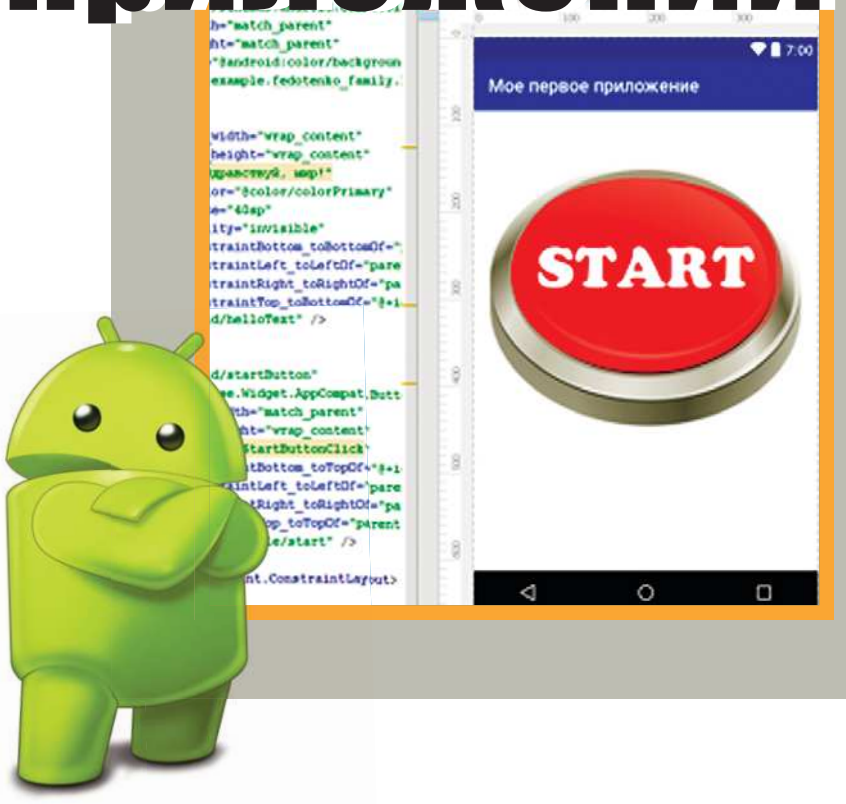
ШКОЛА ЮНОГО ПРОГРАММИСТА

М. А. Федотенко

Разработка мобильных приложений



Первые шаги

Лаборатория ЗНАНИЙ

◀ ШКОЛА ЮНОГО ПРОГРАММИСТА ▶

М. А. Федотенко

Разработка мобильных приложений

Первые шаги

Электронное издание

Под редакцией
В. В. Тарапаты

Москва
Лаборатория знаний
2019

УДК 004.9
ББК 32.97
Ф34

Серия основана в 2018 г.

Федотенко М. А.

Ф34 Разработка мобильных приложений. Первые шаги [Электронный ресурс] / М. А. Федотенко ; под ред. В. В. Тарапаты. — Эл. изд. — Электрон. текстовые дан. (1 файл pdf : 338 с.). — М. : Лаборатория знаний, 2019. — (Школа юного программиста). — Систем. требования: Adobe Reader XI ; экран 10".

ISBN 978-5-00101-640-3

Эта книга — практическое руководство для тех, кто уже делает первые шаги в разработке мобильных приложений под Android или пока только мечтает об этом. С ней вы легко освоите основы разработки, познакомитесь со средой разработки Android Studio, создадите собственные приложения, которыми можно поделиться с друзьями и со всем миром.

Для дополнительного образования в школе и дома. Будет полезна учащимся 8–11 классов школы, учителям информатики, руководителям кружков, студентам и всем, кто хочет войти в мир Android-разработки.

УДК 004.9
ББК 32.97

Деривативное электронное издание на основе печатного аналога: Разработка мобильных приложений. Первые шаги / М. А. Федотенко ; под ред. В. В. Тарапаты. — М. : Лаборатория знаний, 2019. — 335 с. : ил. — (Школа юного программиста). — ISBN 978-5-00101-192-7.

ISBN 978-5-00101-640-3

Оглавление

От автора

Посвящается
Виталию Ивановичу Кулишу

Кажется, что сегодня буквально все можно сделать мобильным приложением. А вы как раз давно хотели стать разработчиком мобильных приложений, но не знали, с чего начать? Смотрите на многочисленные приложения и думаете, что это сложно?

Легко ли создавать приложения для Android™?[1]

Если задать этот вопрос в Интернете — большинство ответов будет «сложно». Но в магазине приложений Google Play миллионы приложений. Если бы это было очень сложно, их было бы тысячи; невероятно сложно — сотни, причем их разработка была бы «по плечу» только крупным компаниям. Но их миллионы! А значит, вы вполне можете стать одним из сотен тысяч разработчиков по всему миру и создать новый продукт, который будут использовать и любить миллионы пользователей.

Не стоит оглядываться на пессимистов!

В чем отличие этой книги от остальных книг по Android-разработке? Она проста и понятна для начинающих, яркая и увлекательная.

Следуя подробным пошаговым инструкциям, вы уже на первом уроке создадите приложение, которое можно запустить и которым можно поделиться с друзьями и не только.

Думаете, нужно заранее досконально изучить несколько языков программирования? Нет, для начала будет достаточно понимания их основ, которое вы получите в процессе чтения данной книги[2]. При этом вы сможете называться настоящими Android-разработчиками, в отличие от тех, кто пользуется различными конструкторами приложений.

Вам не придется на протяжении всей книги идти к одному-единственному результату. Результатом изучения нескольких уроков будет готовый продукт (калькулятор, скетчбук или викторина), который можно совершенствовать и далее[3]. А в конце вы создадите собственный мессенджер и, при желании, сможете развить его до уровня WhatsApp или Viber.

Этому мало где учат. Далеко не во всех, даже ведущих, университетах есть курсы Android-разработки, а значит, с уверенно-

---

[1] Android является товарным знаком Google Inc.

[2] Основы программирования также можно изучить с помощью книг серии «Школа юного программиста» издательства «Лаборатория знаний».

[3] Примеры приложений из этой книги можно найти на ее странице на сайте издательства «Лаборатория знаний».

стью можно сказать, что большинство разработчиков — самоучки. Что же мешает вам?

Задача этой книги — дать вам основные инструменты, с помощью которых вы сами сможете создать собственный ВКонтакте, или... Кто знает?

Успехов вам, будущие Android-разработчики!

Благодарности

Искренняя признательность профессору МПГУ Надежде Николаевне Самылкиной, благодаря которой написание этой книги стало возможным.

Благодарю моего научного руководителя, Марину Леонидовну Соболеву, за ее поддержку и бесценные советы.

Спасибо моим любимым студентам за проявленный интерес и помощь в испытании проектов этой книги на практике.

Отдельная благодарность моим коллегам, Виктору Викторовичу Тарапате и Алене Антоновне Салаховой, а также замечательному коллективу издательства «Лаборатория знаний» — всем, кто работал с рукописью и способствовал ее улучшению.

И самая главная благодарность всем, кто прочитает эту книгу.

Сердечно благодарю мою дорогую семью за всестороннюю помощь и веру в мои силы.

Введение

Эта книга посвящена основам разработки мобильных приложений под Android в интегрированной среде разработки[1] **Android Studio.**

Android Studio — официальная среда разработки мобильных приложений для устройств с операционной системой Android от компании Google. Это доступный и универсальный инструмент. Он обладает широкими возможностями и достаточно прост в освоении.

Работа в Android Studio с помощью этой книги станет намного проще и гораздо интереснее. Например, в разработке мобильных приложений под Android задействованы сразу четыре языка программирования и разметки (Java, XML, SQL и Groovy), однако мы не будем тратить время на изучение основ каждого языка в отдельности — все произойдет в процессе обучения разработке мобильных приложений, а на выходе получится довольно внушительный набор IT-навыков.

Глава 1 этой книги мало похожа на стандартную главу учебника, в ней нет блока вводной теории, вопросов, дополнительных заданий. Она содержит только один урок, скорее напоминающий руководство пользователя IDE Android Studio. Задача главы 1 — провести краткий экскурс в мир разработки мобильных приложений, показать весь процесс от запуска Android Studio до запуска приложения на смартфоне.

Все последующие главы содержат по несколько уроков. В каждом уроке создается мини-приложение с набором основных функций современных мобильных приложений. После прохождения нескольких уроков изученные функции объединяются в одно полноценное приложение. Завершает книгу итоговый проект, в котором мы с помощью всех полученных навыков создадим собственный мессенджер.

Развернутое оглавление книги поможет легко ориентироваться в расположении описания изученных функций. Если в процессе реализации десятого приложения мы забудем, как создавать APK, достаточно заглянуть в оглавление и глоссарий. Его можно найти в Приложениях, так же как Руководство по исправлению основных ошибок и подробное описание структуры проекта Android Studio.

Готовы стать настоящим Android-разработчиком?

Давайте пройдем этот путь вместе!

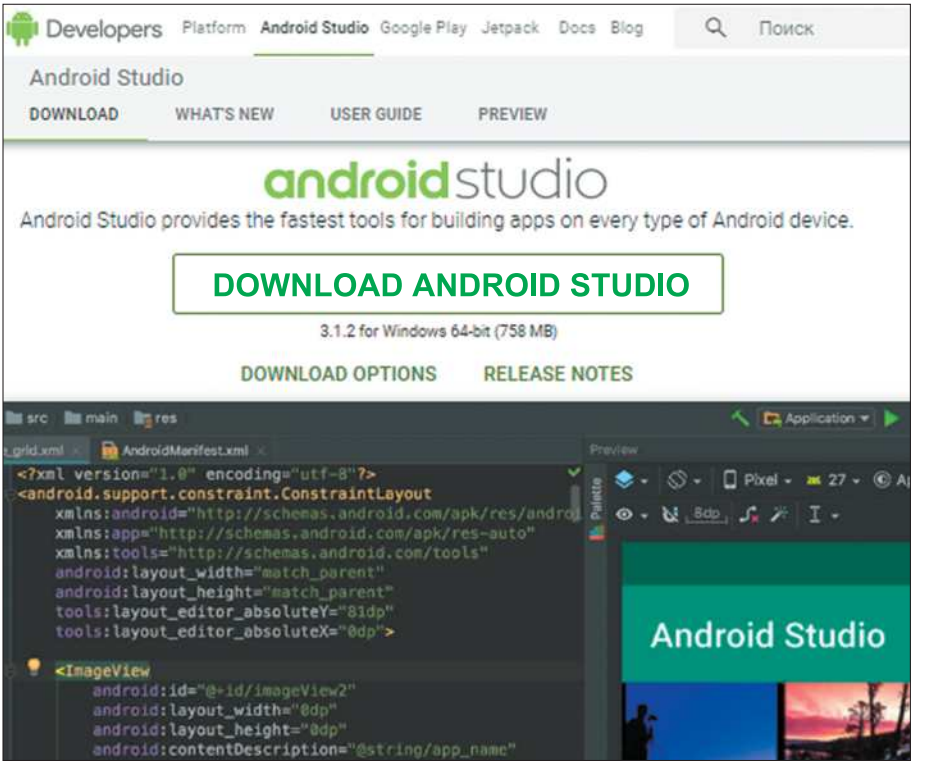
---

[1] Интегрированная среда разработки (Integrated development environment, IDE) — среда разработки, которая уже включает в себя все необходимые для разработки средства, такие как компилятор, текстовый редактор, средство автоматизации сборки и отладчик.

Загрузка и установка Android Studio

Интегрированная среда разработки **Android Studio** является свободно распространяемой. Текущая версия доступна для загрузки на официальном сайте разработчика **https://developer.android.com/studio/index.html**

На открывшейся странице сайта есть кнопка **Download Android Studio** (*скачать Android Studio*). Она запускает скачивание текущей версии для 64-разрядной Windows:



Если на компьютере установлена другая операционная система, нужно пролистать страницу вниз и найти раздел **System Requirements** (*системные требования*). В нем содержатся заявленные разработчиком системные требования:

Для **Windows**:

- Microsoft® Windows® 7/8/10 (32- или 64-разрядная версия);
- минимум 3 Гб ОЗУ (рекомендуется 8 Гб) плюс 1 Гб для эмулятора Android (виртуального устройства для тестирования приложений);
- минимум 2 Гб доступного места на диске (рекомендуется 4 Гб);
- минимальное разрешение экрана 1280×800.

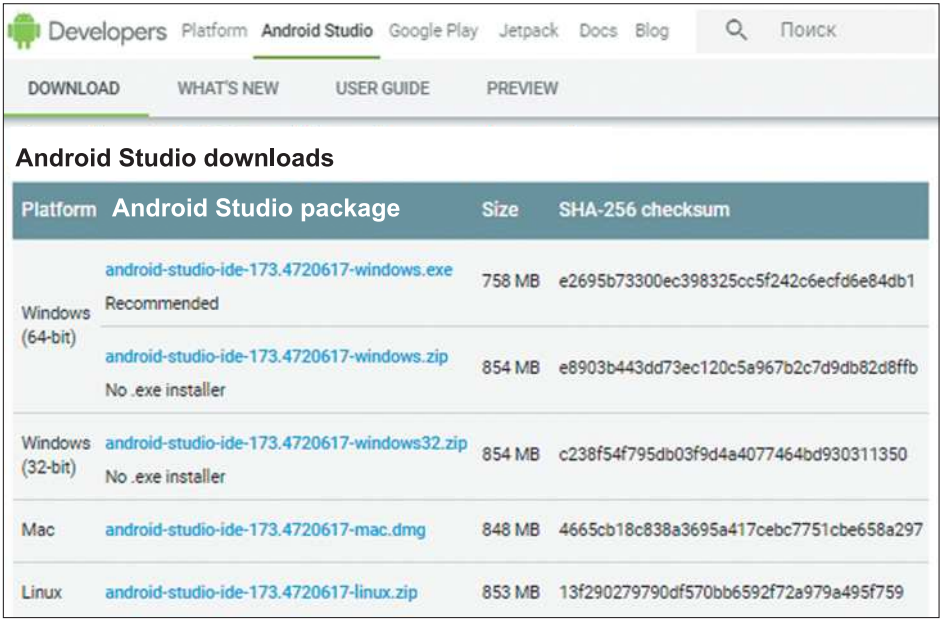
Для **Mac OS**:

- Mac® OS X® 10.10 (Yosemite) или выше;
- минимум 3 Гб ОЗУ (рекомендуется 8 Гб) плюс 1 Гб для эмулятора Android;
- минимум 2 Гб доступного места на диске (рекомендуется 4 Гб);
- минимальное разрешение экрана 1280×800.

Для **Linux**:

- рабочий стол GNOME или KDE;
- библиотека GNU C (glibc) 2.19 или более поздней версии;
- минимум 3 Гб ОЗУ (рекомендуется 8 Гб) плюс 1 Гб для эмулятора Android;
- минимум 2 Гб доступного места на диске (рекомендуется 4 Гб);
- минимальное разрешение экрана 1280×800.

Перед разделом системных требований идет раздел **Android Studio downloads** (*загрузки*), в котором представлены ссылки для скачивания **Android Studio** для разных платформ:

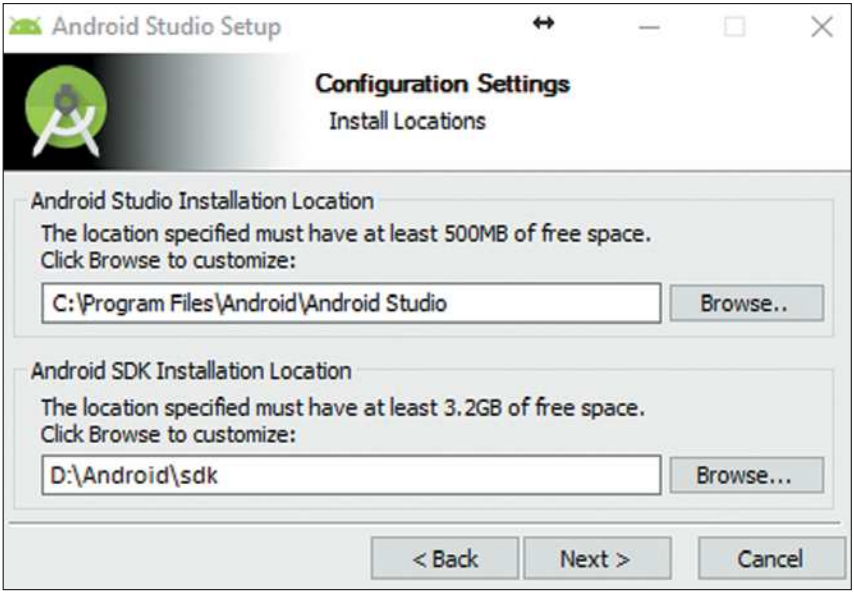


Выбираем свою операционную систему и скачиваем соответствующий установочный пакет из поля **Android Studio package** (*пакет Android Studio*).

В загруженном установочном файле уже имеются все необходимые для разработки инструменты: сама **IDE Android Studio** и **Android SDK**[1]. Таким образом, дополнительно ничего скачивать и устанавливать не требуется.

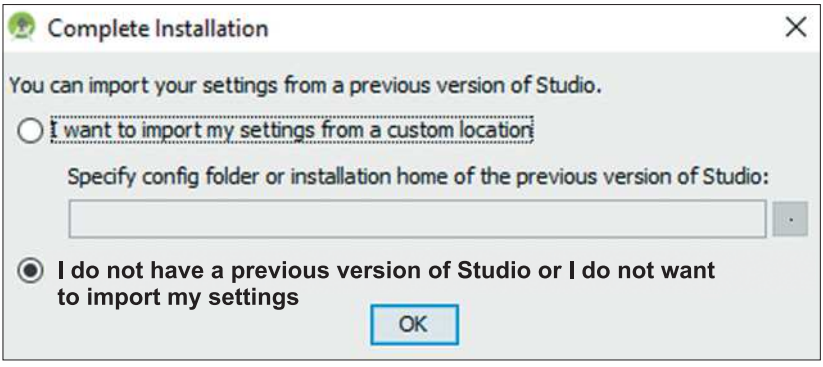
Процесс установки **Android Studio** стандартный, все опции оставляем и принимаем без изменений. Обратить внимание стоит на два момента:

1. При выборе путей установки **Android Studio** и **Android SDK** следует учесть, что:
   - для установки SDK нужно минимум 3.2 Гб свободной памяти на диске;
   - выбранные пути не должны содержать букв кириллицы — это важно.



2. При первом запуске **Android Studio** задаст вопрос об импорте конфигурации. Выбираем второй пункт — **I do not have a previous version of Studio or I do not want to import my settings** (*у меня нет предыдущей версии Android Studio или я не хочу импортировать текущие настройки*):

[1] Android SDK (Software Development Kit) — набор инструментов для разработки программного обеспечения для операционной системы Android. Включает в себя эмуляторы устройств, документацию и все необходимые для разработки пакеты.



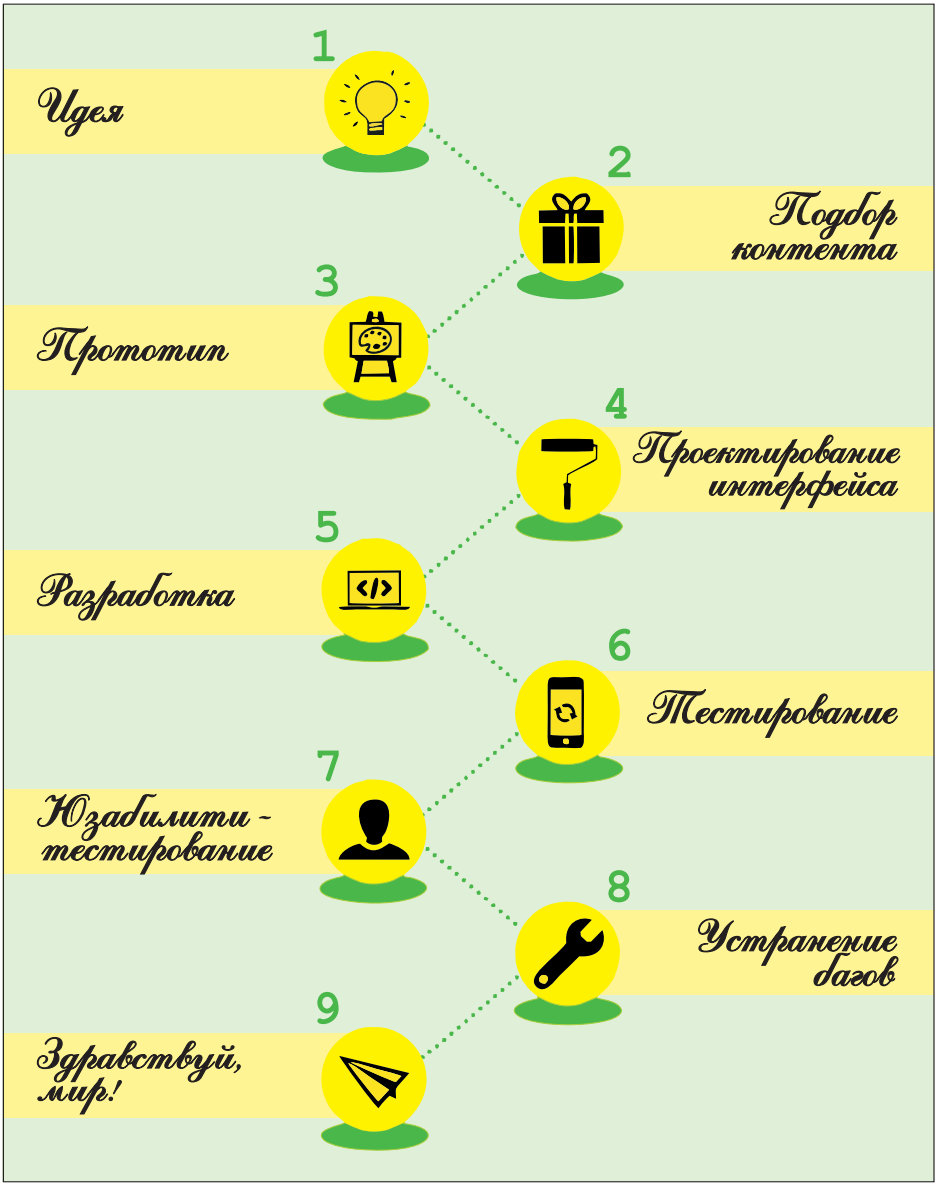
После этого **Android Studio** начнет самостоятельно загружать **Android SDK** из сети Интернет. Если загрузка SDK с первого раза не получится, **Android Studio** предложит повторить попытку — обязательно нажимаем **Retry** (*попробовать снова*).

Когда установка будет полностью завершена, появится приветственное окно Android Studio:

На панели слева будут отображаться уже созданные проекты (при первом запуске она, естественно, будет пуста), в основной рабочей области — возможности для работы с проектами (создание нового проекта, открытие уже существующего, загрузка и импорт проектов), а внизу — **Events** (*события,* например оповещения об обновлениях), **Configure** (переход к настройкам Android Studio) и **Get Help** (меню помощи).

Это означает, что теперь у нас есть все инструменты, необходимые для того, чтобы стать Android-разработчиками. Вперед!

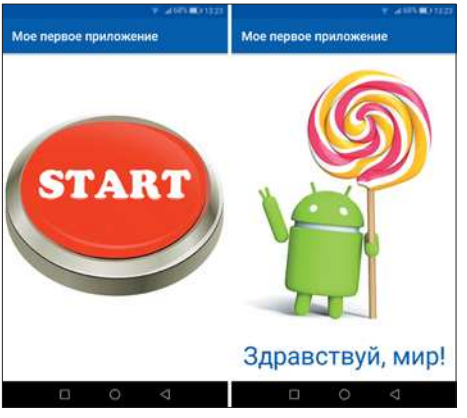
РАЗРАБОТКА ANDROID-ПРИЛОЖЕНИЙ



Глава 1. «Hello, World!», или Первое приложение

Все программисты начинают освоение нового языка программирования с первой программы — «Hello, World!» (в переводе с англ. — «Здравствуй, мир!»). И это не только базовый прием, показывающий программисту особенности нового языка и структуру программы. Это традиция, впервые она была введена в книге «Язык программирования Си» Брайана Кернигана и Денниса Ритчи в 1978 году.

И мы последуем этой традиции при знакомстве с **Android Studio**. Ведь «Hello, World!» — это не просто абстрактное приветствие, на самом деле это оптимистичное и довольно амбициозное заявление: «Здравствуй, мир! Встречай нового Android-разработчика!»[1].



Концепция приложения: при запуске приложения на экране отображается кнопка старта, при нажатии на которую появляется приветственный текст и картинка.

1.1. Запуск Android Studio и создание проекта

Для создания нашего первого (и любого последующего) приложения необходимо создать новый Android Studio проект.

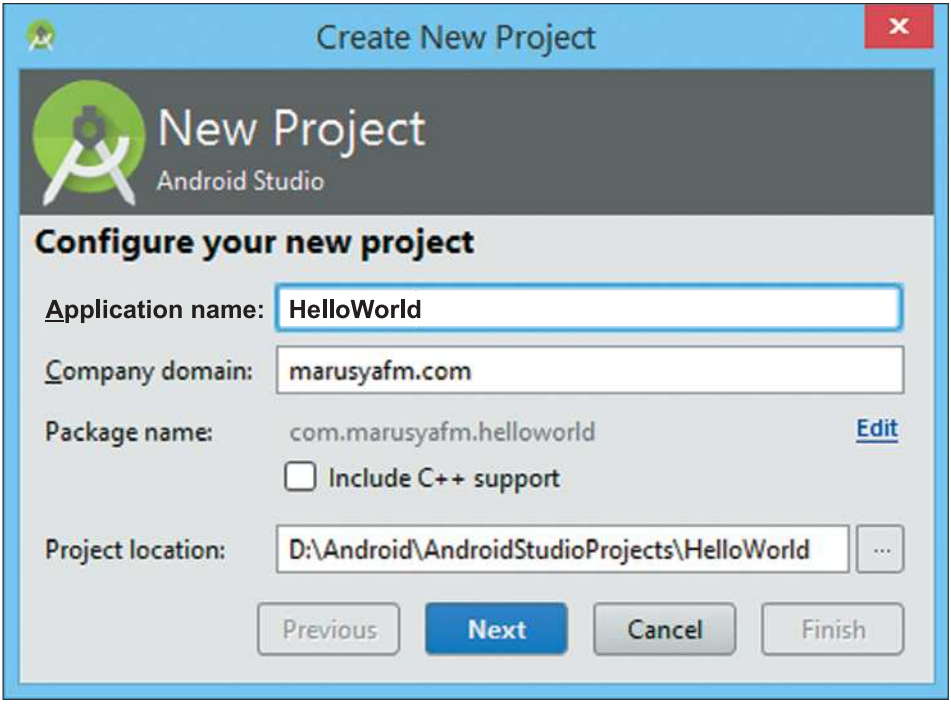
[1] Изображение робота с леденцом — логотип ОС **Android 5.0 Lollipop**.

Чтобы создать проект, в приветственном окне Android Studio выбираем первый пункт — **Start a new Android Studio project** (*начать новый проект Android Studio*).

В открывшемся окне задаем **Application name** (*имя приложения*) — HelloWorld. При первом запуске приложения это имя будет отображаться в «шапке» самого приложения. В дальнейшем его легко изменить, поэтому то, что мы введем сейчас, останется только в качестве названия проекта Android Studio и не будет отображаться ни в магазине приложений Google Play, ни в меню смартфона, ни где-либо еще.

Важно!

- Имя приложения **должно содержать только буквы латиницы, цифры и знаки препинания.**
- Имя приложения **должно иметь смысл и напоминать о содержании приложения.** Для первого приложения это кажется несущественным, но в дальнейшем будет трудно ориентироваться в стандартных MyApplication1, MyApplication123 и MyApplication5678 и так далее.



В **Company domain** (*домен компании*) по умолчанию записывается <имя_компьютера>.com Профессиональные разработчики (особенно крупные компании) указывают в этом поле свой сайт. Если собственного сайта нет — оставляем без изменений.

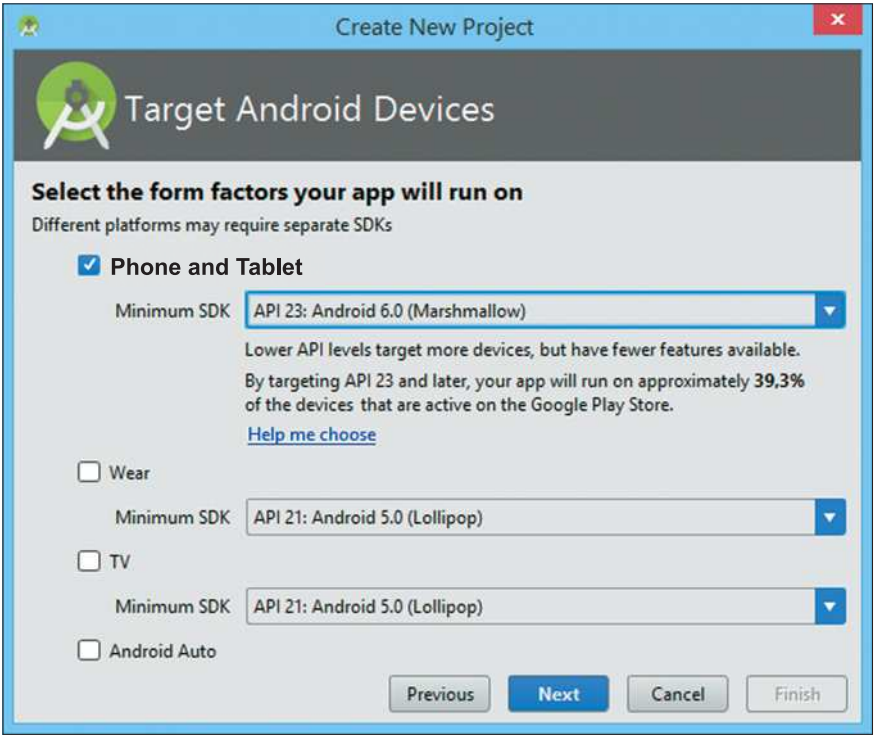
Также при желании можно изменить **Project location** (*расположение проекта — путь к нему*).

Важно!

Путь **не должен содержать букв кириллицы и пробелов.**

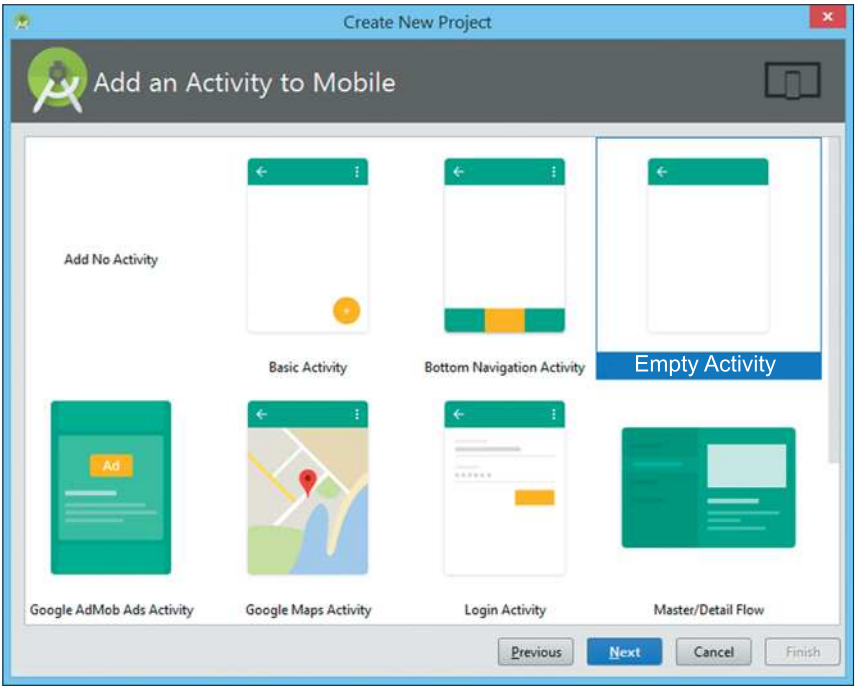
Задаем новый путь (или же оставляем путь, созданный **Android Studio** по умолчанию) и переходим к следующему шагу — нажимаем кнопку **Next.**

В появившемся окне нужно выбрать устройства, для которых будет написано приложение. В среде **Android Studio** можно создавать приложения для самых разных устройств: смартфонов и планшетов, умных часов, телевизоров, автомобилей. Все они выглядят, устроены и функционируют по-разному, а значит, и проекты, и программный код будут иметь свои особенности. На наших уроках речь пойдет о приложениях для смартфонов и планшетов, поэтому выберем **Phone and Tablet** (*смартфоны и планшеты*).

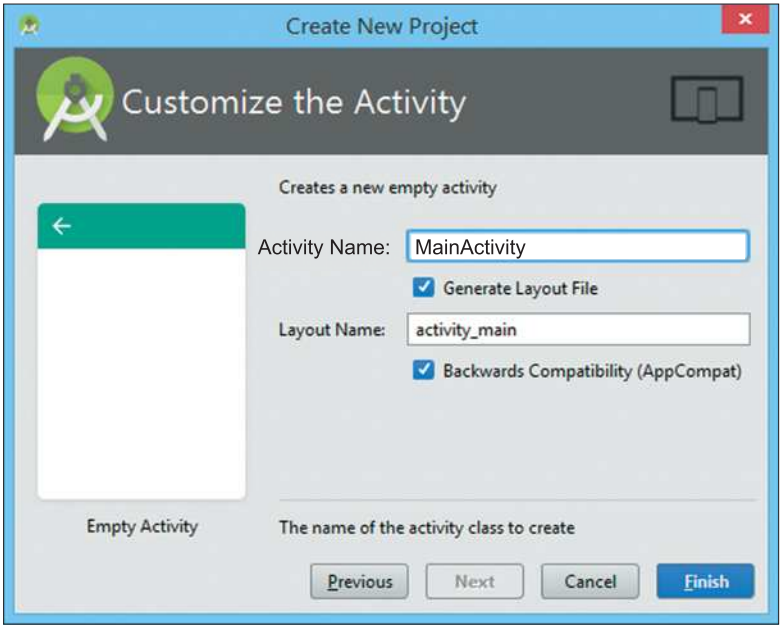


Затем выбираем минимальную версию операционной системы Android. При выборе системы из списка отображается информация о том, что выбор более старой версии операционной системы позволит приложению работать на большей доле смартфонов, чем при выборе недавно анонсированной версии. Однако при этом используется меньше новых функций. Этот параметр впоследствии можно будет изменить в настройках проекта. Выберем, например, **Android 6.0 (Marshmallow)**. **Android Studio** оповещает нас, что приложение, написанное для этой версии, будет работать приблизительно на 39,3% устройств. Эта цифра гораздо ниже реальной, но при разработке приложений для широкой аудитории лучше выбрать версию системы, охватывающую 100% устройств. Переходим к следующему шагу.

Далее нужно выбрать тип Activity. **Activity** («*активность*», *активная область экрана приложения*) — это каждый отдельно взятый экран нашего приложения. Позже мы сможем добавить в приложение сколько угодно Activity, и все они могут быть разного типа. С этой целью **Android Studio** предлагает ряд готовых шаблонов: стандартный экран, пустой экран, широкоформатный экран, экран настроек, экран для работы с картами Google, форма авторизации и другие:



На этапе создания приложения нужно выбрать тип только для одной Activity — экрана, который будет начальным. Выбираем **Empty Activity** (*пустой экран*) и переходим к последнему шагу.



Здесь мы оставляем все параметры без изменений, в том числе **Activity Name** (*имя Activity*) по умолчанию MainActivity (*главная Activity*). Завершаем настройку проекта нажатием кнопки **Finish**.

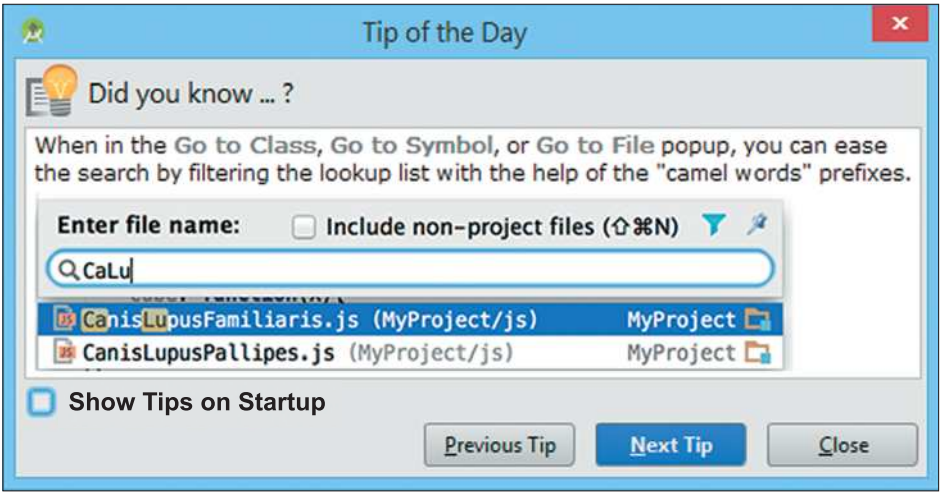
Теперь **Android Studio** понадобится некоторое время на запуск и загрузку всех необходимых для разработки инструментов, и потом еще на создание и сборку проекта. Этот процесс довольно долгий (может занимать до 15 мин) и требует активного подключения к сети Интернет.

Важно!

На этом этапе главное правило — **не начинать работу до того, как Android Studio сообщит о своей готовности**. Когда процесс сборки проекта и загрузки всех модулей завершится, значок **Android Studio** на панели задач будет подсвечен и на нем появится зеленая галочка. Это значит, что мы можем начинать.

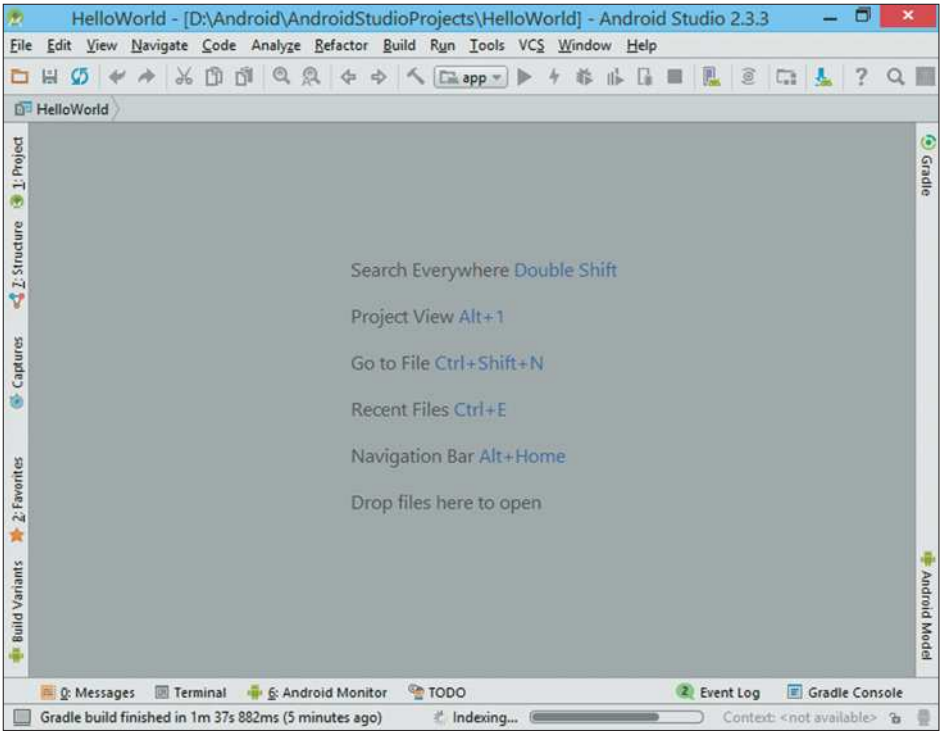
А пока процесс запуска в самом разгаре (он происходит в несколько этапов), поговорим об интерфейсе, возможностях и основных инструментах **Android Studio**.

На первом этапе запуска **Android Studio** загружает непосредственно среду разработки. В начале, как и в любой другой среде разработки, появляется всплывающее окно с подсказками. Снимем галочку (чекбокс) **Show Tips on Startup** (*показывать подсказки при загрузке*), чтобы не видеть это окно при каждом запуске **Android Studio**, и закроем окно.

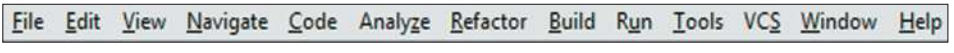


1.2. Знакомство с интерфейсом Android Studio

В заголовке окна мы видим **название нашего проекта** (HelloWorld), его **расположение** (путь к проекту) и **текущую версию Android Studio** (2.3.3).



Ниже располагается **Главное меню** со стандартным для любой среды программирования набором пунктов:



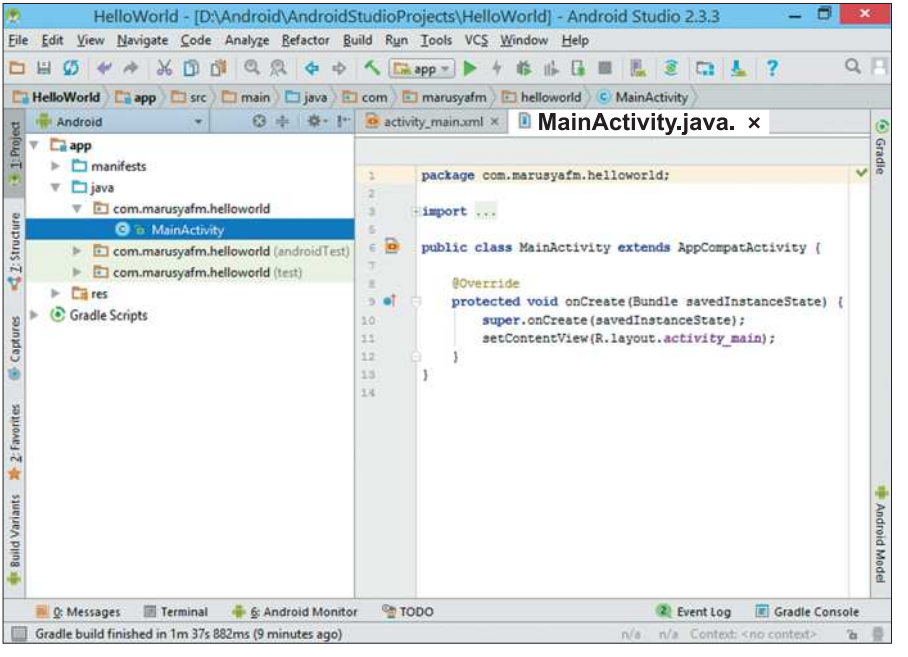
Панель быстрого доступа содержит кнопки для перехода к наиболее востребованным пунктам Главного меню, таким как: создать, сохранить или обновить проект; отменить или повторить действие; вырезать, копировать, вставить; собрать проект, скомпилировать проект, запустить проект (все эти понятия мы рассмотрим в следующих уроках):

Панель навигации отображает путь к текущему открытому файлу. С ее помощью можно легко перемещаться между файлами и папками, образующими довольно сложную структуру проекта **Android Studio:**

В самом низу окна расположена **Панель выполнения и отладки.** С помощью элементов, расположенных на этой панели, можно отслеживать и контролировать все процессы, выполняемые **Android Studio:**

Рабочая область (темно-серая область по центру) перестраивается и изменяет свою структуру в зависимости от типа открытого файла проекта. Появляются дополнительные панели свойств, палитры компонентов, области предварительного просмотра и т. д.

За это время подгрузился и открылся файл **MainActivity.java.**



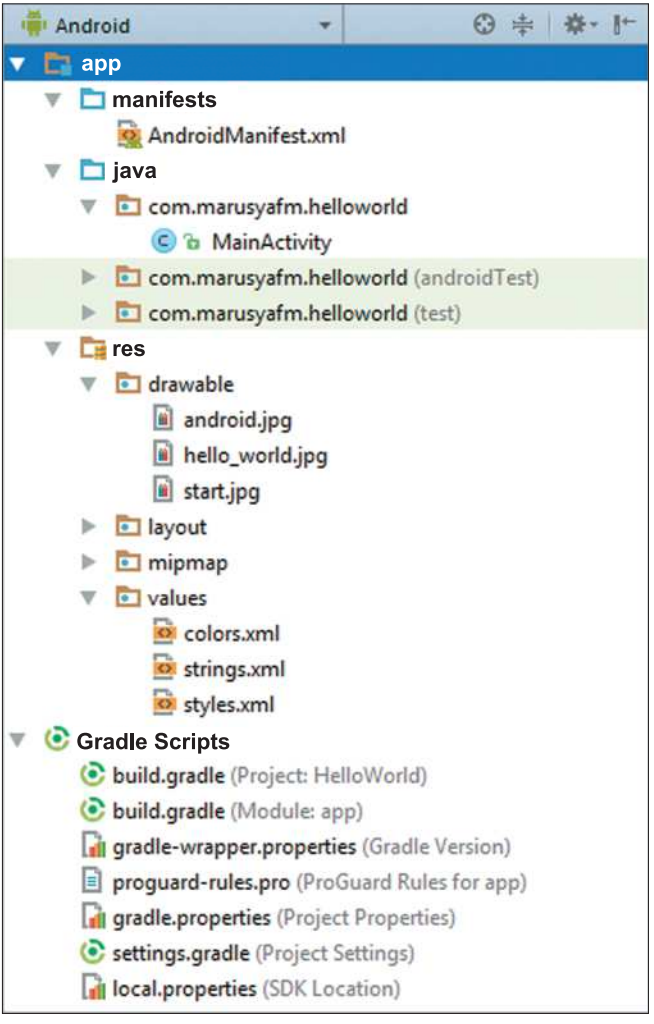
Файлы Java содержат в себе только код на языке программирования Java™ — для них не отображается никаких дополнительных панелей и областей.

Пояснение

Программный код для наших приложений мы будем писать на языке Java. Для того чтобы начать разрабатывать мобильные приложения, совершенно необязательно досконально изучать его заранее. Основ программирования на Java, которые мы получим на этом и последующих уроках, будет вполне достаточно.

Теперь мы можем познакомиться со структурой проекта Android Studio.

1.2.1. Структура проекта Android Studio



Изначально структура проекта Android Studio[1] включает в себя следующие папки:

1. app — корневая папка, в которой хранятся все файлы нашего приложения. Имеет следующую подструктуру:

- **manifests** — в этой папке хранятся **манифесты** — файлы, в которых на языке разметки XML записана основная информация о конфигурации приложения: отображаемое название приложения, структура приложения, иконка и ее миниатюра, полномочия и системные требования. Эта информация нужна операционной системе Android (для установки приложения и его отображения в памяти и на экране смартфона, для проверки версии обновления), а также магазинам приложений, таким как Google Play;
- **java** — место хранения исходного программного кода нашего приложения — все Java-классы, необходимые для работы и тестирования;
- **res** — папка ресурсов. В ней хранятся все картинки (папка **drawable**), файлы xml-разметки (папка **layout**), служебные иконки (папка **mipmap**) и описания всех заданных в проекте значений различных переменных (папка **values**).

2. Gradle Scripts — корневая папка для скриптов системы автоматической сборки **Gradle**. Скрипты написаны на языке Groovy и хранятся в папке **Gradle Scripts**. **Android Studio** производит сборку приложения с помощью этой системы в фоновом режиме, выполняя Gradle-скрипты без необходимости нашего вмешательства как разработчиков.

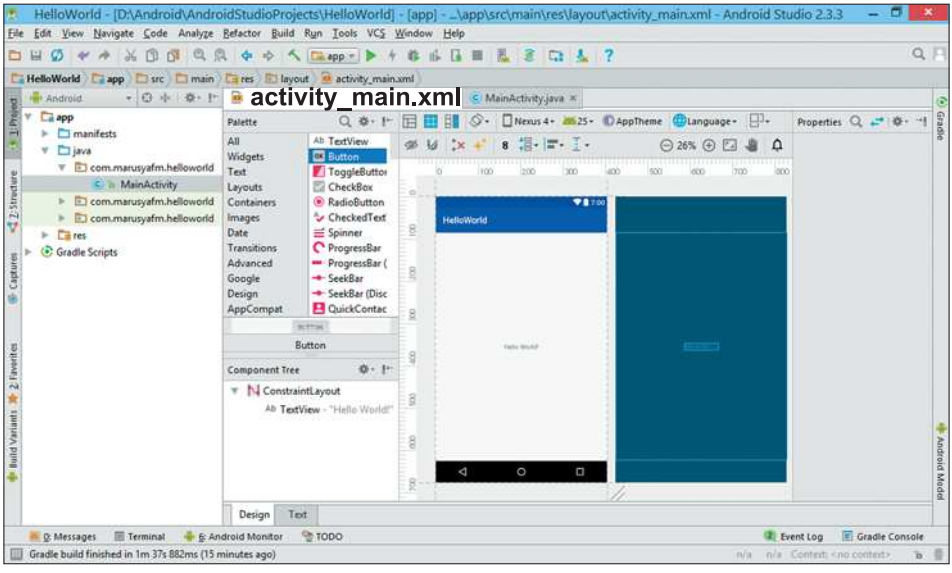
Пояснение

Сборка приложения — процесс получения готового продукта (приложения) из исходного кода, который, как мы видим, написан на разных языках программирования и разметки и «разбросан» по разным файлам, образующим сложную структуру проекта Android Studio.

Процесс сборки включает в себя **компиляцию** (преобразование кода, написанного разработчиком, в машинный код — набор команд, понятных компьютеру) и **компоновку** (преобразование всех файлов проекта в один исполняемый файл, который затем можно будет запустить на смартфоне).

[1] Подробное описание структуры проекта Android Studio см. в Приложении в конце книги.

Тем временем процесс запуска **Android Studio** завершен и наш проект принял следующий вид — открылся файл xml-разметки **activity_main.xml**:



А значит, все готово, и мы можем приступать к разработке.

Файлы xml хранят в себе описания всех элементов приложения (кнопок, надписей, картинок) и их настроек на языке разметки XML.

Пояснение

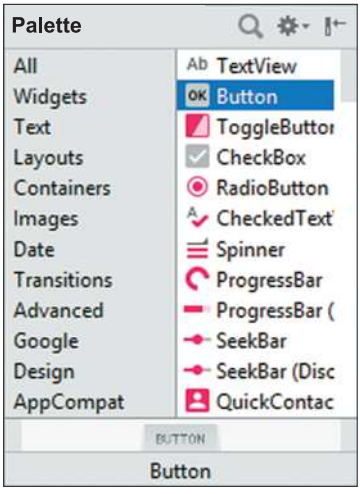
XML (e**X**tensible **M**arkup **L**anguage) — расширяемый язык разметки, созданный для описания данных. В **Android Studio** на языке XML описываются элементы интерфейса (дизайн) приложения, а также некоторые ресурсы (цвета, строки, стили).

Файл XML представляет собой описание элементов интерфейса и их атрибутов (свойств), заключенных в парные теги:

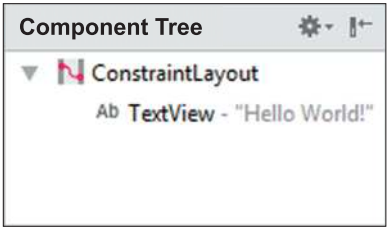
<открывающий тег>…</закрывающий тег>

Рабочая область для xml-файлов в **Android Studio** содержит шесть элементов:

1. Palette (*палитра элементов*). В ее левой части перечислены названия типов элементов (все элементы, виджеты, текстовые элементы, шаблоны и изображения), а в правой — все элементы выбранного типа. Для того чтобы добавить элемент на экран, достаточно перетащить его из палитры элементов:

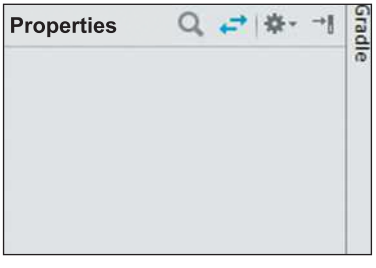


2. Component Tree (*дерево компонентов*). В нем отображаются названия и ID (*идентификаторы*) всех элементов, расположенных на экране.



3. Панель быстрого доступа. На ней собраны кнопки для быстрого перехода к настройкам отображения Activity:

4. Панель **Properties (Attributes,** *панель свойств*). Сейчас она пуста, но при выборе элемента на ней будут отображаться все свойства этого элемента.

