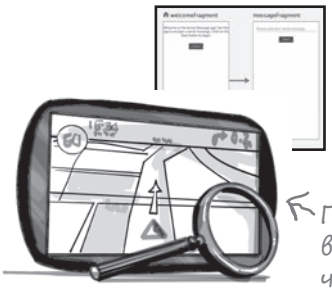
Активности образуют основу любого Android-приложения. Ранее вы видели, как создавать активности и как использовать активности для взаимодействия с пользователем. Но если вы еще не знакомы с **жизненным циклом активностей**, некоторые аспекты их поведения могут вас застать врасплох. В этой главе вы узнаете,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



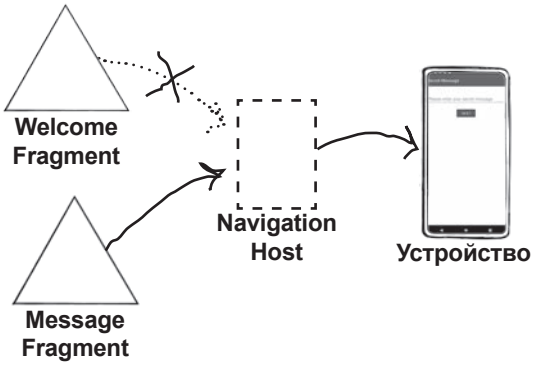
6 Фрагменты и навигация

# 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.




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

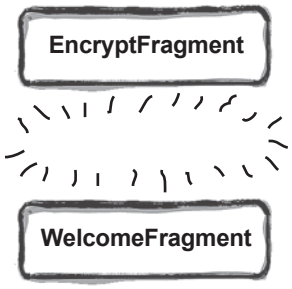


7

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.

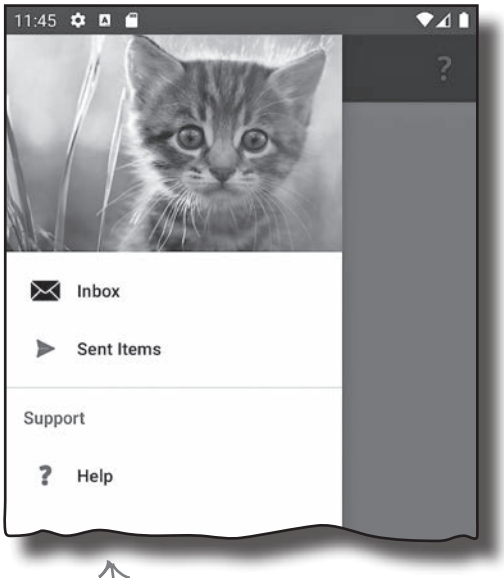
8

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 выдвижную панель, которая появляется на экране сбоку от вашей активности.


**Контроллер навигации**

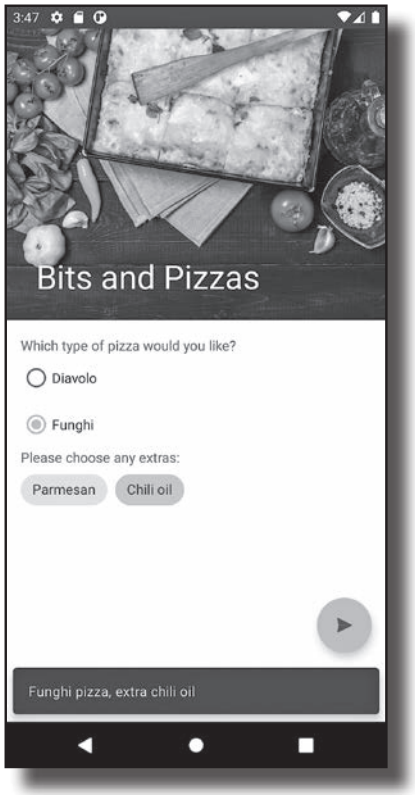



*Если не связать выдвижную панель с контроллером навигации, элементы будут отображаться на выдвижной панели, но по щелчку на них ничего не произойдет.*