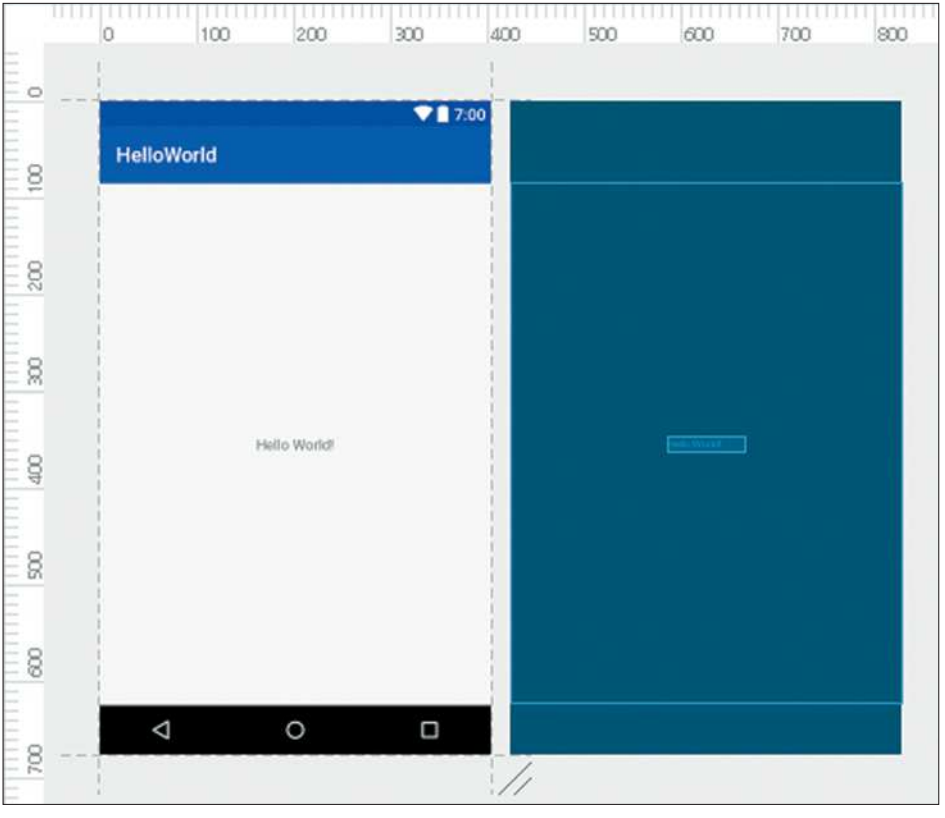


5. Вкладки **Design** и **Text** — для переключения между соответствующими режимами работы (*дизайн* и *текст*, то есть программный код):

6. Отображение выбранного экрана в режиме **предварительного просмотра** (слева) и **макета** (справа).

В режиме **предварительного просмотра** все элементы экрана отображаются так, как их увидит пользователь приложения (с учетом всех заданных настроек — видимости, форматирования, дизайна).

В **режиме макета** элементы экрана отображаются схематично (только границы элементов и первые слова текстового содержимого, если оно есть), для того чтобы на этапе разработки мы видели текущее расположение всех элементов и «не теряли их из вида», если сделаем их невидимыми или расположим один поверх другого.



И здесь нас ждет сюрприз — на экране мы видим текст «Hello World»! Да, действительно, при создании нового проекта среда разработки **Android Studio** по умолчанию придает ему вид Hello-World-приложения. По сути, его уже можно «собрать», запустить и полюбоваться приветственной надписью на экране смартфона. Но мы ведь еще ничего не сделали! Мы даже не знаем, как это приложение работает и как его можно изменить.

Поприветствуем мир по-своему:

1) оставим текст, но изменим его содержание, стиль и свойства;
2) запрограммируем текст так, чтобы он не просто «висел» на экране, а появлялся по нажатию кнопки (которую мы уже самостоятельно добавим и настроим).

Казалось бы, все то же простое приветственное приложение, а уже целый список задач. Приступим!

Итак, по умолчанию открылись два файла проекта: **activity_main.xml** и **MainActivity.java**. Файлы xml в **Android Studio** отвечают за разметку — в них описаны все элементы, которые мы видим на экране (кнопки, тексты, картинки), и их свойства. А в файлах Java описываются возможные для этих элементов действия: что произойдет при нажатии на кнопку, в какой момент появится и пропадет уведомление, как переключиться с одного экрана на другой.

Начнем с xml-разметки — останемся в файле **activity_main.xml** и придадим нашему приложению должный внешний вид. В этом нет ничего сложного, по сути, мы будем «рисовать» приложение.

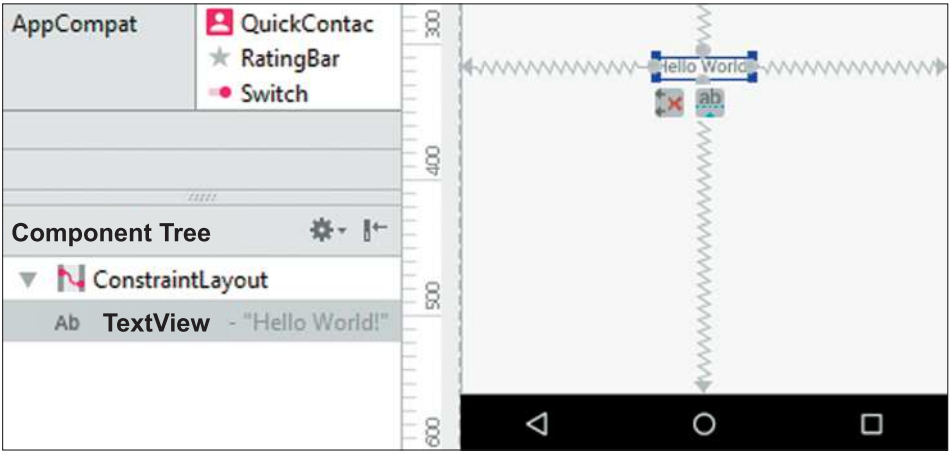
Как мы уже знаем, с файлами xml-разметки в **Android Studio** можно работать в двух режимах: **Design** (*режим дизайна*) и **Text** (*режим кода*). Начнем с дизайна.

1.3. Работа в режиме дизайна

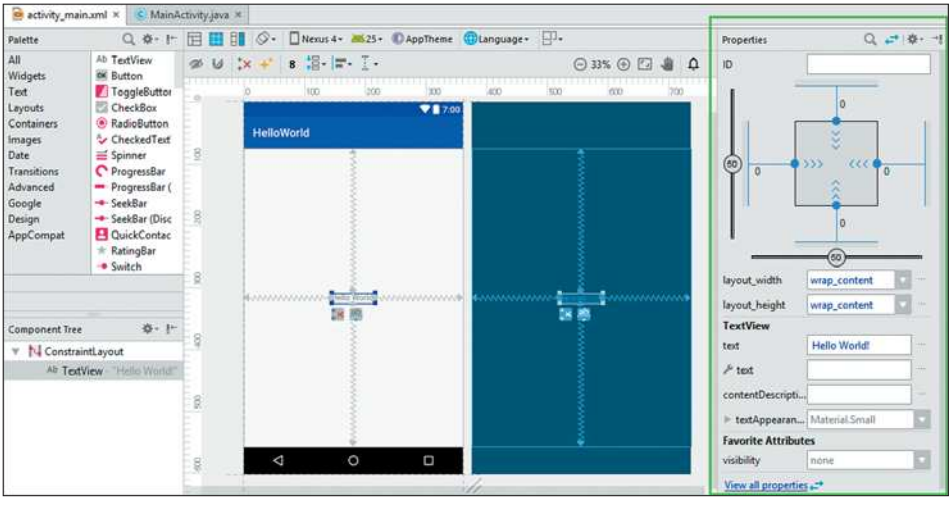
1.3.1. TextView — текстовые элементы

У нас на экране уже есть текст «Hello World» — мы договорились его оставить. Но сейчас размер шрифта слишком мал, не настроен цвет, и текст расположен четко по центру. Изменим это.

Как мы можем увидеть в **Component Tree** (дереве компонентов), этот текст — элемент типа **TextView**:

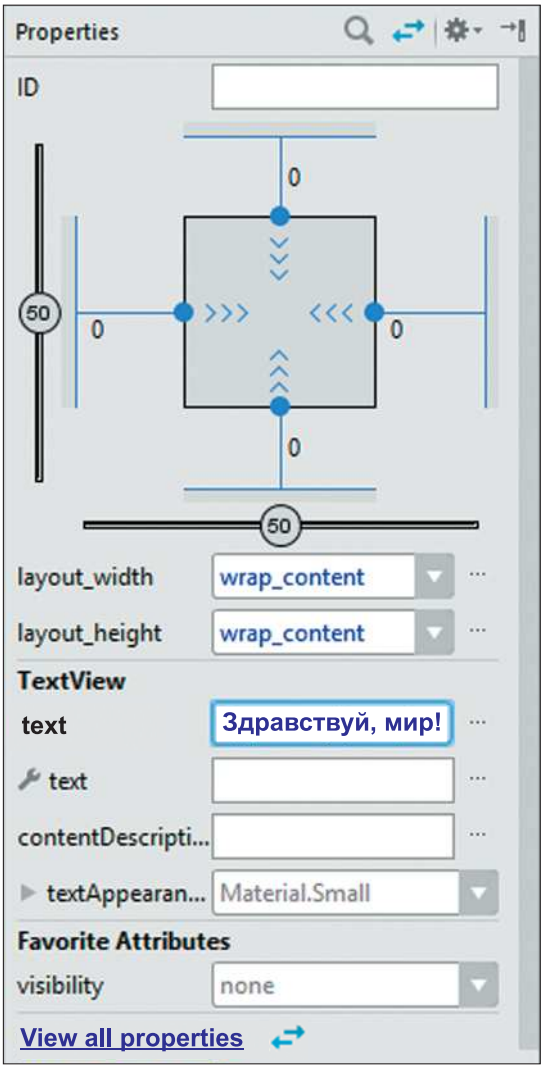


Нажмем на **TextView** на макете или в дереве компонентов, и в правой части рабочей области откроется панель **Properties (Attributes,** *панель свойств*):



Для начала изменим текст. Находим в свойствах атрибут **text** и меняем его значение на Здравствуй, мир!

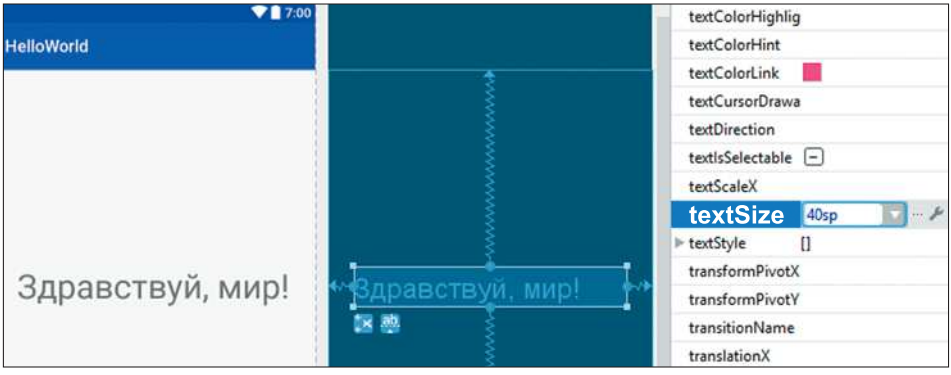
Чтобы изменения свойств вступали в силу, нужно всегда нажимать клавишу **Enter** или кликать по соседним свойствам.



Текст изменился, но он по-прежнему маленький и серый. Для того чтобы изменить эти свойства, нажимаем ссылку **View all properties** (*смотреть все свойства*) внизу панели свойств.

В появившемся списке находим свойство **textSize** (*размер текста*) и вручную задаем ему значение 40sp[1]. Наш текст тут же изменился:

[1] **sp (scale-independent pixels)** — пиксели, независимые от масштабирования. Единицы измерения, предназначенные для работы с текстом, для наиболее корректного отображения шрифтов. Значение задается пользователем вручную и остается неизменным для любого разрешения экрана, делая интерфейс **адаптивным**.



И раз уж мы работаем в режиме «дизайна», то не можем позволить тексту остаться серым. Находим свойство **textColor** (*цвет текста*) и нажимаем на **многоточие** справа от него. Откроется окно **Resources** (*окно ресурсов*).

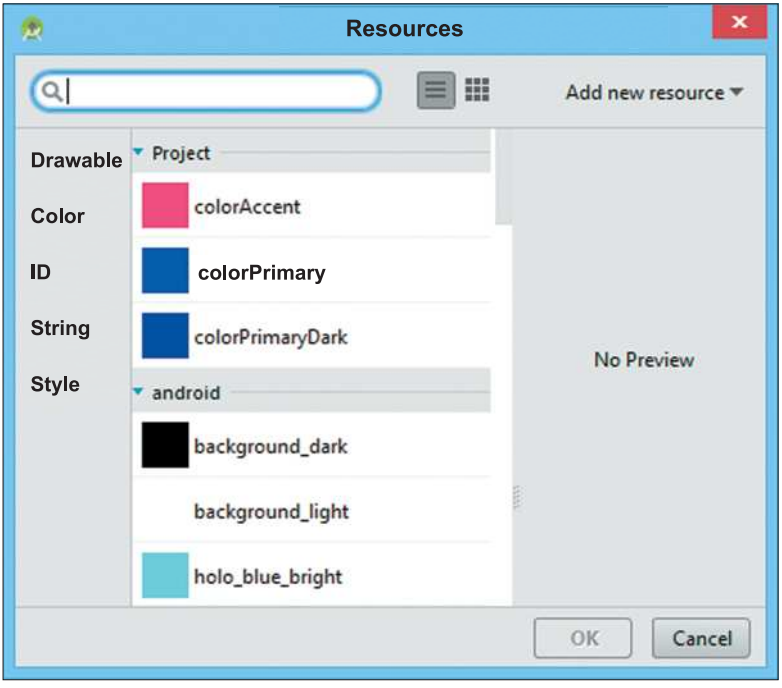
1.3.2. Resources — библиотеки ресурсов

Практически все свойства элементов **Android Studio** можно задавать вручную, а можно для этого использовать ресурсы из библиотек **Resources** (*библиотек ресурсов*). С точки зрения программирования это более грамотно. Ведь если определить и прописать изображение или какой-нибудь текст как ресурс, то к нему можно будет неоднократно обращаться в дальнейшем, указав его уникальный идентификатор **id**, а не переписывая текст каждый раз заново.

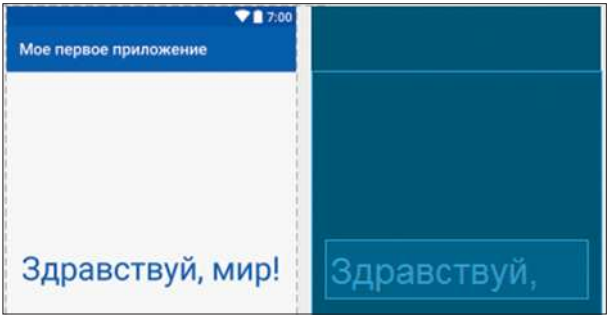
Ресурсами могут быть:

- изображения — вкладка **Drawable**;
- цвета — **Color**;
- сами элементы (определение по id) — **ID**;
- строки (текст) — **String**;
- темы и стили — **Style**.

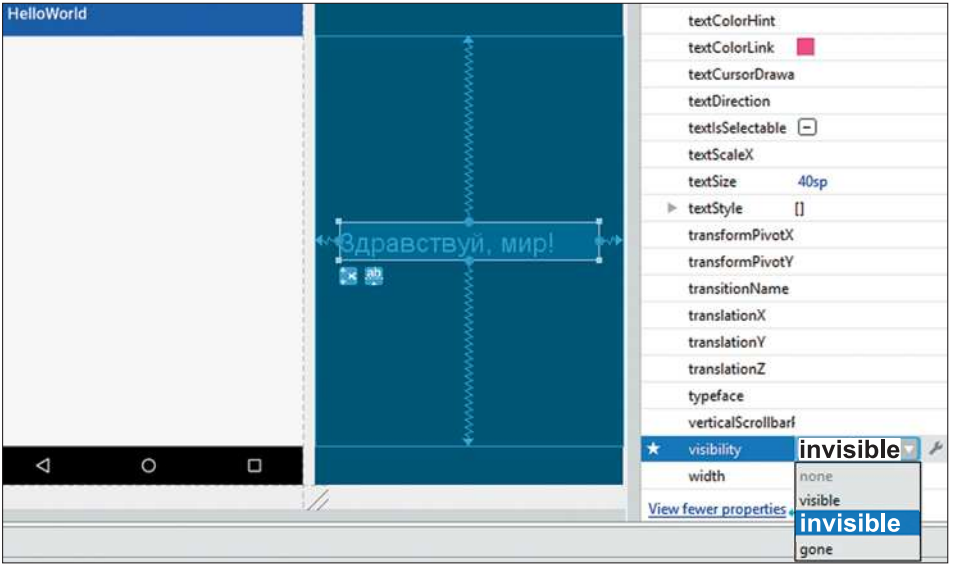
Как уже было сказано, окно ресурсов для любого свойства вызывается нажатием на многоточие справа от названия свойства элемента.



Вернемся к нашему элементу **TextView** — ему нужно задать цвет. Переходим на вкладку **Color** (*цвет*) и выбираем **colorPrimary** — это главный цвет приложения, определенный *по умолчанию*. Его можно изменить, можно добавить новые цвета, но этим мы займемся в следующих главах. А пока нам достаточно, чтобы цвет текста совпадал с цветом заголовка приложения. Нажимаем **OK**. Текст стал синим:



Изменим еще одно свойство — **visibility** (*видимость*) текста. Пусть наш текст не просто все время «висит» на экране, а появляется при нажатии на кнопку. Из выпадающего списка выбираем **invisible** (*невидимый*). Наблюдаем изменения на экране: в режиме предварительного просмотра наш текст исчез, но в режиме макета он остался, чтобы мы знали, что он существует и в каком месте экрана располагается. По такому принципу всегда работают с элементами и областями экрана, которые должны появляться только в определенный момент (например, «спрятанные» вещи или бонусы в играх), а до тех пор оставаться невидимыми.



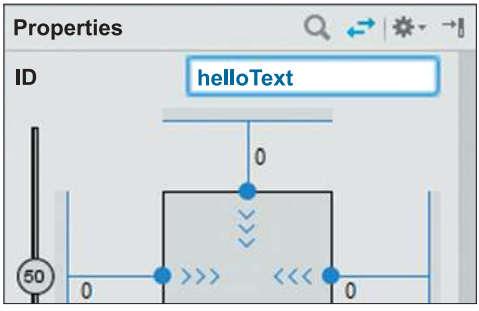
Вернемся в самое начало списка свойств и изменим **ID** (*уникальный идентификатор*) элемента. Внизу списка свойств нажмем ссылку **View fewer properties** (*смотреть меньше свойств*). Первым в коротком списке будет **ID** элемента.

1.3.3. ID — уникальный идентификатор

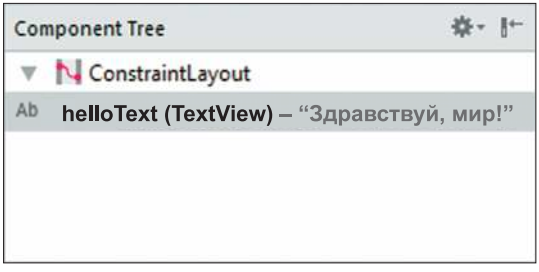
Каждому элементу в **Android Studio** должен быть присвоен **уникальный идентификатор**. По умолчанию присваивается название типа элемента и порядковый номер (например, для кнопок: Button1, Button2, Button3), но по таким ID нам будет сложно идентифицировать элементы, нужно будет все время вспоминать или смотреть, какая кнопка была первая, а какая — десятая. Поэтому грамотные программисты присваивают элементам ID, напоминающие об их содержании и типе элемента. Это упрощает

работу в команде, а также чтение собственного кода, написанного некоторое время назад.

Наш элемент **TextView** содержит приветственный текст, поэтому в качестве **ID** напишем helloText.



Изменения сразу отразятся в дереве компонентов. Теперь в нем отображается **ID** элемента, затем в скобках тип элемента, а за ним содержание элемента.



Отлично! С текстом закончили, займемся кнопкой.

Для этого нам понадобятся две картинки — кнопка и приветствующий нас персонаж. Для кнопки подойдет любое ее изображение с надписью «Start». А приветствовать мир от нашего имени пусть будет робот-андроид, например с логотипа **Android 5.0 Lollipop**:

Кстати, создатели ОС Android очень любят сладкое: каждая очередная версия называется в честь какой-нибудь сладости, причем по алфавиту.


Android 1.0
Apple Pie


Android 1.1
Banana Bread


Android 1.5
Cupcake


Android 1.6
Donut


Android 2.0
Eclair


Android 2.2
Froyo


Android 2.3
Gingerbread


Android 3.0
Honeycomb


Android 4.0
Ice Cream Sandwich


Android 4.1
Jelly Bean


Android 4.4
KitKat


Android 5.0
Lollipop


Android 6.0
Marshmallow

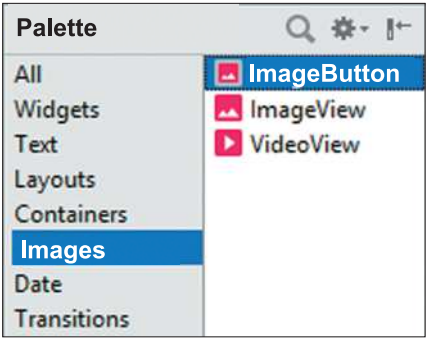

Android 7.0
Nougat


Android 8.0
Oreo

Но вернемся к нашим картинкам.

1.3.4. ImageButton — изображение-кнопка

В **Palette** (*палитре элементов*) выбираем категорию **Images** (*изображения*), а в ней элемент **ImageButton** (*изображение-кнопка*).



Перетаскиваем **ImageButton** на экран. Так как мы работаем с изображением, по умолчанию открывается окно ресурсов на вкладке **Drawable** (*графические объекты*).

Можно выбрать картинку для нашей кнопки из коллекции **Android Studio**, а можно загрузить свою картинку.

Важно!

Разработанные на наших уроках приложения мы можем впоследствии дорабатывать и делиться ими и даже выкладывать в магазины приложений. А значит, при создании приложения лучше сразу соблюсти все правила, например подбирать «лицензионные» картинки. Мы же хотим соблюдать авторские права. Было бы очень неэтично и незаконно присваивать себе чужой графический труд.

Лучше всего использовать в приложениях картинки, распространяемые по лицензии **CC0 (CC0 1.0 Универсальная** — передача в общественное достояние), то есть те, которые авторы сами дарят миру. Это самая подходящая для нас лицензия, потому что она позволяет свободно использовать картинки в любых целях, изменять их по своему усмотрению и при этом не указывать автора.

Для поиска лицензионных картинок существует большое количество сайтов со «стоковыми» изображениями, например **https://pixabay.com/** или **https://www.pexels.com/**

Для того чтобы библиотека ресурсов «увидела» наши картинки и предложила их к использованию в приложении, нужно выполнить два действия:

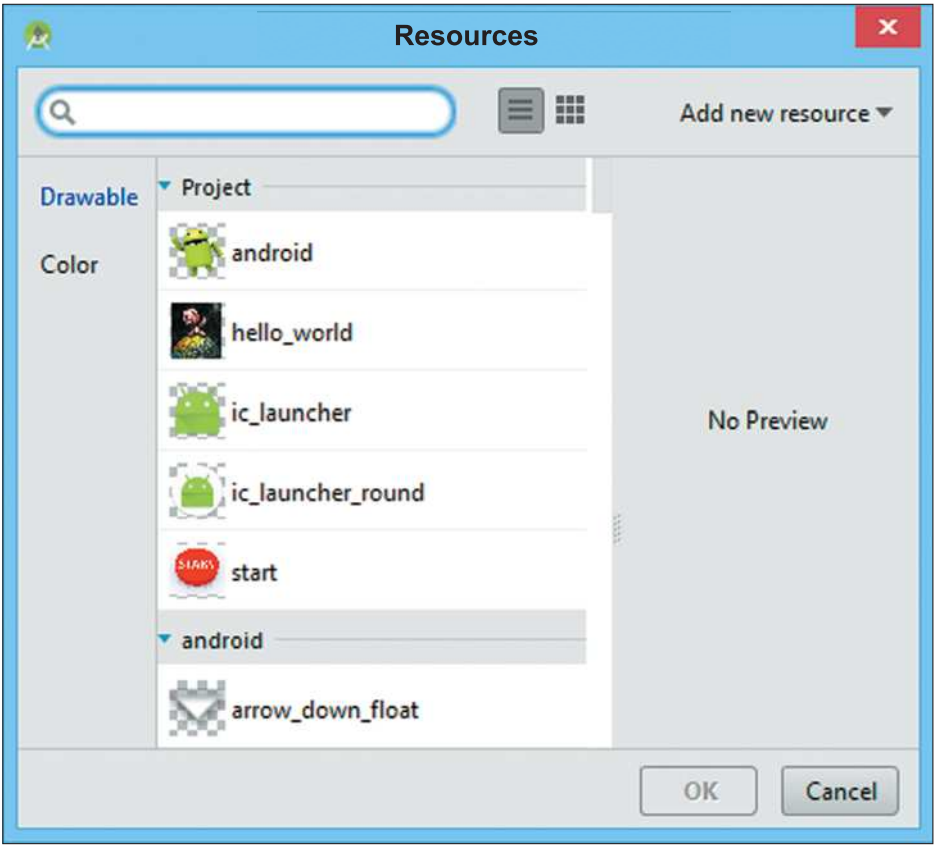
1. Скопировать нужные картинки в папку проекта: **...\HelloWorld\app\src\main\res\drawable**, нажать на папку drawable и выполнить вставку сочетанием клавиш **Ctrl+V**. Это можно сделать как через проводник, так и в самой **Android Studio**.
2. Переименовать картинки и сохранить их в требуемом формате.

Важно!

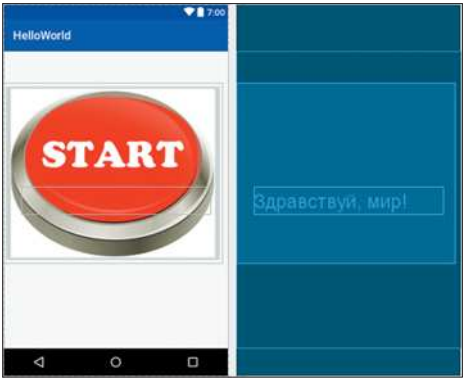
- Названия картинок **не должны содержать** заглавных букв, букв кириллицы и пробелов.
- Названия картинок должны начинаться с буквы.
- Предпочтительный формат — **.png**. Картинки, сохраненные в формате **.jpg**, **IDE Android Studio** просто не распознает.
- **Изменить имя и формат** картинки можно прямо в **Android Studio**. Для этого нужно нажать *правой* кнопкой мыши на картинке в проекте Android Studio, затем в появившемся контекстном меню выбрать пункт **Refactor → Rename**.

Например, стартовую кнопку можно назвать start, а робота — android.

Теперь наши картинки появились в **ресурсах** проекта.

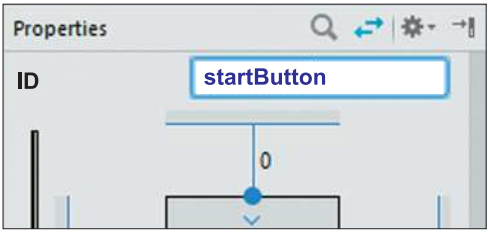


Выбираем изображение кнопки **start**, нажимаем **ОК** — кнопка появилась на экране:

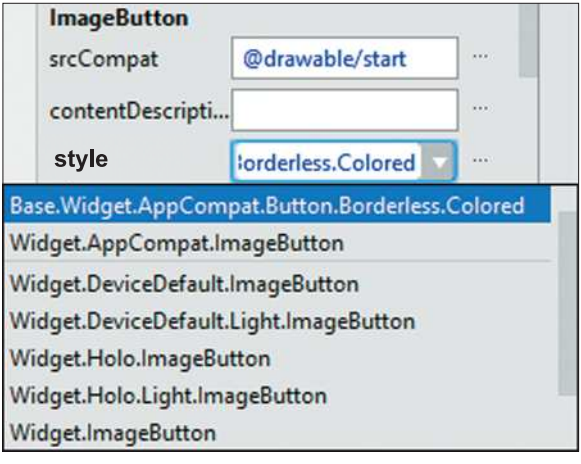


Перетащим наш элемент **TextView** так, чтобы он находился ниже кнопки, а не за ней, как сейчас. Займемся свойствами **ImageButton.**

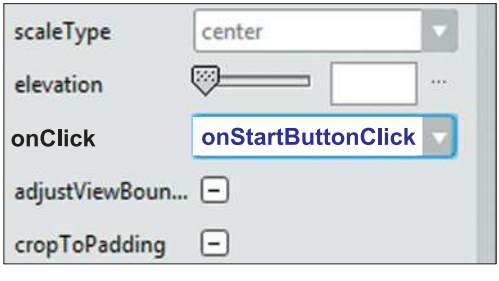
Изменим **ID** на startButton:



В свойстве **style** (*стиль*) пропишем (или найдем в выпадающем списке) @style/Base.Widget.AppCompat.Button.Borderless.Colored — этот стиль уберет тень и границы нашей кнопки:



За действие, происходящее при нажатии на элемент, отвечает свойство **onClick.** Обычно в нем пишут название обработчика событий — **метода Java.** Но описывать этот метод и добавлять в него все необходимые инструкции мы будем чуть позже, а пока в свойстве **onClick** напишем onStartButtonClick (*при нажатии на кнопку Start*).

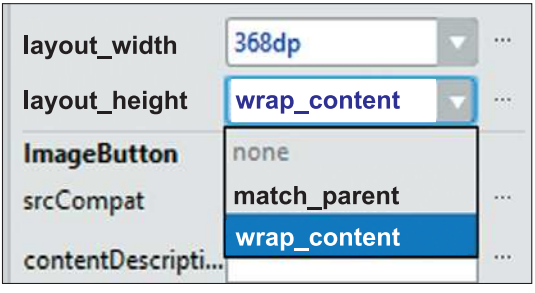


Важно!

Названия методов, так же как и идентификаторы элементов, **должны напоминать об их содержании**. Например, для методов нажатия на кнопки обычно выбирают название, начинающееся на on и заканчивающееся на Click, чтобы в целом получилось «при нажатии на кнопку...».
Названия методов **должны начинаться со строчной** (маленькой) **буквы.**

Изменим размер кнопки. За размер элемента отвечают два свойства: **layout_width** (*ширина*) и **layout_height** (*высота*). Высоту и ширину можно задать конкретным числом пикселей (например, 368dp[1]), а можно выбрать один из вариантов, при которых размер элемента не будет зависеть от конкретного числа пикселей:

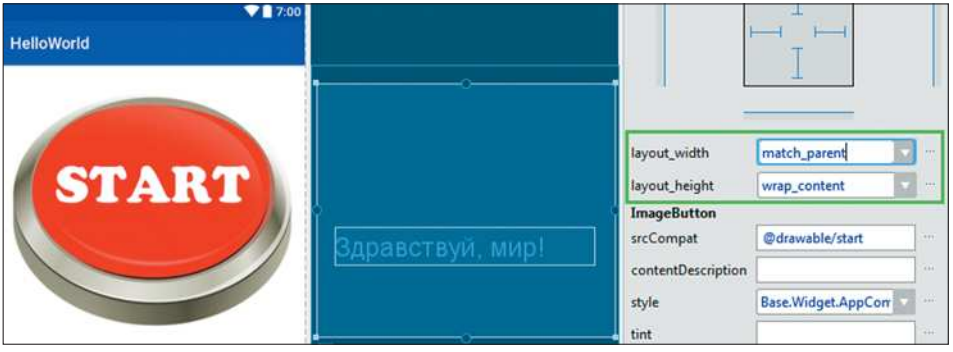
- **match_parent** (*соответствие родительскому элементу*) — элемент займет всю ширину/длину родительского элемента, в нашем случае элемент примет ширину/длину всего экрана;
- **wrap_content** (*соответствие контенту*) — изображение на экране примет размер исходного загруженного изображения. Это значит, что если мы загрузили картинку размером 200×200 пикселей, то при выборе **wrap_content** наш элемент примет длину/ширину 200dp.



Для нашей кнопки зададим ширину **match_parent**, чтобы она занимала весь экран, и высоту **wrap_content**, чтобы при этом сохранить пропорции картинки:

---

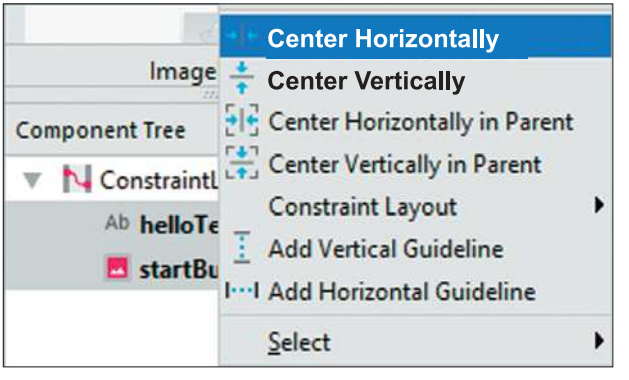
1 **dp (density-independent pixels)** — пиксели, независимые от плотности. Абстрактная единица измерения, основанная на плотности физических пикселей экрана с разрешением 160 пикселей. Для такого экрана 1dp = 1пиксель, для экрана с разрешением 240 пикселей 1 dp = 2 пикселя и далее по пропорции. Используется для того, чтобы сделать интерфейс приложения **адаптивным** и чтобы его элементы выглядели одинаково на всех устройствах (экранах с разными разрешениями).



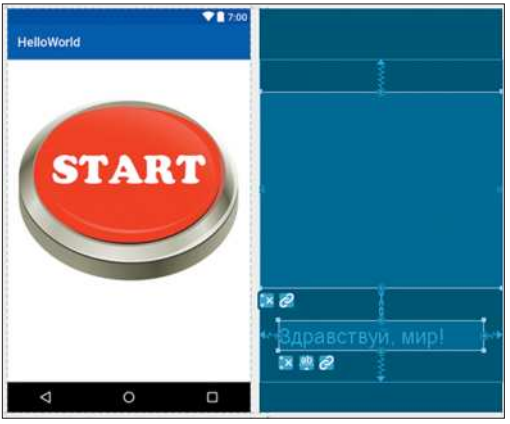
Последнее, что мы можем сделать в режиме дизайна на данный момент, — выровнять расположение элементов относительно друг друга и центра экрана (то есть **центрировать** их). Центрировать элементы нужно обязательно, иначе при запуске приложения все элементы примут начальные координаты (0, 0), то есть сместятся в левый верхний угол экрана и станут накладываться друг на друга. Это правило нашего корневого элемента **(ConstraintLayout)**.

О том, как будут вести себя элементы интерфейса в корневых элементах других типов, мы поговорим в следующем уроке. А пока запомним, что элементы внутри **ConstraintLayout** нужно всегда центрировать.

Удерживая клавишу **Shift** и кликая по элементам, выделим наши текст и кнопку в дереве компонентов и нажмем на них **правой кнопкой мыши**.



В появившемся контекстном меню выберем пункт **Center Horizontally** (*центрировать по горизонтали*). Затем снова нажмем на выделенные элементы правой кнопкой и выберем следующий за ним пункт **Center Vertically** (*центрировать по вертикали*). Элементы расположились точно по центру:



На этом работа в режиме дизайна завершена. Теперь, для того чтобы наш задуманный алгоритм работал и при нажатии на кнопку что-нибудь происходило, настало время писать код. Для этого перейдем на вкладку **Text**, расположенную в нижней части экрана.

1.4. Работа в режиме кода

В режиме кода наш файл **activity_main.xml** имеет следующий вид: все созданные нами элементы и их свойства отражены по порядку строками xml-кода, а справа для нашего удобства открыта область **предварительного просмотра:**

Одним из достоинств **Android Studio** является то, что, работая в режиме как дизайна, так и кода программист имеет совершенно одинаковые возможности. Все, что написано в коде, тут же отражается в свойствах, и любое заданное свойство тут же прописывается в коде.

Рассмотрим xml-код, созданный для нашей кнопки:

```
<ImageButton
    android:id="@+id/startButton"
    style="@style/Base.Widget.AppCompat.Button.Borderless.Colored"
    android:layout_width="368dp"
    android:layout_height="wrap_content"
    android:onClick="onStartButtonClick"
    app:srcCompat="@drawable/start"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintBottom_toTopOf="@+id/helloText"
    app:layout_constraintTop_toTopOf="parent" />
```

В элементе **<ImageButton ... />** по порядку описываются все заданные нами свойства в виде атрибутов. Если идти по строкам, то мы увидим:

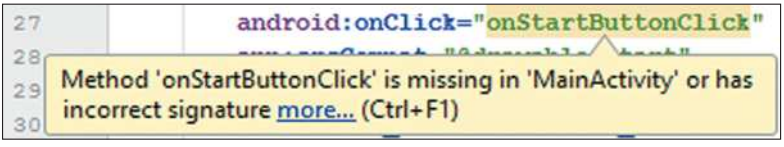
- идентификатор (атрибут **android:id**);
- стиль (атрибут **style**);
- ширину (атрибут **android:layout_width**);
- высоту (атрибут **android:layout_height**);
- событие onClick (атрибут **android:onClick**);
- ресурс — изображение (атрибут **app:srcCompat**);
- последние четыре строки отвечают за расположение элемента, они сформировались в результате наших выравниваний элементов по центру.

Пояснение

Вместо того чтобы искать нужное свойство элемента на панели свойств в режиме дизайна, можно с таким же успехом прописать его в xml-разметке. Иногда это даже удобнее, учитывая, что при вводе нескольких символов **Android Studio** всегда выдает подсказки и возможные варианты.

Проведем эксперимент. Выделим всю разметку элемента **<ImageButton ... />** и вырежем его **(Ctrl+X)**. На боковой панели **предпросмотра** увидим, что стартовая кнопка пропала. Вставим код обратно **(Ctrl+V)** — кнопка появилась на прежнем месте с прежними настройками.

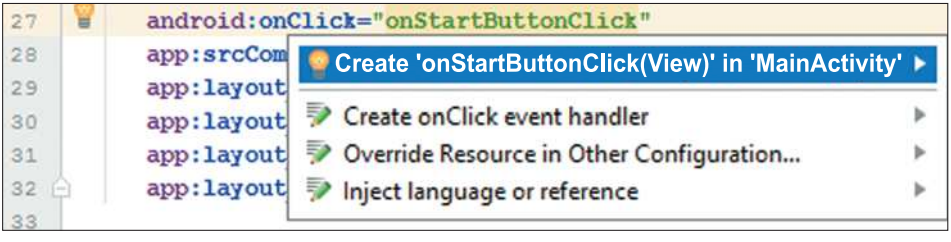
Но вернемся к кнопке. Нам все еще предстоит написать для нее обработчик события нажатия. Находим в коде атрибут **android:onClick** и видим, что его значение подсвечено. Значит, у **Android Studio** есть для нас подсказка на этот счет. Наводим курсор и видим сообщение о том, что указанного метода **onStartButtonClick** нет в Java-классе **MainActivity**:



Пояснение

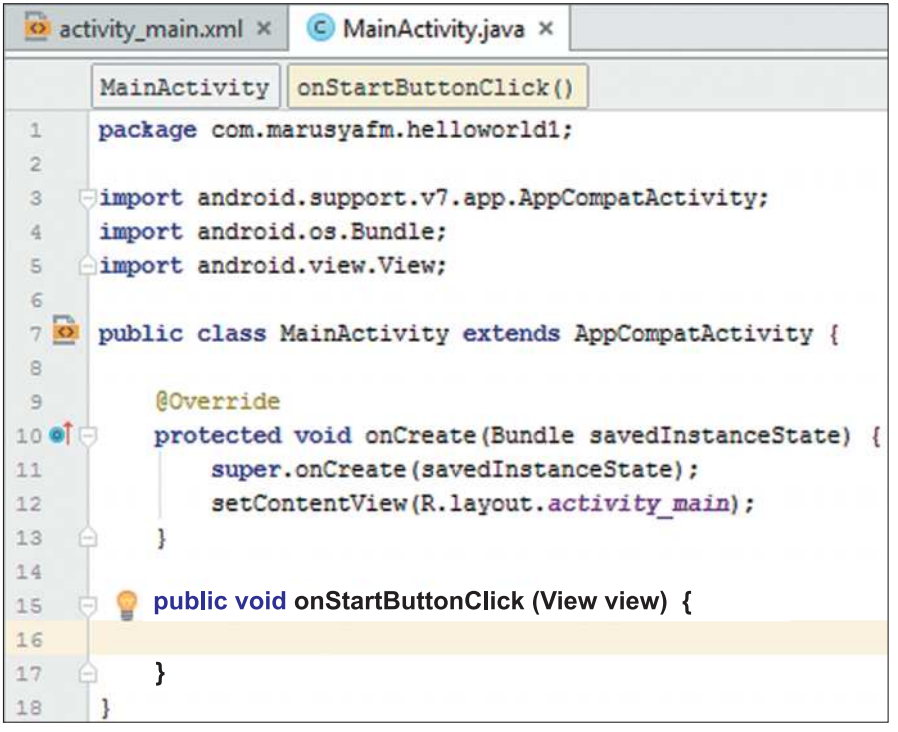
MainActivity.java — это класс, в котором на языке программирования Java описываются обработчики событий всех элементов главного экрана: что должно происходить при открытии экрана, нажатии на кнопки и так далее.

Его там и правда нет — мы его еще не написали. Но искать этот класс в структуре проекта и создавать в нем метод вручную мы тоже не станем, и все благодаря подсказкам **Android Studio**. Нажимаем на названии метода (сейчас оно подсвечено желтым цветом), затем сочетание клавиш **Alt+Enter** и видим целый список предлагаемых действий для данного случая:



Нас интересует первый пункт — **Create ‘onStartButtonClick (View)’ in ‘MainActivity’** (*создать метод в MainActivity*). Выбираем его и нажимаем **Enter**. Если первой опции в списке нет, выбираем **Create onClick event handler** (*создать обработчик события нажатия*). В открывшемся окне выбираем класс **MainActivity** и также нажимаем клавишу **Enter**.

Android Studio самостоятельно открывает файл **Main Activity** и создает в нем метод **public void onStart ButtonClick(View view) {}:**



В этом методе нам остается только прописать, какие именно действия должны выполняться при нажатии на кнопку.

Но как нам удалось совершить такой резкий переход от одного языка программирования к другому? Неужели Java каким-то волшебным образом понимает то, что написано на XML, и может этим самостоятельно оперировать?

Не совсем так — Java пока ничего не знает о созданных нами элементах, и наша задача «рассказать» ему о них, то есть **объявить** их.

Находим класс **public class MainActivity** и в самом его начале объявляем переменную. Для начала объявим наш элемент **TextView**. Вся строка объявления будет иметь вид:

```
private TextView helloText;
```

где TextView — тип переменной (тип нашего элемента), а helloText — заданное нами имя переменной (для наглядности совпадающее с ID элемента).

Пояснение

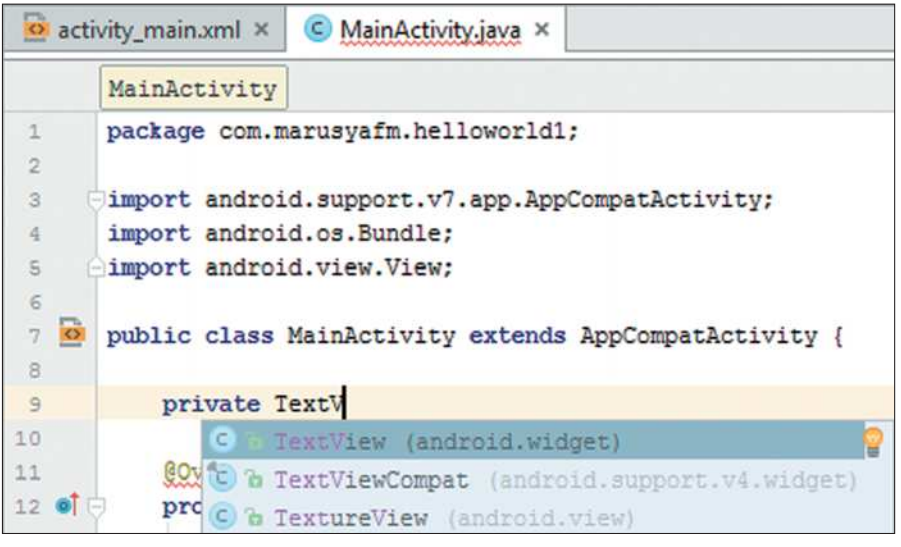
Одно из основных понятий объектно-ориентированного программирования (а Java является объектно-ориентированным языком программирования) — **инкапсуляция**. Инкапсуляция подразумевает объединение данных и всех методов, необходимых для работы с ними, в классы для обеспечения возможности их дальнейшего сокрытия (изоляции).

Реализация **принципа инкапсуляции** — использование **модификаторов доступа** для сокрытия части программного кода от конечного пользователя (обеспечения контроля доступа к классам и их членам).

Модификаторы доступа:

- **public** (*публичный, общедоступный*) — открытый доступ, может использоваться любым классом;
- **private** (*частный, закрытый*) — закрытый доступ, обеспечивается только из текущего класса;
- **protected** (*защищенный, частично скрытый*) — доступ для текущего класса и производных от него.

Здесь нас ждет очень важная подсказка от **Android Studio**. Когда мы будем набирать тип переменной, появится список предлагаемых вариантов.



Если выбрать значение **TextView** из списка и нажать **Enter** (а не вводить самим), **Android Studio** также **автоматически импортирует все необходимые библиотеки** для работы с этим типом. Блок **import** (*импортированных библиотек*) нам не нужно

прописывать самостоятельно, он находится в начале файла, сразу после имени **package** (*пакета*)[1].

Настало время дать понять языку Java, что только что объявленная переменная **helloText** — это именно приветственный текст, расположенный на нашем экране.

С этой целью в конец метода **protected void onCreate** (*при создании*)[2] добавим строку:

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

где переменная **helloText** — элемент с типом **TextView** и **ID** helloText (тот самый приветственный текст, размещенный на экране); **findViewById** (*найти элемент по идентификатору*) — метод поиска элемента в приложении по его ID.

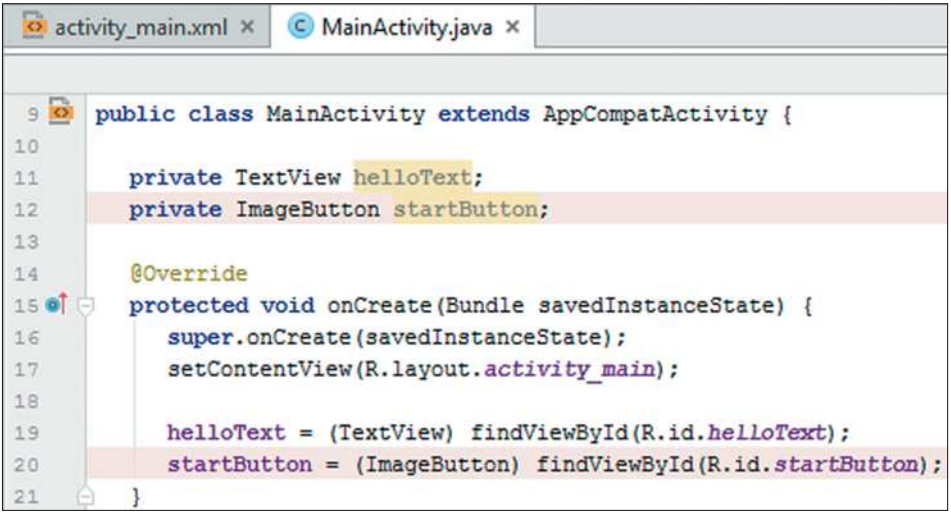
activity_main.xml × MainActivity.java ×

```
1  package com.marusyafm.helloworld1;
2
3  import android.support.v7.app.AppCompatActivity;
4  import android.os.Bundle;
5  import android.view.View;
6  import android.widget.TextView;
7
8  public class MainActivity extends AppCompatActivity {
9
10     private TextView helloText;
11
12     @Override
13     protected void onCreate(Bundle savedInstanceState) {
14         super.onCreate(savedInstanceState);
15         setContentView(R.layout.activity_main);
16
17         helloText = (TextView) findViewById(R.id.helloText);
18     }
19
20     public void onStartButtonClick(View view) {
21
22     }
```

[1] **Пакет** — это именованная группа классов. В случае с **com.marusyafm.helloworld1** пакет имеет имя приложения, а значит, он содержит все классы, входящие в данное приложение.

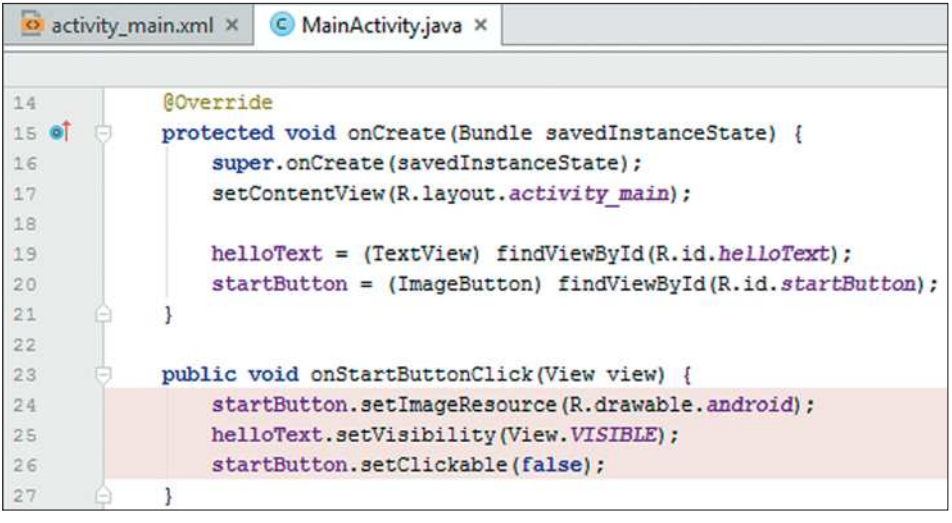
[2] В этом методе описываются все события, которые должны происходить, и элементы, которые должны отображаться на экране при его «создании», то есть все, что увидит пользователь, перейдя к этому экрану.

Аналогичным образом объявляем и «определяем» нашу кнопку. Тип — ImageButton, имя переменной — startButton:



Итак, у нас есть все для написания обработчика события нажатия кнопки. В метод **public void onStartButtonClick** пишем следующие три строки кода (см. также рисунок ниже):

```
startButton.setImageResource(R.drawable.android);
helloText.setVisibility(View.VISIBLE);
startButton.setClickable(false);
```

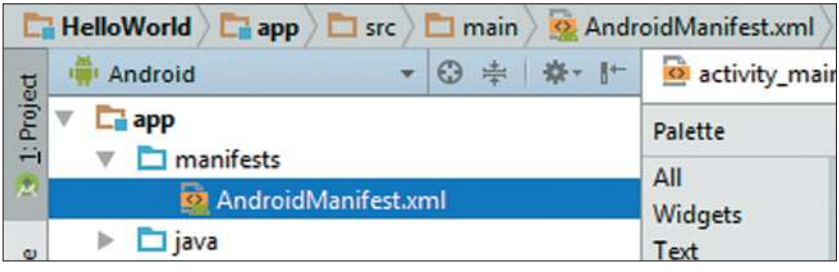


Первой строкой (№ 24) мы устанавливаем для кнопки новую картинку. В самом начале мы загружали в папку **drawable** две картинки — кнопку **(start)** и приветствующего робота **(android)**. Теперь мы прописали название картинки android при нажатии кнопки, то есть после нажатия картинка на кнопке изменится и вместо красной кнопки появится робот.

Второй строкой (№ 25) мы устанавливаем видимость текста Visible, так как по умолчанию он у нас невидимый, а при нажатии на кнопку должен появиться.

И третьей строкой (№ 26) мы снова возвращаемся к кнопке и настраиваем возможность ее нажатия («кликабельность», активность). Выбираем false (ложь), то есть после того как мы нажмем кнопку, ее картинка изменится, на экране появится текст, а сама кнопка станет неактивной.

Наше приложение почти готово к запуску, осталось только поменять его название. За это отвечает файл **AndroidManifest.xml**, который лежит в папке **manifests**. Откроем его.



1.4.1. AndroidManifest — файл манифеста

Файл манифеста **AndroidManifest.xml** содержит основную информацию о нашем приложении. В нем описываются отображаемое название приложения, иконка, миниатюра, разрешения и полномочия (доступ к системным папкам и процессам, файловой системе и так далее), основные компоненты приложения.

Файл манифеста считывается системой Android на нашем устройстве при установке приложения. Именно благодаря этому файлу на нашем экране появляется значок и нужное название приложения, сразу же при установке запрашиваются разрешения на использование сторонних приложений и сервисов (Интернет, камера, внутренняя память и др.). А еще благодаря файлу манифеста Android определяет, нужно ли установить приложение или же оно уже установлено и только нуждается в обновлении.

activity_main.xml × MainActivity.java × AndroidManifest.xml ×

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.marusyafm.helloworld1">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="HelloWorld"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

На данный момент в манифесте нас интересует только название приложения, которое будет отображаться в заголовке приложения, в меню смартфона и в магазине приложений. Чтобы изменить это название, находим атрибут **android:label** и меняем его значение (прописанное в кавычках) на Мое первое приложение.

activity_main.xml × MainActivity.java × AndroidManifest.xml ×

manifest application

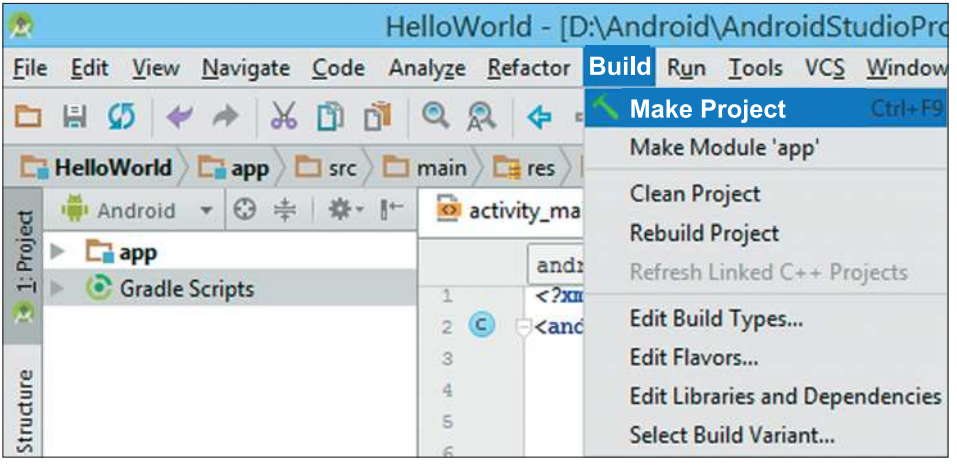
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.marusyafm.helloworld1">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="Мое первое приложение"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
```

Наше первое приложение готово! Теперь его можно запустить и протестировать.

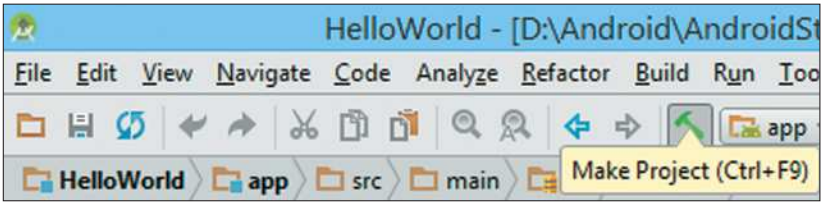
1.5. Сборка проекта

Для того чтобы запустить приложение и посмотреть, что же у нас в итоге получилось, для начала проект Android Studio необходимо «собрать»: скомпилировать, проверить на наличие ошибок (тех, что не видны в коде или просто не замечены разработчиком), создать все недостающие служебные файлы.

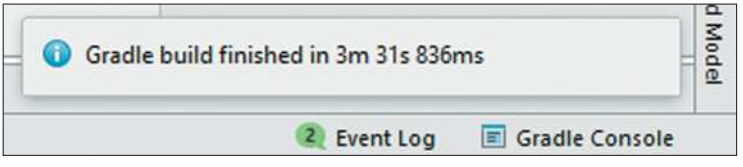
В главном меню выбираем пункт **Build → Make Project** (*Сборка → Собрать проект*):



Это же действие можно выполнить с помощью сочетания клавиш **Ctrl+F9** или же нажатием на значок молотка на панели быстрого доступа:



После некоторого ожидания **Android Studio** проинформирует нас об успешном завершении операции или же о наличии ошибок в проекте (и предложит быстрый переход к их исправлению):



В данном случае проект был собран успешно, ошибок в нем нет. Ура! Можно приступать к самому волнующему этапу — запуску и тестированию нашего первого приложения.

1.6. Тестирование приложения

Тестирование приложения — это процесс его проверки на наличие самых разных ошибок (багов).

Приложение, прежде чем попадет в магазин, должно быть **тщательно и многократно протестировано** на разных устройствах и желательно большим количеством тестировщиков. Это позволит избежать «лавины» отрицательных отзывов.

Тестирование может быть разных типов и разного назначения: тестирование вручную и автоматическое тестирование, тестирование функционала и юзабилити-тестирование. В крупных компаниях, занимающихся разработкой мобильных приложений, существуют целые отделы тестирования. Пишутся сложные тестирующие программы, позволяющие обнаружить даже те ошибки, которые человек может пропустить.

Но мы пока только начинающие Android-разработчики, поэтому для нас тестированием будет простая проверка приложения на «работоспособность».

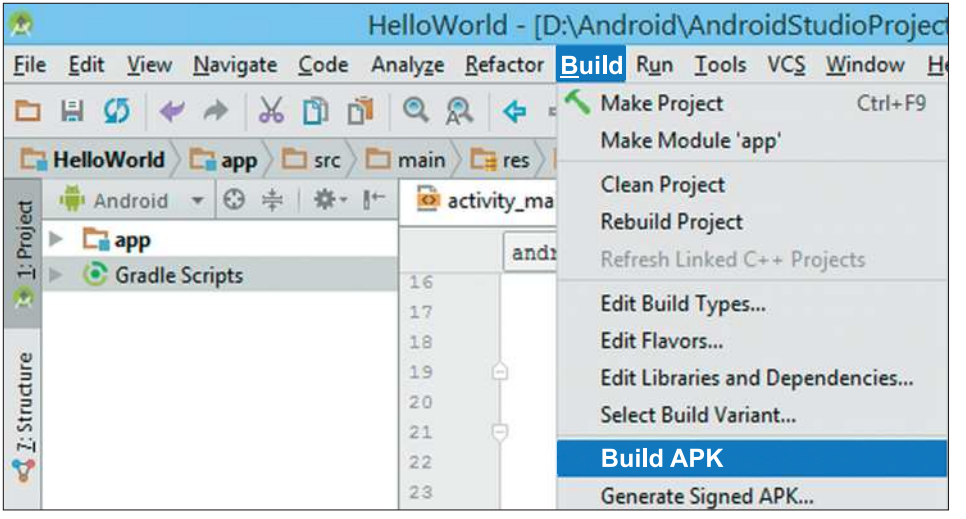
Есть несколько способов тестирования Android-приложений: файл APK, эмулятор и USB-отладка. Начнем, пожалуй, с самого увлекательного — тестирования на реальном устройстве, то есть на нашем собственном смартфоне.

1.6.1. Файл APK

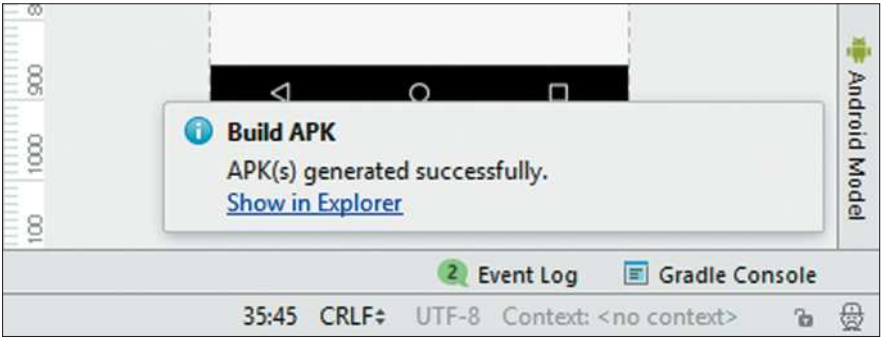
Мы каждый день пользуемся десятками мобильных приложений, и нам важно, чтобы они занимали как можно меньше памяти. А сейчас мы видим перед собой проект Android Studio — со сложной структурой, состоящий из большого количества самых разных файлов, занимающий под сотню мегабайт в памяти компьютера... Неужели все это хранится в наших смартфонах в таком виде и в таком же виде загружается из магазина Google Play? Разумеется, нет.

Для передачи, хранения и распространения приложений создается исполняемый (установочный) файл **APK** (Android Package Kit). По сути, APK это аналог zip-архива. В него упаковывается скомпилированное приложение с его структурой, файлами, ресурсами, библиотеками, но распаковывать его не нужно, поэтому он спокойно хранится в памяти смартфона, занимая несколько кило- или мегабайт.

Для создания APK-файла выбираем **Build** → **Build APK** (*Сборка* → *Собрать APK*):



Когда файл будет создан, **Android Studio** проинформирует нас об успешной генерации и предложит показать его расположение в проводнике (всплывающая подсказка в правом нижнем углу):



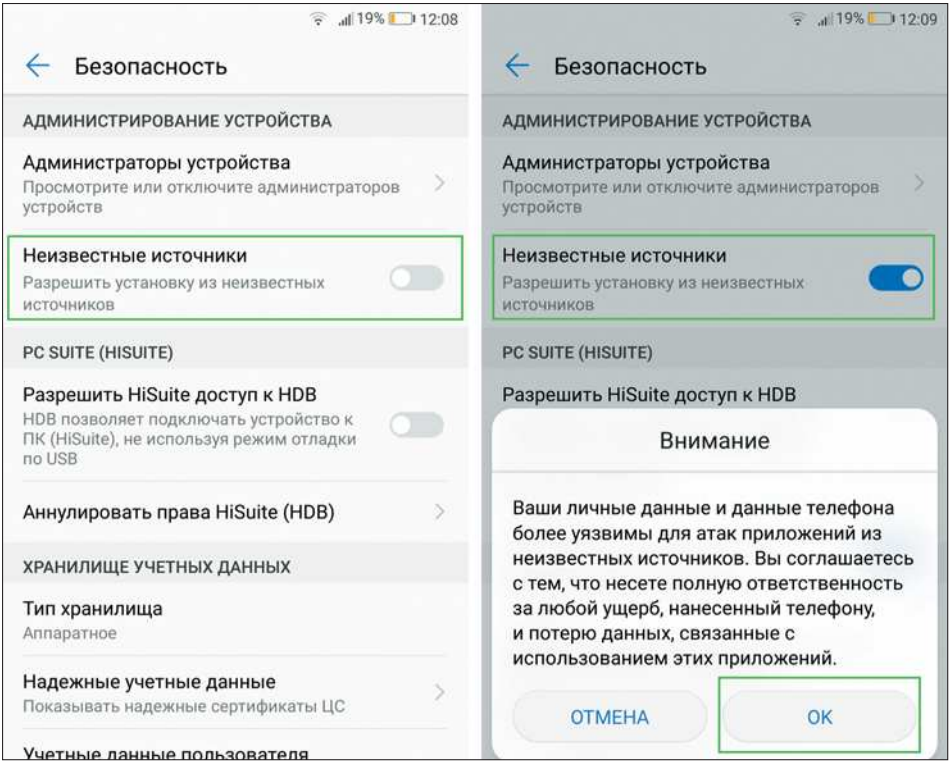
Перейдем по ссылке **Show in Explorer** (или **locate**). В открывшемся проводнике увидим наш файл — **app-debug.apk**.

По умолчанию файлы APK сохраняются в папке проекта: ...\HelloWorld\app\build\outputs\apk

Теперь у нас есть установочный файл нашего приложения. Его можно переименовать (так как по умолчанию все установочные файлы называются app-debug) и отправить на свой смартфон любым удобным способом: через облако, USB, мессенджер или электронную почту.

Но прежде чем запустить приложение, нужно немного изменить **настройки нашего смартфона**. В расширенных настройках

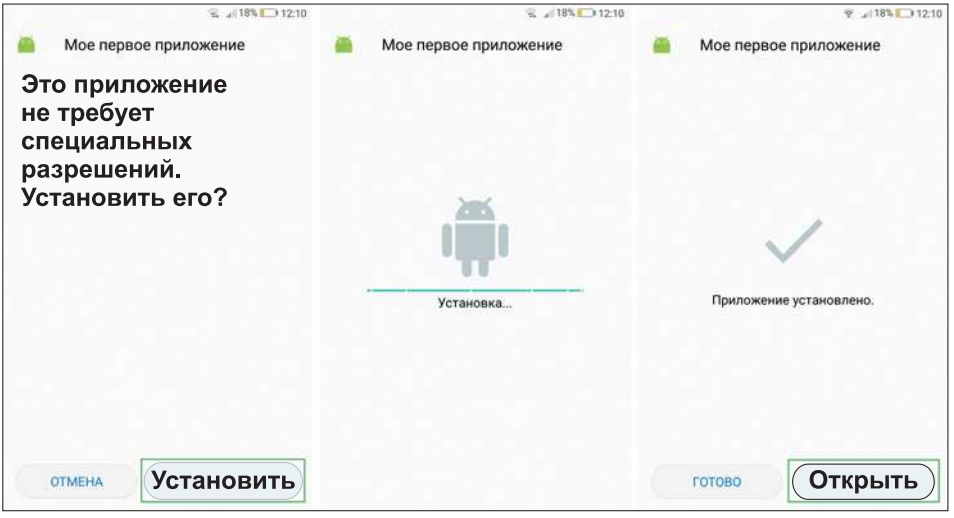
находим пункт **Безопасность** и в нем **Неизвестные источники**. В зависимости от модели смартфона или версии системы меню настроек может выглядеть по-разному, но этот пункт в нем обязательно присутствует. По умолчанию эта опция запрещена (тумблер выключен), но нам необходимо его включить и разрешить установку из неизвестных источников.



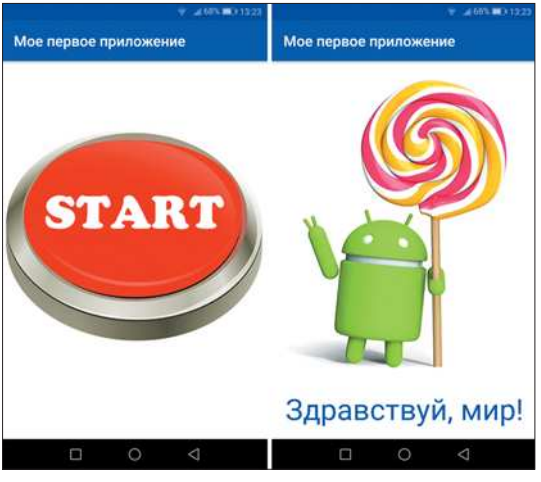
Для чего это нужно? Из соображений безопасности система **Android** разрешает установку программ и приложений только из «известных» ей (официальных) источников, например из магазина приложений Google Play. А наш APK получен из «неизвестного» для системы источника и с ее точки зрения может нести в себе угрозу безопасности.

Но мы-то в своем приложении уверены, в нем только написанный нами код, никаких вредоносных скриптов. Поэтому разрешаем — во всплывающем окне с предупреждением нажимаем **OK**. Позже эти настройки можно будет снова изменить.

Теперь находим наш APK-файл в памяти смартфона. Запускаем его, нажимаем «Установить», наблюдаем за процессом установки и, когда она успешно завершается, нажимаем «Открыть».



Ура! Наше первое приложение запущено, и мы можем смело нажать заветную красную кнопку и прямо со своего смартфона поздороваться со всем миром!



1.6.2. Эмулятор

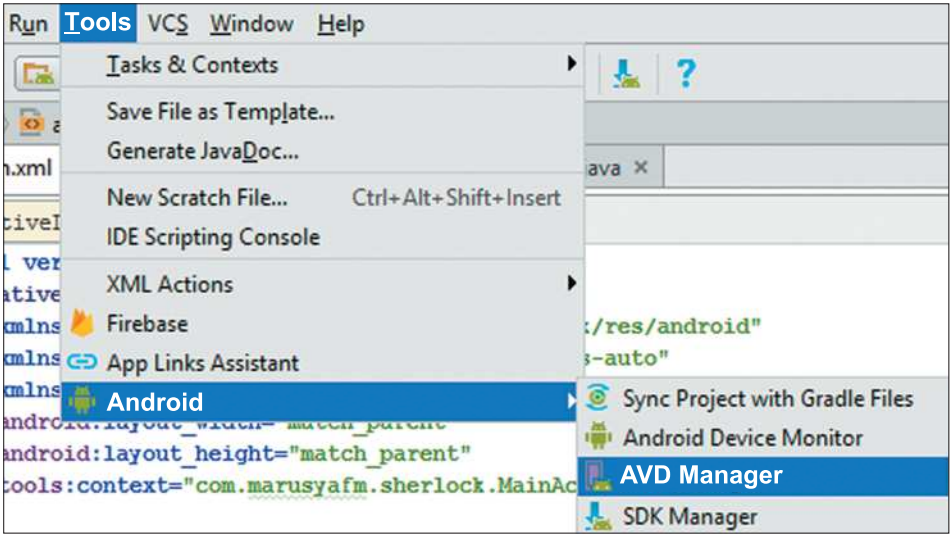
Но каждый раз создавать APK, передавать его куда-то, устанавливать и переустанавливать приложение не слишком удобно и быстро, поэтому так делают в основном для уже готовых приложений. А пока приложение находится в стадии разработки, гораздо удобнее тестировать его прямо на компьютере — с помощью **эмулятора**.

Эмулятор смартфона — это виртуальное устройство, имитирующее работу смартфона, на котором будет запускаться приложение.

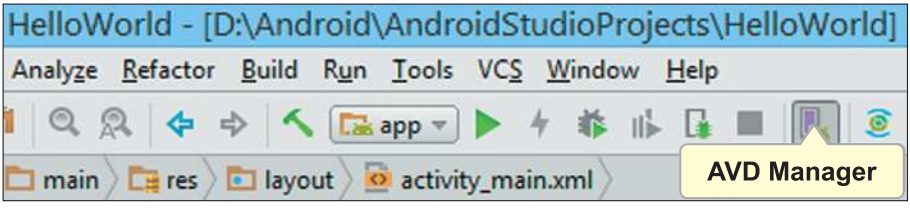
Использовать эмулятор очень удобно, потому что:

1) он вызывается непосредственно из **Android Studio**, что значительно экономит наше время;
2) можно создать виртуальные копии практически всех моделей смартфонов и сразу смотреть, как будет выглядеть приложение на разных дисплеях, с разными настройками и даже версиями операционной системы Android.

Для создания эмулятора в главном меню выбираем пункт **Tools** → **Android** → **AVD Manager** — менеджер виртуальных Android-устройств (Android Virtual Device Manager).

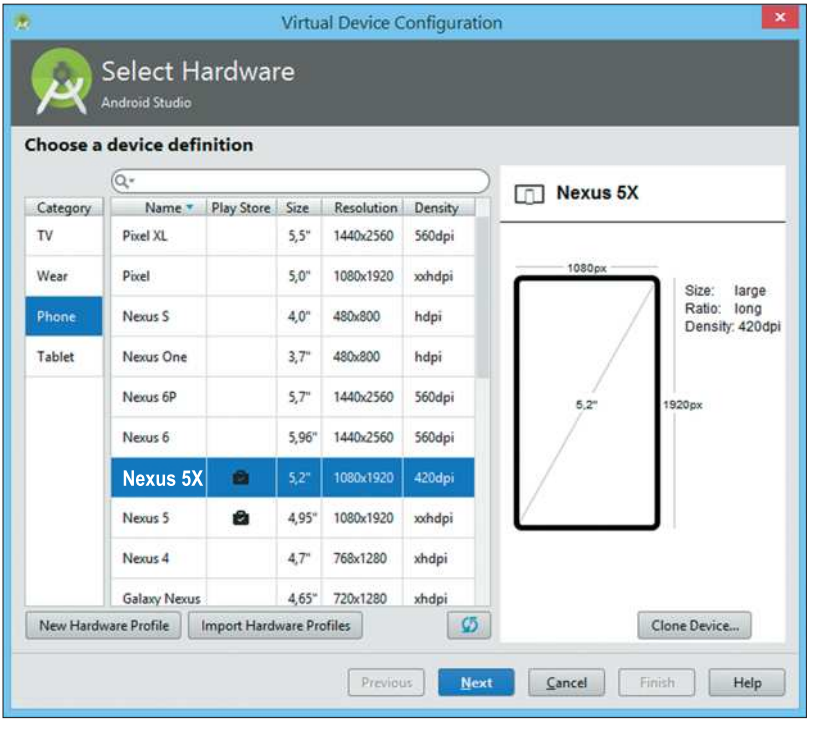


Можно также воспользоваться значком  на панели быстрого доступа:

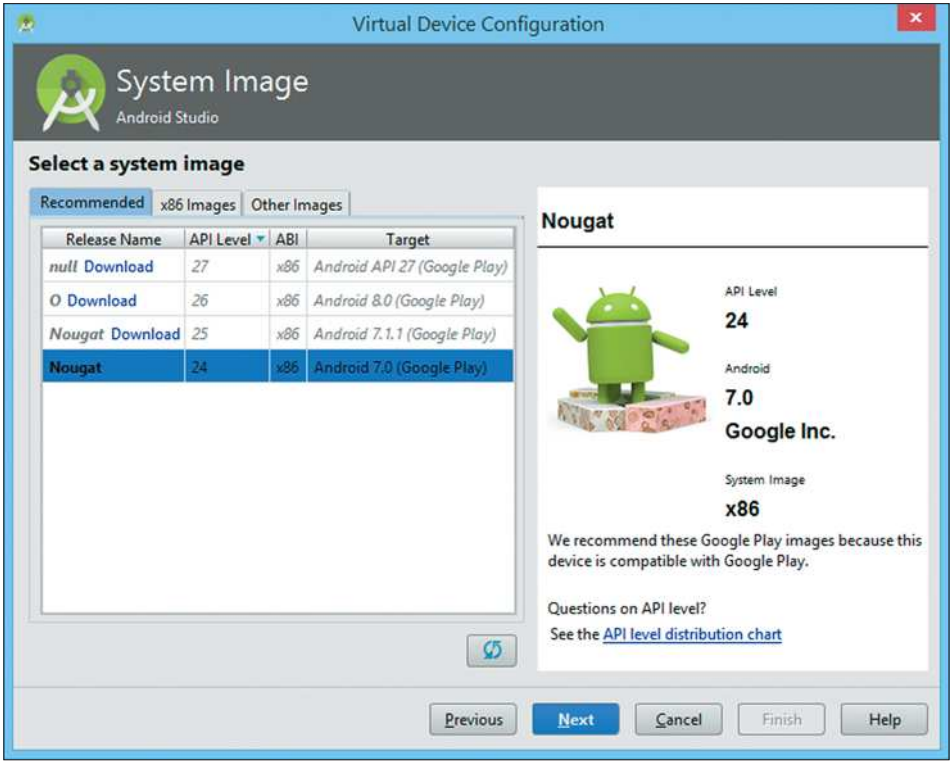


При первом запуске **AVD Manager** предложит создать новое виртуальное устройство. Нажимаем **Create Virtual Device** (*создать виртуальное устройство*):

В открывшемся окне видим список доступных для установки устройств и даже список категорий устройств (смартфоны, планшеты, умные часы, телевизоры). Для начала нам будет достаточно одного устройства. Выберем, например, **Nexus 5X** и перейдем к следующему шагу.

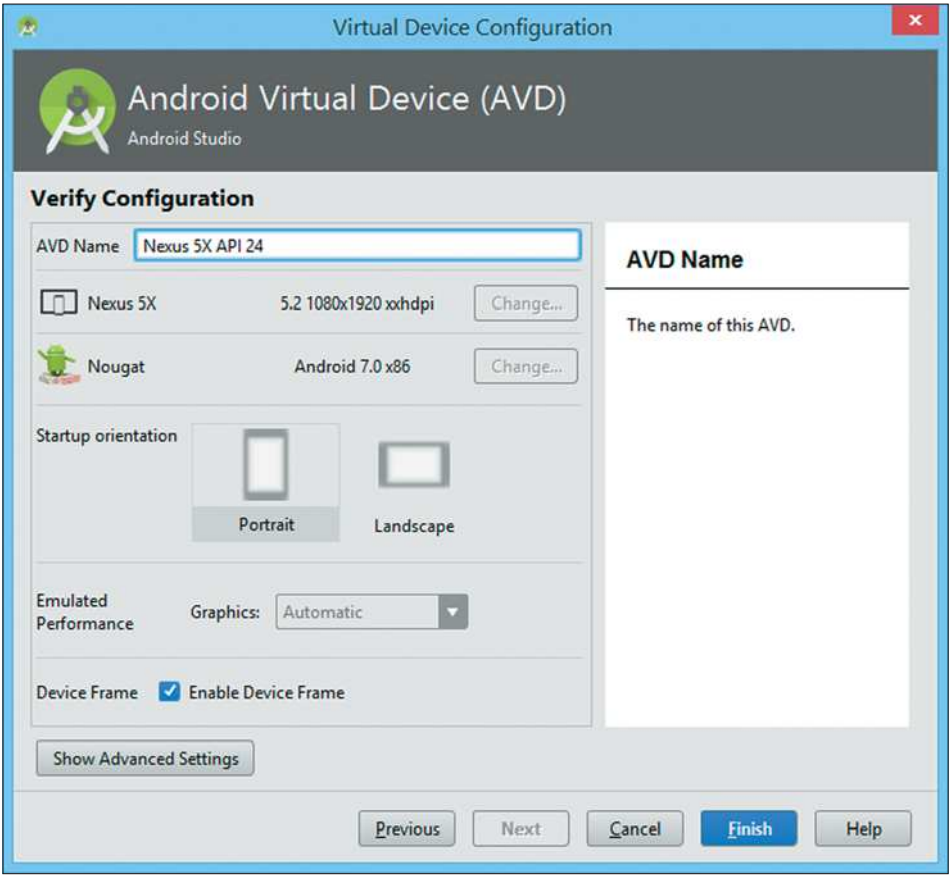


На втором шаге нужно выбрать версию операционной системы Android для нашего виртуального устройства.



Можно выбрать одну из рекомендованных систем (Recommended) или перейти на другие вкладки и выбрать любую версию Android из когда-либо существовавших. Для примера выбираем **Android 7.0 Nougat**, нажимаем **Download** (ссылка рядом с названием системы), дожидаемся загрузки и переходим к последнему шагу.

Здесь нам предоставляется возможность еще раз увидеть набор выбранных настроек (имя виртуального устройства, версия Android, начальная ориентация экрана, графический процессор) и, при необходимости, изменить их:



При разработке серьезных приложений, которые мы захотим выложить в магазин приложений, к вопросу тестирования нужно будет отнестись максимально ответственно и «опробовать» нашу разработку на устройствах с самыми разными диагоналями экрана и операционными системами. И тогда нужно будет следить, чтобы создаваемые нами виртуальные устройства действительно имели разный набор параметров. Но сейчас, при первом запуске, это не так важно.

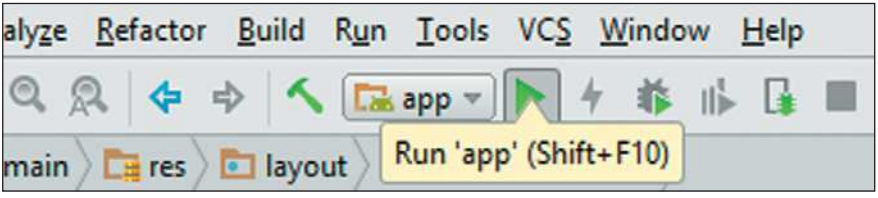
Оставляем все без изменений, нажимаем **Finish**.

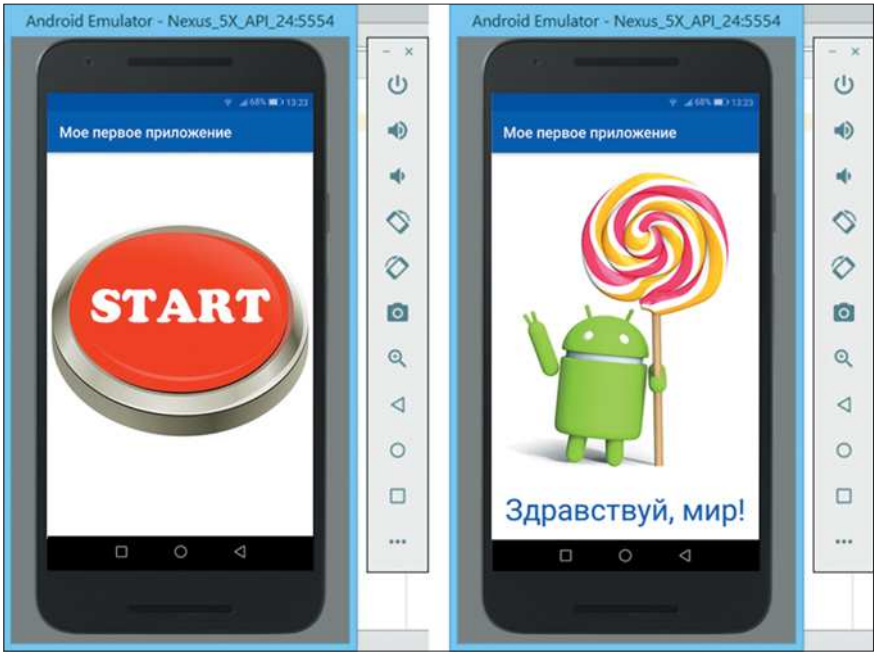
Во вновь появившемся окне **AVD Manager** видим список созданных виртуальных устройств. Их можно настраивать и редактировать их параметры, можно создавать новые устройства. Запускается виртуальное устройство нажатием на кнопку  (**Launch this AVD in the emulator** — *запустить в эмуляторе это виртуальное устройство*):

Далее следует загрузка эмулятора, которая может занять несколько минут.

Готово. Эмулятор запущен.

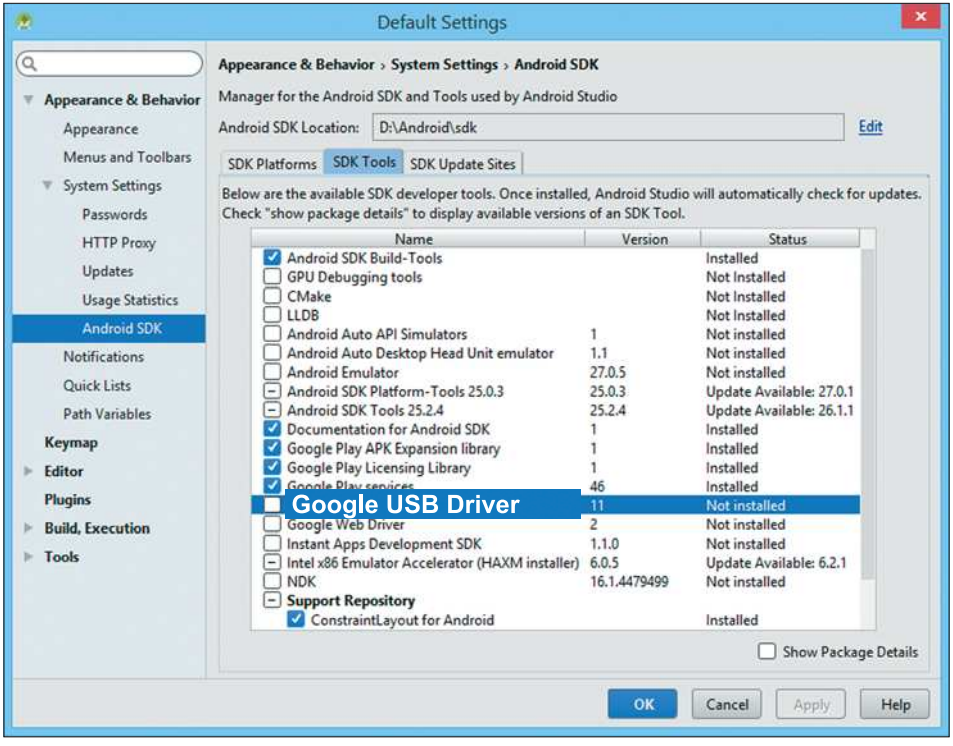
Теперь можно тестировать приложения в онлайн-режиме — вносить изменения в проект и тут же просматривать их. Для этого достаточно нажать на кнопку  на панели быстрого доступа и выбрать устройство для тестирования.



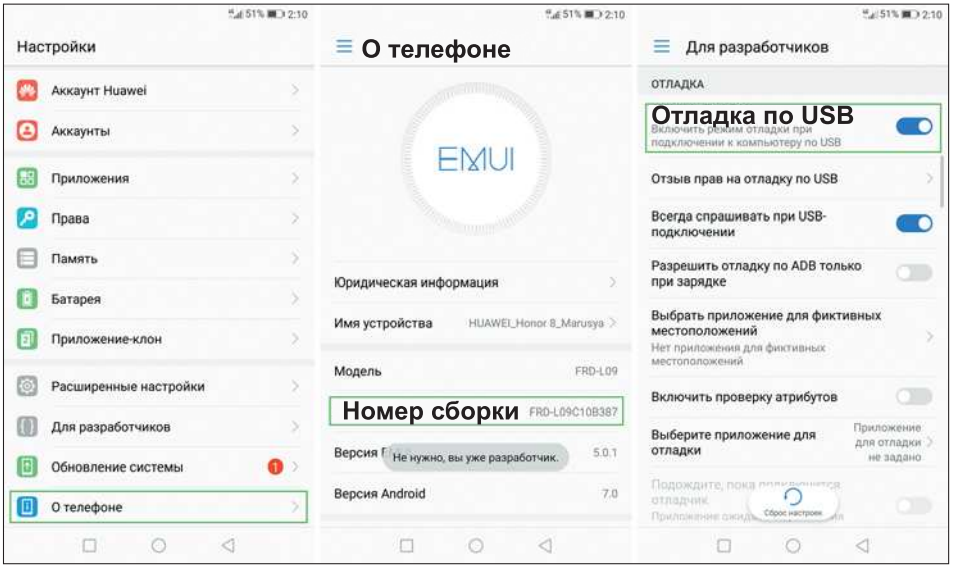


1.6.3. USB-отладка

Вместо эмулятора можно использовать смартфон (или планшет), подключенный к компьютеру через кабель USB. Для этого выбираем пункт **Tools → Android → SDK Manager**, переходим на вкладку **SDK Tools** и убеждаемся, что установлен **Google USB Driver**. Если нет — отмечаем этот пункт и нажимаем кнопку **Apply** (*применить*).



Далее нужно настроить наш смартфон (или планшет) — активировать в нем **режим разработчика.** В разделе настроек **Об устройстве** есть пункт **Номер сборки,** который нужно нажать



7–10 раз до появления сообщения об активации режима разработчика. Это означает, что в настройках устройства появился раздел **Для разработчика**, в котором нужно активировать пункт **Отладка по USB (ADB)**.

Настало время подключить устройство к компьютеру. При следующем запуске приложения в окне **AVD Manager** появится наше устройство, а на его экране запустится тестируемое приложение.

Итоги главы 1

Вот мы и создали наше первое Android-приложение. Несмотря на его кажущуюся внешнюю простоту, мы уже усвоили многие ключевые моменты Android-разработки и научились следующим вещам:

1. Установили новую для себя среду разработки — **Android Studio**.
2. Освоили инструментарий **Android Studio**.
3. Научились работать в режимах дизайна и кода.
4. Освоили основы xml-разметки.
5. Познакомились с основами программирования на Java: объявлением переменных, методами, классами, обработчиками событий.
6. Выполнили отладку и запуск приложения.
7. Создали и настроили эмулятор.
8. Создали файл APK и протестировали приложение на собственном смартфоне.
9. Прошли полный путь от идеи приложения и формулировки требований к нему до получения готового продукта и его запуска.

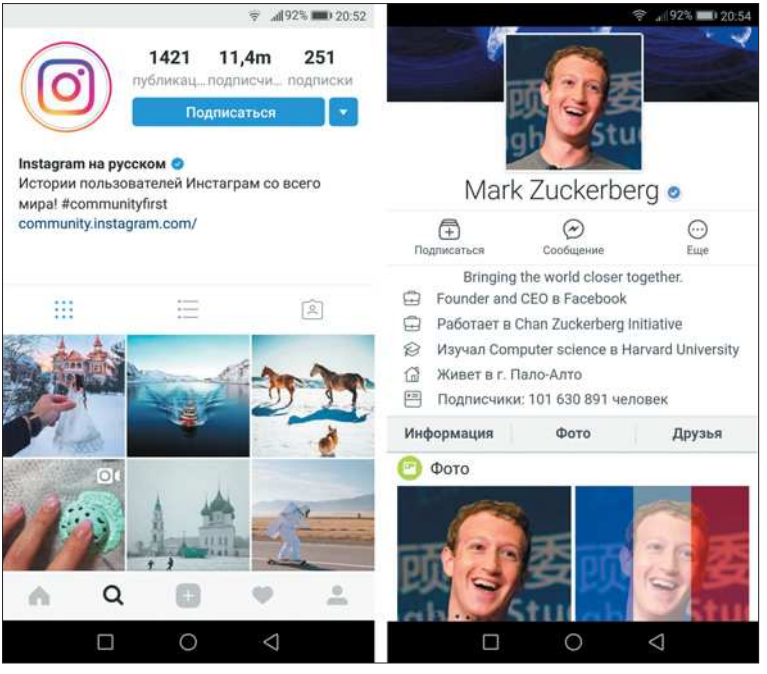
Внушительный список, не правда ли? Нам уже есть чем гордиться. И это был только первый урок Android-разработки — впереди самое интересное!

Глава 2. Основы проектирования интерфейса

2.1. Макеты

Мы уже поприветствовали мир, а теперь представимся ему и создадим наш «профиль начинающего разработчика».

Большинство социальных сетей использует профили, в которых примерно в одном порядке размещается основная информация. На рисунке представлены скриншоты приложений Instagram (официальный аккаунт Instagram на русском @instagramru) и Facebook (официальный аккаунт создателя Facebook Марка Цукерберга facebook.com/zuck):

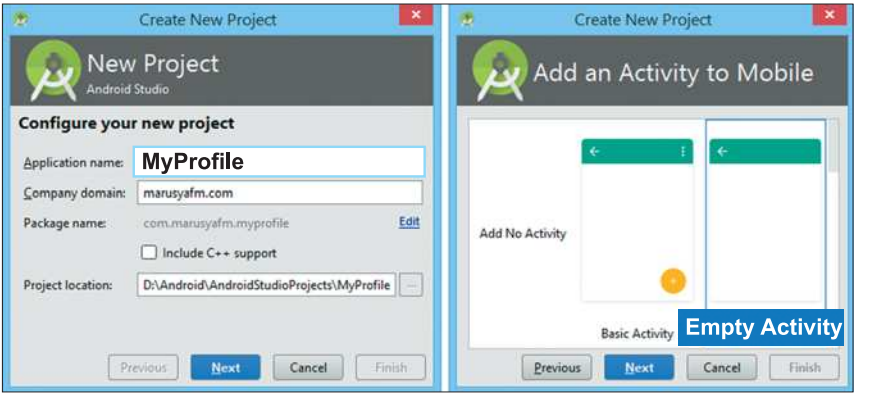


Легко заметить, что информация размещается блоками, в которых элементы следуют в определенном порядке по горизонтали, по вертикали или наслаиваются один на другой.

Создавая такие структурные блоки, мы рассмотрим основные виды компоновки и размещения элементов в **Android Studio**.

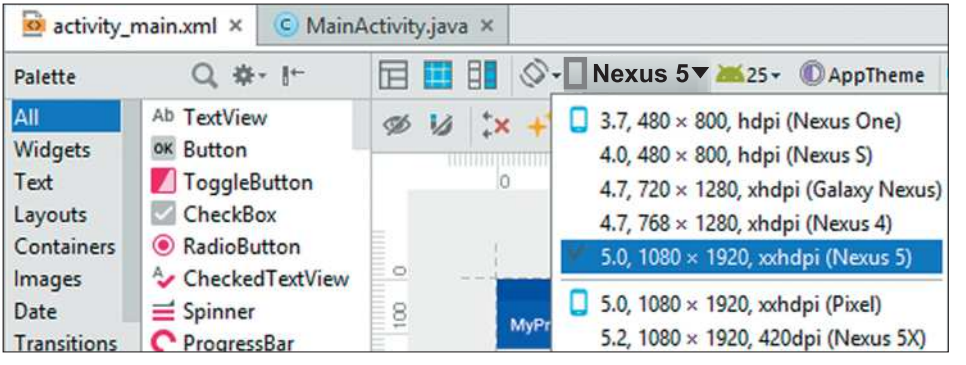
Концепция мини-приложения: «профиль начинающего Android-разработчика». На главной странице блоками размещается информация: фото профиля на фоне обложки, персональные данные, меню, фотографии и так далее.

Создадим новый проект Android Studio с именем MyProfile. На следующих шагах оставляем все настройки без изменений (по умолчанию), в том числе тип **MainActivity** — Empty Activity.



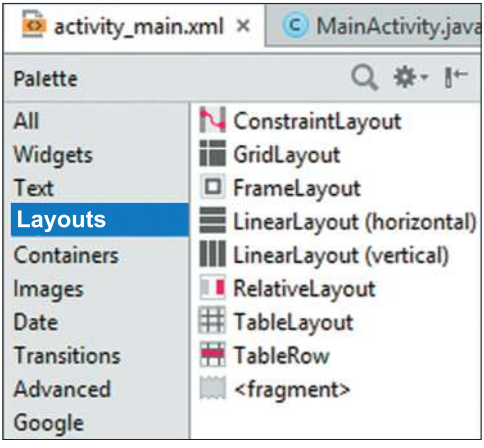
В файле xml-разметки **activity_main** удалим стандартно расположенный там элемент **TextView** HelloWorld.

Большинство смартфонов сегодня имеют диагональ 5 дюймов и больше, поэтому увеличим диагональ для нашего макета. В панели над макетом видим, что сейчас выбран 4.7-дюймовый Nexus 4. В открывшемся по нажатию на этот пункт меню выберем 5-дюймовый Nexus 5. Рекомендуется выбирать его и для следующих проектов.



Приступим к наполнению главного экрана информацией. Мы собираемся разместить на нем большое количество элементов — фотографий и различных надписей. Понятно, что они должны быть не хаотично разбросаны по экрану, а как-то логически скомпонованы.

Для компоновки в **Android Studio** имеются специальные элементы — **Layouts** (*макеты*). Их можно найти в палитре элементов в разделе **Layouts**:

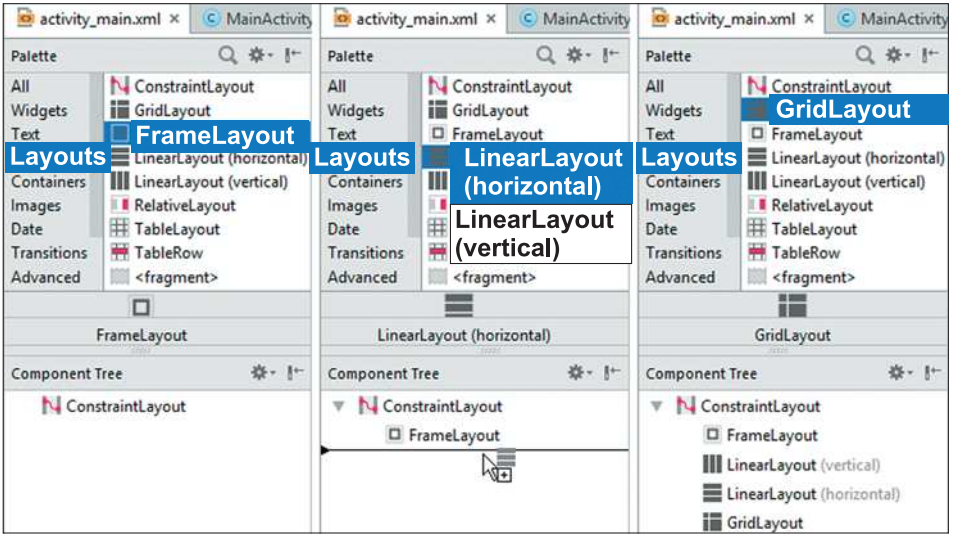


Наиболее часто используются следующие виды макетов:

- **ConstraintLayout** (*макет привязок*) — элементы располагаются с привязкой к расположению других элементов или родительского элемента. В этом макете элементы можно свободно перетаскивать по экрану, но обязательно сохранять привязку к другим элементам.
- **GridLayout** (*макет-сетка*) — элементы располагаются в прямоугольной сетке, похожей на таблицу. Для такой сетки можно задавать количество строк и столбцов, объединять ее ячейки и использовать многие другие возможности, которые обычно предоставляются при работе с таблицами в текстовых и табличных процессорах.
- **FrameLayout** (*рамочный макет*) — элементы располагаются поверх других элементов (стопкой).
- **LinearLayout** (*линейный макет*) — имеет две разновидности:
  1) **LinearLayout (vertical)** — элементы располагаются друг за другом (линейно) *по вертикали* (один над или под другим);
  2) **LinearLayout (horizontal)** — элементы располагаются друг за другом (линейно) *по горизонтали* (один справа или слева от другого).
- **TableLayout** (*табличный макет*) — элементы располагаются в таблице. Этот макет похож на **GridLayout**, но появился гораздо раньше, поэтому на сегодняшний день предоставляет меньше возможностей.
- **RelativeLayout** (*макет отношений*) — элементы располагаются относительно родительского элемента или соседних элементов. С появлением **ConstraintLayout** практически перестал использоваться.

Рассмотрим возможности каждого из макетов.

Перетащим из палитры элементов на дерево компонентов макеты **FrameLayout**, **LinearLayout (vertical)**, **LinearLayout (horizontal)**, **GridLayout** и расположим их в таком порядке внутри созданного по умолчанию **ConstraintLayout**:

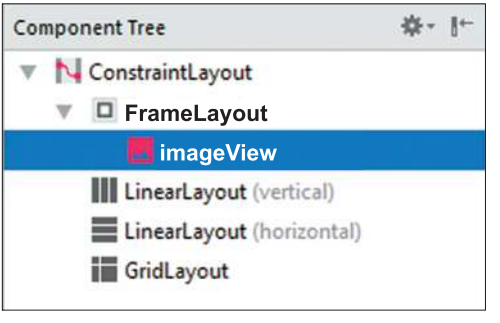


Теперь наполним каждый макет элементами.

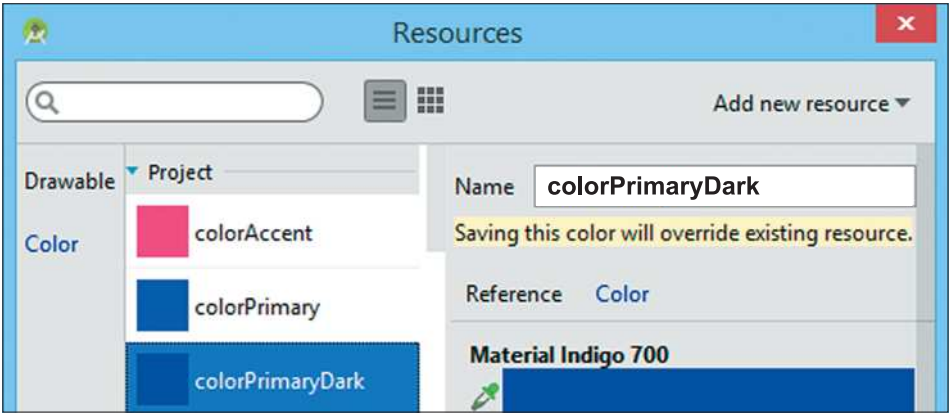
2.1.1. FrameLayout — расположение элементов друг над другом

Элементы во **FrameLayout** располагаются друг за другом (в стопку) по оси *Z*, то есть один наслаивается на другой. В концепции нашего приложения это «фото профиля» и его фон — «фото обложки», как, например, на странице профиля в Facebook.

Из палитры элементов перетаскиваем элемент **ImageView** в макет **FrameLayout** в дереве компонентов.



Элемент, добавленный первым, окажется самым дальним от нас и будет служить фоном (фото обложки). В качестве ресурса выбираем цвет, например colorPrimaryDark.

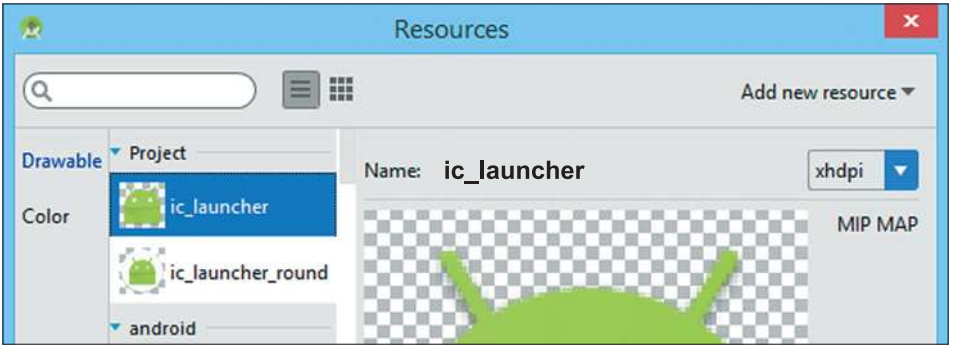


Примечание

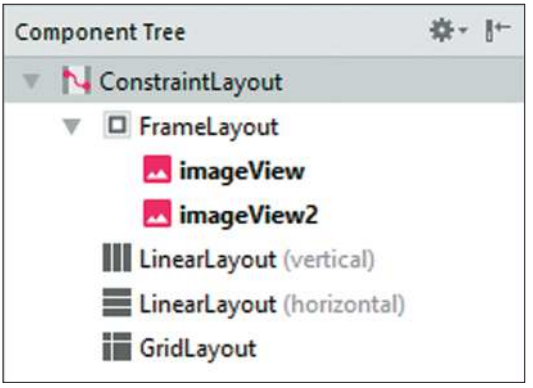
По умолчанию палитра цветов проекта состоит из трех цветов: **colorPrimary** (*основной цвет*), который по умолчанию имеет шапка экрана; **colorPrimaryDark** (*основной темный цвет*) для теней, элементов на заднем плане и панели уведомлений; **colorAccent** (*цвет акцентов*) для различных подчеркиваний и выделения.

Остальные цвета можно найти в библиотеке Android Studio (они идут списком после цветов по умолчанию). Любой цвет можно изменить, нажав на него и выбрав нужный оттенок в цветовой палитре.

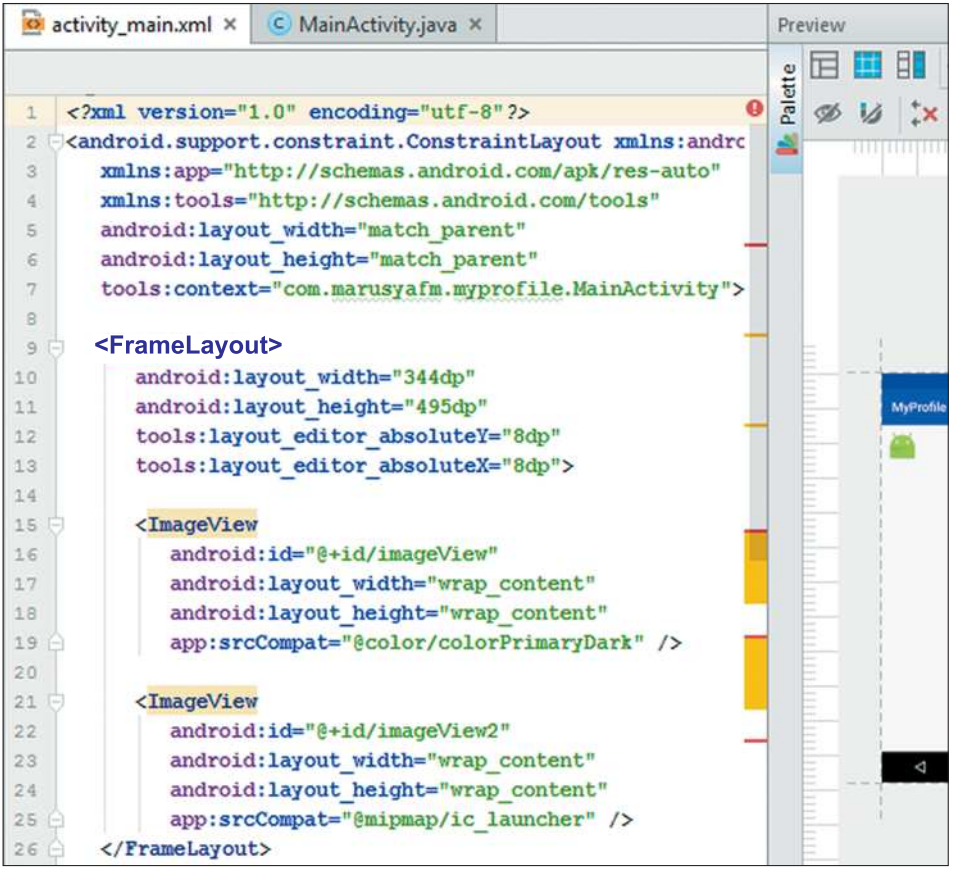
Перетаскиваем во **FrameLayout** еще один элемент **ImageView** — это будет фото профиля. В качестве ресурса для него выбираем графический объект, например стандартный ic_launcher:



В результате дерево компонентов изменилось:



Зададим свойства элементов. Для этого перейдем в режим кода, в котором найдем разметку недавно добавленного нами макета **FrameLayout** (строки кода № 9–26):



Начнем со свойств самого **FrameLayout**. Он должен занимать ширину всего родительского элемента и высоту, скажем, 135dp. Установим **layout_width** и **layout_height** (в строках № 10 и 11) соответственно:

```
android:layout_width=”match_parent”
android:layout_height=”135dp”
```

Свойства **layout_editor_absoluteY** и **layout_editor_absoluteX** (строки № 12 и 13) отвечают за сдвиг элемента по осям *X* и *Y* от начального расположения. Но нам нужно, чтобы макет **FrameLayout** занимал все пространство, без отступов по краям, поэтому удаляем эти строки:

```
~~tools:layout_editor_absoluteY=”8dp”~~
~~tools:layout_editor_absoluteX=”8dp”~~
```

Вместо этого укажем, что левый, правый и верхний края макета должны совпадать с левым, правым и верхним краями родительского элемента. За это отвечают свойства **layout_constraintLeft_toLeftOf** (*макет_привязать левый край_к левому краю*), **layout_constraintRight_toRightOf** (*макет_привязать правый край_к правому краю*) и **layout_constraintTop_toTopOf** (*макет_привязать верхний край_к верхнему краю*):

```
app:layout_constraintLeft_toLeftOf=”parent”
app:layout_constraintRight_toRightOf=”parent”
app:layout_constraintTop_toTopOf=”parent”
```

Затем нужно сделать еще одну важную вещь. Для дальнейшего определения положения **FrameLayout** относительно остальных макетов и элементов нужно сначала задать ему уникальный идентификатор. Сразу после объявления **FrameLayout** (после строки № 9) добавляем строку:

```
android:id=”@+id/frameLayout”
```

Теперь xml-разметка нашего **FrameLayout** выглядит так:

```
9   <FrameLayout
10      android:id="@+id/frameLayout"
11      android:layout_width="match_parent"
12      android:layout_height="135dp"
13      app:layout_constraintLeft_toLeftOf="parent"
14      app:layout_constraintRight_toRightOf="parent"
15      app:layout_constraintTop_toTopOf="parent">
```

Настала очередь изображений.

Первое изображение — это фото обложки, так как оно было добавлено первым:

```
17      <ImageView
18          android:id="@+id/imageView"
19          android:layout_width="wrap_content"
20          android:layout_height="wrap_content"
21          app:srcCompat="@color/colorPrimaryDark" />
```

Фоновое изображение должно занимать все выделенное пространство (в данном случае принять размер макета-родителя). Поэтому в строках № 19 и 20 устанавливаем для **layout_width** и **layout_height** значение match_parent:

```
android:layout_width=”match_parent”
android:layout_height=”match_parent”
```

Второе изображение — фото нашего профиля.