# 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.

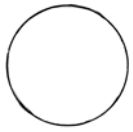


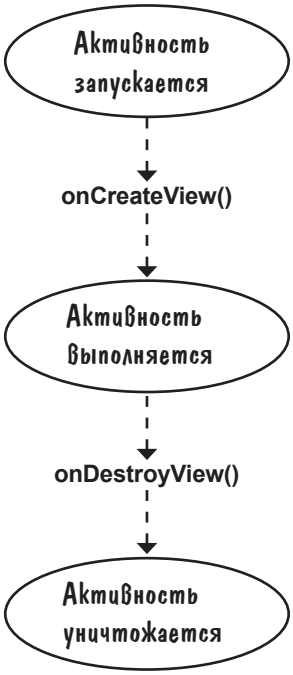
# 10 Связывание представлений

## 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!



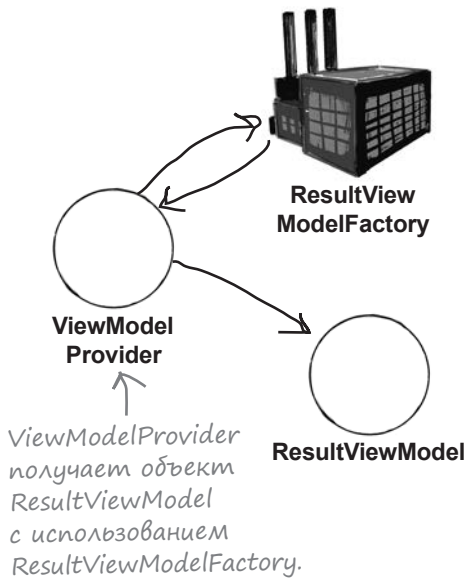



# 11 Модели представлений

## Поведение модели

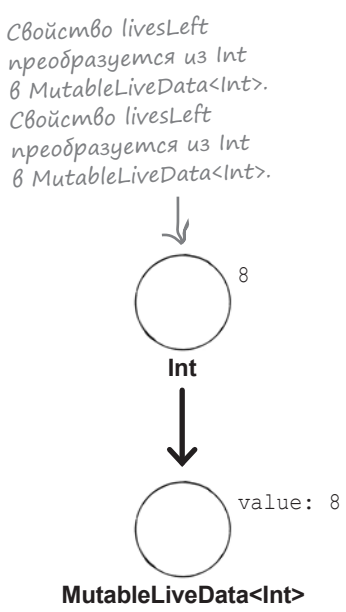
С ростом сложности приложений фрагментам приходится жонглировать все большим количеством объектов. И если не проявить осторожность, это может привести к **разбуханию** кода, который пытается делать все сразу. Бизнес-логика, навигация, управление пользовательским интерфейсом, обработка изменений в конфигурации... 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.



# 12 Живые данные

## В самой гуще событий

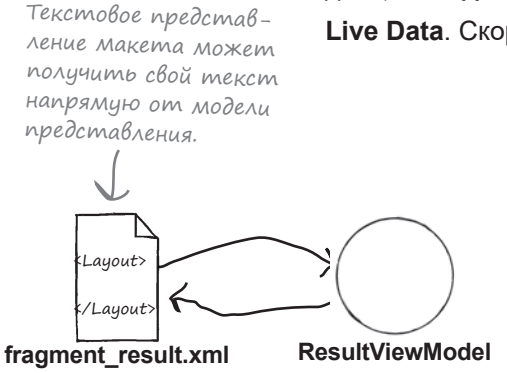
Вашему коду часто приходится реагировать на изменения значений свойств. Например, если свойство модели представления меняет значение, **фрагмент может отреагировать** обновлением своих представлений или переходом. Но **как фрагмент узнает о том, что свойство было обновлено**? В этой главе вы узнаете 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.