```
24      <ImageView
25          android:id="@+id/imageView2"
26          android:layout_width="wrap_content"
27          android:layout_height="wrap_content"
28          app:srcCompat="@mipmap/ic_launcher" />
29  </FrameLayout>
```

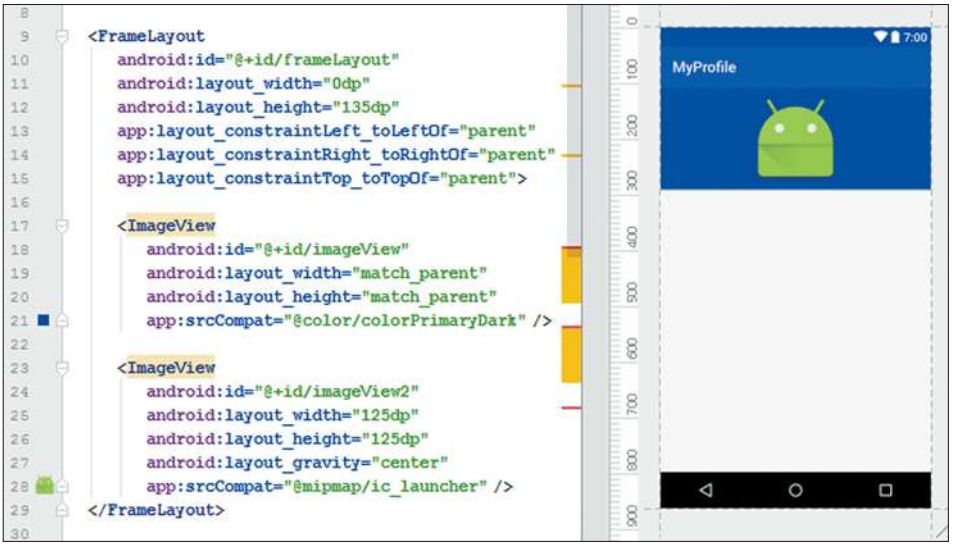
Установим его ширину и высоту 125dp:

```
android:layout_width=”125dp”
android:layout_height=”125dp”
```

Чтобы изображение находилось по центру, добавим свойство **layout_gravity** (*притяжение*) и установим для него значение center (*по центру*):

```
android:layout_gravity=”center”
```

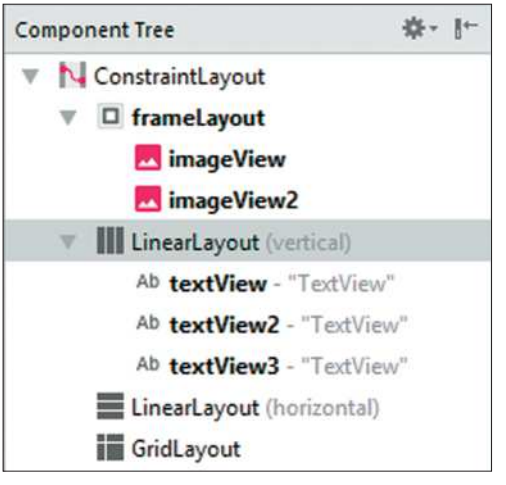
Все изменения мгновенно отображаются в области предварительного просмотра:



2.1.2. LinearLayout (vertical) — линейное расположение элементов по вертикали

В вертикальном **LinearLayout** элементы располагаются один за другим по вертикали, как в простом списке. В нашем приложении это блок персональной информации.

Вернемся в режим дизайна и перетащим из палитры элементов в макет **LinearLayout** (vertical) три элемента **TextView:**



Снова перейдем в режим кода. В строках № 31–55 находится xml-разметка **LinearLayout.**

```
<LinearLayout
    android:layout_width="0dp"
    android:layout_height="0dp"
    android:orientation="vertical"
    tools:layout_editor_absoluteY="8dp"
    tools:layout_editor_absoluteX="8dp">

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

    <TextView
        android:id="@+id/textView2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="TextView" />

    <TextView
        android:id="@+id/textView3"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="TextView" />
</LinearLayout>
```

В первую очередь зададим макету уникальный идентификатор:

android:id="@+id/linearLayout"

Изменим высоту макета на 80dp (теперь это строка № 34):

android:layout_height="80dp"

Удалим сдвиги по осям X и Y:

~~tools:layout_editor_absoluteY="8dp"~~

~~tools:layout_editor_absoluteX="8dp"~~

Вместо них определим расположение макета — левый и правый края совпадают с левым и правым краями родительского элемента:

app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"

А вот верхний край не может совпадать с родительским элементом, потому что в верхней части экрана у нас уже расположился **FrameLayout**. Добавим новое свойство **layout_constraintTop_toBottomOf** (*макет_привязать верхний край_к нижнему краю*) и установим ему значение id, которое мы заранее задали **FrameLayout:**

```
app:layout_constraintTop_toBottomOf="@+id/frameLayout"
```

Теперь **LinearLayout** располагается строго под **FrameLayout.**

Но текст не очень хорошо смотрится, если начинается с самого края экрана, без отступов. Поэтому установим отступы по 8dp справа, слева и сверху с помощью свойств **layout_marginLeft** (*макет_запас слева*), **layout_marginRight** (*макет_запас справа*) и **layout_marginTop** (*макет_запас сверху*)**:**

```
android:layout_marginLeft="8dp"
android:layout_marginRight="8dp"
android:layout_marginTop="8dp"
```

На очереди элементы **TextView**, расположенные внутри **LinearLayout** (строки № 43–59)**:**

```
43      <TextView
44          android:id="@+id/textView"
45          android:layout_width="match_parent"
46          android:layout_height="wrap_content"
47          android:text="TextView" />
48
49      <TextView
50          android:id="@+id/textView2"
51          android:layout_width="match_parent"
52          android:layout_height="wrap_content"
53          android:text="TextView" />
54
55      <TextView
56          android:id="@+id/textView3"
57          android:layout_width="match_parent"
58          android:layout_height="wrap_content"
59          android:text="TextView" />
60  </LinearLayout>
```

Для первого **TextView** меняем текст (свойство **text**, строка № 47)**:**

```
android:text="Имя:"
```

и увеличиваем размер шрифта, добавив свойство **textSize** (*размер текста*):

```
android:textSize="20sp"
```

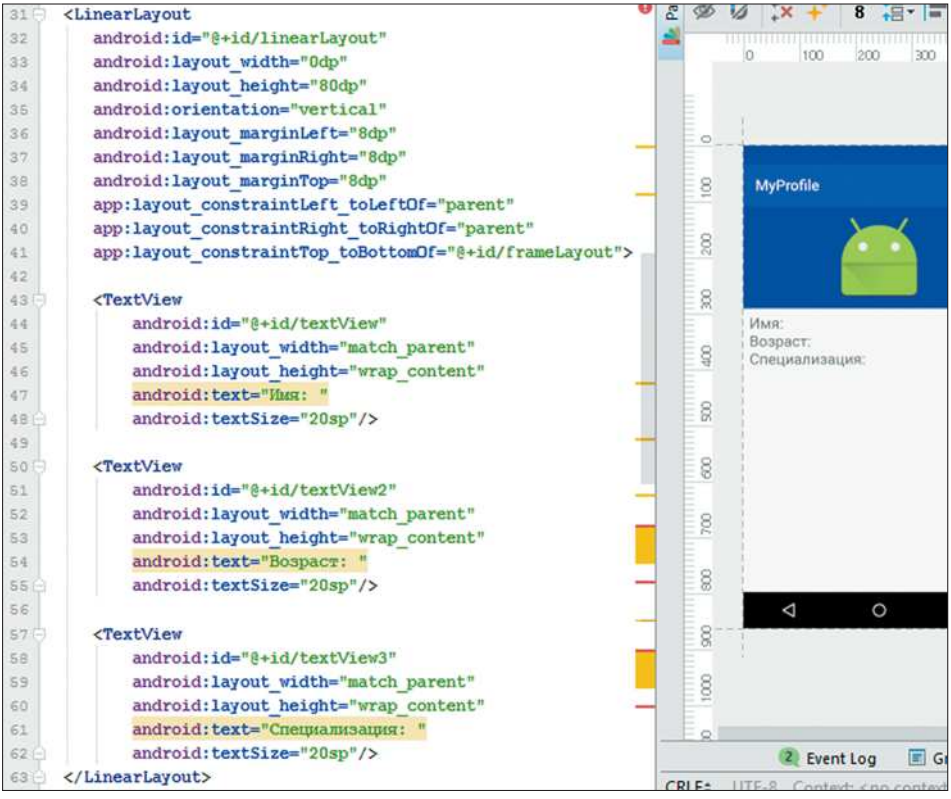
Для второго **TextView** также меняем текст и увеличиваем размер шрифта:

```
android:text="Возраст:"
android:textSize="20sp"
```

Аналогичные изменения введем и для третьего **TextView**:

```
android:text="Специализация:"
android:textSize="20sp"
```

Второй блок нашего профиля готов.

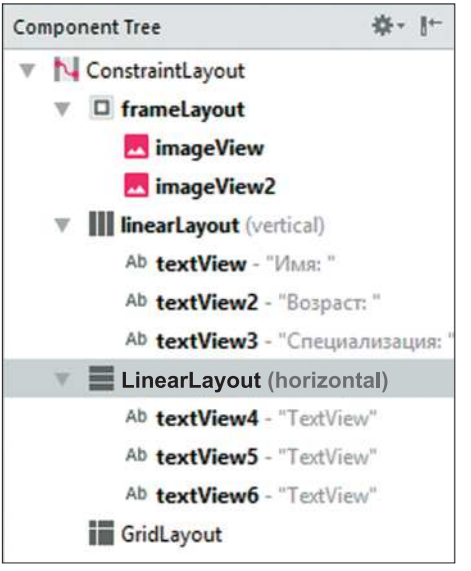



```
<LinearLayout
    android:id="@+id/linearLayout"
    android:layout_width="0dp"
    android:layout_height="80dp"
    android:orientation="vertical"
    android:layout_marginLeft="8dp"
    android:layout_marginRight="8dp"
    android:layout_marginTop="8dp"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/frameLayout">

    <TextView
        android:id="@+id/textView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Имя: "
        android:textSize="20sp"/>

    <TextView
        android:id="@+id/textView2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Возраст: "
        android:textSize="20sp"/>

    <TextView
        android:id="@+id/textView3"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Специализация: "
        android:textSize="20sp"/>
</LinearLayout>
```

2.1.3. LinearLayout (horizontal) — линейное расположение элементов по горизонтали

В горизонтальном **LinearLayout** элементы располагаются по горизонтали, то есть добавляются справа и слева друг от друга. В нашем приложении таким образом будет реализовано горизонтальное меню, как, например, в Facebook и Instagram.

В режиме дизайна перетаскиваем в горизонтальный **LinearLayout** в дереве компонентов три элемента **TextView:**



Вернемся в режим кода. Разметка горизонтального **LinearLayout** — строки № 65–92.

```
65  <LinearLayout
66      android:layout_width="0dp"
67      android:layout_height="0dp"
68      android:orientation="horizontal"
69      tools:layout_editor_absoluteY="8dp"
70      tools:layout_editor_absoluteX="8dp">
71
72      <TextView
73          android:id="@+id/textView4"
74          android:layout_width="wrap_content"
75          android:layout_height="wrap_content"
76          android:layout_weight="1"
77          android:text="TextView" />
78
79      <TextView
80          android:id="@+id/textView5"
81          android:layout_width="wrap_content"
82          android:layout_height="wrap_content"
83          android:layout_weight="1"
84          android:text="TextView" />
```

```
85
86      <TextView
87          android:id="@+id/textView6"
88          android:layout_width="wrap_content"
89          android:layout_height="wrap_content"
90          android:layout_weight="1"
91          android:text="TextView" />
92  </LinearLayout>
```

И снова начинаем с разметки самого макета.

Добавляем макету уникальный идентификатор:

```
android:id="@+id/linearLayout2"
```

Устанавливаем ширину родительского элемента и высоту 30dp:

```
android:layout_width="match_parent"
android:layout_height="30dp"
```

Удаляем сдвиг по осям *X* и *Y*.

```
tools:layout_editor_absoluteY="8dp"
tools:layout_editor_absoluteX="8dp"
```

Вместо этого устанавливаем расположение: правый и левый края совпадают с краями родительского элемента, а верхний край совпадает с нижним краем вертикального **LinearLayout** (то есть находится под ним). Добавляем слева, справа и сверху отступы по 8dp:

```
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toBottomOf="@+id/linearLayout"
android:layout_marginLeft="8dp"
android:layout_marginRight="8dp"
android:layout_marginTop="8dp"
```

Для первого элемента **TextView:**

1) устанавливаем ширину match_parent:

```
android:layout_width="match_parent"
```

2) меняем текст:

```
android:text="Фото"
```

3) устанавливаем размер шрифта 18sp:

```
android:textSize="18sp"
```

4) располагаем элемент по центру:

```
android:gravity="center"
```

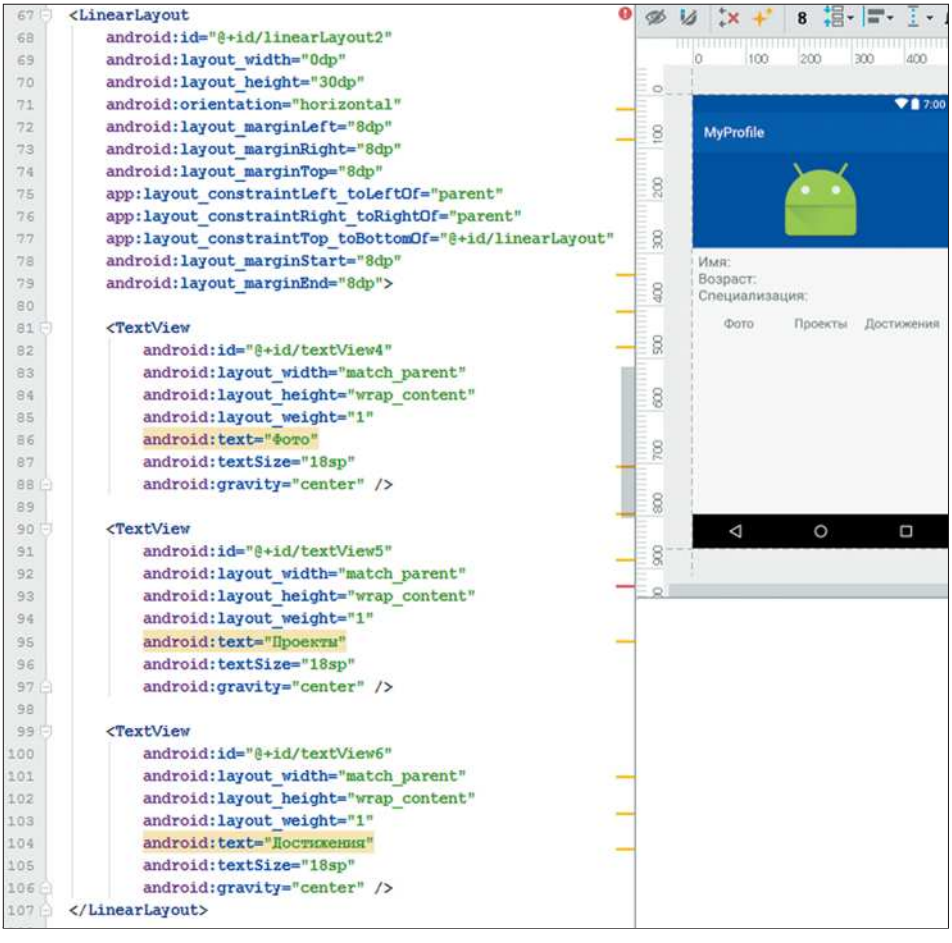
Поскольку элементы в макете **LinearLayout** располагаются только в линейном порядке, для них может быть использован один интересный атрибут — **layout_weight** (*макет_вес*). Вес элемента — это что-то похожее на значимость или приоритет (в ин-

форматике вес имеют ребра графа). Если для элемента установить вес 0, он будет обладать «самым низким приоритетом» и занимать ровно столько пространства, сколько ему нужно для отображения, не больше. Если в то же время другому элементу установить вес 2, он будет иметь «самый высокий приоритет» и займет все оставшееся место в **LinearLayout**. А если для всех элементов установить одинаковый вес 1, то они будут занимать равные части макета.

Последний вариант нам подходит — три элемента **TextView** должны быть одинаковыми по размеру. Поэтому в строке № 81 выставляем для **TextView** вес 1:

```
android:layout_weight="1"
```

Аналогичные изменения необходимо внести для двух оставшихся **TextView**, установив для них текст Проекты и Достижения соответственно:

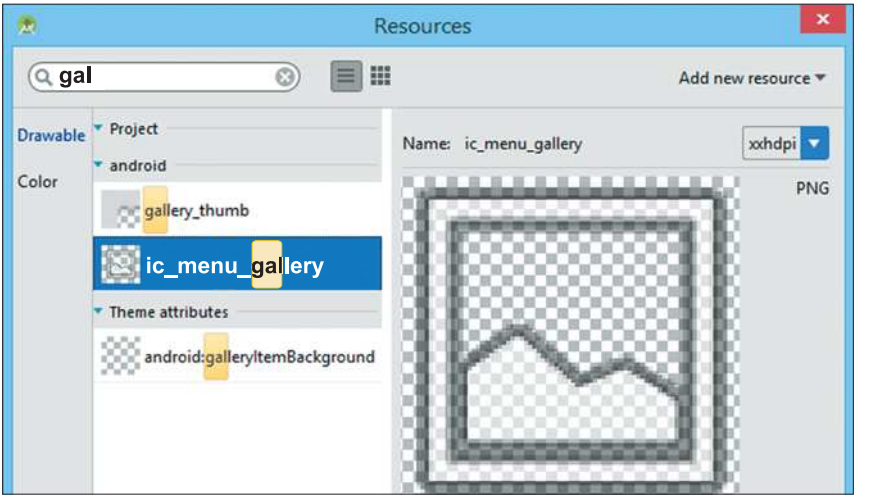


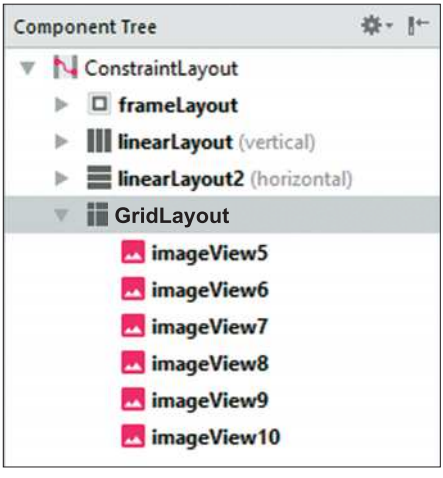
2.1.4. GridLayout — сеточное расположение элементов

Элементы в макете **GridLayout** располагаются в прямоугольной сетке. Она образуется набором тонких линий, разделяющих макет на ячейки, и похожа на таблицу со столбцами и строками.

В нашем приложении в таком «табличном» виде будут представлены фотографии (по аналогии со страницами профиля в Facebook и Instagram).

Снова переводим **activity_main.xml** в режим дизайна и перетаскиваем в **GridLayout** шесть элементов **ImageView**. В качестве ресурса для них выбираем графический объект из коллекции **Android Studio**, например ic_menu_gallery (проще всего воспользоваться поиском).





Возвращаемся в режим кода. Разметка макета **GridLayout** расположена в строках № 109–150.

```
<GridLayout
    android:layout_width="0dp"
    android:layout_height="0dp"
    tools:layout_editor_absoluteY="8dp"
    tools:layout_editor_absoluteX="8dp">

    <ImageView
        android:id="@+id/imageView5"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:srcCompat="@android:drawable/ic_menu_gallery"/>

    <ImageView
        android:id="@+id/imageView6"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:srcCompat="@android:drawable/ic_menu_gallery"/>

    <ImageView
        android:id="@+id/imageView7"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:srcCompat="@android:drawable/ic_menu_gallery"/>

    <ImageView
        android:id="@+id/imageView8"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:srcCompat="@android:drawable/ic_menu_gallery"/>

    <ImageView
        android:id="@+id/imageView9"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:srcCompat="@android:drawable/ic_menu_gallery"/>

    <ImageView
        android:id="@+id/imageView10"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:srcCompat="@android:drawable/ic_menu_gallery"/>
</GridLayout>
```

Начнем традиционно с атрибутов самого макета.

Добавляем ему идентификатор:

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

Устанавливаем длину 250dp и ширину match_parent:

```
android:layout_width="match_parent"
android:layout_height="250dp"
```

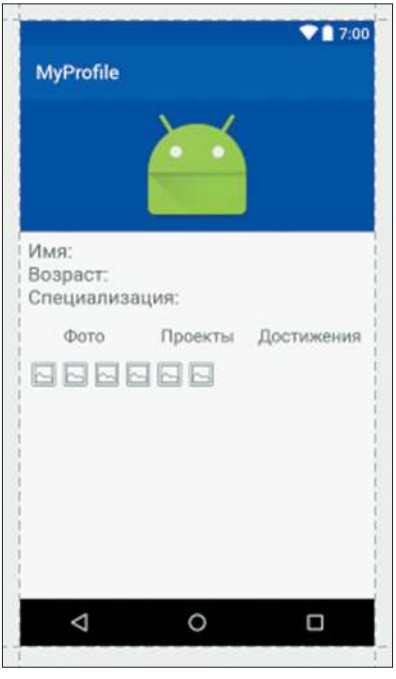
Удаляем сдвиг по осям *X* и *Y*:

```
tools:layout_editor_absoluteY="8dp"
tools:layout_editor_absoluteX="8dp"
```

Устанавливаем расположение — под горизонтальным **LinearLayout**, левый, правый и нижний края совпадают с краями родительского элемента; отступы справа, слева и сверху 8dp:

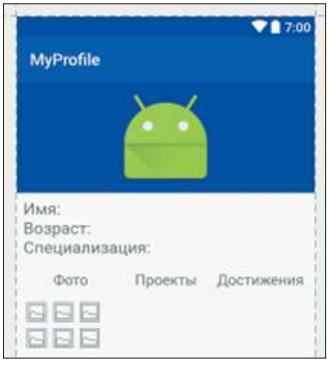
```
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toBottomOf="@+id/linearLayout2"
android:layout_marginLeft="8dp"
android:layout_marginTop="8dp"
android:layout_marginRight="8dp"
```

В предварительном просмотре можем заметить не очень удачное расположение элементов — все шесть изображений в один ряд.



За расположение элементов в **GridLayout** отвечают параметры **rowCount** (*количество строк*) и **columnCount** (*количество столбцов*). Для того чтобы наши изображения располагались, например, по три в ряд, «таблица» должна иметь три столбца. Добавим в описание **GridLayout** соответствующий атрибут:

```
android:columnCount="3"
```

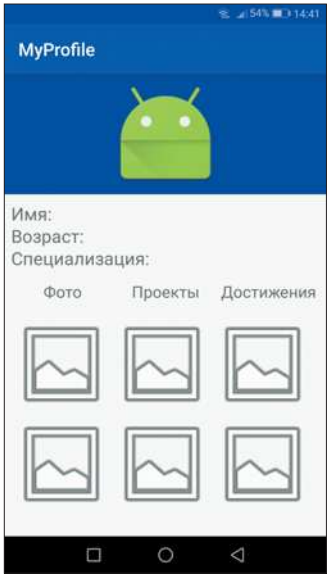


Осталось задать изображениям подходящий размер.

Для каждого из шести **ImageView** устанавливаем ширину и высоту по 112dp:

```
android:layout_width="112dp"
android:layout_height="112dp"
```

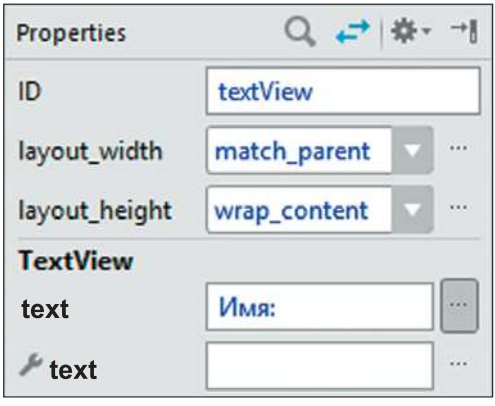
Наш профиль готов. Запускаем приложение, смотрим результат. Получился универсальный шаблон, который нужно наполнить информацией.



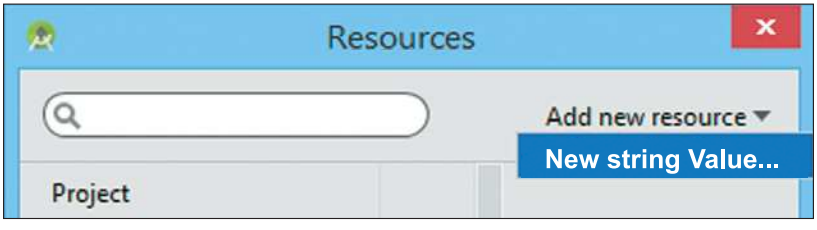
2.1.5. Строковые ресурсы

Грамотные программисты для всех текстов, цветов, стилей и звуков используют **строковые ресурсы**, чтобы иметь возможность обращаться к ним многократно.

Для того чтобы создать строковый ресурс, например для текстового содержания элемента **TextView**, нужно найти в свойствах элемента свойство **text** и нажать на кнопку «...» рядом с ним:

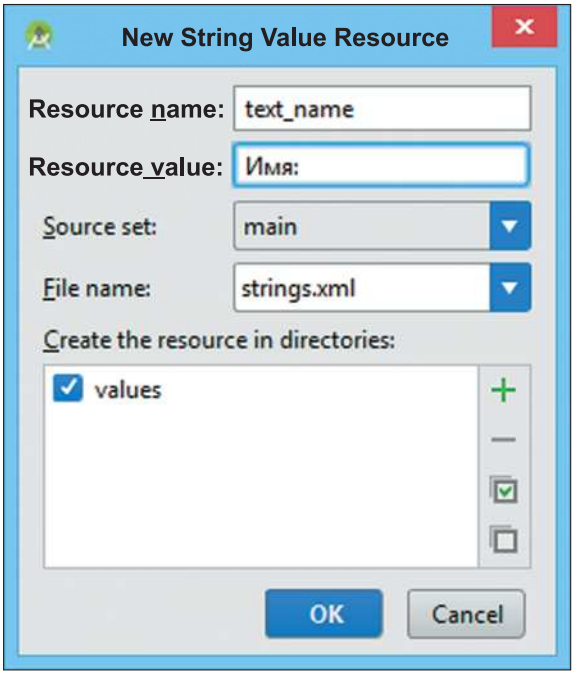


В открывшемся окне ресурсов выбрать **Add new resource** (*добавить новый ресурс*) → **New string Value** (*новое строковое значение*):

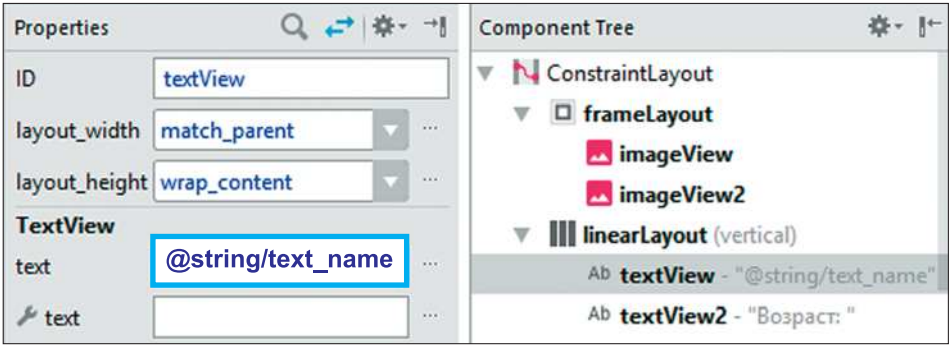


Откроется окно добавления нового строкового значения ресурса, в котором нужно заполнить первые два пункта:

1) **Resource name** (*имя ресурса*) — идентификатор, по которому ресурс будет вызываться;
2) **Resource value** (*значение ресурса*) — текст, который мы хотим отображать в интерфейсе.

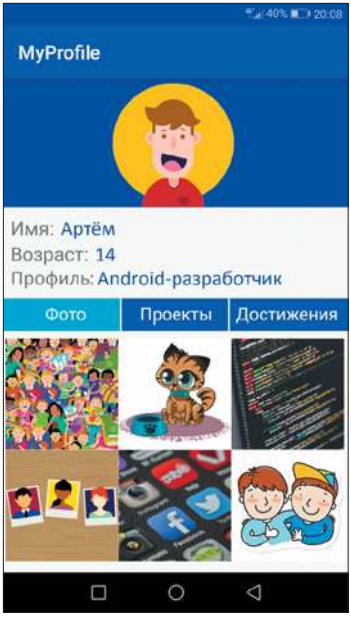


Теперь в свойствах, xml-разметке, дереве компонентов и так далее рядом с этим **TextView** вместо текстового содержания отображается его ресурс:



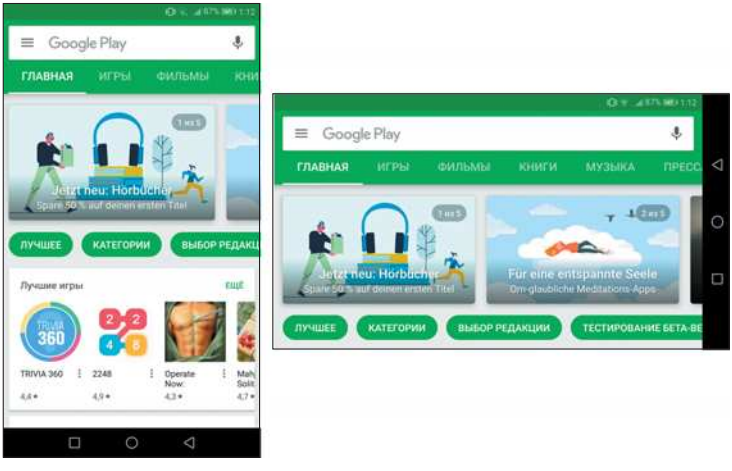
Задания

1. Добавить ресурсы к изображениям.
2. Добавить строковые ресурсы для текстов всех **TextView**.
3. Заполнить поля информацией и добавить еще один макет с блоком информации по своему усмотрению.



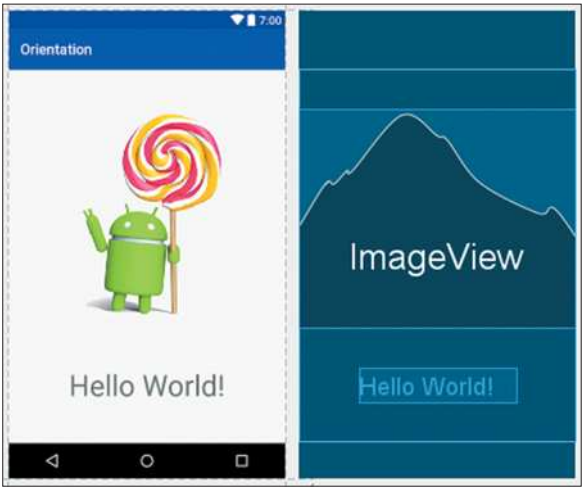
2.2. Ориентация экрана

В зависимости от положения смартфона в руках пользователя большинство приложений может функционировать в двух вариантах ориентации экрана — **портретная (вертикальная)** ориентация и **ландшафтная (горизонтальная)** ориентация:



Создадим новый проект Orientation. Все настройки оставим по умолчанию, для **MainActivity** выбираем тип Empty Activity.

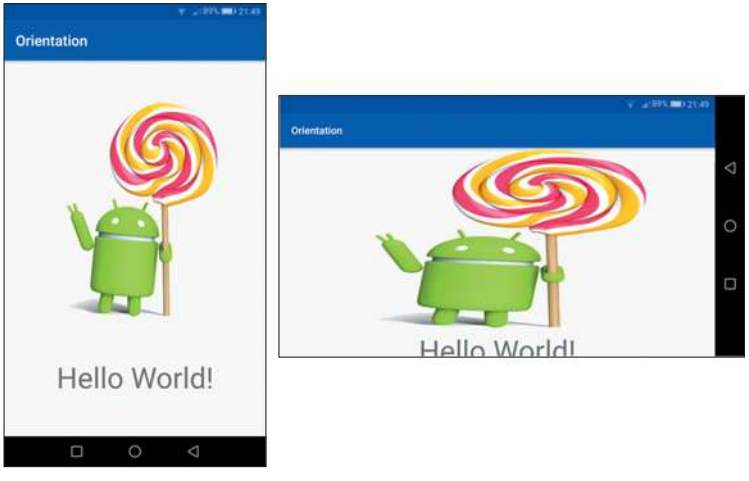
Откроем файл **activity_main.xml** в режиме дизайна и к уже имеющемуся элементу **TextView** добавим новый элемент — изображение **ImageView** (можно выбрать любое изображение из библиотеки ресурсов или загрузить собственное):



Тексту устанавливаем размер 40sp (свойство **textSize**), а изображению — ширину match_parent (свойство **layout_width**) и высоту wrap_content (свойство **layout_height**). Оба элемента центрируем по вертикали и горизонтали (**Center Horizontally** и **Center Vertically**).

Запускаем приложение.

В портретной ориентации приложение выглядит так же, как и на макете, но стоит повернуть смартфон, и все меняется:

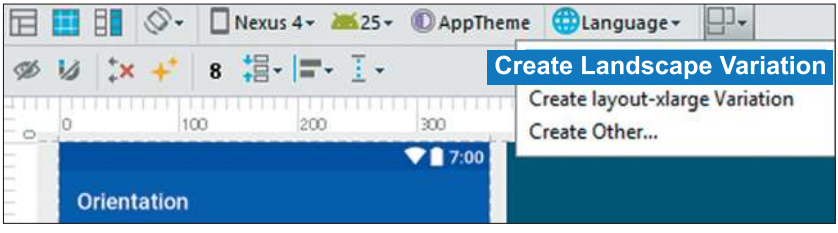


Контент перестал помещаться на экране, изображение растянулось в ширину, но при этом даже нет возможности прокрутки

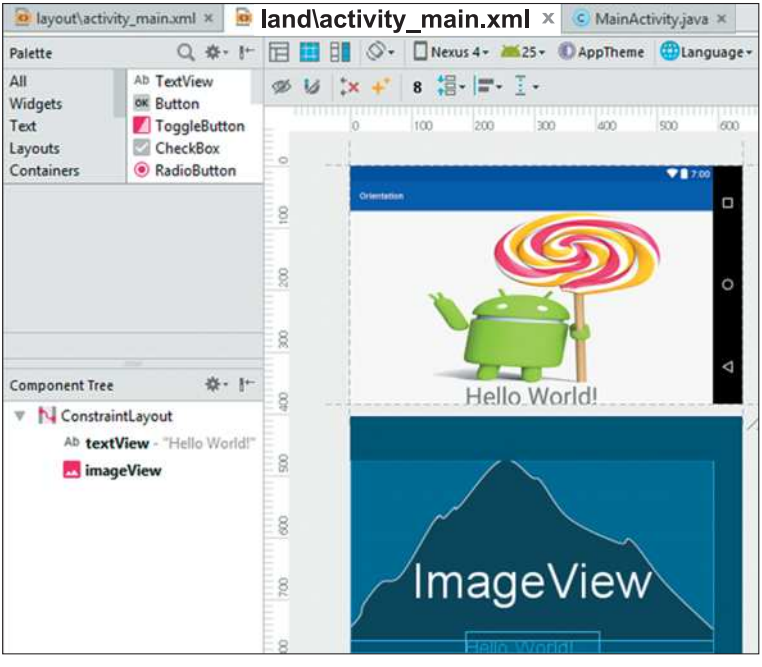
экрана. Это значит, что при создании этого приложения мы не обеспечили реализацию такого важного свойства, как **адаптивность** — способность интерфейса корректно отображаться на различных устройствах и динамически подстраиваться под изменение ориентации и параметры конкретного устройства.

Для того чтобы это исправить, нам нужен еще один вариант xml-разметки для этого же **activity_main**, в котором будет предусмотрено все для корректного отображения приложения в ландшафтной ориентации. Можно написать его вручную, но в **Android Studio** уже имеется такая функция.

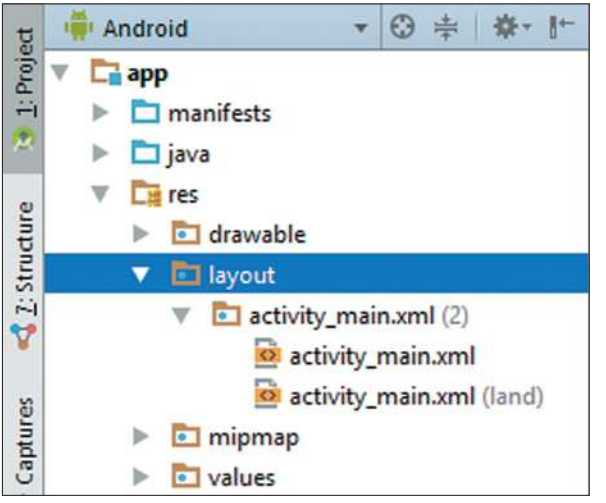
На панели быстрого доступа над областью макета находим пункт **Layout Variants** (*варианты размещения*) и в нем выбираем первый подпункт — **Create Landscape Variation** (*создать горизонтальную вариацию*):



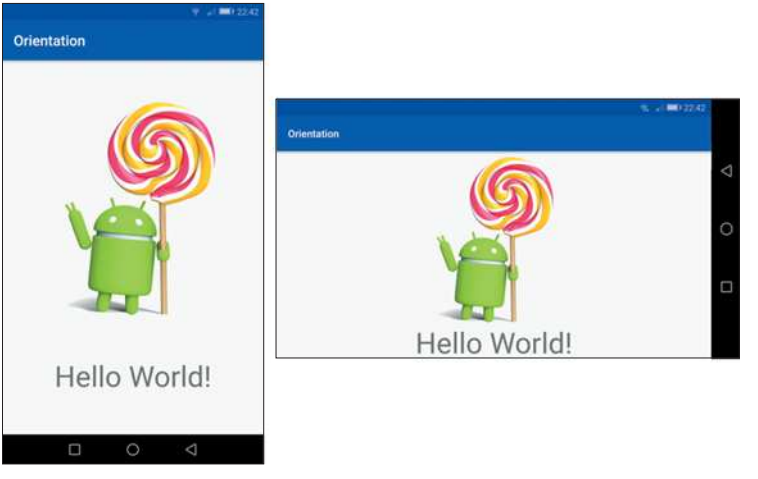
Создается и открывается новый xml-файл.



По сути, **Android Studio** скопировала наш файл **activity_main.xml**, изменив ориентацию на ландшафтную. В структуре проекта новый файл лежит в той же папке, но обозначается пометкой **(land)**:



Теперь можно «привести в порядок» контент для ландшафтного отображения. В режиме дизайна вручную изменяем размеры и расположение элементов таким образом, чтобы они помещались на экране, и снова центрируем их по вертикали и горизонтали. Запускаем приложение:



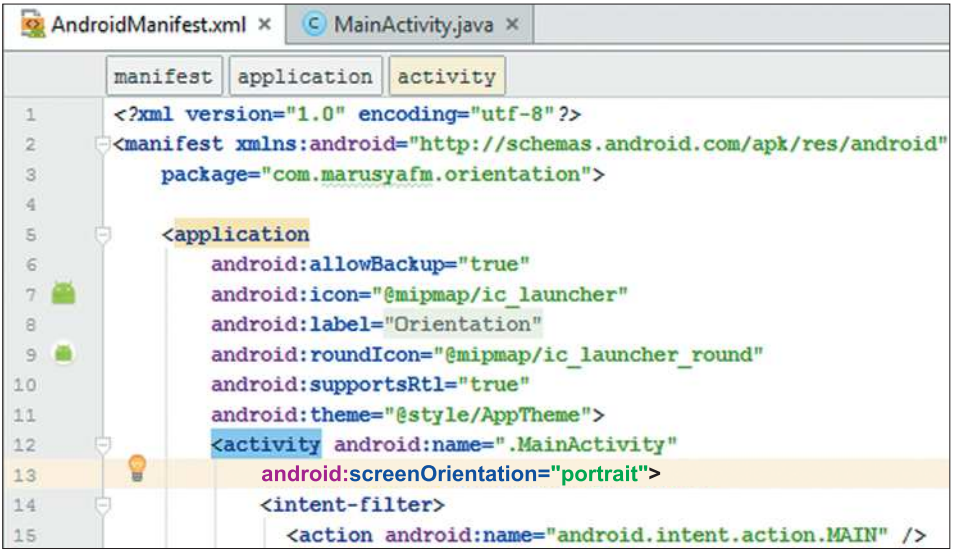
Теперь наш контент выглядит одинаково при любой ориентации экрана.

Задания

Бывают случаи, когда с приложением удобнее работать только в одном варианте ориентации. В **Android Studio** есть возможность запретить смену ориентации, причем как для всего приложения, так и для отдельных его частей.

1. Добавить в файл манифеста **AndroidManifest.xml** атрибут **screenOrientation** (*ориентация экрана*), который разрешит приложению работать только в одном варианте ориентации (строка № 13). Например, установить для него значение portrait для запрета принятия ландшафтной ориентации:

   android:screenOrientation="portrait">



2. Изменить эту строку так, чтобы приложение работало только в ландшафтной (горизонтальной) ориентации.

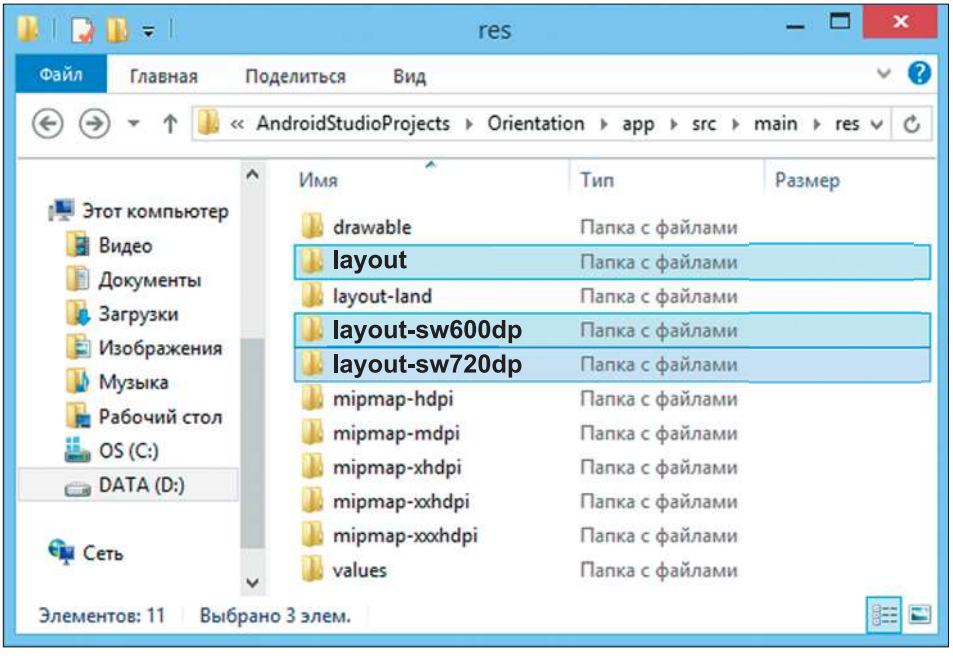
2.3. Разработка приложений для планшетов

Итак, с сохранением корректности отображения контента при изменении ориентации экрана мы разобрались. Теперь запустим это же приложение (**Orientation**), например, на планшете. Можно это сделать на реальном планшете или создать новое виртуальное устройство с помощью модуля **AVD Manager**.

Видим, что на устройстве с диагональю 10 дюймов приложение снова смотрится не так, как на макете в режиме дизайна, значит, интерфейс нашего приложения еще нельзя считать адаптивным.

Чтобы это исправить, нам снова понадобятся изменения в структуре проекта. Можно добавлять новые папки в проект прямо в среде **Android Studio**, а можно это делать через Проводник, что, как правило, быстрее.

В папке проекта **res** (...\AndroidStudioProjects\Orientation\app\src\main\res) находим папку **layout** и дважды копируем ее, присваивая новым папкам имена layout-sw600dp и layout-sw720dp:

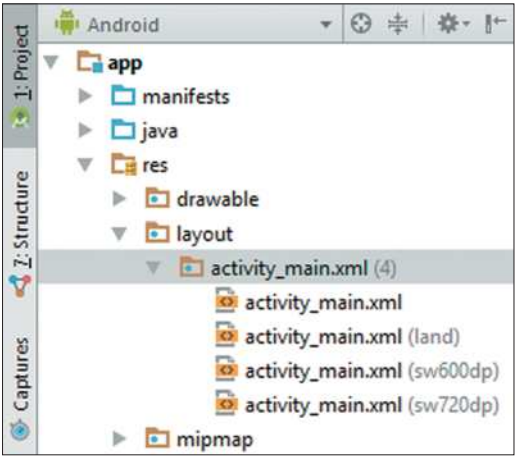


Что такое sw600dp и sw720dp?

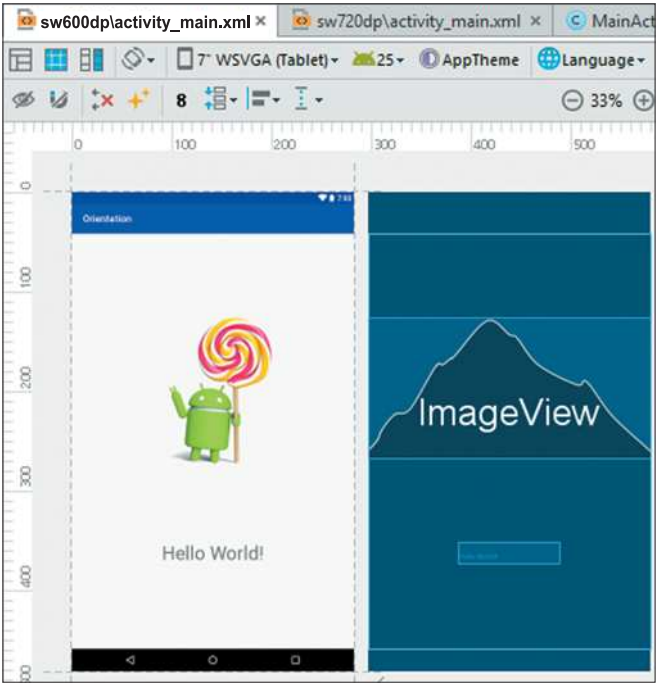
Sw (Smallest-width — *наименьшая ширина*) — это квалификатор, который определяет экраны с заданной минимальной шириной; 600dp — это ширина 7-дюймового экрана, 720dp — ширина 10-дюймового. То есть в папках **layout-sw600dp** и **layout-sw720dp** размещаются файлы разметки для «средних» и «больших» экранов. При запуске приложения на устройстве квалификатор опре-

деляет ширину экрана и отображает разметку, соответствующую параметрам устройства.

Запускаем проект в **Android Studio**. В структуре проекта видим изменения — появились файлы с пометками **sw600dp** и **sw720dp** (так же как в предыдущем уроке появился файл с пометкой **land** для ландшафтной ориентации):



Открываем файл для sw600dp и на макете видим, как наше приложение смотрится на 7-дюймовом экране:



Редактируем макет: устанавливаем для текста размер 70sp, а для изображения — ширину match_parent и подбираем высоту. Центрируем объекты по горизонтали и вертикали. Готово. Запускаем приложение — теперь на 7-дюймовом экране планшета контент смотрится так же, как и на смартфоне.

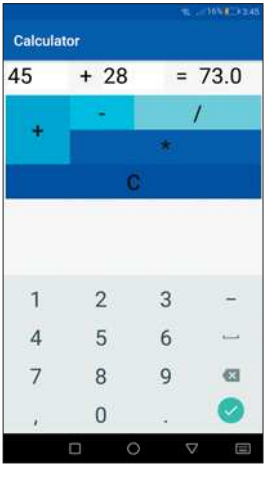
Теперь наш интерфейс может считаться действительно адаптивным. В будущем нужно стараться делать адаптивными интерфейсы всех наших приложений. Но, как несложно догадаться, копировать вариации экрана нужно в последнюю очередь. Так в них придется вносить наименьшее количество изменений.

Задания

1. Внести аналогичные изменения для sw720dp, чтобы контент корректно отображался на 10-дюймовых экранах.
2. Разрешить приложению снова отображаться во всех вариантах ориентации — удалить соответствующую строку из файла манифеста.

2.4. Приложение «Калькулятор»

Для того чтобы обобщить и закрепить все изученное в этой главе, создадим полноценное работающее приложение — калькулятор.



Концепция приложения: простой непрограммируемый калькулятор. Два поля для ввода чисел и одно для вывода результата. Панель с кнопками для проведения различных арифметических операций.

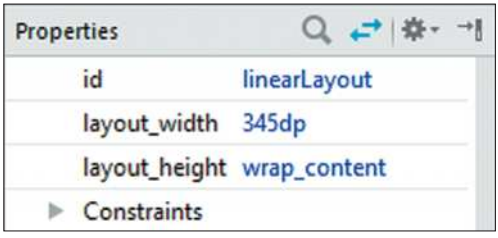
При нажатии на поля ввода появляется числовая клавиатура. При выборе операции автоматически происходит расчет. Кнопка «C» очищает все поля.

Выполнив все описанные ниже шаги, мы получим настоящий «готовый продукт» — собственное приложение, которым смело можно поделиться.

Шаг 1. Создать новый Android Studio проект с именем Calculator и одной Empty Activity.

Шаг 2. Реализовать область вычислений.

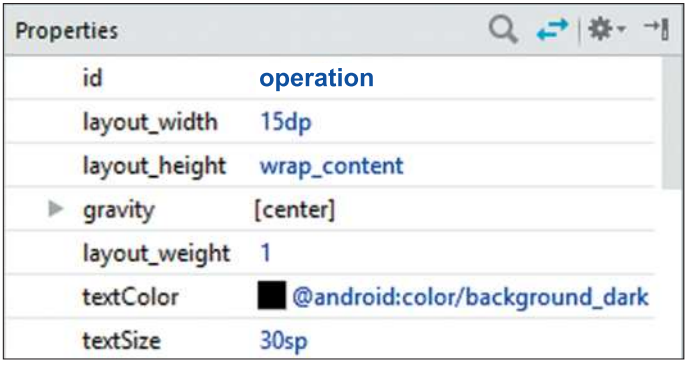
Для этого добавить на экран горизонтальный **LinearLayout**. С помощью панели свойств задать ему идентификатор linearLayout, ширину 345dp и высоту wrap_content:



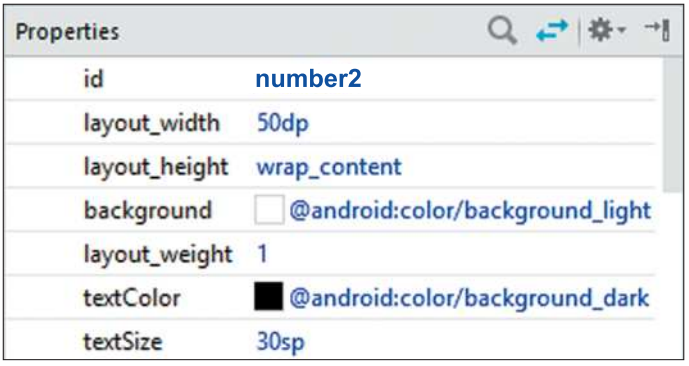
Шаг 3. В добавленном макете **LinearLayout** разместить пять элементов **TextView**:

1) **number1** (*число 1*) — для ввода первого числа:

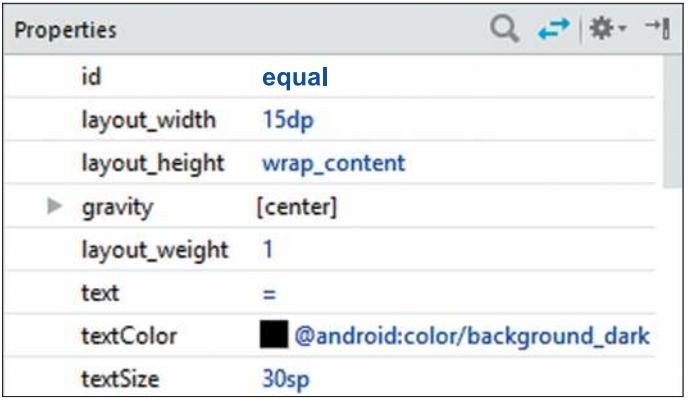
2) **operation** (*операция*) — для назначения операции (умножение, деление, сложение, вычитание):



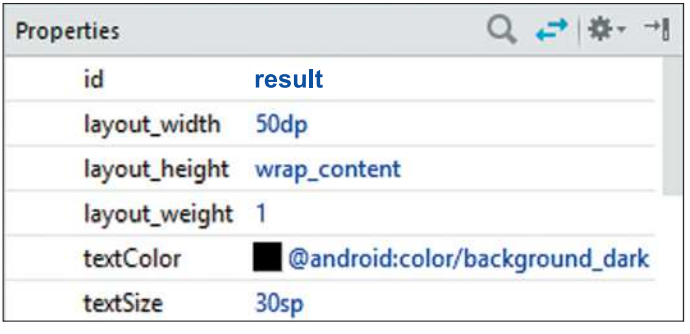
3) **number2** (*число 2*) — для ввода второго числа:



4) **equal** (*равно*) — для вывода символа «=» перед результатом:



5) **result** (*результат*) — для вывода результатов вычислений:



Шаг 4. Изменить элемент **TextView** для числа 1 и числа 2 таким образом, чтобы в них можно было вводить текст.

Перейти в режим кода. Найти элементы и изменить их названия с **TextView** на **EditText**. В конце описания элементов доба-

вить атрибут **inputType** (*тип ввода*) со значением numberSigned (строка № 26).

```
18      <EditText
19          android:id="@+id/number1"
20          android:layout_width="50dp"
21          android:layout_height="wrap_content"
22          android:layout_weight="1"
23          android:background="@android:color/background_light"
24          android:textColor="@android:color/background_dark"
25          android:textSize="30sp"
26          android:inputType="numberSigned"/>
27
```

Этот атрибут отвечает за ввод данных в элемент, то есть какая клавиатура будет отображаться при заполнении элемента. Если не указывать этот атрибут, то отобразится клавиатура, выбранная по умолчанию пользователем для всего смартфона, если numberDecimal — клавиатура только с цифрами, если же numberPassword — клавиатура для ввода числового пароля, которая к тому же заменяет вводимые цифры точками. Мы выбрали **numberSigned** — числовую клавиатуру с возможностью ввода десятичного разделителя (точки или запятой) и минуса, то есть отрицательных и дробных чисел, — то, что нужно для калькулятора.

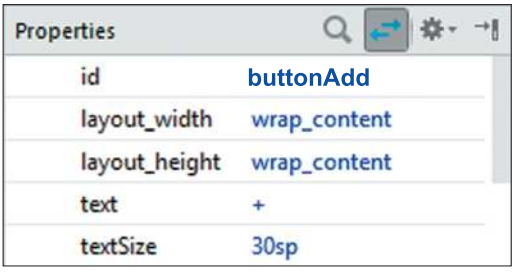
Шаг 5. Реализовать кнопки для выбора операции.

Добавить на экран макет **GridLayout**, с помощью панели свойств задать ему идентификатор gridLayout, ширину 350dp, высоту wrap_content и количество столбцов 3 (свойство **columnCount**):

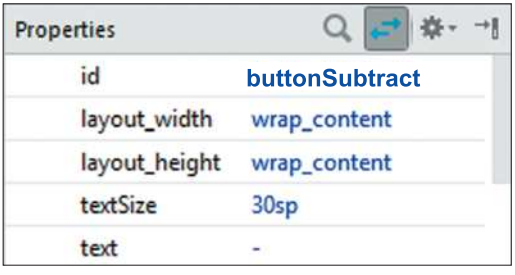
Центрировать оба макета (**LinearLayout** и **GridLayout**) по горизонтали.

Шаг 6. Разместить внутри **GridLayout** пять кнопок:

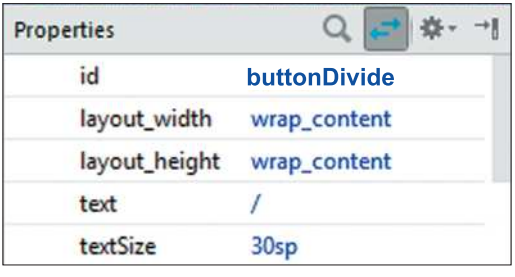
1) **buttonAdd** (*кнопка Сложить*) — кнопка «+»:



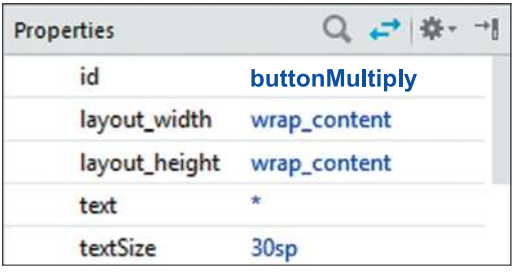
2) **buttonSubtract** (*кнопка Вычесть*) — кнопка «–»:



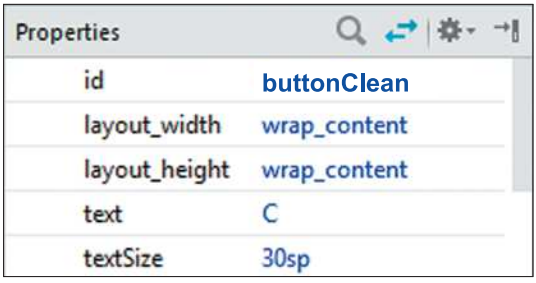
3) **buttonDivide** (*кнопка Разделить*) — кнопка «/»:



4) **buttonMultiply** (*кнопка Умножить*) — кнопка «*»:



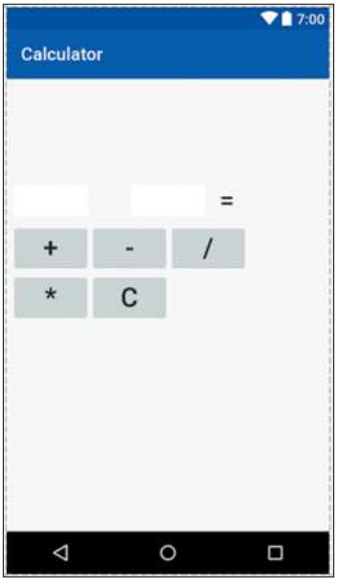
5) **buttonClean** (*кнопка Очистить*) — кнопка «C»:



Шаг 7. Изменить размер и расположение кнопок.

Перейдем в режим кода.

Сейчас наши кнопки расположены не самым удобным и презентабельным образом:



Это поправимо. У макета **GridLayout** есть функция объединения ячеек: элементы в нем могут занимать больше одной ячейки, как при работе с таблицами в текстовых и табличных процессорах (MS Office Word, MS Office Excel и так далее) или как для атрибута span в HTML. За это отвечают три атрибута: **layout_columnSpan** (*макет_диапазон столбцов*) для определения числа столбцов, занимаемых элементом (то есть размер по горизонтали), **layout_rowSpan** (*макет_диапазон строк*) — число строк, **layout_gravity** (*макет_притяжение*) для «растягивания» элемента на выбранное количество ячеек.

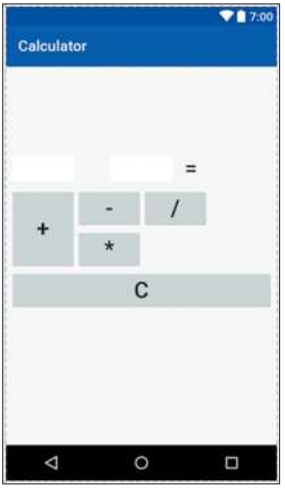
Отведем кнопке «+» две ячейки по вертикали. Для этого найдем ее описание в xml-разметке и добавим в него строки:

```
android:layout_rowSpan="2"
android:layout_gravity="fill_vertical"
```

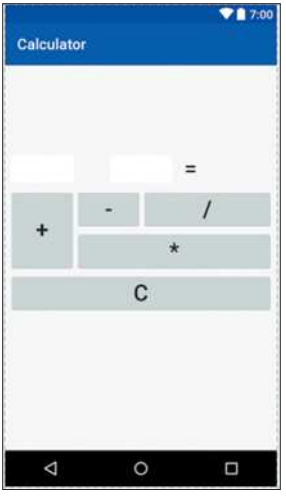
Первой строкой мы определили новую высоту кнопки — две строки, а второй — растянули кнопку на эту высоту по вертикали (**fill_vertical** — *заполнение по вертикали*).

А кнопку «C», например, растянем на три столбца по горизонтали:

```
android:layout_columnSpan="3"
android:layout_gravity="fill_horizontal"
```



Теперь по аналогии нужно изменить разметку оставшихся трех кнопок так, чтобы они расположились следующим образом:



С разметкой мы закончили, осталось написать Java-код, чтобы наш калькулятор работал.

Шаг 8. Объявить переменные и определить элементы.

Откроем файл **MainActivity.java**:

activity_main.xml × MainActivity.java ×

```
1  package com.marusyafm.calculator;
2
3  import android.support.v7.app.AppCompatActivity;
4  import android.os.Bundle;
5
6  public class MainActivity extends AppCompatActivity {
7
8      @Override
9      protected void onCreate(Bundle savedInstanceState) {
10         super.onCreate(savedInstanceState);
11         setContentView(R.layout.activity_main);
12     }
13 }
```

В классе **MainActivity** объявим переменные для всех наших кнопок и текстовых полей (не забывая импортировать предлагаемые **Android Studio** библиотеки):

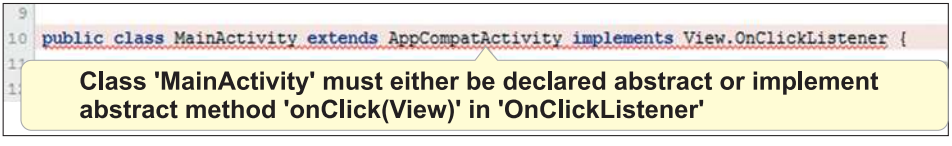
```
3  import android.support.v7.app.AppCompatActivity;
4  import android.os.Bundle;
5  import android.widget.Button;
6  import android.widget.EditText;
7  import android.widget.TextView;
8
9  public class MainActivity extends AppCompatActivity {
10
11     private Button buttonAdd, buttonSubtract, buttonDivide,
12                    buttonMultiply, buttonClean;
13     private TextView operation, result;
14     private EditText number1, number2;
15
16     @Override
17     protected void onCreate(Bundle savedInstanceState) {
```

Затем, как мы уже умеем, с помощью метода **findViewById()** определим элементы в методе **onCreate()**:

```
15    @Override
16    protected void onCreate(Bundle savedInstanceState) {
17        super.onCreate(savedInstanceState);
18        setContentView(R.layout.activity_main);
19
20        buttonAdd = (Button) findViewById(R.id.buttonAdd);
21        // ... остальные кнопки
22        operation = (TextView) findViewById(R.id.operation);
23        number1 = (EditText) findViewById(R.id.number1);
24        // ... остальные текстовые элементы
25    }
26 }
```

Шаг 9. Написать обработчики событий нажатия кнопок.

Для добавления обработчиков событий нажатия кнопок нужно сначала подключить «прослушиватели» нажатий. Для кнопок нажатие является сигналом для начала выполнения заданных действий. Для этого в название класса **MainActivity** допишем **implements View.OnClickListener** (*реализующий прослушиватель нажатий*).



Android Studio подчеркивает измененную строку и сообщает нам, что **Class 'MainActivity' must either be declared abstract or implement abstract method 'onClick(View)' in 'OnClick Listener'** (*класс MainActivity должен быть либо объявлен абстрактным, либо реализовывать абстрактный метод onClick( ) в OnClickListener*). Следуя его подсказкам, нажимаем **Alt+Enter** и выбираем первый пункт — **Implement methods** (*имплементировать, реализовать методы*).

В конце программы появится метод **onClick()**, в котором мы сможем описать необходимые действия при нажатии любой кнопки.

Примечание

Java-классы бывают обычными и абстрактными.

Абстрактные классы в первую очередь существуют для того, чтобы предоставлять свои методы другим классам в качестве шаблона. Абстрактные классы могут содержать как обычные, так и абстрактные методы. Абстрактные методы — методы без прописанного внутри них функционала.

Таким образом, ***имплементируя*** (реализуя) методы абстрактного класса, мы получаем что-то вроде черновика для написания кода и доступ к использованию всех возможностей абстрактного класса внутри нашего класса.

Прежде чем что-то прописать в методе **onClick()**, нужно установить «прослушиватели» нажатия для всех кнопок в методе **onCreate()**. Если кнопок много, они расположены в одном блоке и выполняют однотипные действия, то прописывать отдельный метод **onClick()** для каждой кнопки не всегда удобно. В таких случаях прописывают только один **onClick()**, в который помещают инструкции для всех кнопок. А определением, какая именно кнопка была нажата, занимаются прослушиватели.

Прослушиватели нажатия кнопок объявляются в методе **onCreate()** с помощью метода **setOnClickListener()** (*установить прослушиватель нажатия*). В качестве параметра передадим методу ключевое слово this (*этот*), указывающее на текущий объект данного класса:

```
16    @Override
17    protected void onCreate(Bundle savedInstanceState) {
18        super.onCreate(savedInstanceState);
19        setContentView(R.layout.activity_main);
20
21        buttonAdd = (Button) findViewById(R.id.buttonAdd);
22        buttonSubtract = (Button) findViewById(R.id.buttonSubtract);
23        buttonDivide = (Button) findViewById(R.id.buttonDivide);
24        buttonMultiply = (Button) findViewById(R.id.buttonMultiply);
25        buttonClean = (Button) findViewById(R.id.buttonClean);
26        operation = (TextView) findViewById(R.id.operation);
27        result = (TextView) findViewById(R.id.result);
28        number1 = (EditText) findViewById(R.id.number1);
29        number2 = (EditText) findViewById(R.id.number2);
30
31        buttonAdd.setOnClickListener(this);
32        buttonSubtract.setOnClickListener(this);
33        // ... остальные кнопки
34    }
```

Затем в методе **onClick()** объявляем переменные, необходимые нам для расчетов: num1 для первого числа, num2 для второго числа и res для результата вычислений. Используем тип данных **float**, потому что работаем не только с целыми значениями:

```
38      @Override
39      public void onClick(View v) {
40          float num1 = 0;
41          float num2 = 0;
42          float res = 0;
43      }
44  }
```

Действия для кнопок установим через оператор выбора **switch**.

Switch (*переключатель*) — условный оператор языка Java, который в отличие от операторов **if** и **if-else** применяется для списка известных значений, а также предусматривает ситуацию по умолчанию **(default)**. Действия для каждого возможного значения описываются в своем операторе **case** (*случай*). Принятое на вход выражение сравнивается со значением каждого **case**, и, если совпадение найдено, выполняется команда именно этого **case**. В конце каждого **case** обычно ставится команда **break** для «выхода» из оператора **switch** и перехода к выполнению кода, стоящего после него. Для случаев, не описанных ни в одном **case**, выполняются действия, описанные в **default** (по умолчанию).

Но в операторе **switch** могут использоваться только типы данных int, char, byte, short, enum и String. Использование типа float приведет к ошибке выполнения. Поэтому, прежде чем писать **switch**, добавим в метод **onClick()** строки, в которых получим значения из num1 и num2 и «переведем» их из типа данных float в String (строки № 44–45):

```
38      @Override
39      public void onClick(View v) {
40          float num1 = 0;
41          float num2 = 0;
42          float res = 0;
43
44          num1 = Float.parseFloat(number1.getText().toString());
45          num2 = Float.parseFloat(number2.getText().toString());
46      }
47  }
```

Метод **getText()** (*получить текст*) «извлекает» текстовое содержание **TextView** (то, что прописано в свойстве **text** элемента).

Затем метод **toString()** (*в строку*) определяет этот извлеченный текст как строку. А метод **parseFloat()** (*распознать тип данных float*) «распознает» в получившейся строке число с плавающей точкой.

Вернемся к оператору **switch**. На его вход в качестве выражения для сравнения подается идентификатор нажатой кнопки (**getId()** — *получить идентификатор*). Значения для оператора **case** — идентификаторы всех имеющихся у нас кнопок: buttonAdd, buttonSubtract, buttonDivide, buttonMultiply и buttonClean. И поскольку мы опишем все возможные значения, для действия по умолчанию остается только выход из оператора:

```
44        num1 = Float.parseFloat(number1.getText().toString());
45        num2 = Float.parseFloat(number2.getText().toString());
46
47        switch (v.getId()){
48            case R.id.buttonAdd:
49                operation.setText("+");
50                res = num1 + num2;
51                break;
52            // ... по аналогии для "-", "*" и "/"
53            case R.id.buttonClean:
54                number1.setText("");
55                operation.setText("");
56                number2.setText("");
57                result.setText("");
58                break;
59            default:
60                break;
61        }
62    }
63 }
```

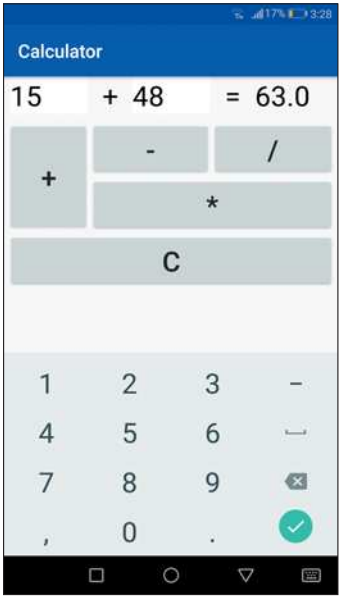
Для кнопки сложения в **TextView operation** устанавливаем текстовое значение «+» (строка № 49), а переменной res присваиваем значение суммы num1 и num2 (строка № 50). По аналогии опишем значения **case** для кнопок вычитания, умножения и деления. Для кнопки очистки всем текстовым полям устанавливаем пустое значение (строки № 54–57).

В самом конце метода **onClick()**, после оператора **switch**, добавляем строку вывода результата:

```
result.setText(res+"");
```

Готово!

Шаг 10. Собрать проект, запустить приложение и проверить работу нашего калькулятора через эмулятор или на реальном устройстве.



Задания

1. Доработать приложение до корректного отображения при горизонтальной ориентации экрана.
2. Доработать приложение до корректного отображения на больших экранах.
3. Добавить функцию возведения в степень.
4. Проработать дизайн приложения.

Итоги главы 2

В главе 2 мы освоили одну из основополагающих вещей Android-разработки — основы проектирования интерфейса. Если не уделить этому вопросу должного внимания, каким бы грамотным ни был код — элементы могут начать наслаиваться друг на друга или выстраиваться не в том порядке. В результате с ними будет гораздо сложнее работать на этапе разработки приложения. Само же приложение не будет корректно отображаться на устройствах с разной диагональю экрана и не будет реагировать на изменение ориентации экрана.

Мы научились:

1. Грамотно выстраивать структуру интерфейса — группировать элементы с помощью макетов.
2. Создавать и использовать строковые ресурсы.
3. Делать наши приложения адаптивными — создавать горизонтальную вариацию ориентации (и устанавливать ориентацию по умолчанию), а также добавлять вариации sw600dp и sw720dp для корректного отображения на самых разных смартфонах и планшетах.

Глава 3. Способы оповещения пользователей

3.1. Всплывающие сообщения

Toast Notification (*уведомление о тосте, тост-уведомление*) или просто **Toast** — всплывающее сообщение для обеспечения простейшей обратной связи с пользователем.

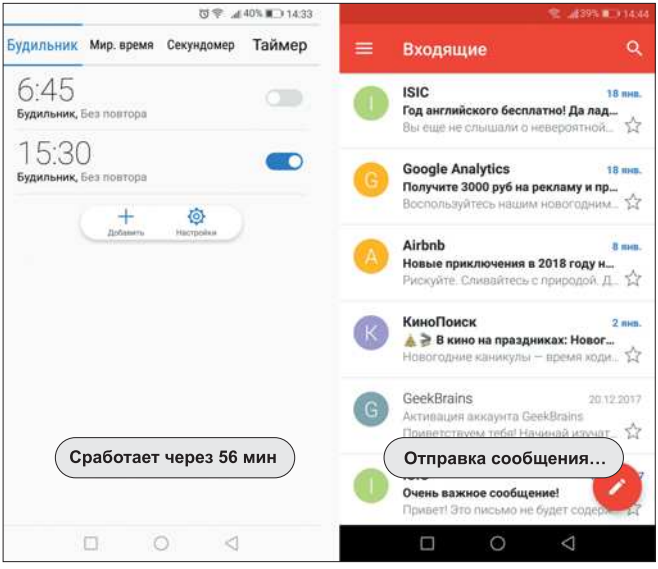
В основе идеи лежат именно тосты — поджаренные кусочки хлеба, которые «выпрыгивают» из тостера и тем самым «уведомляют» нас о своей готовности.

Выглядит **Toast** как бледно-серое сообщение, которое появляется поверх приложения. Оно несет чисто информирующую функцию и не требует непосредственной реакции пользователя. Например, «Папка пуста», «Будильник сработает через 56 мин», «Отправка сообщения...», «Сообщение не было отправлено», «Изменения успешно сохранены».

Сообщение **Toast** принимает размер содержащегося в нем текста, не содержит кнопок и других элементов управления, отображается поверх приложения (но никогда не «мешает» работе с ним) и автоматически исчезает по истечении заданного времени (по тайм-ауту). Основная идея таких сообщений — уведомлять пользователя о чем-либо, не мешая ему дальше пользоваться нашим приложением.

Важно!

Если от пользователя ожидается подтверждение или любое другое действие, Toast для этого не подходит, нужно использовать уведомления или диалоговые окна, которые мы также рассмотрим в этой главе.

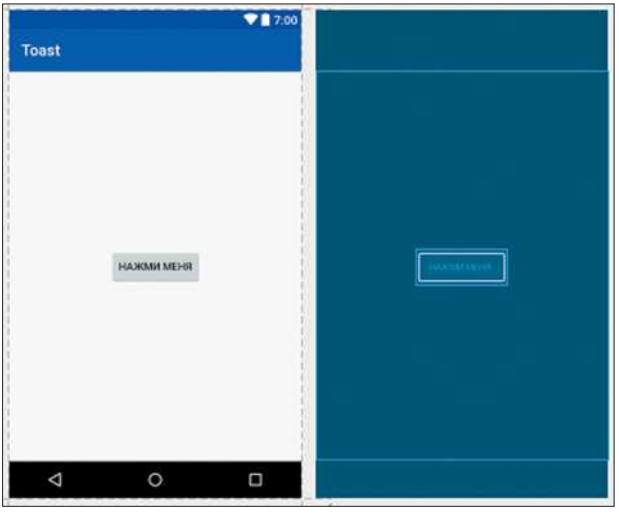


Перейдем к практике.

Концепция мини-приложения: на главном экране размещена кнопка, при нажатии на которую отображается всплывающее сообщение.

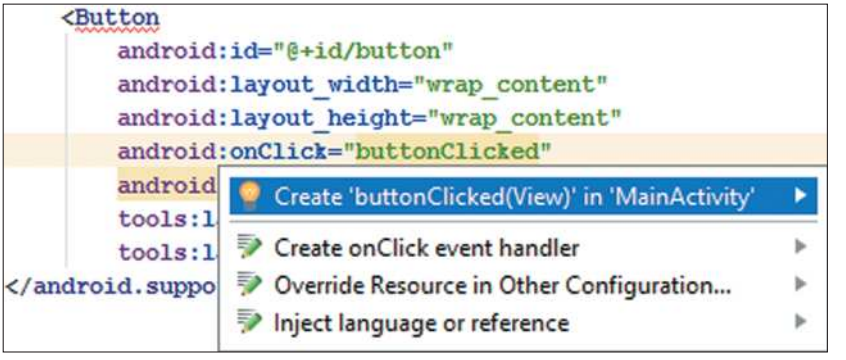
Создадим новый проект Android Studio с именем Toast, параметрами по умолчанию и одной Empty Activity.

В файле xml-разметки **activity_main.xml** в режиме дизайна удалим элемент **TextView** Hello World. Вместо него разместим на экране кнопку, установим для нее текст Нажми меня и отцентрируем кнопку по вертикали и горизонтали:



С помощью панели свойств добавим buttonClicked — обработчик события нажатия кнопки **onClick.**

Перейдем в режим кода и с помощью подсказок **Android Studio** создадим метод **buttonClicked()** в главном Java-классе **MainActivity:**



Теперь нужно добавить само всплывающее сообщение. Однако **Toast** — это не графический объект, его не найти в палитре элементов, его можно создать и вызвать только с помощью Java-кода.

В открывшемся файле **MainActivity.java** находим только что созданный нами метод **buttonClicked()** и, следуя подсказкам **Android Studio**, добавляем следующую строку:

```
Toast myToast = Toast.makeText(getApplicationContext(), “Ура!
Всплывающие сообщения работают“, Toast.LENGTH_LONG);
```

В ней мы объявляем всплывающее сообщение типа Toast с именем myToast (**Android Studio** предложит импортировать библиотеку **android.widget.Toast**, что обязательно нужно сделать) и с помощью метода **makeText()** («*сделать*», *то есть задать текст*) задаем его параметры:

- **контекст сообщения** — задается методом **getApplicationContext()** (*получить контекст приложения*). С помощью этого метода наше всплывающее сообщение будет использоваться на уровне всего приложения, то есть отображаться поверх приложения вне зависимости от выполняемых в приложении процессов, но будет закрыто при выходе из приложения. Также могут быть задействованы методы **getBaseContext()** (*получить базовый контекст*) — сообщение будет отображаться даже при сворачивании приложения или выходе из него, и **this** (*этот*) — сообщение отображается только поверх текущего экрана приложения (то есть в рамках работы текущей Activity) и пропадает при переходе к другому экрану;
- **отображаемый текст** (задается в кавычках) — «Ура! Всплывающие сообщения работают»;

- **продолжительность отображения** — Toast.LENGTH_ LONG. Для этого параметра могут быть использованы только две константы: **Toast.LENGTH_LONG** (*долгое отображение*, 3,5 сек) и **Toast.LENGTH_SHORT** (*короткое отображение*, 2 сек).

Чтобы сообщение начало отображаться, обратимся к методу **show()** (*показать*) и добавим вторую строку:

```
myToast.show();
```

activity_main.xml × MainActivity.java ×

```
8  public class MainActivity extends AppCompatActivity {
9
10     @Override
11     protected void onCreate(Bundle savedInstanceState) {
12         super.onCreate(savedInstanceState);
13         setContentView(R.layout.activity_main);
14     }
15
16     public void buttonClicked(View view) {
17         Toast myToast = Toast.makeText(getApplicationContext(),
18                         "Ура! Всплывающие сообщения работают",
19                         Toast.LENGTH_LONG);
20         myToast.show();
21     }
```

Готово! Запускаем приложение:

Убеждаемся, что при нажатии на кнопку внизу экрана появляется наше всплывающее сообщение, что оно не препятствует работе с приложением (например, повторному нажатию кнопки) и исчезает через 3,5 сек.

Дополнительно можно настроить расположение всплывающего сообщения на экране. За это отвечает метод **setGravity()** (*установить «притяжение»*, то есть положение относительно краев экрана). Метод имеет три параметра:

1) **тип стандартного расположения на экране** — по центру (Gravity.CENTER), в нижней части экрана (Gravity.BOTTOM), в верхней части экрана (Gravity.TOP), заполнять все свободное пространство в разных направлениях (Gravity.FILL_HORIZONTAL), центрироваться по любому из направлений (Gravity.CENTER_VERTICAL) и так далее;
2) **смещение по горизонтали** (по оси X) от выбранного расположения — задается числом пикселей dp;
3) **смещение по вертикали** (по оси Y) от выбранного расположения — задается числом пикселей dp.

Расположим наше всплывающее сообщение в верхней части экрана, добавив в метод **buttonClicked()** еще одну строку (№ 20):

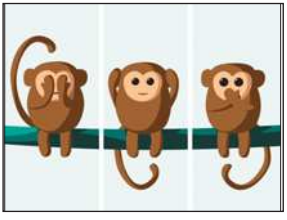
```
myToast.setGravity(Gravity.TOP, 0, 0);
```

```
17  public void buttonClicked(View view) {
18      Toast myToast = Toast.makeText(getApplicationContext(),
19                      "Ура! Всплывающие сообщения работают", Toast.LENGTH_LONG);
20      myToast.setGravity(Gravity.TOP, 0, 0);
21      myToast.show();
22  }
```

По аналогии можно задать любое место появления всплывающих сообщений.

Задания

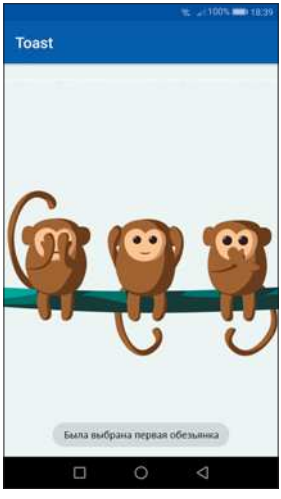
1. Разместить на экране еще две кнопки и поместить их в горизонтальный макет **LinearLayout**.
2. Подобрать изображение с несколькими отдельно стоящими персонажами (например, обезьянки, сидящие на ветке). Любым удобным способом разделить изображение на части, чтобы каждый персонаж оказался в отдельном изображении.



Все три изображения переместить в папку проекта **drawable** и установить в качестве фона (свойство **background**) к расположенным на экране кнопкам.

3. Дописать класс **MainActivity** таким образом, чтобы при нажатии на одну из кнопок всплывающее сообщение информировало пользователя о том, какой именно персонаж был выбран.

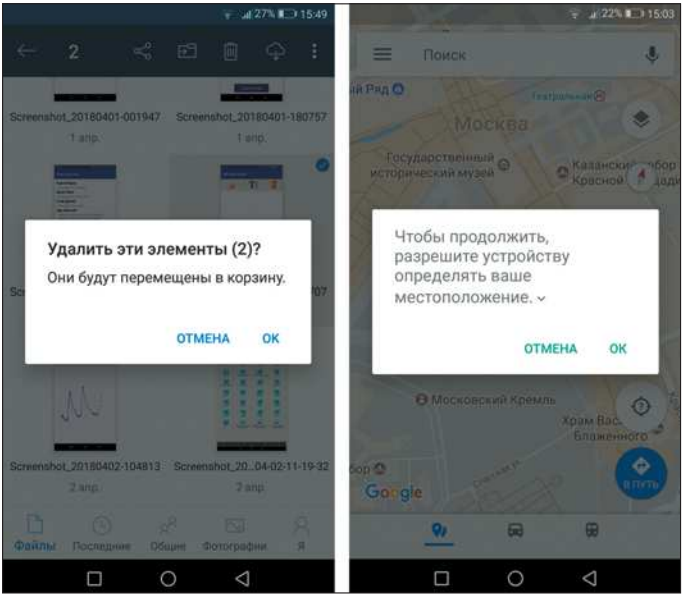
Подсказка: нужно написать обработчик событий для каждой кнопки.



3.2. Диалоговые окна

В случаях когда от пользователя требуется некоторая «реакция», то есть выполнение или подтверждение какого-либо действия, следует применять диалоговые окна.

Диалоговые окна нужны для ввода дополнительной информации или запроса на принятие некоторого решения пользователем. Обычно содержат вопрос и кнопки с предлагаемыми вариантами ответа.



Создадим новый проект Android Studio Dialog.

По центру экрана разместим кнопку с текстом **Закрыть приложение** (не забудем добавить для текста строковый ресурс). В классе **MainActivity** добавим обработчик события нажатия кнопки — метод **onCloseButtonClick()**. В нем объявим наше диалоговое окно (строка № 17), а точнее, его **builder** (*построитель, конструктор, сборщик*), в котором собственно и задаются основные параметры диалогового окна:

```
8  public class MainActivity extends AppCompatActivity {
9
10     @Override
11     protected void onCreate(Bundle savedInstanceState) {
12         super.onCreate(savedInstanceState);
13         setContentView(R.layout.activity_main);
14     }
15
16     public void onCloseButtonClick(View view) {
17         AlertDialog.Builder builder = new AlertDialog.Builder(MainActivity.this);
18
19     }
```

Далее устанавливаем эти параметры:

```
19  public void onCloseButtonClick(View view) {
20      AlertDialog.Builder builder = new AlertDialog.Builder(MainActivity.this);
21      builder.setTitle("Выход из приложения")
22              .setIcon(R.drawable.image_close)
23              .setMessage("Вы уверены, что хотите закрыть приложение?")
24              .setPositiveButton("OK",
25                      new DialogInterface.OnClickListener(){
26                          public void onClick(DialogInterface dialog, int id) {
27                              finish();
28                          }
29                      });
30  }
31  }
```

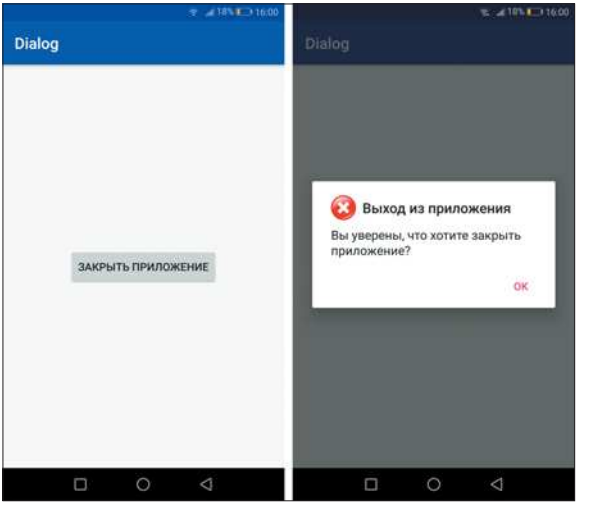
Метод **setTitle()** (*установить заголовок*) устанавливает заголовок диалогового окна, метод **setIcon()** (*установить иконку*) — иконку в заголовке, метод **setMessage()** (*установить сообщение*) — сообщение в области содержимого окна. Метод **setPositiveButton()** (*установить «положительную» кнопку*) добавляет в область действия кнопку, отвечающую за положительный ответ на запрос приложения. В нем прописывается название кнопки, а также обработчик события нажатия на эту кнопку. В данном случае это метод **finish()** (*завершить*), который отвечает за завершение работы приложения.

Теперь осталось только дописать две строки для создания нашего окна и его отображения при нажатии на кнопку:

```
19  public void onCloseButtonClick(View view) {
20      AlertDialog.Builder builder = new AlertDialog.Builder(MainActivity.this);
21      builder.setTitle("Выход из приложения")
22              .setIcon(R.drawable.image_close)
23              .setMessage("Вы уверены, что хотите закрыть приложение?")
24              .setPositiveButton("OK",
25                      new DialogInterface.OnClickListener(){
26                          public void onClick(DialogInterface dialog, int id) {
27                              finish();
28                          }
29                      });
30      AlertDialog alert = builder.create();
31      alert.show();
32  }
```

В строке № 30 методом **create()** (*создать*) мы создаем диалоговое окно с именем alert (*предупреждение*) со всеми параметрами, прописанными в построителе. Следующей строкой № 31 с помощью метода **show()** (*показать*) активируем его отображение.

Запускаем приложение.



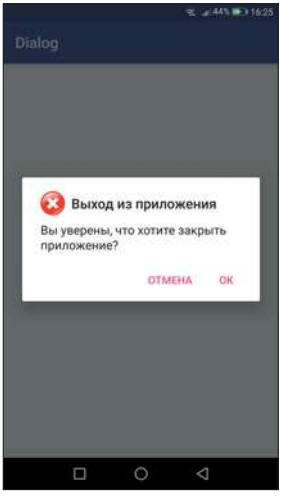
При нажатии кнопки **OK** приложение закрывается, а нажатие в любом месте экрана вокруг окна приводит к закрытию диалогового окна.

Организуем пользователю выбор — добавим кнопку **Отмена**. Окно невозможно будет закрыть, пока выбор не сделан.

Способ добавления «отрицательной» кнопки аналогичен способу добавления «положительной» — метод **setNegativeButton()**, а за невозможность закрытия окна отвечает метод **setCancelable()** (*установить «отменяемость»*):

```
19  public void onCloseButtonClick(View view) {
20      AlertDialog.Builder builder = new AlertDialog.Builder(MainActivity.this);
21      builder.setTitle("Выход из приложения")
22              .setIcon(R.drawable.image_close)
23              .setMessage("Вы уверены, что хотите закрыть приложение?")
24              .setCancelable(false)
25              .setPositiveButton("OK",
26                      new DialogInterface.OnClickListener(){
27                          public void onClick(DialogInterface dialog, int id) {
28                              finish();
29                          }
30                      })
31              .setNegativeButton("Отмена",
32                      new DialogInterface.OnClickListener(){
33                          public void onClick(DialogInterface dialog, int id) {
34                              dialog.cancel();
35                          }
36                      });
37      AlertDialog alert = builder.create();
38      alert.show();
39  }
```

Итак, в диалоговом окне две кнопки: одна закрывает окно, а другая — все приложение, и окно уже не закрывается нажатием на произвольную область экрана.

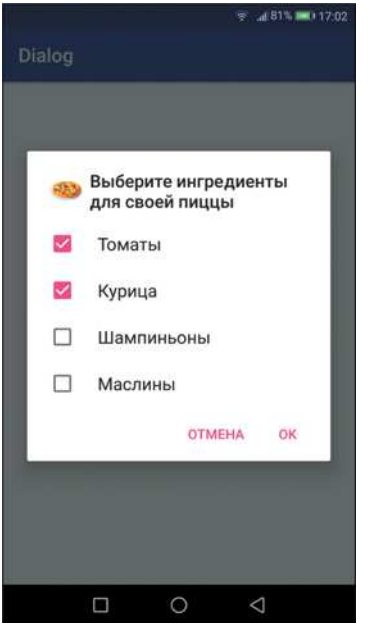


3.2.1. Диалоговые окна с множественным выбором

Диалоговые окна можно организовать также в виде списка или списка с множественным выбором. Для реализации списка нужно добавить в программу два массива (один для хранения значений списка, а второй для хранения состояния каждого пункта списка) и метод **setMultiChoiceItems()** (*установить элементы множественного выбора*):

```
21  public void onCloseButtonClick(View view) {
22      // массив значений для отображаемых в окне строк
23      String[] mIngredients = { "Томаты", "Курица", "Шампиньоны", "Маслины" };
24      // массив для хранения выбора пунктов (по умолчанию ни один пункт не выбран, поэтому false)
25      final boolean[] mSelectedIngredients = { false, false, false, false };
26
27      AlertDialog.Builder builder = new AlertDialog.Builder(MainActivity.this);
28      builder.setTitle("Выберите ингредиенты для своей пиццы")
29          .setIcon(R.drawable.pizza)
30          .setCancelable(false)
31          // множественный выбор
32          .setMultiChoiceItems(mIngredients, null, new DialogInterface.OnMultiChoiceClickListener(){
33              @Override
34              public void onClick(DialogInterface dialog, int which, boolean isChecked){
35                  mSelectedIngredients[which] = isChecked;
36              }
37          })
38          .setPositiveButton("OK",
39                  new DialogInterface.OnClickListener(){
40                      public void onClick(DialogInterface dialog, int id) {
41                          // ...
42                      }
43                  })
```

В результате в диалоговом окне доступен множественный выбор:

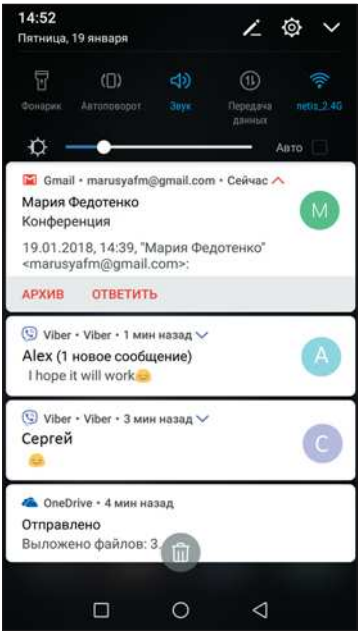


Для управления состоянием каждого чекбокса можно использовать метод **setChecked()** (*установить выбранным*). Чтобы узнать текущее состояние чекбокса, вызывают метод **isChecked()** (*выбран*), как ранее в коде. Далее, в зависимости от логики программы, с помощью конструкции **if-else** прописывают дальнейшие действия.

3.3. Уведомления

Уведомления — это сообщения, которые выводятся за пределами самого приложения для привлечения внимания пользователя. Сначала они отображаются в виде значков в области уведомлений (в верхней части экрана):

Для просмотра полного текста уведомления или его сокращенного варианта (в зависимости от приложения) нужно развернуть свайпом **панель уведомлений**:



Уведомления в обязательном порядке содержат **заголовок, иконку** и **текст.**

Создадим новый проект Notifications со стандартными параметрами. На главном экране разместим кнопку **ImageButton**, подберем для нее в качестве изображения колокольчик и создадим обработчик события нажатия — метод **onClick():**

Перейдем в файл **MainActivity.java** и первым действием объявим переменную для нашего уведомления — его идентификатор NOTIFICATION_ID (идентификатору может быть присвоено любое числовое значение, обычно начинают с единицы):

```
13  public class MainActivity extends AppCompatActivity {
14
15      public static final int NOTIFICATION_ID = 1;
16
17      @Override
18      protected void onCreate(Bundle savedInstanceState) {
19          super.onCreate(savedInstanceState);
20          setContentView(R.layout.activity_main);
21      }
```

Далее объявим построитель уведомления **NotificationCompat.Builder** mBuilder и в нем зададим три обязательных параметра уведомления (строки № 24–28):

1) **setSmallIcon()** (*установить маленькую иконку*) — иконку можно выбрать из библиотеки Android Studio или загрузить свою;

Важно!

К иконкам для уведомлений предъявляются достаточно четкие требования: **изображения иконок должны быть небольшого размера** и **иметь прозрачный фон** (соответственно должны быть сохранены в формате **.png**). Иначе они будут отображаться сплошным темным квадратом или не будут отображаться вовсе.

2) **setContentTitle()** (*установить заголовок контента*);
3) **setContentText()** (*установить текст контента*).

```
23  public void onClick(View view) {
24      NotificationCompat.Builder mBuilder =
25          (NotificationCompat.Builder) new NotificationCompat.Builder(this)
26                  .setSmallIcon(R.drawable.icon123)
27                  .setContentTitle("Колокольчик")
28                  .setContentText("Колокольчик звенит, колокольчик зовет");
29
30  }
```

Для того чтобы при нажатии на уведомление открывалось наше приложение, нужно использовать объект **Intent**.

Intent (*намерение*) — это абстрактное описание одной выполняемой операции, с помощью которого можно запросить выполнение некоторого действия у другой Activity или даже стороннего приложения.

Объявим **Intent** и укажем, что открываться должен главный экран нашего приложения — **MainActivity** (строка № 30).

```
23  public void onClick(View view) {
24      NotificationCompat.Builder mBuilder =
25          (NotificationCompat.Builder) new NotificationCompat.Builder(this)
26                  .setSmallIcon(R.drawable.icon123)
27                  .setContentTitle("Колокольчик")
28                  .setContentText("Колокольчик звенит, колокольчик зовет");
29
30      Intent resultIntent = new Intent(this, MainActivity.class);
31  }
```

Для выстраивания навигации при взаимодействии с уведомлением объявляется объект **TaskStackBuilder** (*построитель стека задач*) stackBuilder — своего рода список, в который будут заноситься задачи по мере их выполнения, в нашем случае — вызовы Activity (строка № 32):

```
30      Intent resultIntent = new Intent(this, MainActivity.class);
31
32      TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
33      stackBuilder.addParentStack(MainActivity.class);
34      stackBuilder.addNextIntent(resultIntent);
35
36      PendingIntent resultPendingIntent =
37          stackBuilder.getPendingIntent(0, PendingIntent.FLAG_UPDATE_CURRENT);
38      mBuilder.setContentIntent(resultPendingIntent);
39      NotificationManager mNotificationManager =
40          (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
41  }
42  }
```

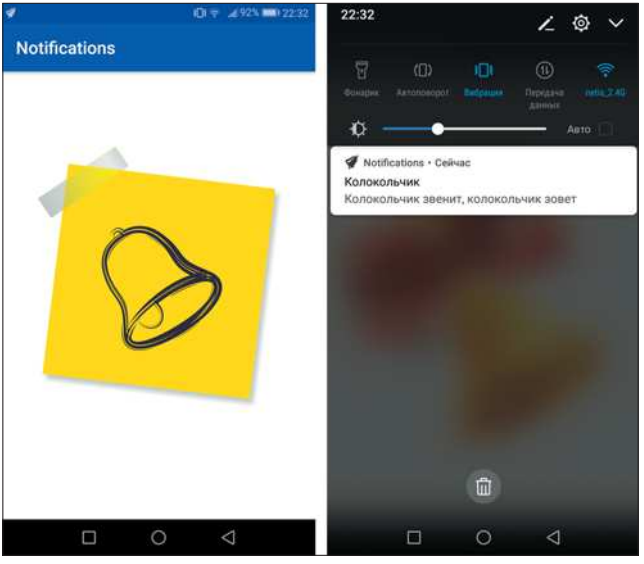
Метод **addParentStack()** (*добавить родительский стек*) устанавливает в качестве родительского стека класс **MainActivity** (строка № 33). Но у нас в приложении сейчас только один экран — **MainActivity**, поэтому он упоминается и как родительский стек, и как **resultIntent** (*результирующее намерение*). Если бы их было несколько, то при нажатии на уведомление мы попадали бы на экран, заданный как **resultIntent**, а при нажатии кнопки «назад» попадали бы на «родительский», то есть домашний, экран.

Объект **PendingIntent** (*в ожидании намерения*) представляет собой некоторую «оболочку» для объекта **Intent** и содержит его описание и определение действий, которые будут над ним выполняться в дальнейшем. В нашем случае, при нажатии на уведомление будет открываться Activity, которая находилась в стеке задач, сформированном в объекте **TaskStackBuilder**, то есть наш единственный экран **MainActivity**.

Для того чтобы «собрать» и отобразить уведомление и иметь возможность обновлять его в дальнейшем, добавим в конец метода **onClick()** еще одну строку:

```
mNotificationManager.notify(NOTIFICATION_ID, mBuilder.build());
```

Запустим приложение:



При нажатии на колокольчик в области уведомлений появляется значок нашего уведомления. При открытии панели видим иконку, название приложения, время уведомления, название и полный текст уведомления. Если нажать на уведомление, открывается главный экран нашего приложения. Но при этом уведомление не пропадает после того, как мы по нему перешли, что по идее должно происходить, ведь мы уже ознакомились с информацией. Его можно только «смахнуть» или удалить вместе со всеми уведомлениями. А еще, например, если нажать на колокольчик несколько раз, то уведомления будут «накладываться» друг на друга и мы будем постоянно видеть только последнее.

Исправим это и рассмотрим еще некоторые возможности работы с уведомлениями.

3.3.1. Удаление уведомлений

Для того чтобы уведомление удалялось после прочтения пользователем, нужно в построителе уведомления использовать метод **setAutoCancel()** (*установить автоматическое завершение*). При выборе true уведомление будет автоматически закрываться после

прочтения, false — будет продолжать «висеть» в панели уведомлений, например до выполнения какого-либо условия:

```
.setAutoCancel(true)
```

3.3.2. Большая иконка

Уведомления могут также содержать большую иконку. Например, в мессенджерах это обычно фото пользователя, который прислал нам сообщение. За большую иконку в построителе уведомления отвечает метод **setLargeIcon()** (*установить большую иконку*).

Обратим внимание, что изображение для больших иконок запрашивается в модели **bitmap**, а чтобы не менять само изображение, можно «декодировать» его ресурс в нужный формат прямо внутри метода **setLargeIcon()**:

```
.setLargeIcon(BitmapFactory.decodeResource(res,R.drawable.icon123))
```

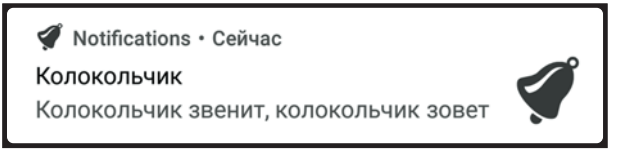
Пояснение

Bitmap (*битовая карта*) — это объект, используемый для работы с изображениями и определяющий их как набор пикселей, каждый из которых описан набором битов.
Изображения, которые мы помещаем в папку ресурсов drawable, через код можно только отобразить, а сформировав программно объект **bitmap**, мы можем совершать над ним различные операции (изменение размеров, кадрирование, «наполнение» некоторым содержимым, например сделанным скриншотом экрана).

При этом для использования ресурсов не забудем объявить их в начале метода **onClick()**:

```
Resources res = this.getResources();
```

Теперь наше уведомление имеет также большую иконку.

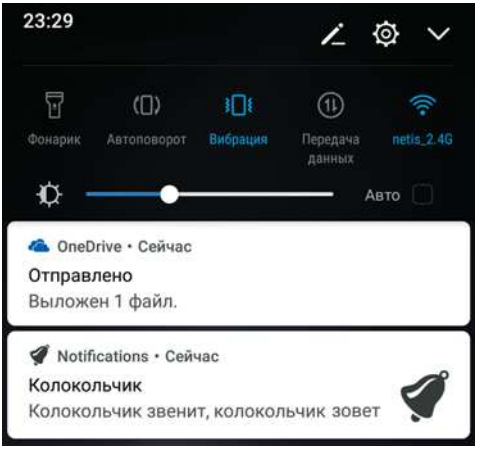


3.3.3. Приоритет уведомлений

Все уведомления имеют свой приоритет, по этой причине некоторые отображаются первыми вне зависимости от времени получения.

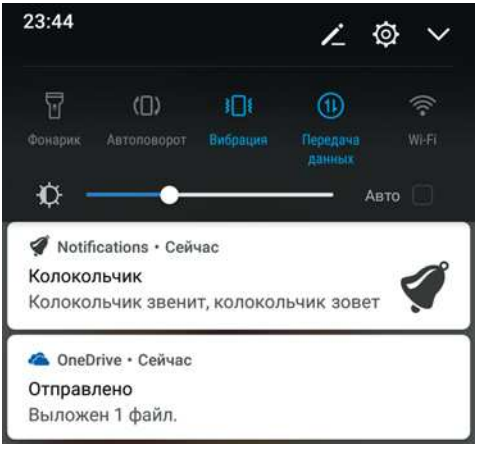
По умолчанию задан средний приоритет, поэтому, скорее всего, наше уведомление будет медленно смещаться в конец списка

уведомлений, уступая уведомлениям различных мессенджеров или файловых менеджеров.



Но если в построителе уведомления вызвать метод **setPriority()** (*установить приоритет*) и задать ему значение PRIORITY_HIGH (*высокий приоритет*), то наше уведомление будет отображаться одним из первых на панели уведомлений:

```
.setPriority(NotificationCompat.PRIORITY_HIGH)
```



3.3.4. Звуковое и световое оповещение

Для более эффективного привлечения внимания пользователя уведомления часто сопровождаются звуковым сигналом, вибрацией и миганием светового индикатора.

Эти события устанавливаются методом **setDefault()**: значение DEFAULT_LIGHTS активирует световой индикатор, DEFAULT_

SOUND устанавливает звуковое сопровождение (стандартный рингтон уведомлений), а DEFAULT_VIBRATE отвечает за вибрацию. Можно совсем не использовать эти параметры, а можно использовать сразу все (DEFAULT_ALL):

```
.setDefaults(NotificationCompat.DEFAULT_LIGHTS)
.setDefaults(NotificationCompat.DEFAULT_SOUND)
.setDefaults(NotificationCompat.DEFAULT_VIBRATE)
```

3.3.5. Уведомление, отображающее ход выполнения

Иногда в уведомлении требуется отображать ход выполнения какой-либо операции, например загрузки файла. За это в построителе уведомления отвечает метод **setProgress()** (*установить прогресс*):

```
.setProgress(0, 0, true)
```

Метод **setProgress()** имеет три параметра:

1) **max** (*максимум*) — максимальное значение прогресса;
2) **progress** (*прогресс*) — начальное значение прогресса;
3) **indeterminate** (*неопределенность*) — будет ли показано завершение выполнения.

Задаются все три значения, но некоторые из них могут быть нулевыми. Значения (0, 0, true), соответствующие рисунку, дают нам имитацию некоторого бесконечного процесса, у которого нет начальной и конечной точки, а также какого-то финального состояния. Обычно такой вариант используют, пока загрузка еще не началась, либо вместо скучного сообщения из серии «Подождите, идет загрузка...».

3.3.6. Уведомления на экране блокировки

В построитель уведомления можно добавить метод **setVisibility()** (*установить видимость*):

```
.setVisibility(NotificationCompat.VISIBILITY_PUBLIC)
```

При значении VISIBILITY_PUBLIC (*общая видимость*) на экране блокировки будет показано полное содержимое уведомления; при значении VISIBILITY_PRIVATE (*«частная», то есть частичная видимость*) будет показана часть уведомления, например название или

имя отправителя; при VISIBILITY_SECRET (*видимость секретная*) уведомление не будет отображаться на экране блокировки вовсе:

3.4. Звуковые эффекты

Здесь речь пойдет не о создании музыкального плеера, а о звуковом сопровождении некоторых элементов приложения или некоторых действий пользователя. Например, это могут быть звуки, издаваемые персонажами в играх.

Концепция мини-приложения: «кинохлопушка». На экране размещено изображение-кнопка, по нажатию на которое воспроизводятся звуки: хлопки и фразы помощника режиссера типа «сцена-один, дубль-один».

Создадим новый проект Sounds со стандартными параметрами и одной Empty Activity.

Подготовим ресурсы. Нам понадобится одно изображение кинохлопушки (поместим его в папку **res/drawable**) и два звуковых файла: звук самой хлопушки и запись фразы из серии «сцена-один дубль-один» (эти ресурсы нужно поместить в папку **res/raw**, при необходимости создав ее). Звуковые файлы можно записать самостоятельно или, так же как и изображения, искать на стоках (рекомендуется брать файлы, распространяемые по лицензии CC0).

На главном экране разместим кнопку **ImageButton**, в качестве ресурса установим для нее изображение кинохлопушки, разме-

стим по центру экрана, добавим обработчик события нажатия — метод **onClick()**:

Перейдем в файл **MainActivity.java**.

Для получения возможности загрузки, воспроизведения и работы со звуками в приложении нам нужен класс **SoundPool** (*звуковой пул, объединение*). Для этого к нашему классу **MainActivity** нужно подключить метод **OnLoadCompleteListener** (*прослушиватель завершения загрузки*), входящий в класс **SoundPool**. «Подключение» осуществляется путем написания команды **implements** («*имплементировать*», *реализовать*) после имени класса (строка № 9):

```
3  import android.media.AudioManager;
4  import android.media.SoundPool;
5  import android.support.v7.app.AppCompatActivity;
6  import android.os.Bundle;
7  import android.view.View;
8
9  public class MainActivity extends AppCompatActivity
10                            implements SoundPool.OnLoadCompleteListener {
```

Android Studio предлагает имплементировать соответствующие методы, то есть реализовать их в нашем коде. Соглашаемся, в конце программы видим появившийся метод **onLoadComplete()** (*по завершении загрузки*), но в нем мы пока ничего писать не будем.

Далее в классе **MainActivity** объявляем переменные для **SoundPool** и двух наших звуковых файлов:

```
9  public class MainActivity extends AppCompatActivity
10                            implements SoundPool.OnLoadCompleteListener {
11     private SoundPool mSoundPool;
12     private int mClapSound, mVoiceSound;
13
14     @Override
15     protected void onCreate(Bundle savedInstanceState) {
16         super.onCreate(savedInstanceState);
17         setContentView(R.layout.activity_main);
```

Теперь определяем **SoundPool** в методе **onCreate()** (строка № 20). На вход ему передаются три параметра: максимальное количество одновременно воспроизводимых звуковых файлов (в нашем случае два), используемый аудиопоток (задаем стандартный STREAM_MUSIC из класса **AudioManager**) и параметр качества, который принято задавать равным 0. Также для **SoundPool** устанавливается прослушиватель окончания загрузки (строка № 21):

```
15  @Override
16  protected void onCreate(Bundle savedInstanceState) {
17      super.onCreate(savedInstanceState);
18      setContentView(R.layout.activity_main);
19
20      mSoundPool = new SoundPool(2, AudioManager.STREAM_MUSIC, 0);
21      mSoundPool.setOnLoadCompleteListener(this);
22  }
```

Среда разработки **Android Studio** в последних версиях «осуждает» использование класса **SoundPool**, зачеркивает его в коде и отмечает как устаревший, а взамен предлагает класс **SoundPool.Builder**. Однако на практике оба эти класса продолжают свободно использоваться, к тому же **SoundPool.Builder** не будет работать на более ранних версиях системы Android. Поэтому предупреждение можно просто игнорировать, а можно добавить перед методом **onCreate()** строку, которая запретит показывать подобные предупреждения для данного метода:

```
@SuppressWarnings(«deprecation»)
```

Определим и с помощью метода **load()** (*загрузить*) загрузим наши звуковые файлы из ресурсов в **SoundPool**:

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

    mSoundPool = new SoundPool(2, AudioManager.STREAM_MUSIC, 0);
    mSoundPool.setOnLoadCompleteListener(this);

    mVoiceSound = mSoundPool.load(this, R.raw.voice, 1);
    mClapSound = mSoundPool.load(this, R.raw.clap, 1);
}
```

Осталось организовать запуск музыкальных файлов в определенный момент, в данном случае — по нажатию на **ImageButton**. Для этого в методе **onClick()** пропишем строки:

```
public void onClick(View view) {
    mSoundPool.play(mVoiceSound, 1, 1, 0, 0, 1);
    mSoundPool.play(mClapSound, 1, 1, 0, 0, 1);
}

@Override
public void onLoadComplete(SoundPool soundPool,
                           int sampleId, int status) {
}
}
```

Метод **play()** получает на вход следующие параметры:

- **идентификатор воспроизводимого звукового файла**;
- **громкость левого канала** — значение от 0 до 1;
- **громкость правого канала** — значение от 0 до 1;
- **приоритет воспроизведения**;
- **количество повторов**;
- **скорость воспроизведения** — коэффициент от 0,5 до 2.