# 13 Связывание данных

## Построение умных макетов

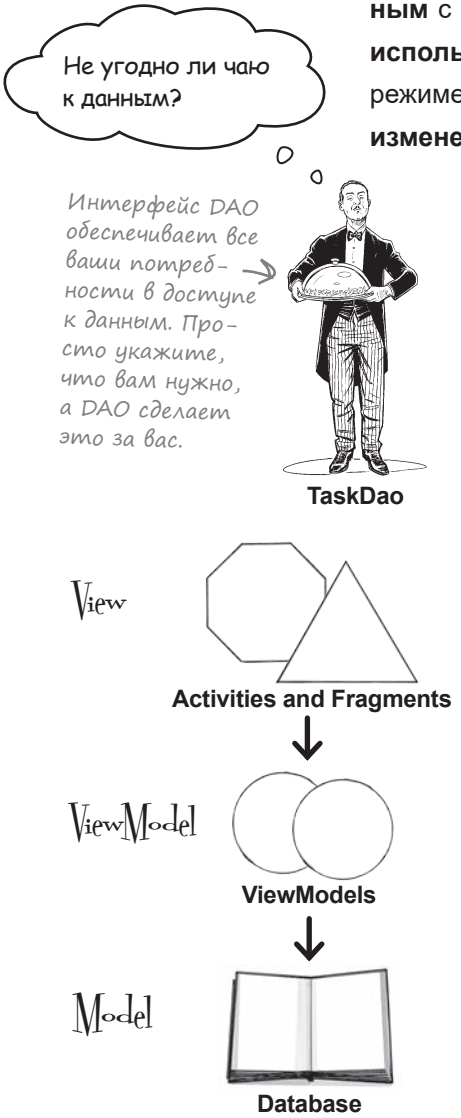
Возможности макетов не ограничиваются управлением внешним видом вашего приложения. У всех макетов, написанных нами ранее, поведение должно было определяться кодом активности или фрагмента. Но только представьте, что макеты могли бы думать самостоятельно и принимать собственные решения. В этой главе представлен 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.




# 14 Базы данных Room

## Номер с видом

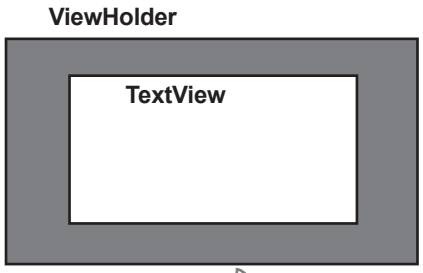
В большинстве приложений используются данные. Но если вы не позаботитесь о том, чтобы эти данные были где-то сохранены, **данные будут навсегда потеряны** при закрытии приложения. В мире приложений Android информация обычно ***хранится в базах данных***, поэтому в этой главе мы представим **библиотеку 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()***.



# 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.*

# 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.**

# 17 Навигация в представлениях с переработкой

## Карточные фокусы

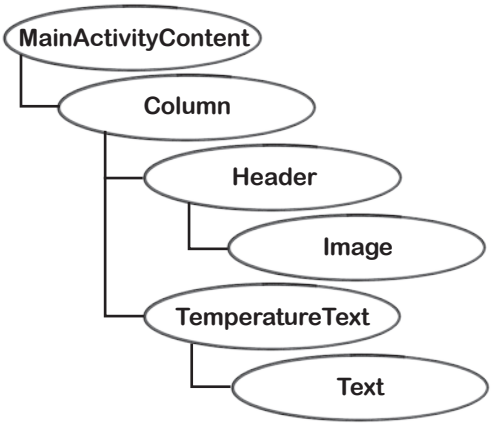
Некоторые приложения требуют, чтобы пользователь выбрал элемент из списка. В этой главе вы узнаете, **как сделать представления с переработкой центральной частью структуры вашего приложения**, для чего следует организовать **обработку щелчков на их элементах**. Вы увидите, как **реализовать навигацию 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**.

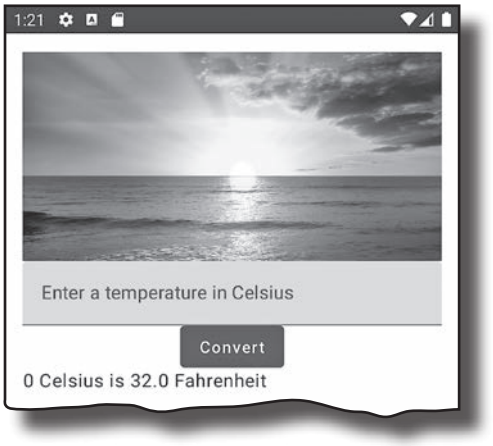
# 18 Jetpack compose

## 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:*

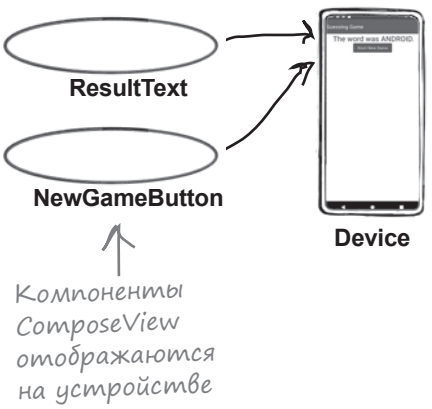


# 19 Интеграция с представлениями

## 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.**



# 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**.