Запустим приложение и убедимся, что при нажатии на кинохлопушку мы одновременно слышим хлопок и фразу, звуки имеют одинаковую громкость и воспроизводятся однократно.

Задания

1. Выставить приоритет воспроизведения, чтобы сначала звучал текст, а затем — хлопок.
2. Воспроизводить хлопок дважды, а скорость воспроизведения фразы увеличить в два раза.

3. Реализовать возможность остановки воспроизведения: добавить на экран еще одну кнопку, и в обработчике события ее нажатия прописать метод **stop()** для каждого звука:
   mSoundPool.stop(идентификатор_звукового_файла);

3.5. Приложение «Маленький принц»

В данном приложении мы применим все, чему научились в этой главе, включив в него все виды оповещений пользователя.

В начале 2000-х годов была популярна игрушка «Тамагочи» — виртуальный питомец, который нуждался в непрерывном уходе и постоянно уведомлял хозяина о своих многочисленных потребностях. Этот принцип все еще популярен — сейчас в магазине приложений Google Play множество подобных приложений с самыми разными «питомцами».

Мы разработаем похожее приложение и обратимся для этого к мировой литературе, к герою, чей обычный день был больше похож на алгоритм. Это **Маленький принц**, главный герой одноименной книги Антуана де Сент-Экзюпери. Утром он приводил в порядок свою планету. «*Есть такое твердое правило, — сказал мне после Маленький принц. — Встал поутру, умылся, привел себя в порядок — и сразу же приведи в порядок свою планету. Непременно надо каждый день выпалывать баобабы, как только их уже можно отличить от розовых кустов: молодые ростки у них почти одинаковые. Это очень скучная работа, но совсем не трудная*».

Днем Маленький принц должен был обязательно поливать свою Розу и ухаживать за ней. А вечером укрывать ее от сквозняков ширмой и накрывать колпаком. «*Нет, тигры мне не страшны, но я ужасно боюсь сквозняков. У вас нет ширмы?* ‹...› *Когда настанет вечер, накройте меня колпаком. У вас тут слишком холодно*».

Ночью Маленький принц мог сколько угодно раз любоваться закатом. «*А на твоей планетке тебе довольно было передвинуть стул на несколько шагов. И ты опять и опять смотрел на закатное небо, стоило только захотеть...*» По сути, он мог «менять» время суток по своему усмотрению.

Попробуем частично воспроизвести этот «алгоритм».

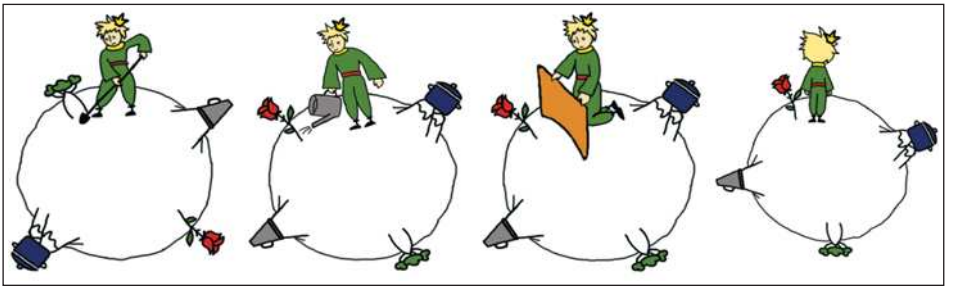
Концепция приложения: «один день из жизни Маленького принца на астероиде Б-612». С помощью кнопок навигации пользователь может перемещаться по локациям «утро», «день», «вечер» и «ночь», получая уведомления о том, что нужно делать в

это время суток (привести в порядок планету, укрыть Розу ширмой, полюбоваться закатом и так далее), и всплывающие сообщения о находящихся в локации предметах и персонажах.

Шаг 1. Создать проект Le Petit Prince с параметрами по умолчанию и одной Empty Activity.

Шаг 2. Добавить локации.

В нашем приложении будет четыре «локации» — утро, день, вечер и ночь. Для каждой локации подберем соответствующую картинку (из классических или современных иллюстраций, распространяемых по CC0, либо нарисуем сами, как сделал автор), присвоим им имена morning, day, evening и night соответственно и сохраним в папке проекта **res/drawable**.



Эти локации можно расположить друг за другом, поскольку они будут отображаться по очереди. Для этого из палитры элементов перетаскиваем на дерево компонентов и размещаем внутри уже имеющегося макета **ConstraintLayout** новый макет **FrameLayout**.

Зададим ему идентификатор frameLayout и следующие свойства:

```
<FrameLayout
    android:id="@+id/frameLayout"
    android:layout_width="match_parent"
    android:layout_height="470dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent">

</FrameLayout>
```

Начинаем добавлять локации. В соответствии со свойствами **FrameLayout** элемент, добавленный первым, отображается последним, значит, первой будет ночь.

Внутри макета **FrameLayout** размещаем элемент **ImageView**, присваиваем ему идентификатор imgNight и устанавливаем свойства:

```
17      <ImageView
18          android:id="@+id/imgNight"
19          android:layout_width="360dp"
20          android:layout_height="wrap_content"
21          android:visibility="invisible"
22          app:srcCompat="@drawable/night" />
```

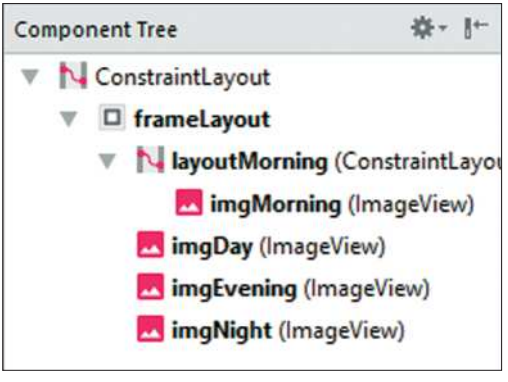
Для вечерней локации по аналогии добавляем **ImageView** с идентификатором imgEvening:

```
17      <ImageView
18          android:id="@+id/imgEvening"
19          android:layout_width="360dp"
20          android:layout_height="wrap_content"
21          android:visibility="invisible"
22          app:srcCompat="@drawable/evening" />
```

Для дневной локации размещаем **ImageView** imgDay с похожими свойствами:

```
17      <ImageView
18          android:id="@+id/imgDay"
19          android:layout_width="360dp"
20          android:layout_height="wrap_content"
21          android:visibility="invisible"
22          app:srcCompat="@drawable/day" />
```

Для утренней локации, так как она отображается первой, мы в дальнейшем добавим дополнительные возможности, поэтому сейчас **ImageView** imgMorning поместим внутрь **ConstraintLayout** layoutMorning:



```
<android.support.constraint.ConstraintLayout
    android:id="@+id/layoutMorning"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:visibility="visible">

    <ImageView
        android:id="@+id/imgMorning"
        android:layout_width="360dp"
        android:layout_height="wrap_content"
        android:visibility="visible"
        app:srcCompat="@drawable/morning" />
</android.support.constraint.ConstraintLayout>
```

Шаг 3. Реализовать навигацию.

Добавим возможность менять локации, то есть выбирать время суток, как это делал Маленький принц. За это будут отвечать четыре кнопки — Утро, День, Вечер и Ночь, которые мы расположили в верхней части приложения и скомпоновали в горизонтальный **LinearLayout**.

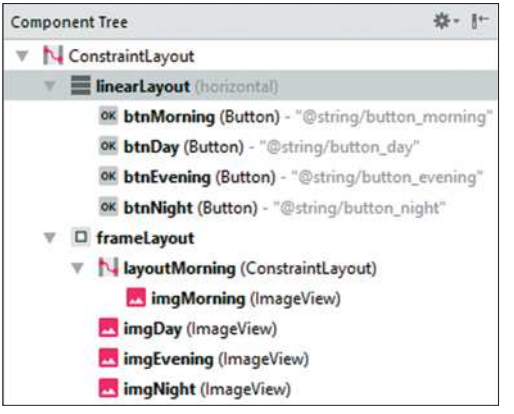
```
<LinearLayout
    android:id="@+id/linearLayout"
    android:layout_width="360dp"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    app:layout_constraintBottom_toTopOf="@+id/frameLayout"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent">
```

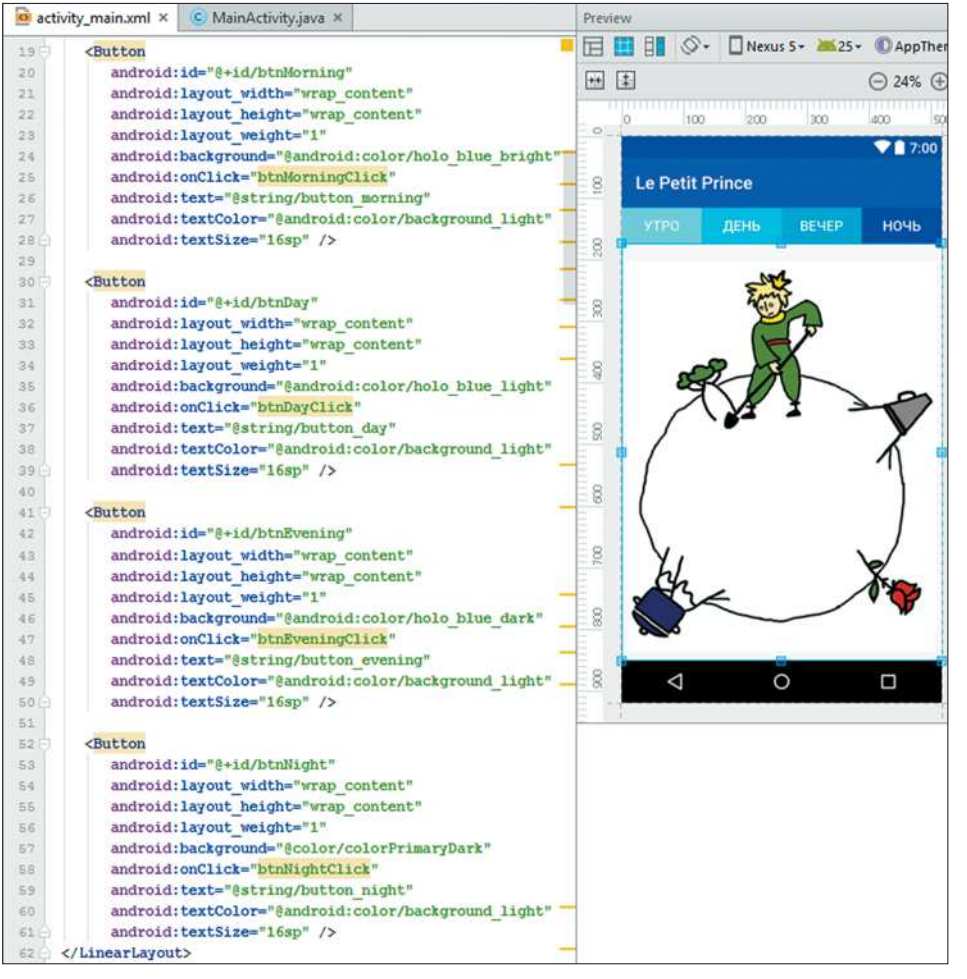
Для текстового содержания наших кнопок создадим строковые ресурсы — пропишем их в файле строковых ресурсов **strings.xml**, хранящемся в папке **res/values**.

activity_main.xml × | strings.xml × | MainActivity.java ×

```
<resources>
    <string name="app_name">Le Petit Prince</string>

    <string name="button_morning">Утро</string>
    <string name="button_day">День</string>
    <string name="button_evening">Вечер</string>
    <string name="button_night">Ночь</string>
</resources>
```

Теперь задаем их свойства и добавляем обработчики событий нажатия:






```
<Button
    android:id="@+id/btnMorning"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:background="@android:color/holo_blue_bright"
    android:onClick="btnMorningClick"
    android:text="@string/button_morning"
    android:textColor="@android:color/background_light"
    android:textSize="16sp" />

<Button
    android:id="@+id/btnDay"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:background="@android:color/holo_blue_light"
    android:onClick="btnDayClick"
    android:text="@string/button_day"
    android:textColor="@android:color/background_light"
    android:textSize="16sp" />

<Button
    android:id="@+id/btnEvening"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:background="@android:color/holo_blue_dark"
    android:onClick="btnEveningClick"
    android:text="@string/button_evening"
    android:textColor="@android:color/background_light"
    android:textSize="16sp" />

<Button
    android:id="@+id/btnNight"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:background="@color/colorPrimaryDark"
    android:onClick="btnNightClick"
    android:text="@string/button_night"
    android:textColor="@android:color/background_light"
    android:textSize="16sp" />
</LinearLayout>
```

Переходим в **MainActivity.java**.

Объявляем и определяем все наши кнопки, изображения и макет **ConstraintLayout**:

```
9  public class MainActivity extends AppCompatActivity {
10
11     private Button btnMorning, btnDay, btnEvening, btnNight;
12     private ImageView imgMorning, imgDay, imgEvening, imgNight;
13     private ConstraintLayout layoutMorning;
14
15     @Override
16     protected void onCreate(Bundle savedInstanceState) {
17         super.onCreate(savedInstanceState);
18         setContentView(R.layout.activity_main);
19
20         btnMorning = (Button) findViewById(R.id.btnMorning);
21         btnDay = (Button) findViewById(R.id.btnDay);
22         btnEvening = (Button) findViewById(R.id.btnEvening);
23         btnNight = (Button) findViewById(R.id.btnNight);
24         imgMorning = (ImageView) findViewById(R.id.imgMorning);
25         imgDay = (ImageView) findViewById(R.id.imgDay);
26         imgEvening = (ImageView) findViewById(R.id.imgEvening);
27         imgNight = (ImageView) findViewById(R.id.imgNight);
28         layoutMorning = (android.support.constraint.ConstraintLayout)
29                         findViewById(R.id.layoutMorning);
30     }
```

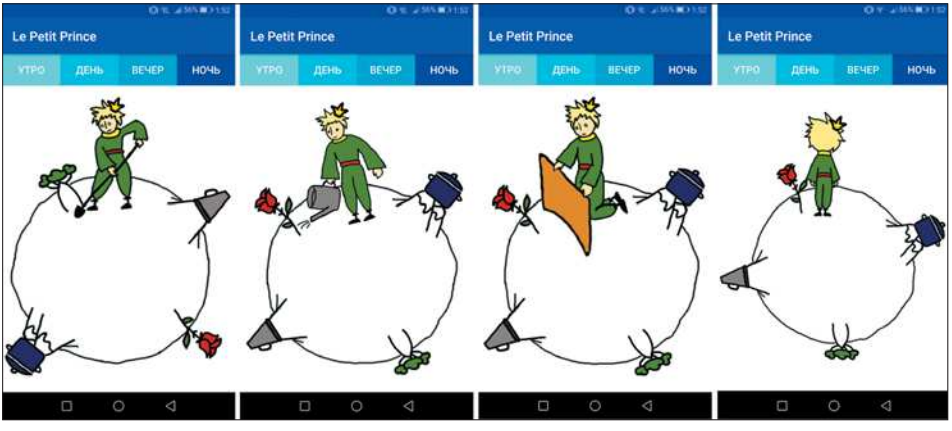
Для осуществления навигации по локациям нужно сделать так, чтобы при нажатии на кнопку отображалось только изображение, соответствующее ее локации.

Установим всем локациям видимость с помощью метода **setVisibility()** (*установить видимость*). Например, если для кнопки **btnMorning** у макета **layoutMorning** мы устанавливаем значение VISIBLE (*видимый*), то для кнопок **imgDay**, **imgEvening** и **imgNight** — значение INVISIBLE (*невидимый*). И также по аналогии для кнопок **btnDay**, **btnEvening** и **btnNight**.

```
32     public void btnMorningClick(View view) {
33         layoutMorning.setVisibility(View.VISIBLE);
34         imgDay.setVisibility(View.INVISIBLE);
35         imgEvening.setVisibility(View.INVISIBLE);
36         imgNight.setVisibility(View.INVISIBLE);
37     }
```

```
38
39      public void btnDayClick(View view) {
40      }
41
42      public void btnEveningClick(View view) {
43      }
44
45      public void btnNightClick(View view) {
46      }
47  }
```

Запустим приложение. Теперь при нажатии на одну из кнопок происходит навигация по локациям.



Шаг 4. Организовать оповещение пользователя с помощью уведомлений.

Как мы уже выяснили, по утрам Маленький принц должен приводить в порядок свою планету, днем поливать Розу, вечером огораживать ее ширмой, а ночью любоваться закатами. Обо всех этих «обязанностях» нам нужно уведомлять пользователя при переходе к соответствующей локации.

Сначала подберем иконки для уведомлений, потому что они являются обязательными параметрами. Нам подойдут изображения-символы метлы, розы, ширмы и заката. Изображения должны быть двухцветные (например, черно-белые) и с прозрачным фоном. Назовем их соответственно cleaning, rose, home, sunset и поместим в папке проекта **res/drawable**.

В классе **MainActivity** объявим четыре уведомления и присвоим им идентификаторы:

```
10  public class MainActivity extends AppCompatActivity {
11
12      private Button btnMorning, btnDay, btnEvening, btnNight;
13      private ImageView imgMorning, imgDay, imgEvening, imgNight;
14      private ConstraintLayout layoutMorning;
15
16      public static final int notificationMorning = 1;
17      public static final int notificationDay = 2;
18      public static final int notificationEvening = 3;
19      public static final int notificationNight = 4;
20
21      @Override
```

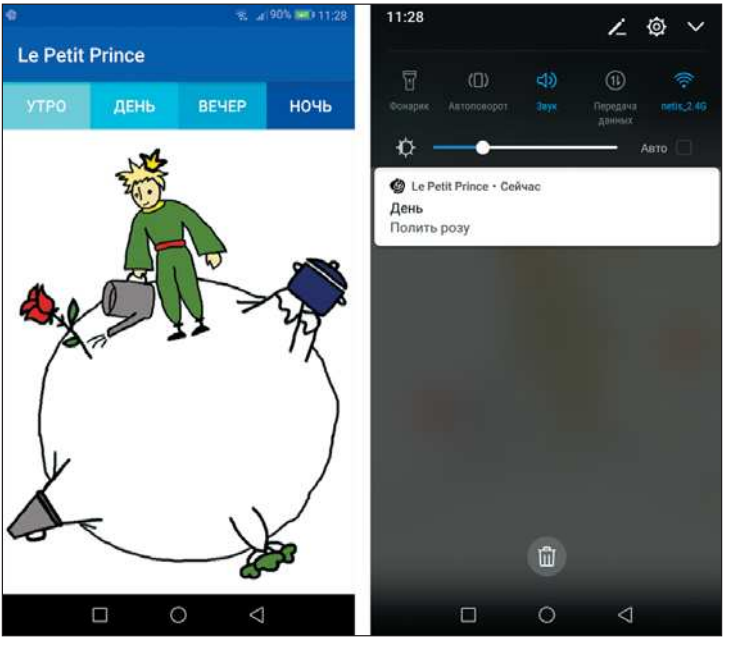
Для того чтобы днем оповестить пользователя о необходимости полить Розу, построим соответствующее уведомление в методе **btnDayClick()**:

- маленькая иконка — изображение розы rose;
- заголовок — День;
- текст — Полить розу;
- автоматическое завершение после прочтения пользователем;
- оповещение вибрацией и звуковым сигналом;
- полная видимость, в том числе на экране блокировки;
- переход к главному экрану нашего приложения по нажатию.

```
50  public void btnDayClick(View view) {
51      imgDay.setVisibility(View.VISIBLE);
52      layoutMorning.setVisibility(View.INVISIBLE);
53      imgEvening.setVisibility(View.INVISIBLE);
54      imgNight.setVisibility(View.INVISIBLE);
55
56      NotificationCompat.Builder mBuilder =
57              (NotificationCompat.Builder) new NotificationCompat.Builder(this)
58                      .setSmallIcon(R.drawable.rose)
59                      .setContentTitle("День")
60                      .setContentText("Полить розу")
61                      .setAutoCancel(true)
62                      .setDefaults(NotificationCompat.DEFAULT_ALL)
63                      .setVisibility(NotificationCompat.VISIBILITY_PUBLIC);
```

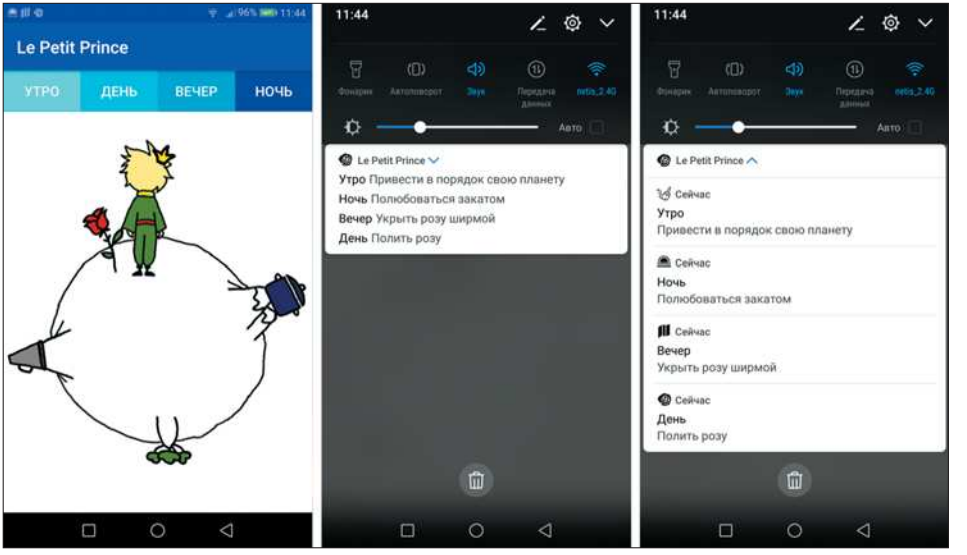
```
64
65      Intent resultIntent = new Intent(this, MainActivity.class);
66
67      TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
68      stackBuilder.addParentStack(MainActivity.class);
69      stackBuilder.addNextIntent(resultIntent);
70
71      PendingIntent resultPendingIntent =
72              stackBuilder.getPendingIntent(0, PendingIntent.FLAG_UPDATE_CURRENT);
73      mBuilder.setContentIntent(resultPendingIntent);
74      NotificationManager mNotificationManager =
75              (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
76
77      mNotificationManager.notify(notificationDay, mBuilder.build());
78  }
```

Запускаем приложение и убеждаемся, что уведомление работает:



По аналогии добавляем уведомления для остальных локаций:

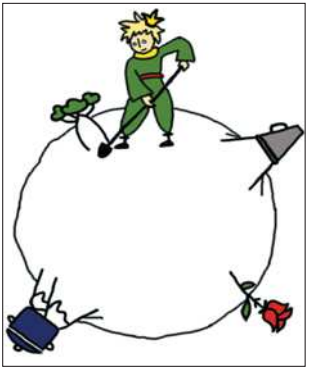
- **btnMorningClick()** — иконка cleaning, заголовок Утро, текст Привести в порядок свою планету;
- **btnEveningClick()** — иконка home, заголовок Вечер, текст Закрыть розу ширмой;
- **btnNightClick()** — иконка sunset, заголовок Ночь, текст Полюбоваться закатом.



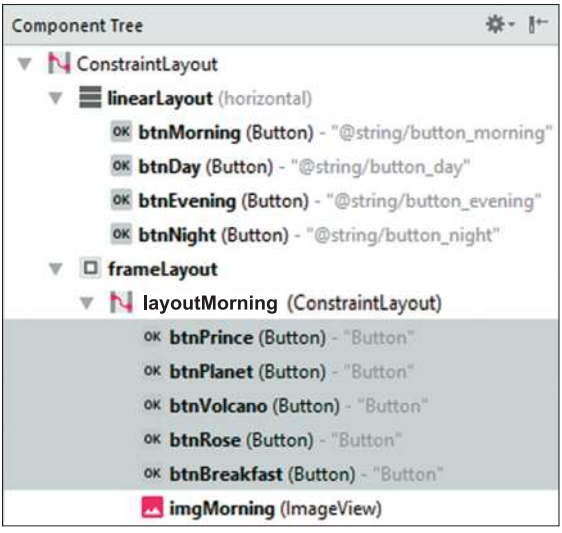
Шаг 5. Добавить всплывающие сообщения.

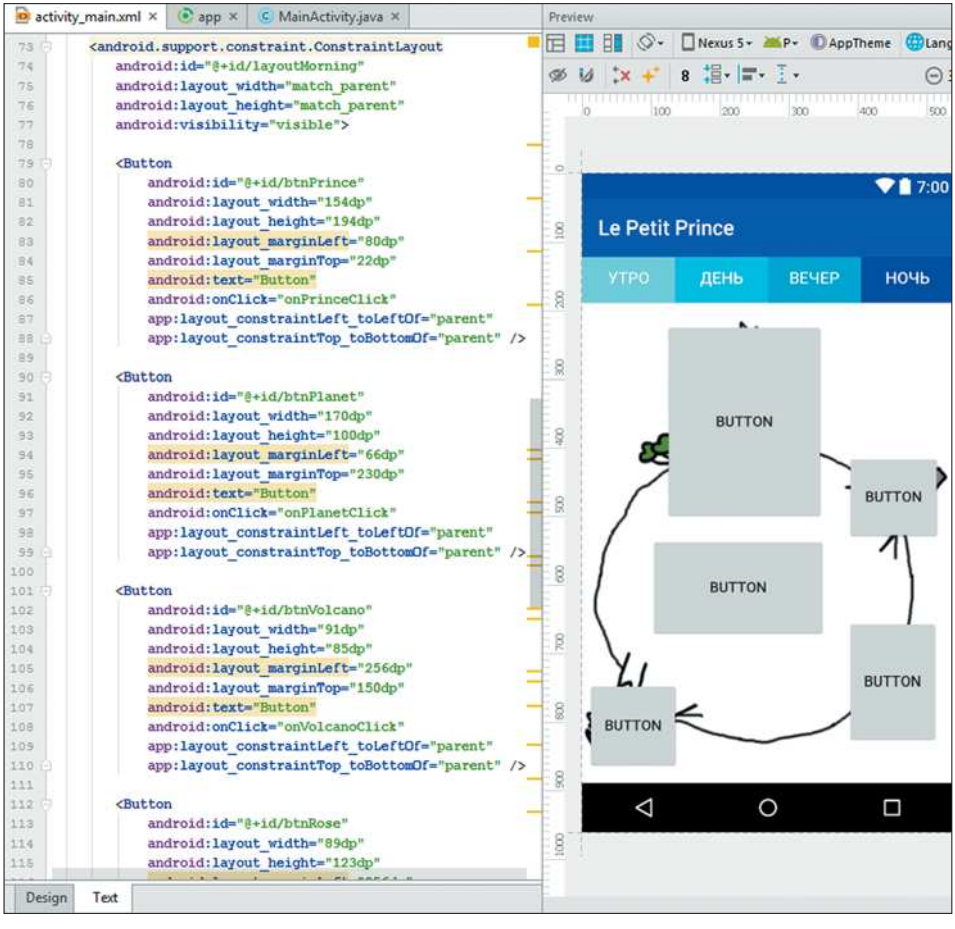
В этом приложении мы файлы не загружаем и сообщения не отправляем, поэтому будем «знакомить» пользователя с персонажами утренней локации с помощью всплывающих сообщений **Toast**. Однако дробить картинку на части (как в уроке про всплывающие сообщения) мы не будем, а используем для этой цели кнопки, которые расположим поверх локации.

На картинке для утренней локации изображены: Маленький принц, его планета, потухший вулкан, Роза и завтрак, разогревающийся на действующем вулкане.



Перетащим в дерево компонентов и расположим внутри макета **layoutMorning** пять соответствующих кнопок:

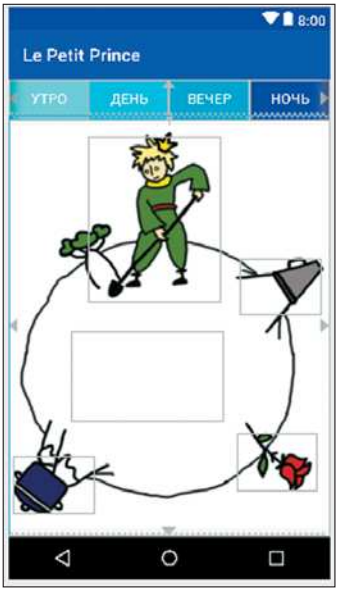

Component Tree
ConstraintLayout
linearLayout (horizontal)
btnMorning (Button) - "@string/button_morning"
btnDay (Button) - "@string/button_day"
btnEvening (Button) - "@string/button_evening"
btnNight (Button) - "@string/button_night"
frameLayout
layoutMorning (ConstraintLayout)
btnPrince (Button) - "Button"
btnPlanet (Button) - "Button"
btnVolcano (Button) - "Button"
btnRose (Button) - "Button"
btnBreakfast (Button) - "Button"
imgMorning (ImageView)


activity_main.xml
app
MainActivity.java
Preview
<android.support.constraint.ConstraintLayout
android:id="@+id/layoutMorning"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:visibility="visible">
<Button
android:id="@+id/btnPrince"
android:layout_width="154dp"
android:layout_height="194dp"
android:layout_marginLeft="80dp"
android:layout_marginTop="22dp"
android:text="Button"
android:onClick="onPrinceClick"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintTop_toBottomOf="parent" />
<Button
android:id="@+id/btnPlanet"
android:layout_width="170dp"
android:layout_height="100dp"
android:layout_marginLeft="66dp"
android:layout_marginTop="230dp"
android:text="Button"
android:onClick="onPlanetClick"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintTop_toBottomOf="parent" />
<Button
android:id="@+id/btnVolcano"
android:layout_width="91dp"
android:layout_height="85dp"
android:layout_marginLeft="256dp"
android:layout_marginTop="150dp"
android:text="Button"
android:onClick="onVolcanoClick"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintTop_toBottomOf="parent" />
<Button
android:id="@+id/btnRose"
android:layout_width="89dp"
android:layout_height="123dp"
Design
Text
Nexus 5
AppTheme
7:00
Le Petit Prince
УТРО
ДЕНЬ
ВЕЧЕР
НОЧЬ
BUTTON
BUTTON
BUTTON
BUTTON
BUTTON

точно над своими «персонажами», добавим обработчики событий нажатия.

Как видим (см. рисунок на с. 136), кнопки заслоняют своих персонажей — это нехорошо! Уберем их тексты и в качестве фона зададим цвет transparent (*прозрачный*):

```
android:text="Button"
android:background="@android:color/transparent"
```



Осталось в обработчиках событий нажатия прописать появление всплывающих сообщений.

Для Маленького принца в методе **onPrinceClick()** добавим **Toast** myToast1 с текстом Маленький принц, долгим отображением и расположением внизу экрана (по центру без сдвигов).

```
165  public void onPrinceClick(View view) {
166      Toast myToast1 = Toast.makeText(getApplicationContext(),
167                       "Маленький принц", Toast.LENGTH_LONG);
168      myToast1.setGravity(Gravity.BOTTOM, 0, 0);
169      myToast1.show();
170  }
171
172  public void onPlanetClick(View view) {
173
174  }
```

Аналогичные всплывающие сообщения добавляем и для остальных кнопок:

- **onPlanetClick()** — сообщение myToast2 с текстом: Астероид Б-612. Планета, на которой живет Маленький принц;
- **onVolcanoClick()** — сообщение myToast3 с текстом: Потухший вулкан. О нем тоже нужно заботиться;
- **onRoseClick()** — сообщение myToast4 с текстом: Роза. Ее нужно поливать, а на ночь укрывать ширмой и колпаком;
- **onBreakfastClick()** — сообщение myToast5 с текстом: Действующий вулкан. На нем удобно разогревать завтрак.

Готово!

Задания

1. Добавить в приложение диалоговые окна, например с запросами на подтверждение действий при нажатии на уведомления.
2. Добавить в приложение звуковые эффекты, например при нажатии на персонажей.
3. Доработать приложение до корректного отображения на планшетах.

Итоги главы 3

В главе 3 мы научили наши приложения «разговаривать» с пользователем, а сами научились:

1. Создавать всплывающие сообщения для информирования пользователя о том, что происходит в приложении в данный момент. Это очень важно, потому что пользователь не любит ждать, а тем более ждать неизвестно чего.
2. Строить простые и сложные диалоговые окна, чтобы предложить пользователю выбор или запросить у него дополнительную информацию.
3. Создавать уведомления для привлечения внимания пользователя, даже когда приложение закрыто.
4. Сопровождать действия пользователя в приложении звуковыми эффектами, работать с несколькими звуковыми потоками, настраивать различные параметры воспроизведения звуковых файлов.

Глава 4. Дизайн и юзабилити мобильных приложений

4.1. Дизайн и юзабилити

В магазине Google Play миллионы приложений. И когда мы выбираем какое-то одно приложение из ряда похожих, то обычно руководствуемся интуитивными понятиями «красиво» и «удобно». С «красотой» все понятно, речь идет о **дизайне** приложения. Существуют требования к дизайну приложений (и мы с ними познакомимся в следующих уроках) и современные тренды. А «удобство» использования и все, что мы под ним подразумеваем, входит в понятие **юзабилити.**

Слово **usability** от прилагательного **usable** — *годный к употреблению, удобный, практичный* на русский язык переводится как «удобство использования», «степень удобства и простоты использования». Но, как и большинство терминов IT-сферы, в какой-то момент его перестали переводить и начали активно использовать в речи и на письме, как заимствованное слово.

В нашем случае **юзабилити** — это способность приложения быть удобным для пользователя, практичным, эстетичным, простым и интуитивно понятным в использовании, позволять в кратчайшее время достигать желаемого результата и при этом обеспечивать пользователю чувство удовлетворенности от использования именно этого приложения.

Известный специалист по юзабилити Якоб Нильсен еще в 1990 году предложил «10 эвристик», или правил, обеспечивающих юзабилити системы. Эти эвристики не теряют своей актуальности и полностью могут быть применены к мобильным приложениям:

1. Видимость статуса системы. Наш пользователь должен постоянно понимать, что происходит с приложением. Для этого используются всплывающие сообщения, уведомления, рассылки, загрузочные экраны и различные отображения статуса.

2. Соответствие между системой и реальным миром. Приложение должно говорить с пользователем на понятном ему языке, больше ориентируясь на повседневную речь, чем на специализированные термины.

3. Контролируемость системы пользователем и свобода его действий. У пользователя всегда должна быть возможность после фразы «Упс! Я выбрал что-то не то» отменить действие или вернуться на предыдущий шаг и сделать все, как он задумывал.

4. Согласованность понятий и соблюдение стандартов. При наполнении приложения контентом мы не должны называть одни

и те же элементы или действия разными словами. Лучше всего соблюдать общепринятые наименования и следовать стандартам.

5. Предотвращение ошибок. Какими бы понятными мы ни сделали сообщения об ошибках — пользователь предпочтет их не видеть вовсе. При разработке приложения лучше продумать его так, чтобы свести количество возможных ошибок к минимуму. Часто ошибки можно предотвратить подсказками и примерами.

6. Видеть — лучше, чем вспоминать. Пользователь не должен вспоминать, как что-то сделать, перейдя в другой раздел приложения. Мы должны обеспечить быстрый доступ к самым востребованным функциям приложения.

7. Гибкость и эффективность использования. Опытные пользователи нашего приложения и новички должны находиться в равных условиях: вторым нужно четко понимать «что и как делать», и при этом первых не должны раздражать слишком подробные инструкции. Упрощение и настройка часто повторяемых действий помогают.

8. Эстетичный и минималистичный дизайн. В приложении не должно быть лишних элементов и информации. Особенно это актуально для смартфонов с их небольшими экранами, где каждое слово занимает ощутимую часть пространства.

9. Помощь пользователю в понимании и исправлении ошибок. Сообщения об ошибках должны быть сформулированы простым языком, не содержать технической информации и предлагать конкретное решение.

10. Помощь и документация. Разумеется, к мобильным приложениям сейчас никто не пишет руководство пользователя и прочую сопроводительную документацию, но в процессе работы с нашим приложением пользователь должен видеть некоторые подсказки или знать, где их можно найти (например, в соответствующем разделе).

Все эти эвристики кажутся вполне логичными, многие из них даже очевидны, но их изучение позволит нам в будущем создавать приложения, которые действительно полюбят пользователи. А проверить выполнение эвристик юзабилити достаточно просто — для этого существует **юзабилити-тестирование**.

Самый простой и популярный способ проведения юзабилити-тестирования — **сценарий «целевой пользователь»**. Как правило, первыми и главными тестировщиками приложения являемся мы (сами разработчики) и участники нашей команды. Но мы как разработчики провели со своим приложением много времени, у нас в голове идеальное представление, но часто замыленный глаз. Поэтому мы можем упустить что-то важное.

Для этого пишутся **сценарии**, в которых представляется типичный потенциальный пользователь приложения, его история, цель, задачи в приложении и последовательность действий. Например, если мы разработали приложение по поиску репетиторов для школьников, то в сценарий можно поместить следующую информацию:

«***Пользователь***: Аня, 15 лет, ученица 10 класса.

Цель: поиск репетитора по информатике.

Задача в приложении: выбрать репетитора и записаться на первое занятие.

Требования: Аня собирается сдавать ЕГЭ по информатике, поэтому репетитор должен быть школьным учителем, не из школы № 30 (в которой учится Аня), но из Басманного района (чтобы Ане было легко добираться). Аня не готова платить больше XX руб/час, хочет заниматься 1 раз в неделю, свободна по средам после 17:00 и готова начать уже на этой неделе.

Сценарий:
1. *Запуск приложения.*
2. *Переход в раздел поиска.*
3. *Заполнение параметров поиска.*
4. *Изучение предложенных вариантов.*
5. *Корректировка поискового запроса.*
6. *Выбор подходящего варианта.*
7. *Заказ.*
8. *Получение уведомлений (обратной связи).*

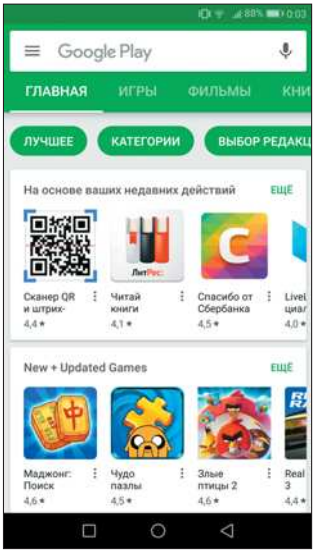
... »

Такие сценарии выдаются знакомым или кому угодно, согласившемуся поучаствовать в тестировании. Оценивается, сколько времени у человека ушло на выполнение задания из сценария и какие трудности при этом возникали.

4.2. Логотип приложения

4.2.1. Назначение и роль логотипа

Логотип приложения отображается в магазине Google Play, а также в меню смартфона, на экране в качестве ярлыка быстрого доступа к приложению, в менеджере управления приложениями:



По умолчанию **Android Studio** присваивает приложениям стандартный логотип с изображением робота-андроида:

Но ведь все приложения не могут выглядеть одинаково. К тому же использование стандартного логотипа не разрешено политикой Google. Поэтому при создании приложения нужно уделить особое внимание выбору его логотипа.

4.2.2. Виды логотипов

Логотипы можно разделить на три группы.

1. Текстовые логотипы. Обычно такие логотипы для приложений используют компании, названия которых в жизни и являются логотипами этих компаний (ebay, NewYorker, H&M, Booking):

2. Знаковые логотипы. На них нет текста, только изображение, явно ассоциирующееся с компанией или самим приложением и его назначением (Twitter, YouTube, Instagram, WhatsApp, Angry Birds).

3. Комбинированные логотипы. Разработчики игр Star Wars, ВКонтакте, Duolingo, Burger King, AliExpress и многих других приложений считают, что лучшим способом привлечения внимания пользователя будет размещение на логотипе и изображения, и названия.

На сегодняшний день для мобильных приложений в основном используются знаковые и комбинированные логотипы, потому что логотип всегда сопровождается полным названием (как в магазине приложений, так и в меню смартфона), а картинки на небольшом экране лучше привлекают внимание.

4.2.3. Создание логотипа

При подборе изображения для логотипа приложения нужно следовать ряду правил:

1. Логотип должен отражать суть и содержание приложения. Например, если мы создаем приложение для управления роботом с помощью смартфона, вряд ли стоит размещать на логотипе что-то кроме робота и, максимум, смартфона.

Не следует размещать на логотипе много текста и несколько изображений. Если же логотип содержит группу объектов, она должна быть зрительно неделимой и представлять собой единый силуэт. Прежде чем использовать для логотипа готовое изображение из Интернета, следует сначала найти в магазине приложений приложения с похожей тематикой и проверить, не воспользовался ли этим изображением другой разработчик.

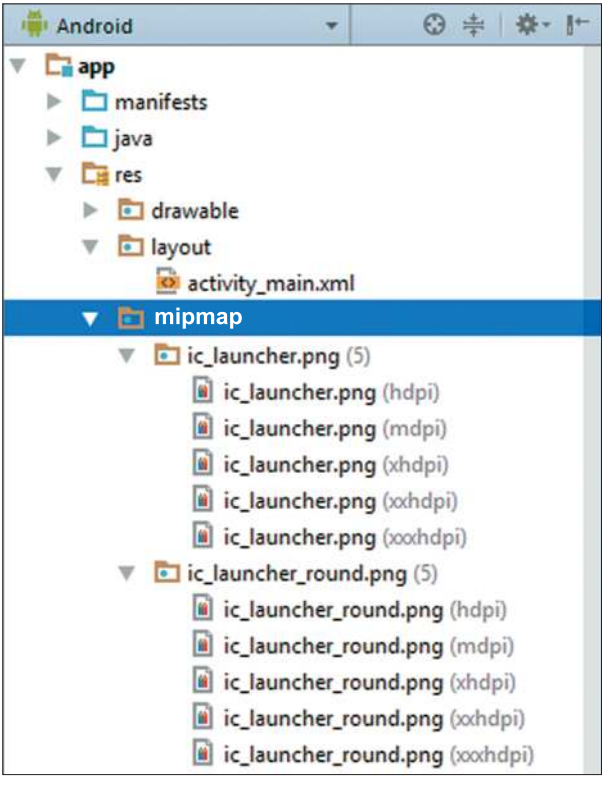
2. Логотип должен хорошо и четко смотреться в разных размерах и при разных цветах фона. Но не стоит устанавливать в качестве логотипа фотографии.

3. Логотип приложения должен гармонично смотреться с остальными логотипами и иметь те же размеры.

4. При создании логотипа внимательно следить за текущими тенденциями и отслеживать логотипы похожих приложений, чтобы достойно смотреться на их фоне и в то же время выгодно отличаться.

В проекте Android Studio логотип приложения располагается в папке **res/mipmap** в двух вариациях: «обычный» логотип (по умолчанию носит имя **ic_launcher.png**) и круглый логотип (**ic_launcher_round.png**). Вторая вариация создается потому, что в последних версиях Android использует адаптивные значки. Соответственно там, где требуется, отобразит предоставленный нами круглый логотип, а в остальных случаях возьмет «обычный» и адаптирует его форму и размер в соответствии с версией системы, моделью смартфона или установленной темой.

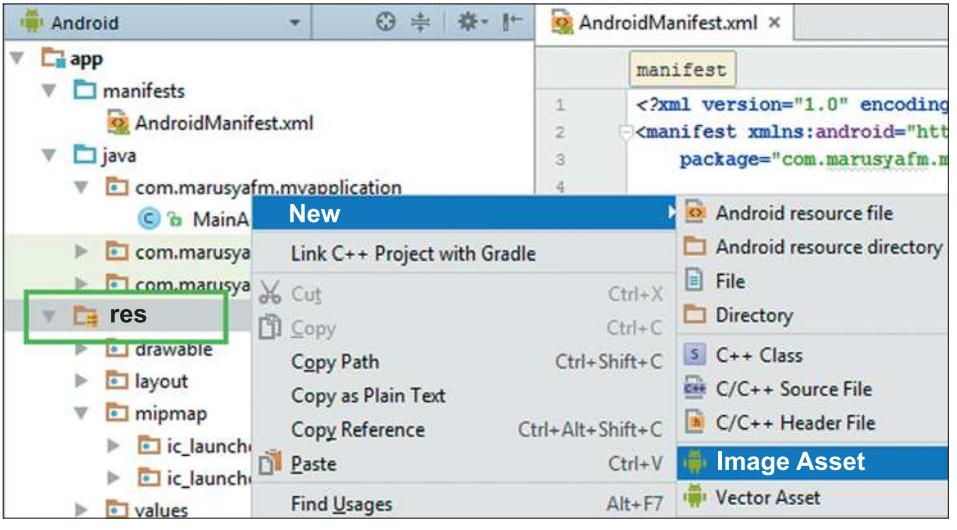
Вместе с каждой вариацией хранится несколько ее копий для использования в различных разрешениях.



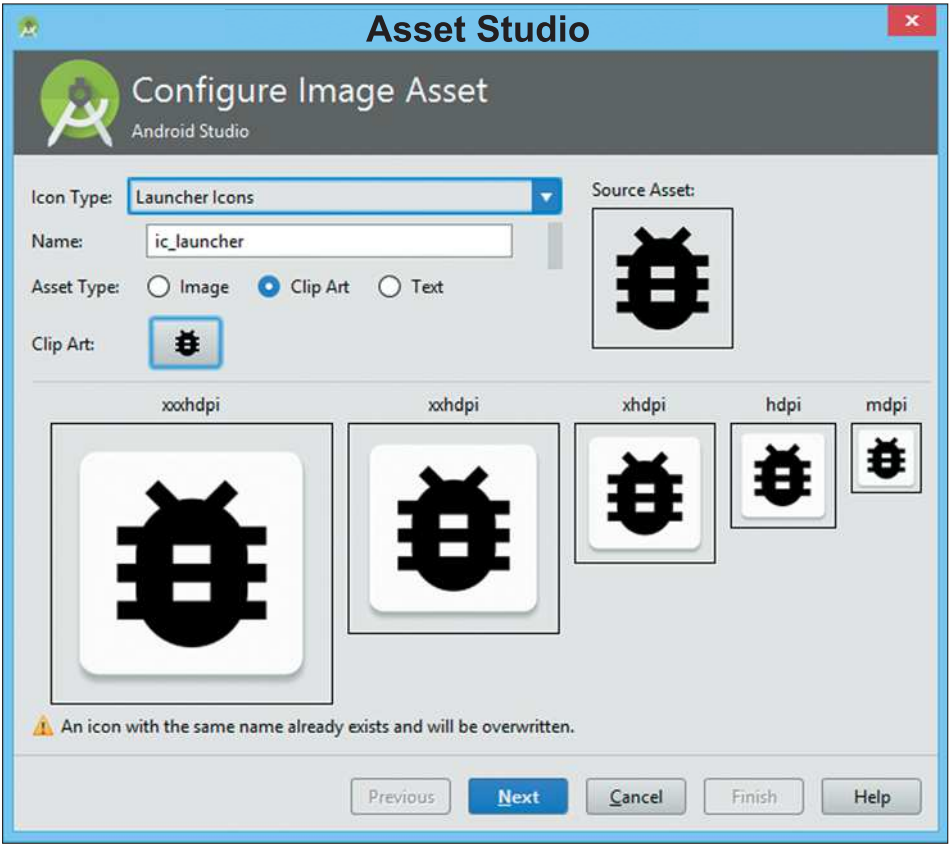
Для создания логотипа приложения можно воспользоваться любым известным редактором (например, Adobe Photoshop, Logo Design Studio, CorelDraw, Inkscape), одним из многочисленных онлайн-редакторов и даже мобильных приложений.

Но можно создать логотип и непосредственно в **Android Studio** с помощью инструмента **Image Asset Studio**. Он предназначен для простого и быстрого создания различных значков (для логотипа приложения, для пунктов меню, кнопок и так далее), соответствующих всем требованиям для использования в мобильных приложениях.

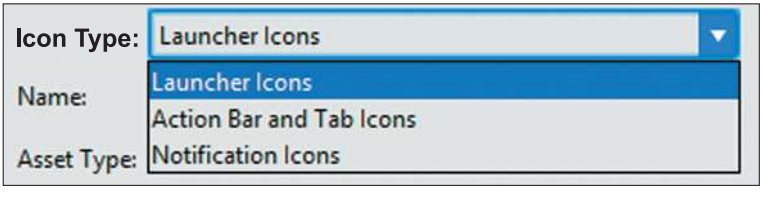
Для того чтобы создать изображение для логотипа, нажмем правой кнопкой мыши на папке проекта **res**, затем выберем **New** → → **Image Asset** (*Новый* → *Изображение-ресурс*):



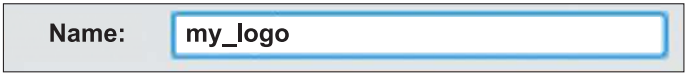
В открывшемся окне **Asset Studio** настраиваются параметры изображения:



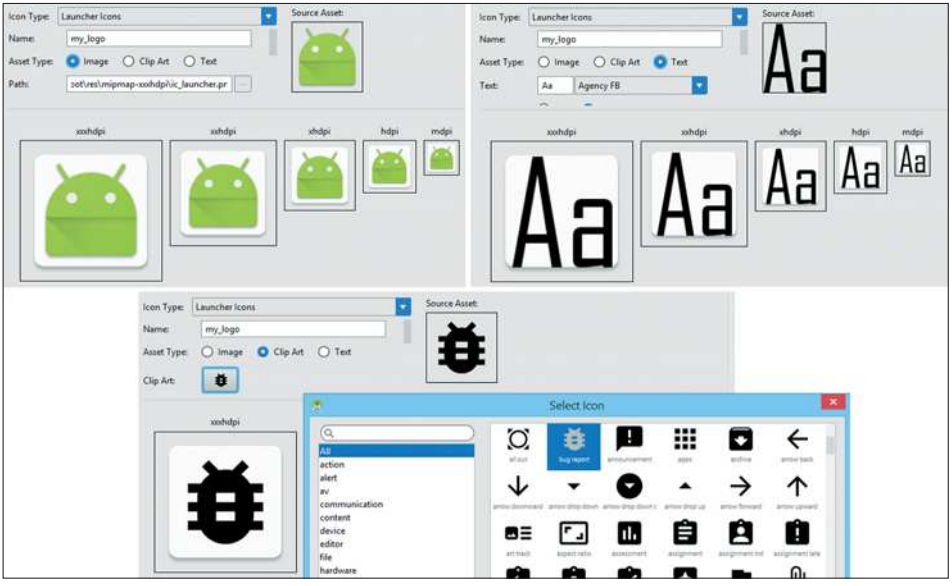
В поле **Icon Type** (*тип значка*) можно выбрать тип создаваемого значка по назначению: **Launcher Icons** (*значок запуска*) — это как раз наш логотип, **Action Bar and Tab Icons** (*значки для пунктов меню и вкладок*), **Notification Icons** (*значки для уведомлений*). Для создания логотипа выбираем **Launcher Icons:**



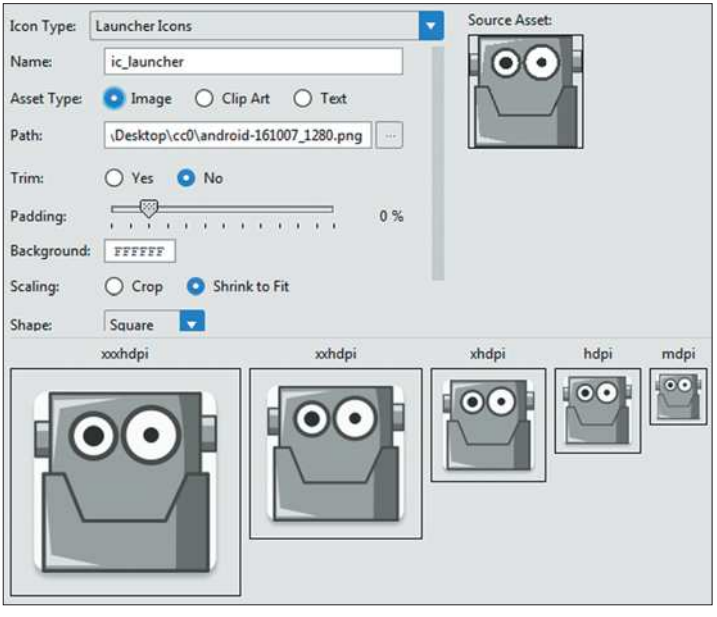
В поле **Name** (*имя*) задаем имя значка, например my_logo.



В поле **Asset Type** (*тип ресурса*) выбирается тип ресурса, используемого для создания значка: **Image** (*изображение*, которое мы загрузим сами), **Clip Art** (*клипарт*) — выбор одной из стандартных иконок, **Text** (*текст*) — для создания текстовых логотипов.



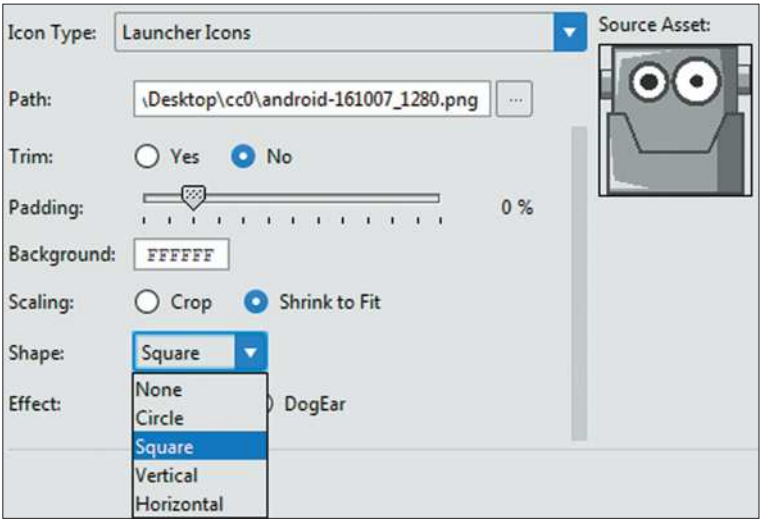
Выбираем **Image**. В поле **Path** (*путь*) прописываем путь к картинке и смотрим, как изменилось наше изображение:



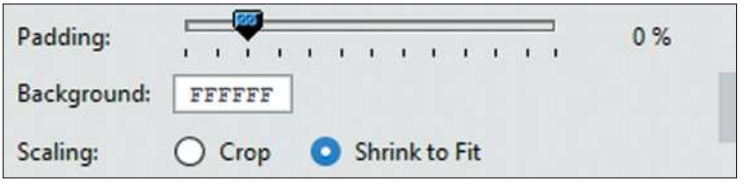
Теперь с помощью полосы прокрутки перейдем в конец списка параметров.

Поле **Effect** (*эффект*) предоставляет на выбор два значения: **None** (*ничего*) — без эффекта и **DogEar** (*собачье ухо*) — с загнутым уголком. Оставляем **None**.

В поле **Shape** (*форма*) можно выбрать фигуру, в которую будет вписано изображение. Здесь по умолчанию выбран **Square** (*квадрат*), но доступны еще круг, вертикальный и горизонтальный прямоугольники, а также отсутствие фоновой фигуры. Выберем **None** — отсутствие фигуры и фона в принципе.

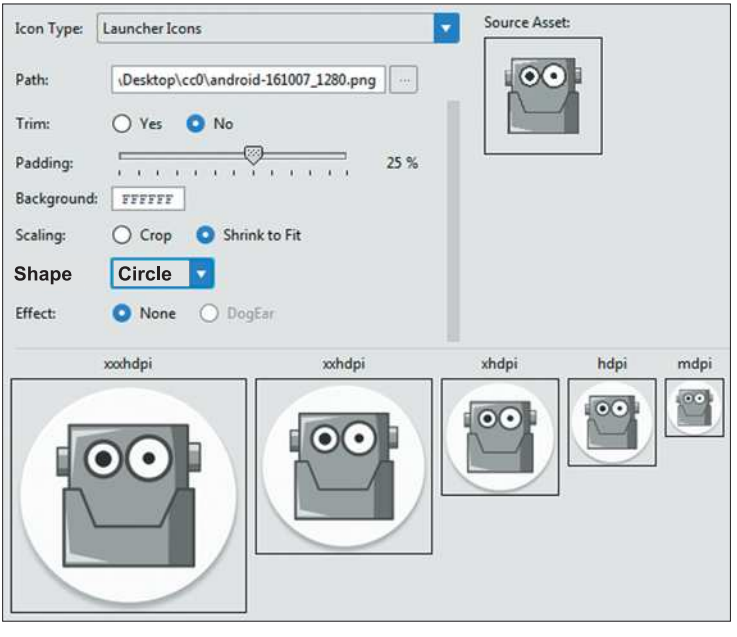


Поле **Padding** (*набивка, заполнение*) определяет, на сколько процентов фигура заполняет пространство значка, поле **Background** (*фон*) позволяет установить цвет фона для изображения. В зависимости от значения поля **Scaling** (*обмер, рамка*) система обрежет края нашего значка или полностью впишет изображение в значок без возможности изменения границ.



Когда логотип сформирован, можно перейти на следующий шаг (кнопка **Next**) — выбрать папку для сохранения созданного значка. Выберем папку **main**, и наш логотип будет автоматически создан во всех необходимых разрешениях и форматах. Нажимаем кнопку **Finish**.

По аналогии создаем второй логотип — **круглый**, выбрав в параметре **Shape** фигуру **Circle** (*круг*):



Готово! Осталось установить созданные изображения в качестве логотипа нашего приложения.

4.2.4. Установка логотипа приложения

Как мы уже знаем, название приложения меняется в файле манифеста **AndroidManifest.xml**. Здесь же прописан и логотип.

AndroidManifest.xml ×

manifest | application

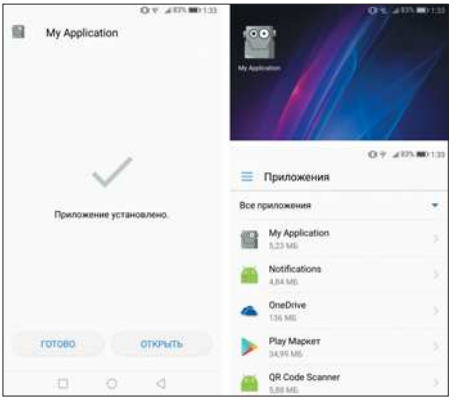
```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      package="com.marusyafm.myapplication">
4
5      <application
6          android:allowBackup="true"
7          android:icon="@mipmap/ic_launcher"
8          android:label="My Application"
9          android:roundIcon="@mipmap/ic_launcher_round"
10         android:supportsRtl="true"
11         android:theme="@style/AppTheme">
```

В атрибуте **icon** (*иконка*, строка № 7) прописан идентификатор логотипа, а в атрибуте **roundIcon** (*круглая иконка*, строка № 9) — идентификатор круглого логотипа.

Как только мы пропишем в эти атрибуты идентификаторы созданных изображений, изменится логотип приложения. Изменения можно наблюдать сразу, прямо в коде справа от номера строки:

```
5      <application
6          android:allowBackup="true"
7          android:icon="@mipmap/ic_launcher"
8          android:label="My Application"
```

Запускаем приложение. Видим измененный логотип и в процессе установки приложения, и на экране, и в менеджере управления приложениями:



4.3. Загрузочный экран приложения

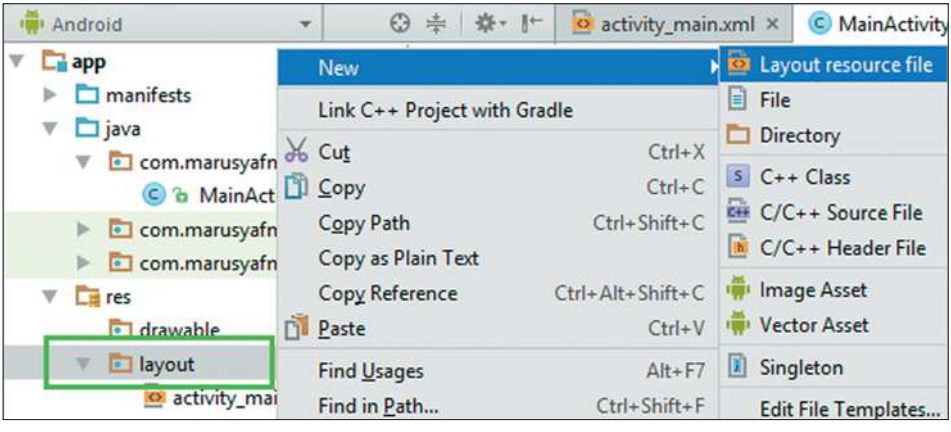
Загрузочный экран приложения — это экран-заставка **Splash Screen**, который демонстрируется пользователю во время загрузки приложения.

На первый взгляд он не несет особой информационной нагрузки, на нем пользователь не может ничего нажать и сделать, даже много текста на нем не разместить. Без загрузочного экрана можно вообще обойтись, что и делают многие разработчики. С другой стороны, если нашему приложению при первом включении требуется время на загрузку данных или проверку подключений, по загрузочному экрану пользователь поймет, что приложение не «зависло» и не «умерло», и, вместо того чтобы закрыть его и бежать писать плохой отзыв, подождет требуемые несколько секунд и останется нашим пользователем.

Но нам, как разработчикам, всегда следует помнить, что современный пользователь не любит ждать (даже секунду) и не хочет видеть то, что не способствует достижению его цели. Поэтому, если загрузочный экран все-таки нужен, желательно разместить на нем радующую глаз графику, напоминающую о содержании приложения, анимацию, а лучше всего отображать статус или процесс загрузки.

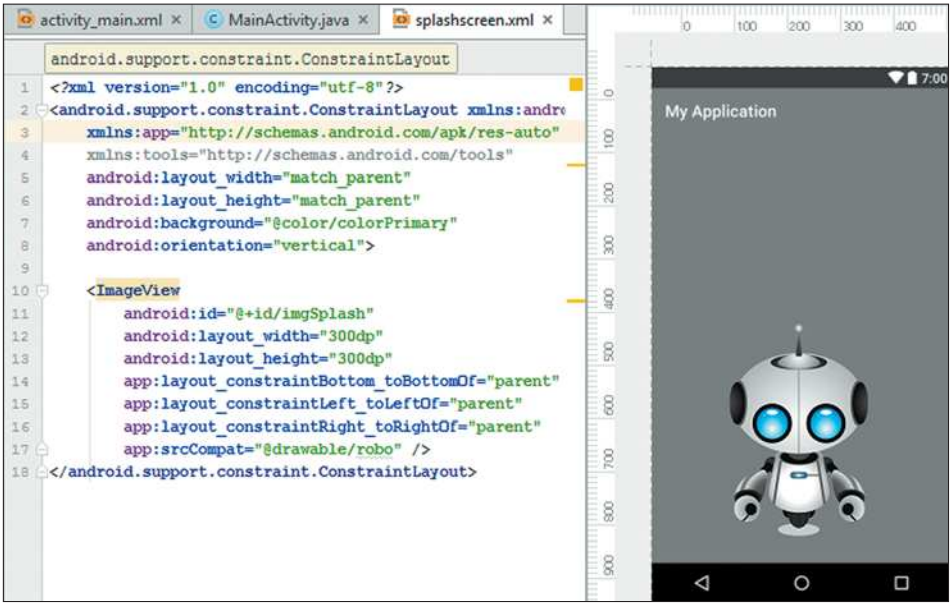
Вернемся к нашему приложению (можно продолжить работать над предыдущим проектом или создать новый).

Для того чтобы описать загрузочный экран, создадим новый файл **xml-разметки** по аналогии с **activity_main.xml**. Правой кнопкой мыши нажимаем на папку проекта **res/layout**, далее выбираем пункт **New → Layout resource file** (*файл ресурсов макета*).

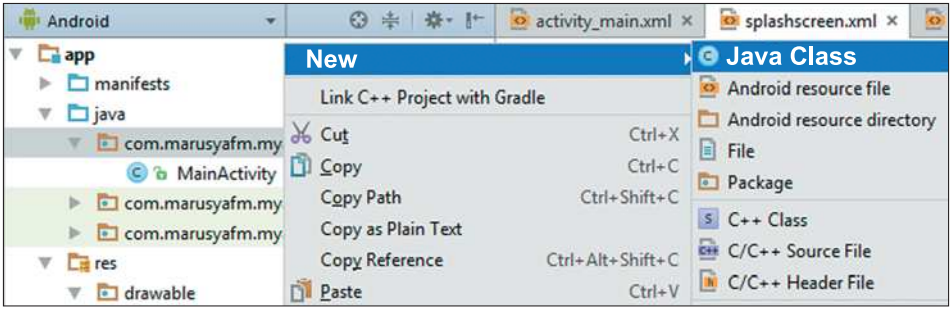


Зададим файлу имя splashscreen и нажмем **Ок**.

Придадим загрузочному экрану нужный вид, например разместим на нем изображение, как на логотипе приложения:



Для описания действий с этим экраном (например, его появления и последующего перехода к главному экрану приложения) нужно создать **новый Java-класс**: нажимаем правой кнопкой мыши на папке, в которой хранится класс **MainActivity**, и выбираем пункт **New → Java Class** (*Java-класс*).



Зададим классу имя SplashActivity и нажмем **Ок**.

В методе **onCreate()** объявляем новое намерение **Intent**, которое после выполнения всех действий класса **SplashActivity** «передаст управление» классу **MainActivity**. А поскольку **MainActivity**, как «главный», всегда запускается по умолчанию, воспользуемся методом **startActivity()** (*запустить Activity*), чтобы первым запустился загрузочный экран, как описано в intent.

activity_main.xml × splashscreen.xml × SplashActivity.java ×

```
1  package com.marusyafm.myapplication;
2
3  import android.content.Intent;
4  import android.os.Bundle;
5  import android.support.v7.app.AppCompatActivity;
6
7  public class SplashActivity extends AppCompatActivity {
8
9      @Override
10     protected void onCreate(Bundle savedInstanceState) {
11         super.onCreate(savedInstanceState);
12
13         Intent intent = new Intent(SplashActivity.this, MainActivity.class);
14         startActivity(intent);
15         finish();
16     }
17
18 }
```

В этом случае загрузочный экран будет отображаться столько времени, сколько нужно для загрузки приложения. Это самый правильный вариант с точки зрения разработки. Однако наше приложение пока слишком «легкое» — в нем всего один экран и совсем нет данных. В результате на загрузку ему потребуется доля секунды и мы не сможем оценить нашу работу над заставкой.

Бывают случаи, когда необходимо показывать загрузочный экран какое-то время, например, если на нем все же есть некоторая важная информация. Представим, что у нас именно такой случай, и установим отображение заставки в течение 3 секунд.

Допишем в класс **SplashActivity** следующий код:

```
8  public class SplashActivity extends AppCompatActivity {
9
10     Handler handler;
11
12     @Override
13     protected void onCreate(Bundle savedInstanceState) {
14         super.onCreate(savedInstanceState);
15         setContentView(R.layout.splashscreen);
16
17         handler=new Handler();
18         handler.postDelayed(new Runnable() {
19             @Override
20             public void run() {
21                 Intent intent = new Intent(SplashActivity.this, MainActivity.class);
22                 startActivity(intent);
23                 finish();
24             }
25         },3000);
26     }
```

где **Handler** (*обработчик*) — это класс для создания и управления фоновыми потоками, выполняющимися в любое заданное время); метод **postDelayed()** (*задержка сообщения*, строка № 18) нужен для задержки выполнения на определенное время; наш **Intent** (строка № 21) помещен внутри метода **run()**; **3000** — время задержки, то есть увеличение времени отображения заставки в миллисекундах (3 секунды = 3*1000 миллисекунд).

Еще одна деталь — заставки обычно отображаются на полный экран, то есть без «шапки» приложения, которая называется **ActionBar** (*панель действий*). Чтобы это реализовать, нужно установить загрузочному экрану свой стиль. Речь о стилях подробнее пойдет в следующем уроке, а пока откроем файл **styles.xml** (папка **res/values**), скопируем код уже прописанной там основной темы и изменим атрибут **name** (*имя темы*) на SplashScreenTheme, а атрибут **parent** (*родительская тема*) на Theme.AppCompat.NoActionBar (стандартная тема без панели действий).

activity_main.xml × | splashscreen.xml × | SplashActivity.java × | **styles.xml** ×

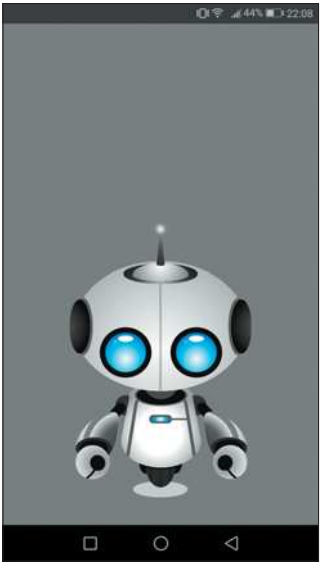
```
1  <resources>
2
3      <!-- Base application theme. -->
4      <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
5          <!-- Customize your theme here. -->
6          <item name="colorPrimary">@color/colorPrimary</item>
7          <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
8          <item name="colorAccent">@color/colorAccent</item>
9      </style>
10
11     <style name="SplashScreenTheme" parent="Theme.AppCompat.NoActionBar">
12         <item name="colorPrimary">@color/colorPrimary</item>
13         <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
14         <item name="colorAccent">@color/colorAccent</item>
15     </style>
16
17 </resources>
```

Осталось прописать информацию о нашем загрузочном экране в файле манифеста. Значение атрибута **name** уже прописанной там Activity изменим на .SplashActivity, добавим новый атрибут **theme** (*тема*) с названием только что созданной нами темы. Информацию об Activity **MainActivity** добавляем в новом теге (строка № 21).

splashscreen.xml × SplashActivity.java × AndroidManifest.xml ×

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.marusyafm.myapplication">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="My Application"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity
            android:name=".SplashActivity"
            android:theme="@style/SplashScreenTheme">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".MainActivity" />
    </application>
```

Готово! Запускаем приложение и в течение трех секунд любуемся созданной заставкой, прежде чем откроется главный экран приложения.



4.3.1. Анимация элементов и класс AnimationUtils

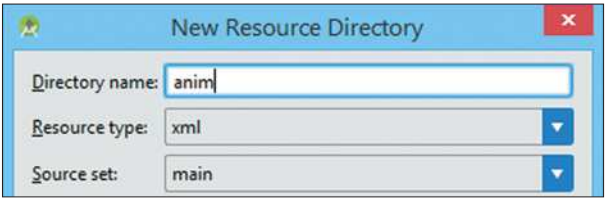
Можно сделать так, чтобы картинка плавно проявлялась или, например, мигала. За анимацию элементов отвечает класс **AnimationUtils**.

Один из его наследников, класс **AlphaAnimation** (*анимация прозрачности*), отвечает за изменение прозрачности объекта, иначе говоря, его выцветание. Это изменение можно настроить «в обе стороны», и элемент будет появляться или исчезать.

Есть еще классы **ScaleAnimation** (*анимация масштабирования*) для изменения размеров элемента, **RotateAnimation** (*анимация вращения*) для управления вращением элемента, **TranslateAnimation** (*анимация перемещения*) — для изменения местоположения элемента.

Каждое действие анимации прописывается в отдельном xml-файле, и все эти xml-файлы вместе хранятся в папке проекта **res/anim**.

Если такой папки в проекте нет, ее нужно создать: нажимаем правой кнопкой мыши на папку **res**, затем **New** → **Android Resource Directory** (*папка ресурсов Android*). В открывшемся окне задаем параметры **Directory name** anim и **Resource type** xml. Оставив остальные параметры по умолчанию, нажимаем **OK**.



В папку **anim** добавляем новый файл для описания анимации нашего изображения — его появления. Правой кнопкой мыши щелкнем по папке **anim**, затем **New** → **Animation resource file** (*файл ресурсов анимации*). Таким файлам обычно задают стандартные имена в соответствии с типом анимации, поэтому назовем файл fadein (*проявляться, выцветать*).

В открывшемся файле прописываем атрибуты нашей анимации:

SplashActivity.java × fadein.xml ×

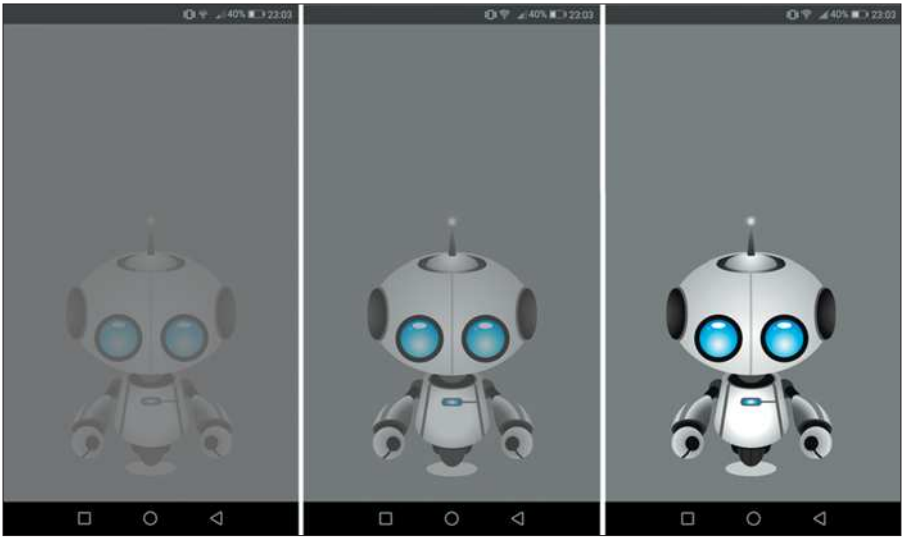
```
1  <?xml version="1.0" encoding="utf-8"?>
2  <alpha xmlns:android="http://schemas.android.com/apk/res/android"
3      android:fromAlpha="0.0"
4      android:toAlpha="1.0"
5      android:duration="4000">
6
7  </alpha>
```

Атрибуты **fromAlpha** и **toAlpha** задают начало и конец анимации выцветания, им присваиваются значения от 0.0 (полная прозрачность) до 1.0 (полная непрозрачность) и **duration** (*продолжительность*) — время анимации в миллисекундах.

Теперь нужно упомянуть нашу анимацию в Java-коде. В классе **SplashActivity** объявляем и определяем изображение, для которого собственно задается анимация. Затем объявляется сама анимация как объект класса **Animation**, для которой методом **loadAnimation()** (*загрузить анимацию*) в качестве ресурса устанавливается созданный нами файл **fadein**. Запускается анимация с помощью метода **startAnimation()** (*начать анимацию*).

```
11  public class SplashActivity extends AppCompatActivity {
12
13      private ImageView imgSplash;
14
15      Handler handler;
16
17      @Override
18      protected void onCreate(Bundle savedInstanceState) {
19          super.onCreate(savedInstanceState);
20          setContentView(R.layout.splashscreen);
21
22          imgSplash = (ImageView) findViewById(R.id.imgSplash);
23          Animation animation = AnimationUtils
24                                  .loadAnimation(getApplicationContext(),R.anim.fadein);
25          imgSplash.startAnimation(animation);
26
```

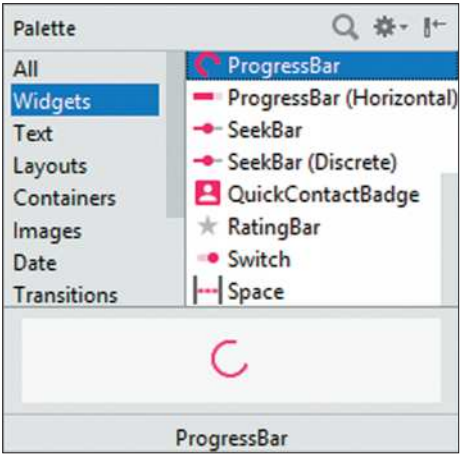
Запустим приложение и убедимся, что теперь наш робот плавно проявляется на загрузочном экране.



По аналогии нужно добиться, чтобы изображение исчезало или мерцало. Не забываем про описание элементов анимации в отдельных xml-файлах.

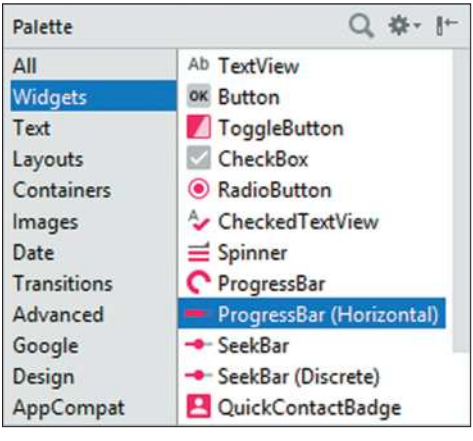
4.3.2. ProgressBar — индикатор загрузки

Элемент **ProgressBar** (*индикатор загрузки*) отображает ход загрузки или выполнение какого-либо процесса в **Android Studio**. Он бывает круговой и горизонтальный. **ProgressBar** помещен в группу **Widgets** (виджетов) — интерактивных элементов, изменяющихся с течением времени или по заданным условиям.

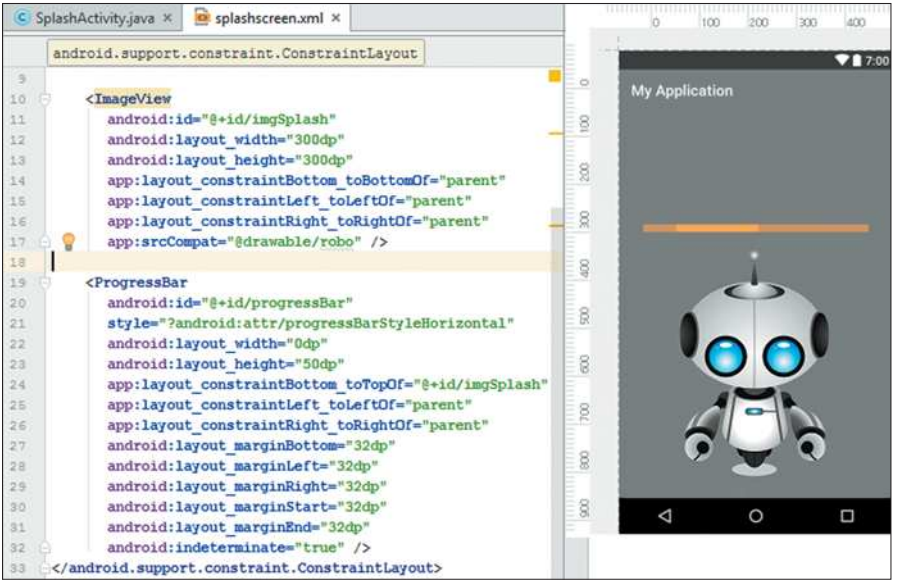


Рассмотрим его возможности на примере горизонтального.

Перетаскиваем на загрузочный экран из палитры элементов элемент **ProgressBar (Horizontal).**

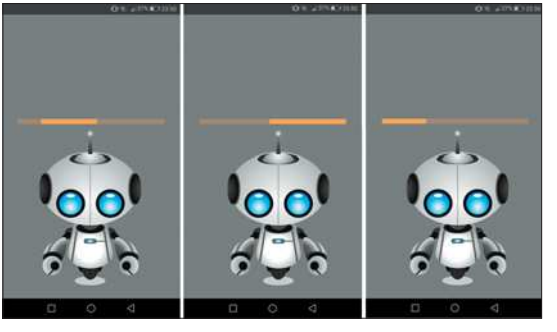


Задаем его свойства:



С атрибутами, отвечающими за размеры и расположение, мы уже давно знакомы, а с последним атрибутом **indeterminate** (*неопределенность*) нет. Этот атрибут с заданным значением true означает, что индикатор загрузки будет бесконечно «бегать по кругу», отображая сам факт загрузки, но не процент выполнения. Если этому атрибуту задать значение false, то пользователь увидит ход загрузки, но в этом случае нужны еще два параметра: атрибуты **max** (максимальное значение процента выполнения, обычно 100) и **progress** (начальный прогресс, может быть 0, 10, 20 и так далее).

Запускаем приложение и наблюдаем за появившимся на заставке индикатором процесса загрузки.



Задания

1. По аналогии сделать круговой **ProgressBar**.
2. Наполнить контентом главный экран приложения.

4.4. Темы и стили

Возвращаясь к вопросам дизайна, поговорим о **стилях** и **темах** в приложении.

4.4.1. Стили

Стиль (в разработке мобильных приложений) — это набор настроек отдельно взятого элемента приложения, которые можно отнести к его дизайну (цвет, шрифт, цвет фона и так далее).

Создадим проект Styles со стандартным набором настроек. На главном экране разместим произвольный набор элементов, например четыре текста и две кнопки.

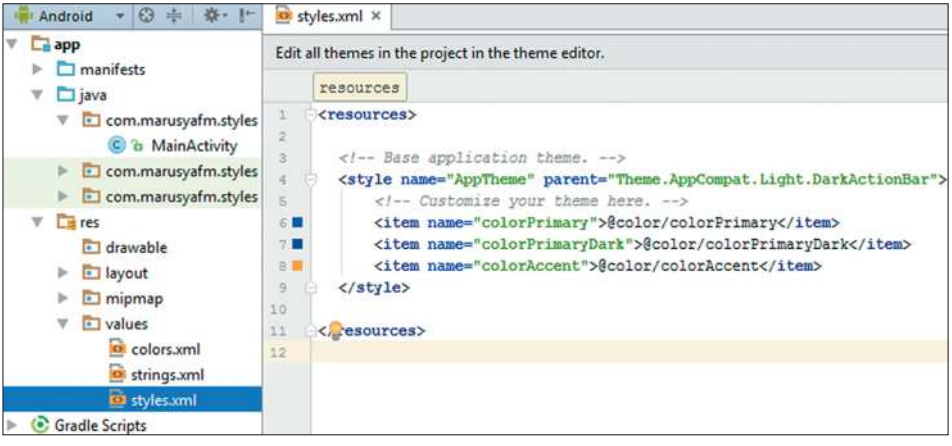
Зададим свойства первого элемента **TextView** — цвет, размер текста, начертание:

```
<TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Первый TextView"
    android:textColor="@color/colorAccent"
    android:textSize="24sp"
    android:textStyle="bold"
    app:layout_constraintBottom_toTopOf="@id/textView2"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

Теперь, если мы захотим задать второму (и последующим) **TextView** такие же свойства, то можем снова задать их по одному в режиме дизайна или скопировать нужные строки в режиме кода. Второй способ явно быстрее, к тому же не нужно запоминать, какие именно значения мы задавали для каких свойств.

Но есть и третий способ — самый быстрый и правильный: объединить все требуемые значения свойств в **стиль** и в следующий раз искать и задавать не несколько свойств, а только одно — **style**.

Стиль — это ресурс, поэтому описания всех стилей приложения хранятся в файле ресурсов **styles.xml** в папке **res/values**:



Сейчас в этом файле описана главная тема приложения, но о темах мы поговорим чуть позже.

Стиль имеет два атрибута: обязательный **name** (*имя*) и необязательный **parent** (*родитель*) — стиль или тема, от которой создаваемый стиль будет наследовать характеристики.

Добавим новый стиль и назовем его, например, BigOrangeTextStyle.

```
    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>

    <style name="BigOrangeTextStyle">

    </style>

</resources>
```

Каждое свойство, входящее в стиль, прописывается как отдельный пункт — **item**. Каждому пункту задается имя и значение.

Чтобы убедиться, насколько удобно использование стилей, зададим в описании этого стиля сразу шесть свойств — цвет, размер и стиль текста, заполнение по центру, ширину и высоту:

```
<style name="BigOrangeTextStyle">
    <item name="android:textColor">@color/colorAccent</item>
    <item name="android:textSize">24sp</item>
    <item name="android:textStyle">bold</item>
    <item name="android:gravity">center</item>
    <item name="android:layout_width">wrap_content</item>
    <item name="android:layout_height">wrap_content</item>
</style>
```

Вернемся к файлу **activity_main.xml**. Найдем разметку последнего элемента **TextView** и зададим ему всего один атрибут — **style**. Судя по подсказкам **Android Studio**, есть уже довольно много стандартных стилей. Еще в одном из первых уроков мы убирали у кнопки тень и границы, выставив стиль Base.Widget.AppCompat.Button.Borderless, а сейчас пропишем название созданного нами стиля:

```
<TextView
    android:id="@+id/textView4"
    android:text="Последний TextView, теперь точно!"
    app:layout_constraintTop_toBottomOf="@id/textView3"
    app:layout_constraintBottom_toTopOf="@id/button"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    style="@style/BigOrangeTextStyle"/>

<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Кнопка ОК"
    app:layout_constraintTop_toBottomOf="@id/textView4"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintRight_toLeftOf="@id/button2"
    app:layout_constraintLeft_toLeftOf="parent"/>
```

В режиме предварительного просмотра видим, как сразу же изменился стиль текста. Таким образом с помощью всего одного атрибута можно задавать стили всем элементам по всему приложению.

Задание

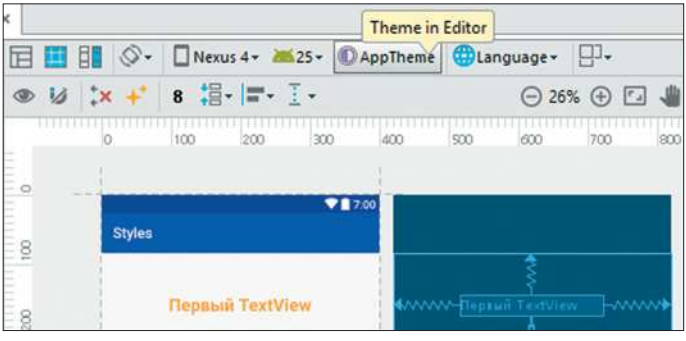
Создать еще один стиль и применить его к расположенным на главном экране кнопкам.

4.4.2. Темы

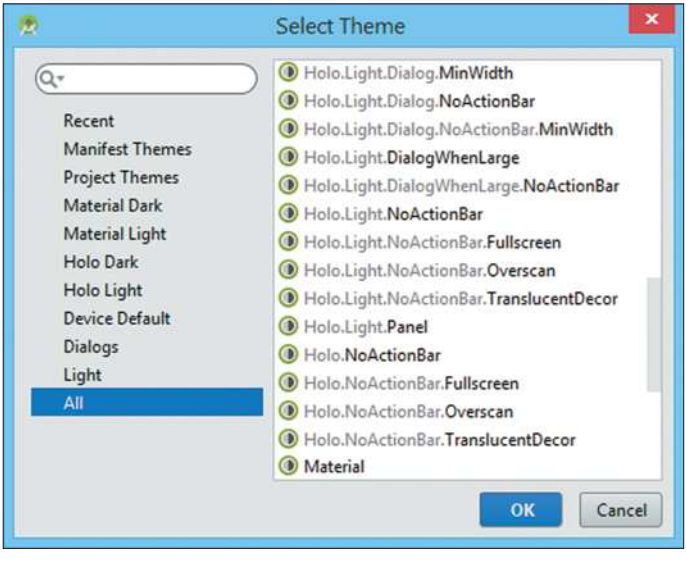
Тема (в разработке мобильных приложений) — это стиль, который применяется ко всему приложению или как минимум ко всей Activity.

Вернемся к файлу **styles.xml**. Как мы уже говорили, первой в нем прописана главная тема приложения. Она названа AppTheme и является дочерней темой Theme.AppCompat.Light.DarkActionBar. По умолчанию в ней прописаны три главных цвета темы: colorPrimary, colorPrimaryDark и colorAccent.

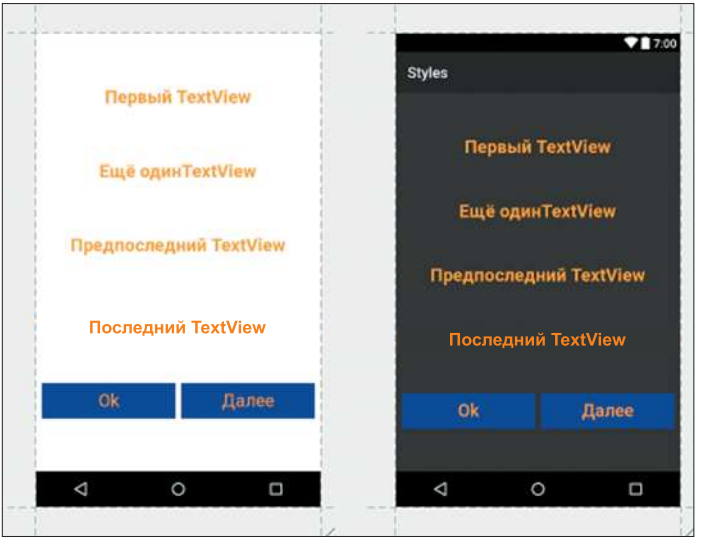
Здесь мы можем добавлять и удалять параметры темы, равно как и прописывать новые темы, но, как всегда, у **Android Studio** для нас уже есть варианты готовых тем. Переход к ним осуществляется по кнопке **AppTheme**, расположенной над макетом в файлах xml-разметки:



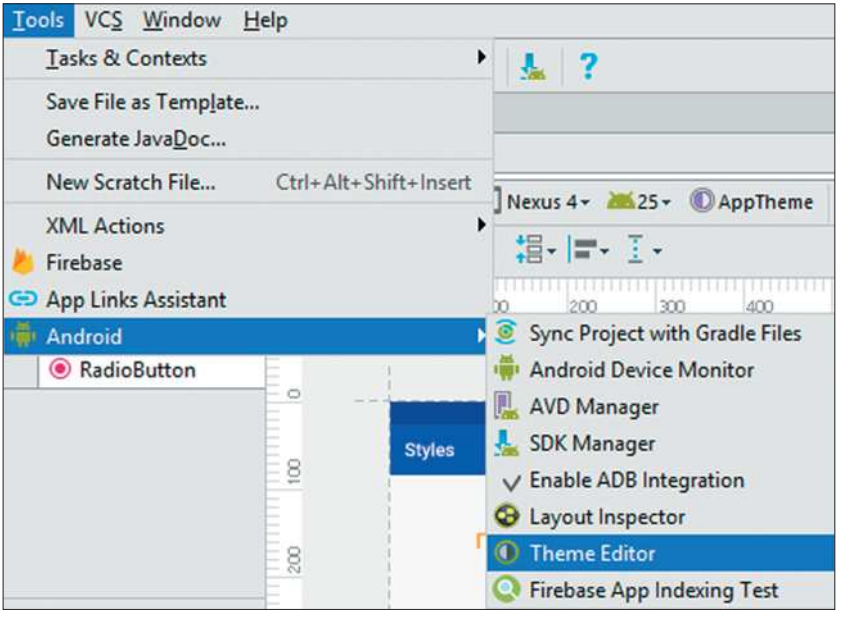
В открывшемся окне можно увидеть список тем, разделенных на категории:



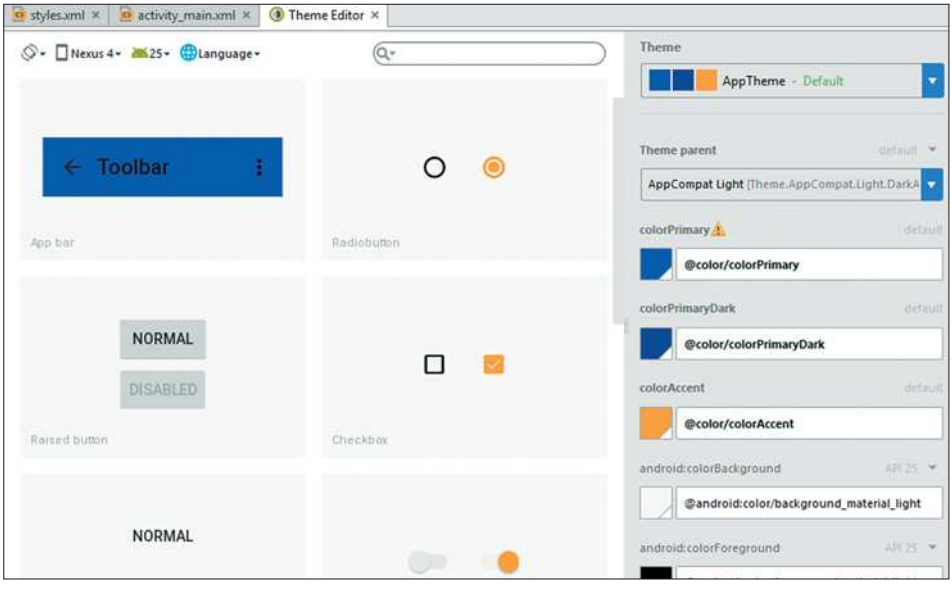
Например, тема Holo.Light.NoActionBar.Fullscreen убирает панель ActionBar и даже панель состояния, позволяя приложению занимать весь экран смартфона; тема Material.Dark делает экран и ActionBar темными; тема Theme.AppCompat.DayNight вообще меняется со светлой на темную в зависимости от времени суток. И таких вариантов достаточно много.



Кроме того, выбранную тему можно настроить и без файла ресурсов — через **Theme Editor** (*редактор тем*):



В редакторе тем можно настраивать различные параметры тем и тут же оценивать изменения в режиме предварительного просмотра.



Выбранную тему можно применить как ко всему приложению, так и к отдельной Activity. Это делается в файле манифеста **AndroidManifest.xml**

Тема, примененная ко всему приложению по умолчанию, записана в теге **application** (строка № 11).

styles.xml × activity_main.xml × AndroidManifest.xml ×

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      package="com.marusyafm.styles">
4
5      <application
6          android:allowBackup="true"
7          android:icon="@mipmap/ic_launcher"
8          android:label="Styles"
9          android:roundIcon="@mipmap/ic_launcher_round"
10         android:supportsRtl="true"
11         android:theme="@style/AppTheme">
12         <activity android:name=".MainActivity">
13             <intent-filter>
14                 <action android:name="android.intent.action.MAIN"/>
15
```

```
16              <category android:name="android.intent.category.LAUNCHER"/>
17          </intent-filter>
18      </activity>
19  </application>
20
21  </manifest>
```

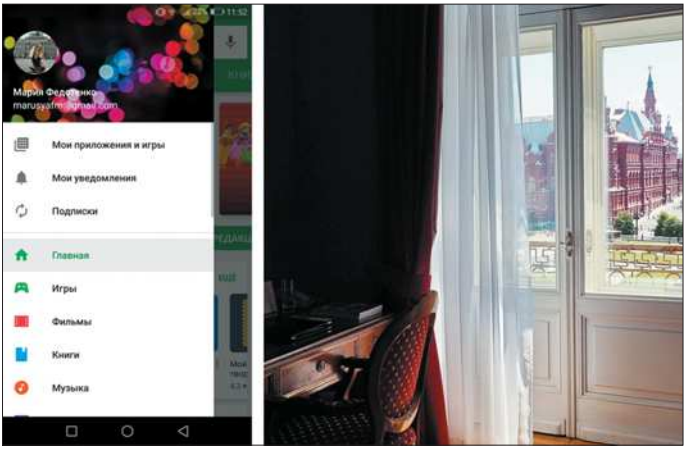
Если же нужно применить отдельную тему к конкретной Activity, атрибут **theme** прописывается в теге **activity** после атрибута **name** (строка № 12).

4.5. Меню. Виды меню

Навигация по приложению (а точнее, качество ее организации) также является одним из важнейших аспектов юзабилити. Один из вариантов реализации навигации — **меню.**

На сегодняшний день в мобильных приложениях используется три вида меню:

1. Меню-шторка. Обычно это основное меню, которое разворачивается на бóльшую часть экрана и вызывается либо свайпом (проведением пальцем) от левого края экрана, либо нажатием на кнопку с тремя полосками, которая обычно располагается в левом верхнем углу, в ActionBar приложения.



Кстати, в официальной терминологии Google эта кнопка называется **«гамбургер»**.

2. Главное меню. Вызывается обычно из правого верхнего угла ActionBar, как, например, в приложении для изучения иностранных языков Duolingo. Кнопка его вызова обозначается тремя точками и называется **«кебаб»**[1].

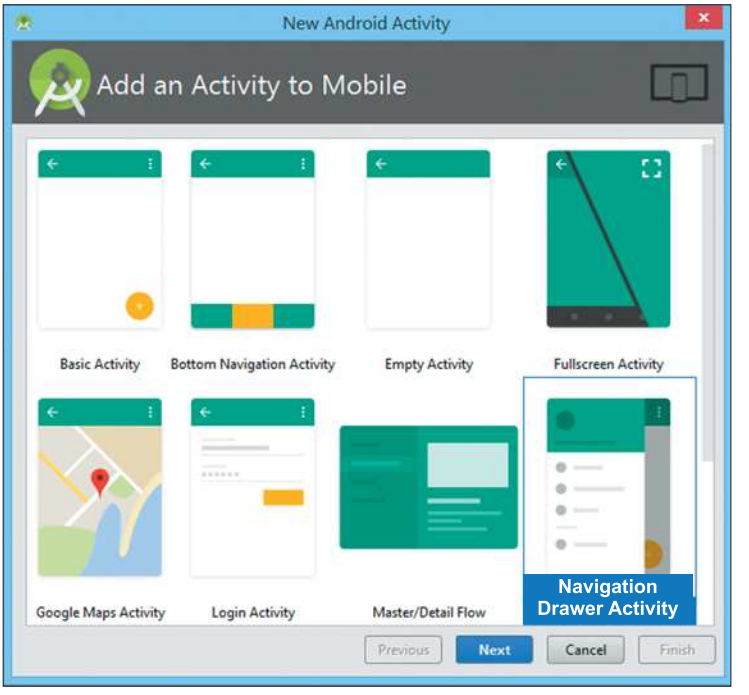
3. Контекстное меню. Это меню может вызываться по нажатию на любой элемент приложения (кнопку, текст, системную кнопку и так далее), как правило, долгим нажатием. Представляет собой аналог контекстного меню, которое мы часто встречаем при работе за компьютером, вызывая его нажатием правой кнопкой мыши.

На сегодняшний день в своем классическом виде в мобильных приложениях практически не используется. Его функции распределяются между кнопкой **FloatingActionButton** и остальными видами меню (путем их некоторого видоизменения). Кнопка **FloatingActionButton** располагается в правом нижнем углу поверх всего приложения. Иногда по нажатию на нее возникают еще кнопки, образуя новый вид меню. Поэтому в данном уроке создание контекстного меню мы рассматривать не будем.

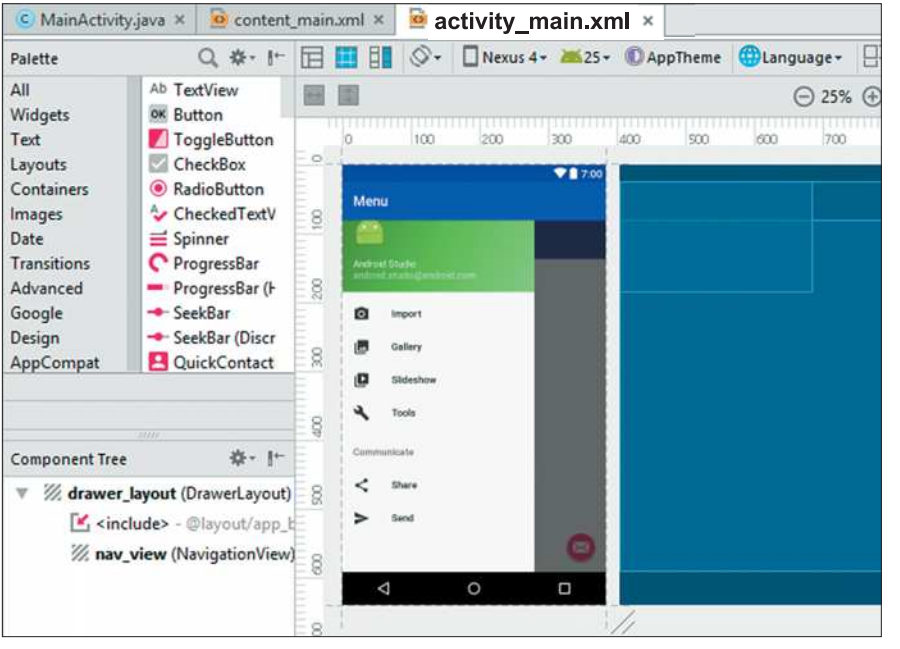
Рассмотрим методы создания разных меню. Любой из видов можно создать написанием Java-кода вручную, но в **Android Studio** достаточно инструментов, чтобы сделать это быстрее и проще.

Создадим новый проект Android Studio с именем My Menu. Параметры оставим по умолчанию, но на шаге выбора Activity выберем **Navigation Drawer Activity** (*экран–построитель навигации*):

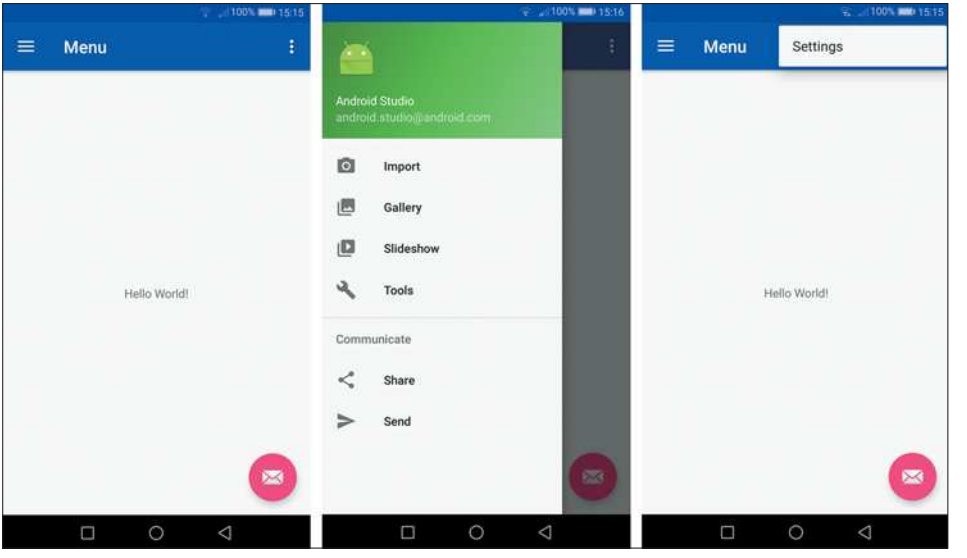
[1] Google — американская компания, а в американском английском языке под словом «kebab» подразумеваются практически все приготовленные на огне мясные блюда кавказской, ближневосточной и азиатской кухни. Поэтому, если быть точным, прототипом кнопки стал шашлык.



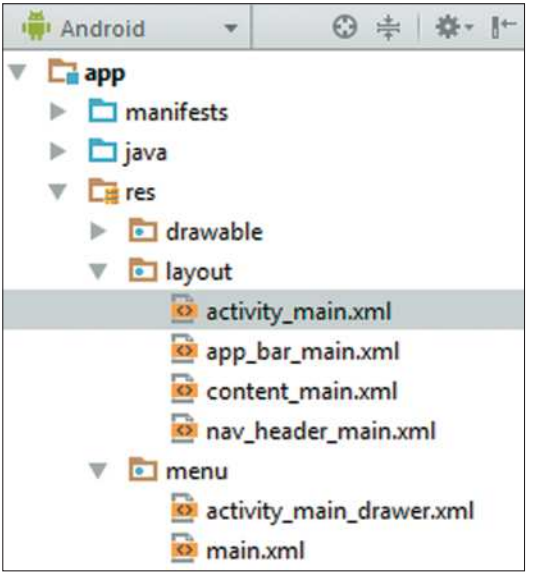
Открывшийся файл **activity_main.xml** выглядит необычно и немного странно: на макете элементы перекрывают друг друга, а в дереве компонентов нет ни одного знакомого названия:



Чтобы разобраться, «что здесь происходит», запустим проект. Мы видим стартовый экран с привычным для нас текстом «HelloWorld!», но в ActionBar приложения появились две новые иконки, те самые, которые мы привыкли видеть во всех приложениях для вызова меню — «гамбургер» и «кебаб»:



Теперь, чтобы разобраться, где это все хранится и как работает, обратимся к структуре проекта:

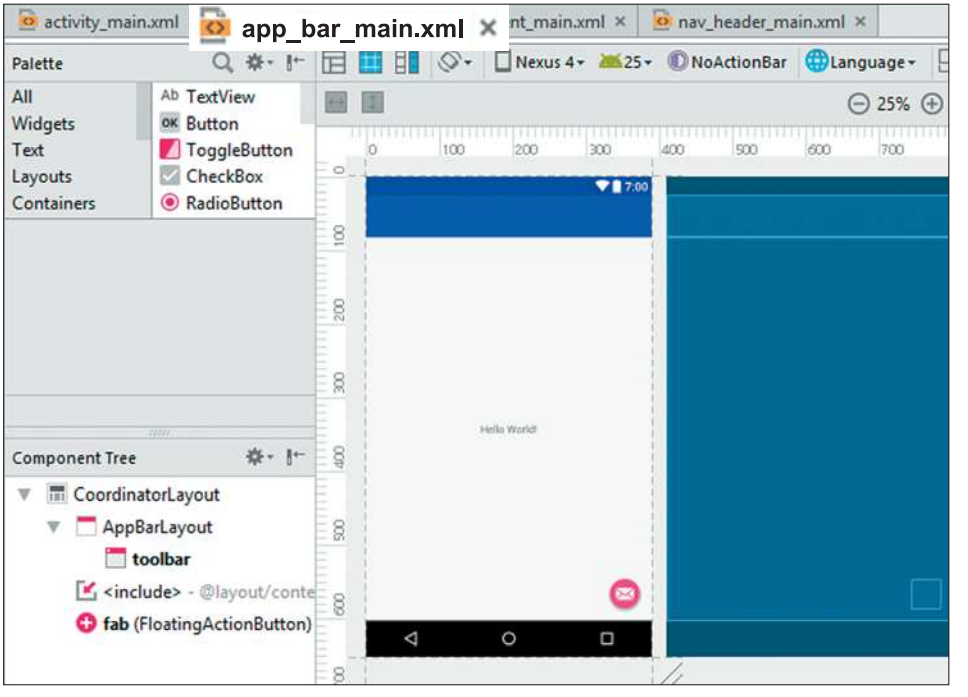


Откуда взялись сразу четыре xml-файла в папке **layout** и два в **menu**?

Рассмотрим каждый из них.

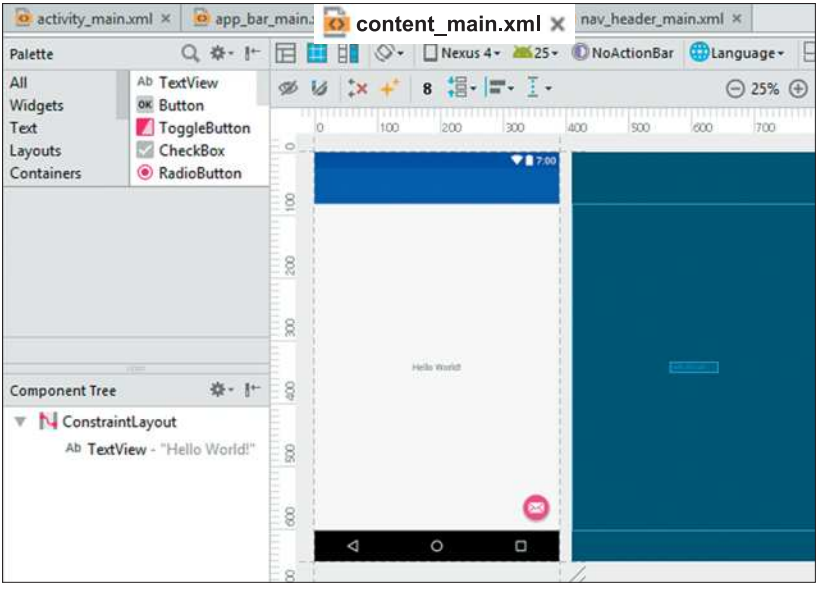
Файл **activity_main.xml** мы уже видели. В режиме дизайна показан практически реальный вид меню-шторки. Основная особенность — в этой Activity мы почти ничего не можем изменить, потому что в ней хранятся только ссылки на все составные части.

Файл app_bar_main.xml



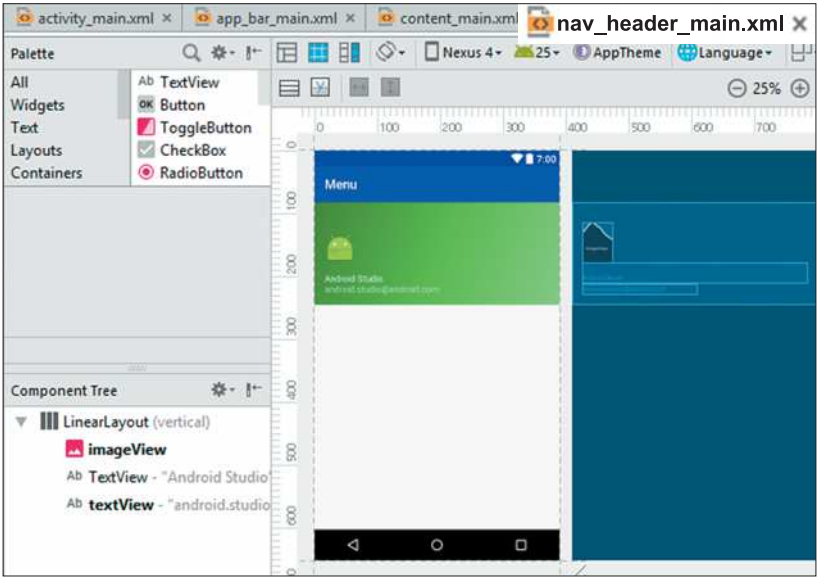
В этом файле хранится разметка для элементов, общих для всего приложения, точнее для экранов, на которые распространяет свое действие меню-шторка. Сейчас это кнопка **FloatingActionButton** (*плавающая кнопка действия*) в правом нижнем углу экрана, а также панель ActionBar (внутри макета AppBarLayout) и ссылка на контейнер, содержащий основной контент страницы. Кнопка **FloatingActionButton** предназначена для того, чтобы самые важные функции приложения всегда находились «под рукой» у пользователя.

Файл content_main.xml



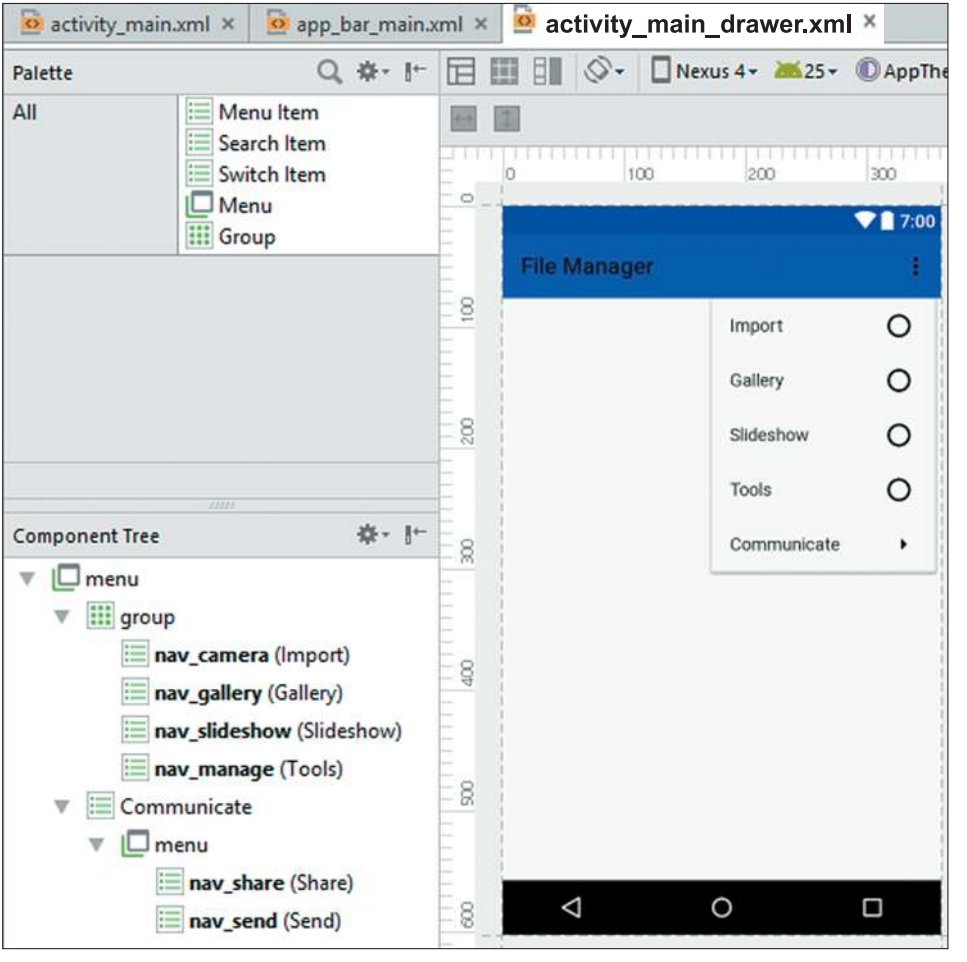
Здесь можно изменить контент главной страницы, то есть именно те элементы, которые отображаются на ней. Сейчас это текст HelloWorld.

Файл nav_header_main.xml



Этот файл разметки хранит в себе **header** (*шапку, заголовочную часть*) меню-шторки. Сейчас в нем логотип приложения и два текстовых элемента.

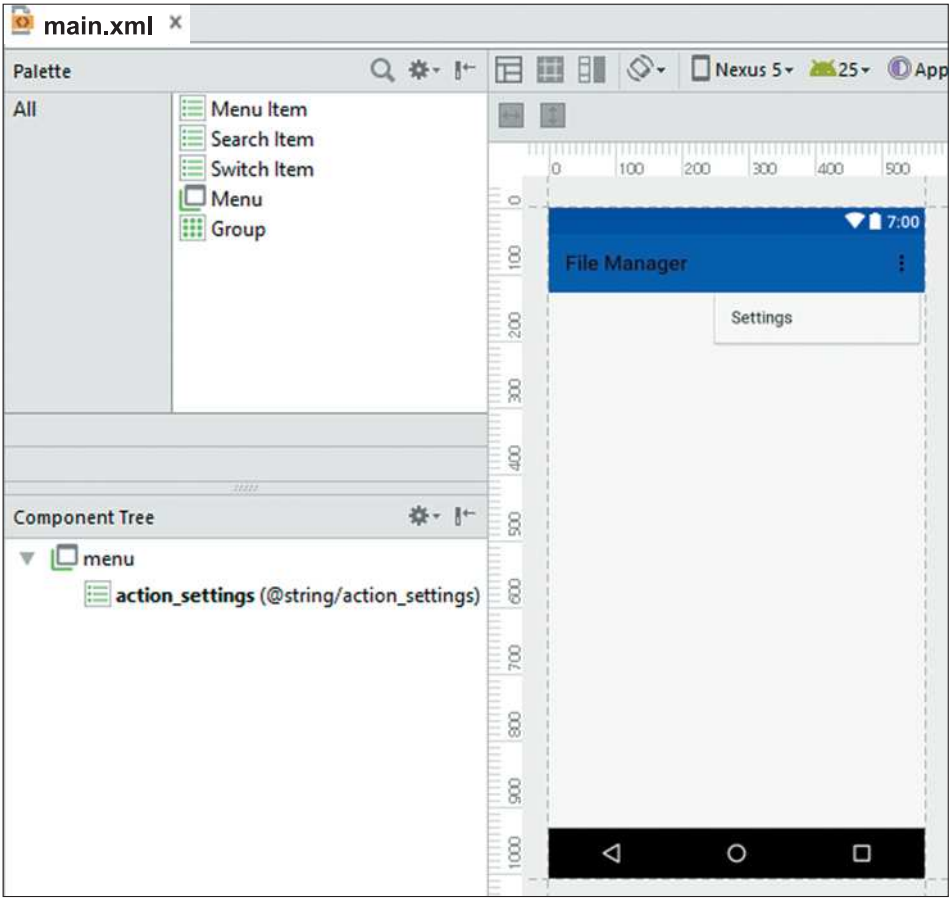
Файл activity_main_drawer.xml



Здесь хранится структура меню-шторки: названия, порядок и свойства его пунктов.

Файл main.xml

В этом файле хранится макет главного меню. Сейчас в нем всего один пункт — **Settings** (*настройки*). В официальной терминоло-

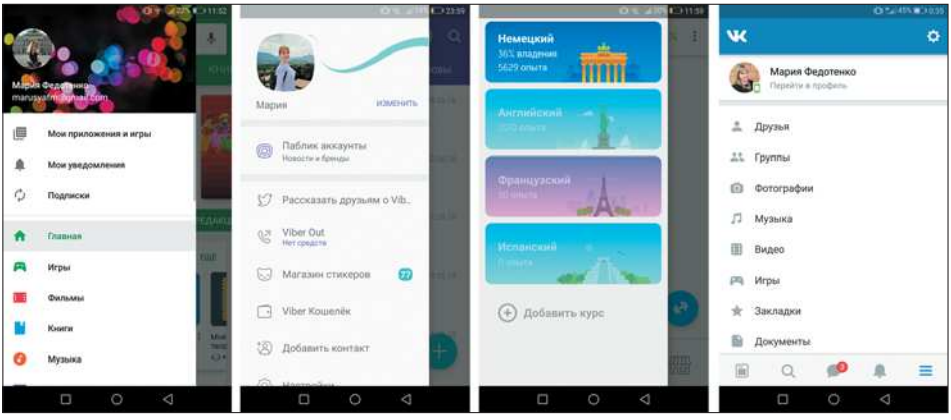


гии Google это меню все еще называется главным, но на сегодняшний день в него обычно выносят такие пункты, как «Настройки», «О приложении», «Оценить приложение» и так далее, то есть моменты, которые несут в себе информацию о приложении, но напрямую не относятся к его основному содержанию.

Итак, со структурой проекта мы разобрались, но, чтобы наверняка понять, как это все работает, воспроизведем меню каждого типа известных приложений.

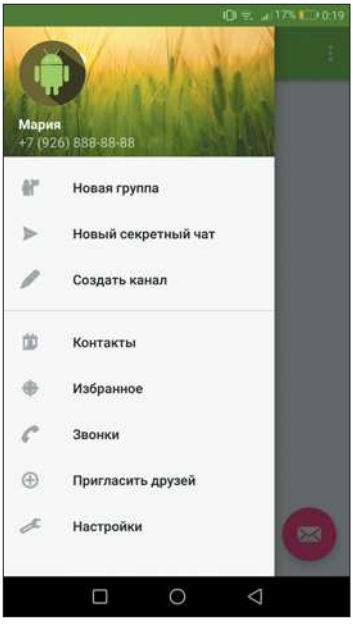
4.5.1. Меню-шторка

Меню-шторку используют практически все приложения от Google (Google Play, Google-переводчик и другие), Viber, Duolingo, ее аналог использует ВКонтакте.



Задание

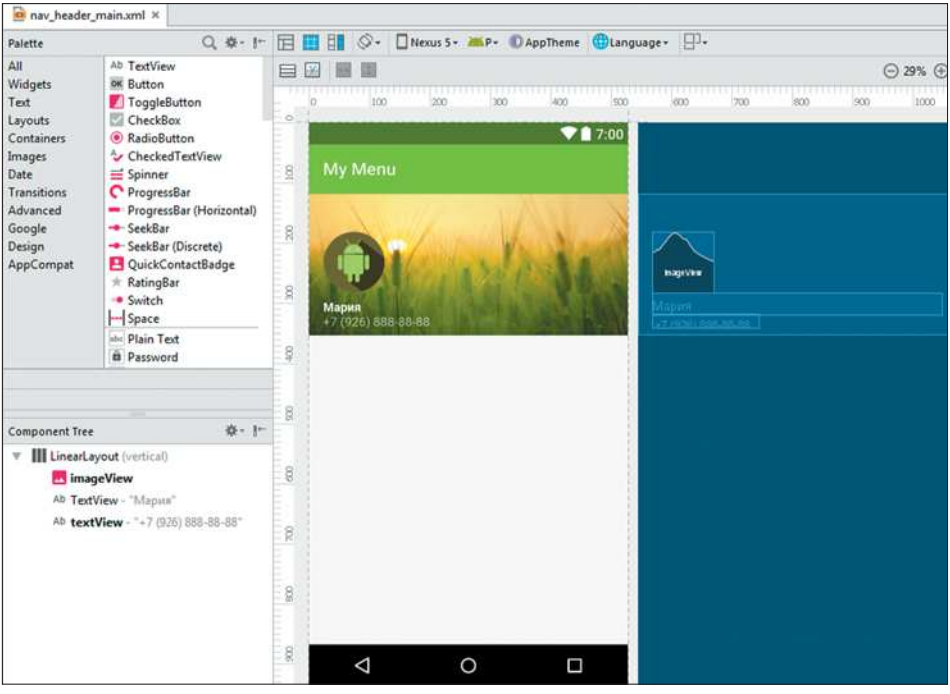
Создать меню-шторку для мессенджера[1].



Для начала подготовим ресурсы: изображения для аватара и фона (значки для пунктов меню можно взять стандартные). Далее перейдем в **nav_header_main.xml** и изменим «шапку» нашего ме-

[1] Изображение робота Android используется здесь в соответствии с лицензией Creative Commons версии 3.0 и является копией или модификацией логотипа, созданного Google.

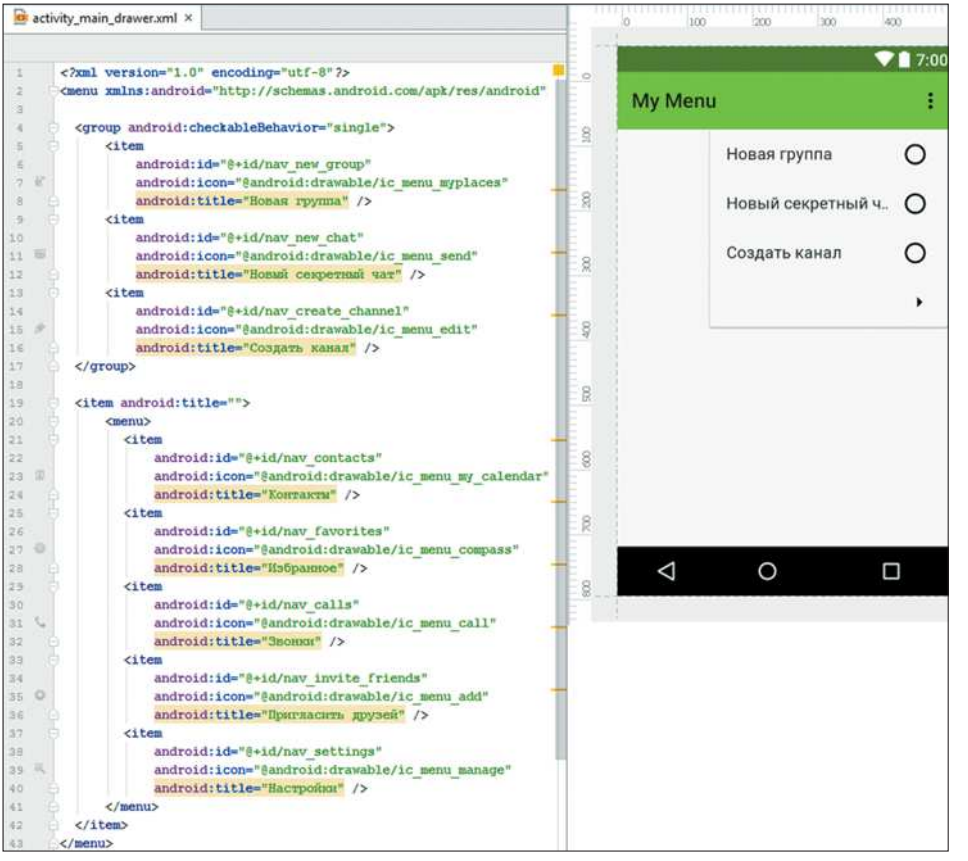
ню-шторки. Изображение-ресурс содержащегося в нем элемента **ImageView** заменим нашим изображением-аватаром. В первом **TextView** зададим наше имя, а во втором — номер телефона. В качестве фона для макета **LinearLayout** (свойство **background**) установим изображение, выбранное нами ранее для фона:



Для изменения пунктов меню-шторки перейдем в файл **activity_main_drawer.xml**. Сейчас в меню по умолчанию четыре пункта в первом блоке (Import, Gallery, Slideshow и Tools), затем разделительная черта, название второго блока Communicate и в нем еще два пункта (Share и Send).

В нашем меню тоже два блока, но в первом только три пункта, а во втором — пять, и не отображается название. Давайте заменим меню по умолчанию нашим:

- в режиме кода удалим из первого блока один пункт и добавим три пункта во второй;
- зададим пунктам идентификаторы и названия;
- установим свои изображения в качестве значков пунктов меню и удалим название второго блока.



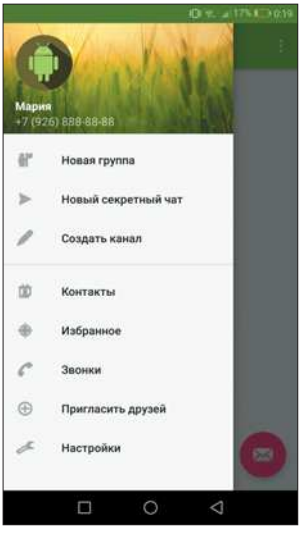
После изменения идентификаторов пунктов меню в разметке xml нужно изменить эти идентификаторы и в Java-классе **MainActivity**, чтобы избежать ошибок при сборке проекта:

activity_main_drawer.xml × MainActivity.java ×

```
77  @SuppressWarnings("StatementWithEmptyBody")
78  @Override
79  public boolean onNavigationItemSelected(MenuItem item){
80      // Handle navigation view item clicks here.
81      int id = item.getItemId();
82
83      if (id == R.id.nav_new_group) {
84          // Handle the camera action
85      } else if (id == R.id.nav_new_chat) {
86
87      } else if (id == R.id.nav_create_channel) {
```

```
88
89      } else if (id == R.id.nav_contacts) {
90
91      } else if (id == R.id.nav_favorites) {
92
93      } else if (id == R.id.nav_calls) {
94
95      } else if (id == R.id.nav_invite_friends) {
96
97      } else if (id == R.id.nav_settings) {
98
99      }
```

Запускаем приложение и видим, что меню выглядит в точности, как задумывалось:



Остался еще один вопрос — при нажатии на пункты меню они выделяются, но при этом ничего не происходит, экран не меняется. Это объясняется тем, что основной экран у нас один, а в обработчиках событий нажатия (как мы уже видели в **MainActivity**) ничего не прописано.

4.5.2. Фрагменты

Меню выбранного нами шаблона относится только к данной Activity, поэтому при нажатии на его пункты мы не можем переходить к другой Activity — в ней этих пунктов меню нет. Для таких случаев в **Android Studio** предусмотрен еще один вид эле-

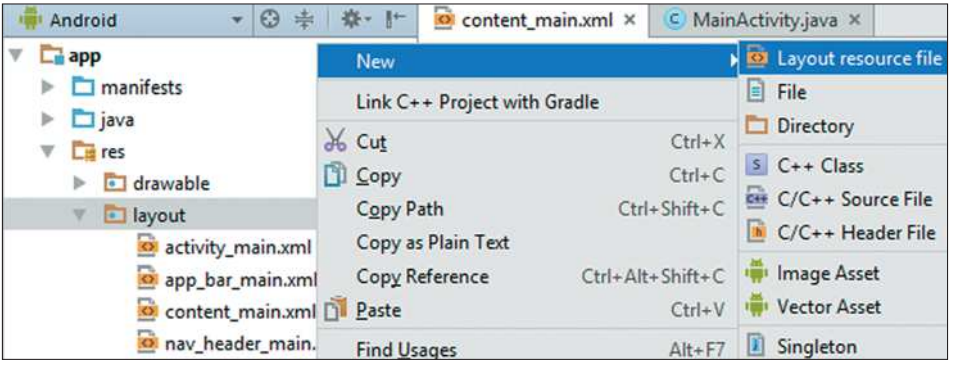
ментов-контейнеров — **Fragments** (*фрагменты*). Созданные нами фрагменты будут располагаться в контейнере, выделенном для контента, и сменять друг друга при выборе соответствующих пунктов меню.

Откроем файл **content_main.xml**, удалим из него элемент **TextView**, а самому макету **ConstraintLayout** зададим **id** container.

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

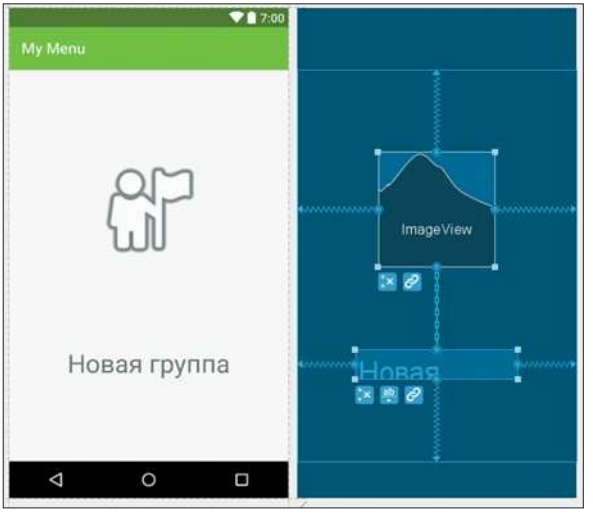
Теперь фрагменты будут сменять друг друга внутри макета.

Xml-разметка для фрагментов создается так же, как и для Activity, — щелчком правой кнопки мыши по папке **layout**, затем **New → Layout resource file:**



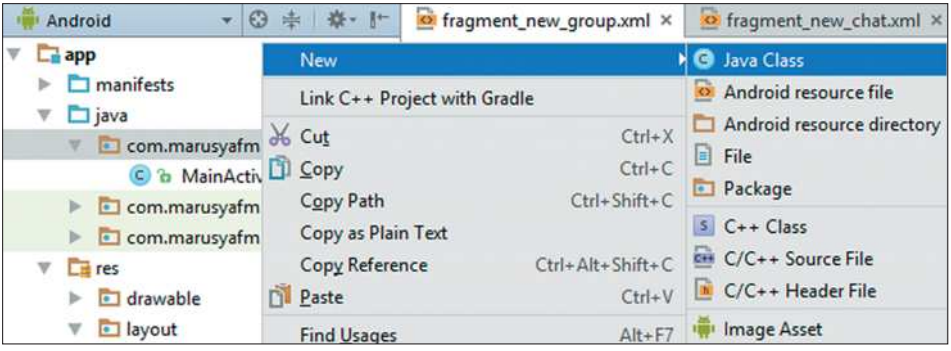
Зададим файлу имя fragment_new_group, **Root element** (*корневой элемент*) — ConstraintLayout и нажмем **OK**.

В созданном макете фрагмента разместим, например, изображение и текст, как в соответствующем пункте меню.



По аналогии создадим еще семь макетов фрагментов для остальных пунктов меню.

Для описания действий, совершаемых внутри каждого фрагмента, нужно создать Java-классы для их макетов:



Первому классу (для фрагмента **fragment_new_group**) присвоим имя NewGroupFragment, остальные его параметры оставим по умолчанию.

В открывшемся классе пропишем следующий код:

fragment_new_group.xml × NewGroupFragment.java ×

```
1  package com.marusyafm.mymenu;
2
3  import android.os.Bundle;
4  import android.support.v4.app.Fragment;
5  import android.view.LayoutInflater;
6  import android.view.View;
7  import android.view.ViewGroup;
8
9  public class NewGroupFragment extends Fragment {
10   @Override
11   public View onCreateView(LayoutInflater inflater, ViewGroup container,
12                            Bundle savedInstanceState) {
13       return inflater.inflate(R.layout.fragment_new_group, container, false);
14   }
15 }
```

Поясняем: для того чтобы созданный нами класс определялся именно как класс фрагмента, расширяем его классом **Fragment** (строка № 9). А в методе **onCreateView()** в качестве возвращаемого значения (строка № 13) прописываем объект inflater. Метод **inflate()** (*надувать*) предназначается для наполнения контейнера содержимым. В данном случае «наполнением» будет служить фрагмент **fragment_new_group**. Мы запишем его в качестве первого параметра.

По аналогии создадим классы для остальных фрагментов.

Теперь реализуем переключение между фрагментами в зависимости от выбранного пункта меню. В классе **MainActivity.java** найдем метод **onNavigationItemSelected(MenuItem item)** (*по выбору пункта навигационного меню*) и добавим в него несколько строк кода для управления нашими фрагментами:

```
79  @SuppressWarnings("StatementWithEmptyBody")
80  @Override
81  public boolean onNavigationItemSelected(MenuItem item) {
82
83      Fragment fragment = null;
84      Class fragmentClass = null;
85
86      // Handle navigation view item clicks here.
87      int id = item.getItemId();
88
89      if (id == R.id.nav_new_group) {
90          fragmentClass = NewGroupFragment.class;
91      } else if (id == R.id.nav_new_chat) {
92      } else if (id == R.id.nav_create_channel) {
93      } else if (id == R.id.nav_contacts) {
94      } else if (id == R.id.nav_favorites) {
95      } else if (id == R.id.nav_calls) {
96      } else if (id == R.id.nav_invite_friends) {
97      } else if (id == R.id.nav_settings) {
98      }
99
100     try {
101         fragment = (Fragment) fragmentClass.newInstance();
102     } catch (Exception e) {
103         e.printStackTrace();
104     }
105
106     FragmentManager fragmentManager = getSupportFragmentManager();
107     fragmentManager.beginTransaction().replace(R.id.container, fragment).commit();
108     item.setChecked(true);
109     setTitle(item.getTitle());
```

В строках № 83–84 объявляем новый фрагмент fragment и соответствующий ему Java-класс fragmentClass. По умолчанию задаем им значения null, дальше они будут изменяться.

Строка № 90 означает, что при нажатии на пункт меню **nav_new_group** (Новая группа) текущим классом фрагмента становится класс **NewGroupFragment**. По аналогии нужно прописать классы для остальных пунктов меню (строки № 91–98).

В строках № 100–104 описана обработка исключений. **Исключение** — это нештатная ситуация (ошибка), возникшая во время выполнения программы. Для обработки исключений используются ключевые слова **try** (*пытаться*) и **catch** (*поймать*). Еще

применяются **throw**, **throws** и **finally**, но нам они не понадобятся. В блок **try** записывается программный код, который нужно защитить от исключений. В блоке **catch** по умолчанию прописывается команда e.printStackTrase() (*выводить трассировку стека*, в который по мере выполнения программы заносятся все выполняемые методы). По этой команде печатается несколько строк с описанием пойманного исключения в консоли вывода Android Studio. Благодаря этому при тестировании разработчик поймет, что именно пошло не так. Сюда можно дописать еще вывод уведомления пользователя о возникшей ошибке или любое другое действие.

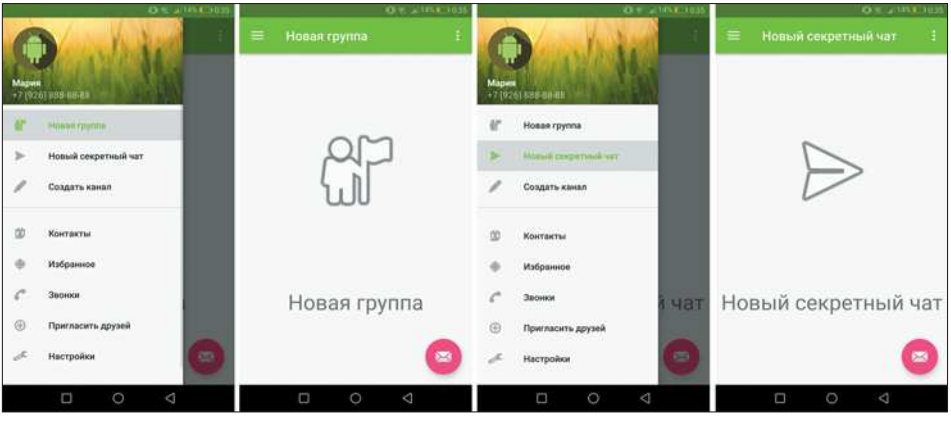
За смену фрагментов (строки № 106–107) отвечает класс **FragmentManager** (*менеджер фрагментов*). Для объекта этого класса (назовем его fragmentManager) пропишем методы:

- **beginTransaction()** (*начать транзакцию*);
- **replace()** (*заместить*) — первый параметр этого метода определяет контейнер, внутри которого происходит замещение, а второй содержит имя фрагмента, которым нужно заместить текущий;
- **commit()** (*зафиксировать*), чтобы изменения вступили в силу.

Метод **setChecked()** (*установить выбранным*) со значением true для выбранного пункта меню (item) выделяет этот пункт меню цветом. Это делается для того, чтобы пользователь понимал, в какой части приложения он находится (строка № 108).

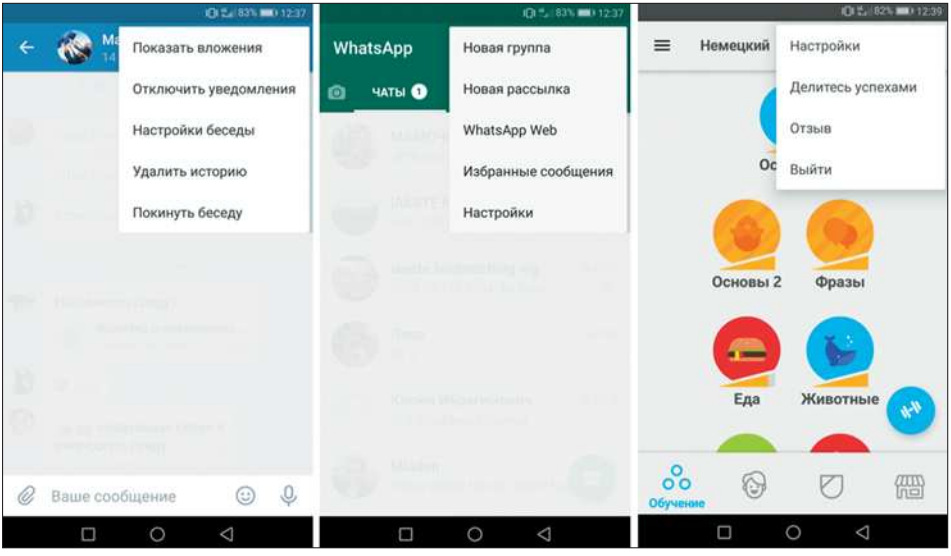
Последний в этом фрагменте кода (строка № 109) метод **setTitle()** (*установить название*) устанавливает в шапке Activity название выбранного пункта меню. Например, при выборе пункта «Настройки» экран получит это же название, а не название приложения по умолчанию.

Готово! Запускаем приложение и убеждаемся, что при нажатии на пункт меню отображается соответствующий фрагмент.

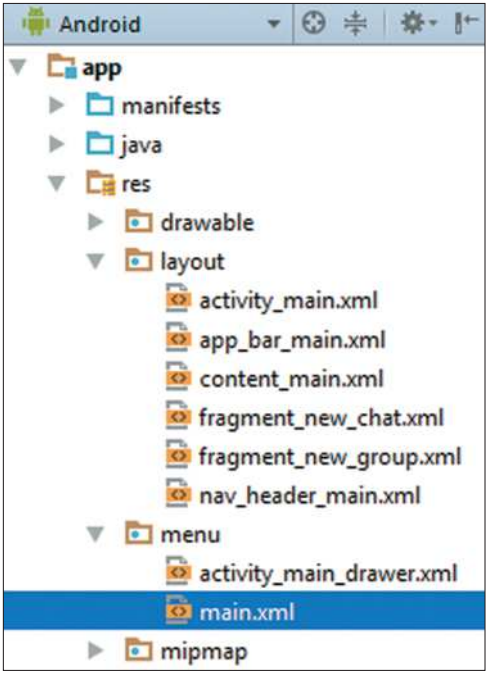


4.5.3. Главное меню

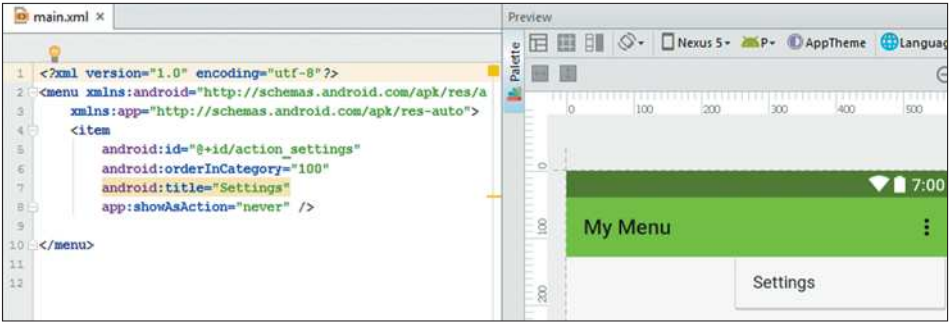
Как уже говорилось, главное меню используется многими известными приложениями: ВКонтакте, WhatsApp, Duolingo и другими.



Для его создания снова обратимся к структуре проекта:



Откроем файл разметки главного меню **main.xml**, который лежит в папке **menu**, и рассмотрим в режиме кода атрибуты, заданные по умолчанию.



Сейчас в меню есть один пункт — Settings (*настройки*), имеющий идентификатор **(id)**, название **(title)** и еще один важный атрибут — **showAsAction**. Этот атрибут отвечает за изменения в панели ActionBar при выборе соответствующего пункта: при значении always пункт меню будет отображаться в ActionBar рядом с тремя точками, при значении never — только внутри меню и будет виден только при нажатии на три точки, при ifRoom — только если поместится по размеру.

Задание

Переименовать существующий пункт Settings в Настройки и добавить еще два пункта — «О приложении» и «Оценить приложение».

Навигация с помощью этого меню реализуется по аналогии с меню-шторкой, но описывается в уже созданном методе **onOptionsItemSelected(MenuItem item)** главного Java-класса **MainActivity**.

4.6. Навигация. Переключение между несколькими экранами

В прошлом уроке, при рассмотрении различных видов меню, мы освоили переключение между фрагментами. Однако навигация может осуществляться не только внутри одной Activity, но и между разными Activity. И у этого способа тоже немало преимуществ, потому что другая Activity может иметь другой тип

и стиль, другое название, отображаемое в панели ActionBar (или вовсе не иметь ActionBar), и так далее.

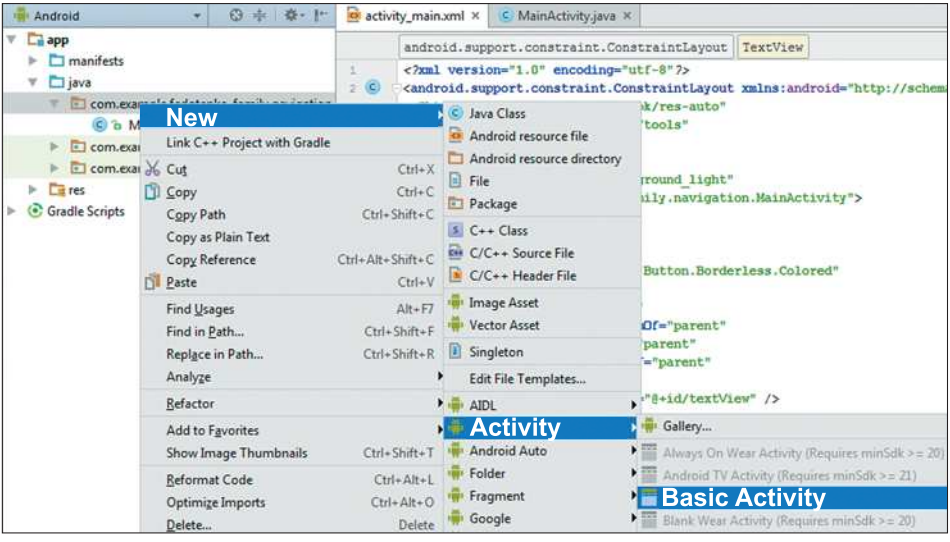
Вспомним наше первое приложение (HelloWorld). В нем при нажатии на кнопку появлялись и изменялись элементы. Но элементов было всего два, и такая «анимация» была уместна. Если же добавить больше элементов, станет гораздо сложнее размещать их один за другим и выстраивать появление одних в зависимости от других; проще расположить элементы по разным экранам и переходить от одного к другому.

Давайте рассмотрим переключение между двумя Activity с помощью элементов одной из них.

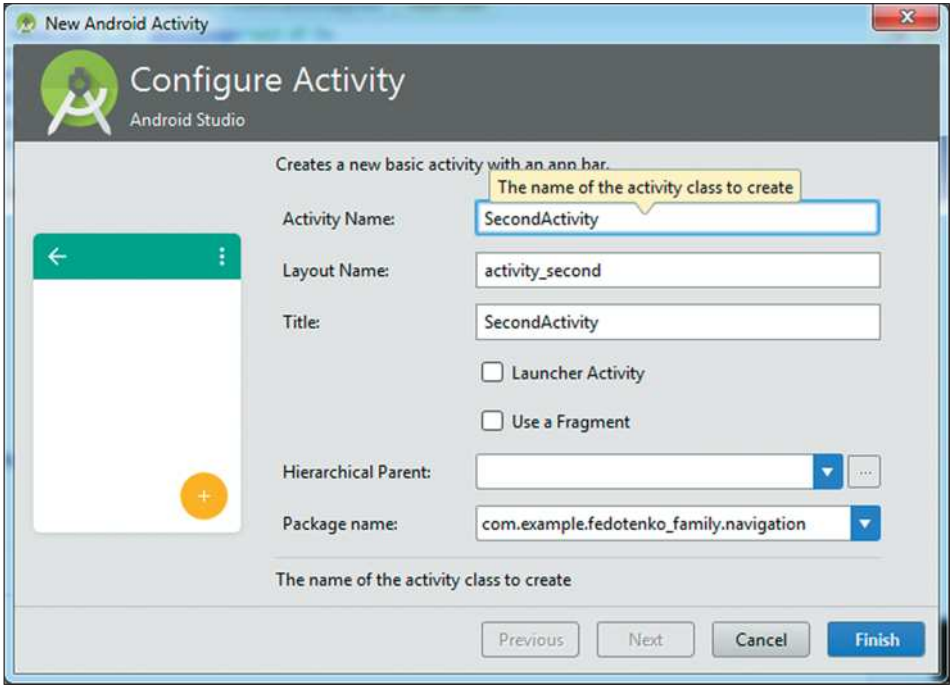
Концепция мини-приложения: наше первое приложение (HelloWorld), в котором элементы расположены в двух Activity (разного типа и с разными темами). Переход между Activity осуществляется по нажатию на кнопку.

Создадим новый проект Navigation и в качестве первой Activity выберем **Empty Activity**. Итак, первый экран у нас есть. Добавим на него произвольный текст и кнопку для будущего перехода ко второму экрану:

Но сам второй экран еще нужно добавить. Для создания новой Activity следует нажать правой кнопкой мыши на папке, в которой лежат наши Java-классы, и выбрать **New** → **Activity** → → **BasicActivity** (или любой другой шаблон):

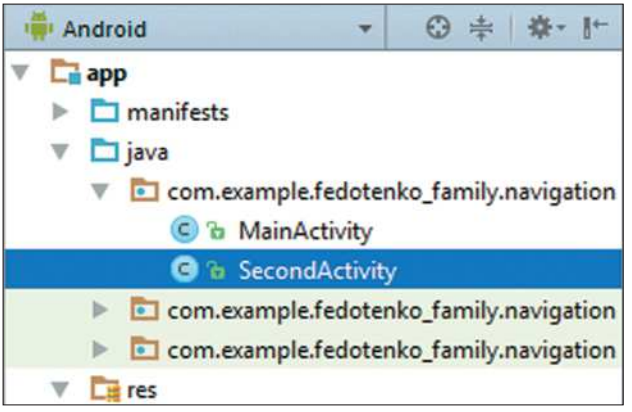


В открывшемся окне нужно задать имя новой Activity, например SecondActivity, и ее параметры:



Layout Name (*имя макета*) можно оставить по умолчанию. Галочку **Launcher Activity** (*экран запуска*) устанавливают для экрана, который должен отображаться первым при запуске приложения, а мы создаем второй, поэтому ее нужно снять.

Готово! Теперь в нашем проекте два Java-класса и три макета, потому что для Basic Activity создается два макета.



Начнем со второй Activity. В ней не предполагается никаких действий, поэтому соответствующий ей Java-класс **SecondActivity** можно закрыть. Файл его xml-разметки **content_second** можно наполнить контентом по своему усмотрению — разместить изображение, текст, кнопки или оставить пустым.

Основным «действующим лицом» в этом уроке будет первый экран.

Откроем его Java-класс **MainActivity.java**. Затем объявим и определим кнопку для перехода ко второму экрану, добавим метод **onStartButtonClick()** для обработки события ее нажатия.

activity_main.xml × content_second.xml × MainActivity.java ×

```
1  package com.example.fedotenko_family.navigation;
2
3  import ...
7
8  public class MainActivity extends AppCompatActivity {
9
10     private ImageButton startButton;
11
12     @Override
13     protected void onCreate(Bundle savedInstanceState) {
14         super.onCreate(savedInstanceState);
15         setContentView(R.layout.activity_main);
16
17         startButton = (ImageButton) findViewById(R.id.startButton);
18     }
19
20     public void onStartButtonClick(View view) {
21     }
22 }
```

Именно в этом методе мы и пропишем переход ко второй Activity.

Переключение между экранами осуществляется через намерение **Intent**, а алгоритм мы уже рассматривали в уроке про загрузочный экран:

```
21 public void onStartButtonClick(View view) {
22     Intent intent = new Intent(MainActivity.this, SecondActivity.class);
23     startActivity(intent);
24 }
```

Обратим внимание также на изменения в файле манифеста. В нем два тега <activity></activity> — для каждого из наших экранов, и для каждого тега описаны основные параметры:

AndroidManifest.xml ×

manifest | application

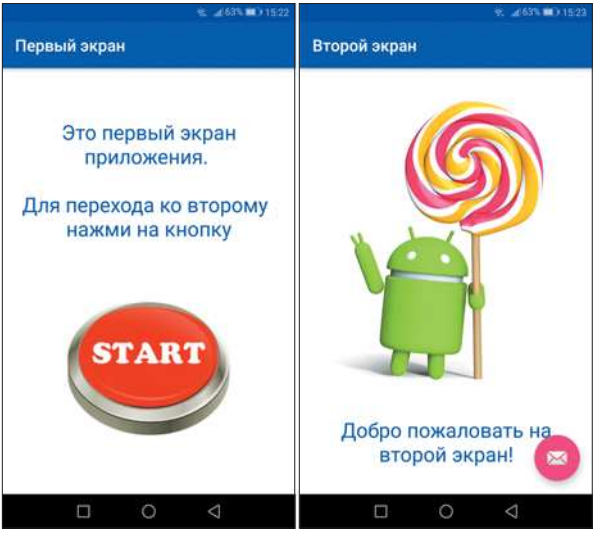
```
12         <activity android:name=".MainActivity">
13             <intent-filter>
14                 <action android:name="android.intent.action.MAIN" />
15
16                 <category android:name="android.intent.category.LAUNCHER"/>
17             </intent-filter>
```

```
18          </activity>
19          <activity
20              android:name=".SecondActivity"
21              android:label="SecondActivity"
22              android:theme="@style/AppTheme.NoActionBar">
23          </activity>
24      </application>
```

Для того чтобы у каждого экрана в ActionBar отображалось свое название, изменим (или добавим) атрибут **label** (строки № 14 и 23):

```
12      <activity
13          android:name=".MainActivity"
14          android:label="Первый экран">
15          <intent-filter>
16              <action android:name="android.intent.action.MAIN" />
17
18              <category android:name="android.intent.category.LAUNCHER" />
19          </intent-filter>
20      </activity>
21      <activity
22          android:name=".SecondActivity"
23          android:label="Второй экран"
24          android:theme="@style/AppTheme.NoActionBar">
25      </activity>
```

Запустим приложение и убедимся, что при нажатии на кнопку происходит переход на другой экран и у каждого из экранов свое название:



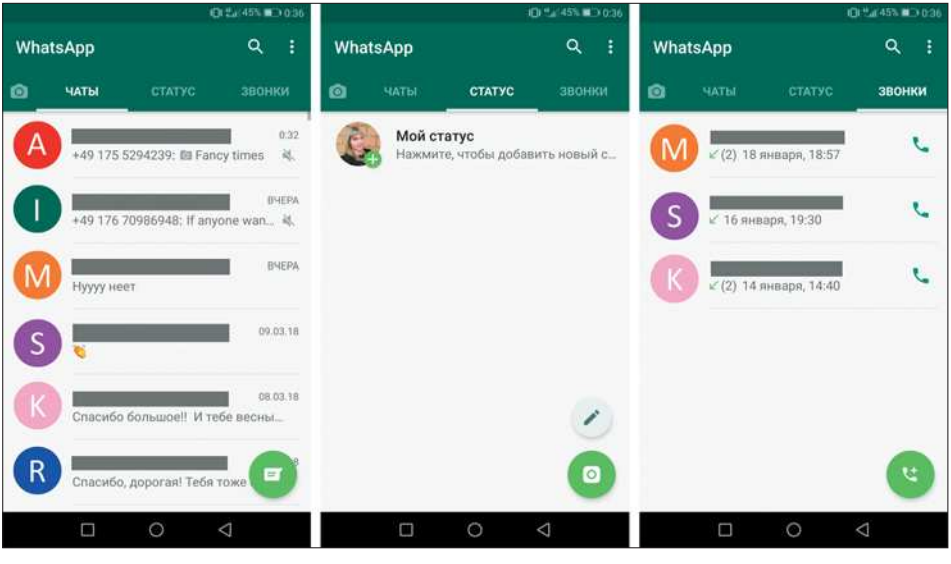
Теперь можно задать обеим Activity разные стили и сделать их уникальными, непохожими друг на друга, с разной структурой, типом и назначением.

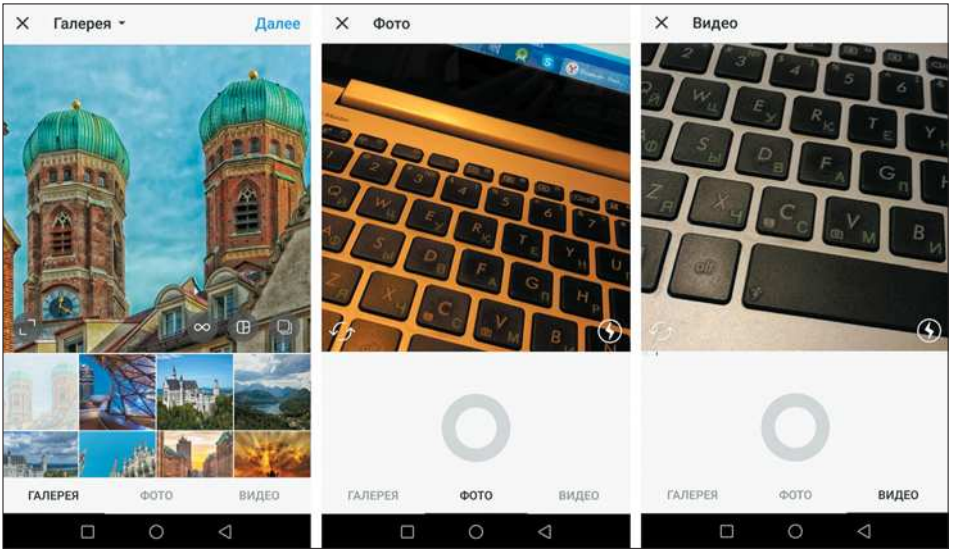
Задания

Мы уже знаем, что в приложении можно переключаться между несколькими экранами или фрагментами с помощью меню, кнопок и других элементов интерфейса. Но в **Android Studio** представлено еще несколько шаблонов для осуществления навигации, которые сейчас популярны и используются во многих приложениях.

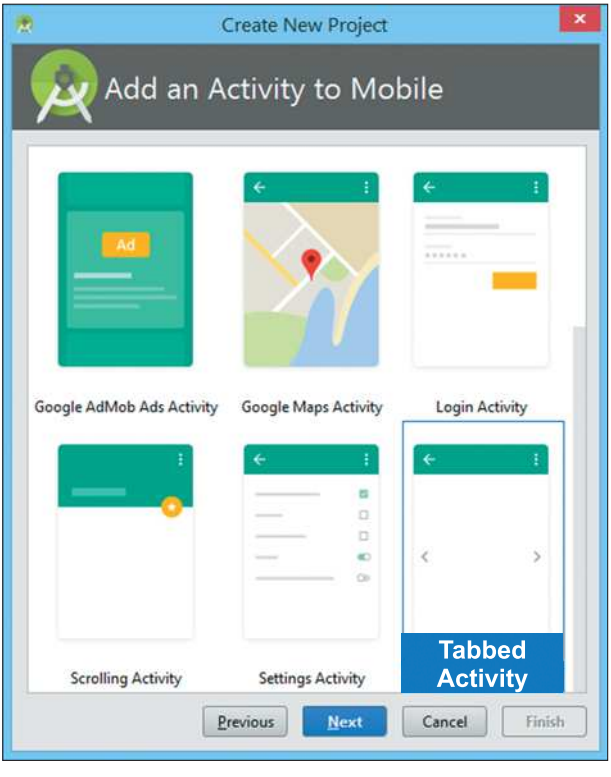
В качестве заданий к этому уроку нам предстоит освоить возможности этих шаблонов самостоятельно.

1. Воспроизвести фрагмент любого известного приложения с использованием шаблона **Tabbed Activity**. Это шаблон, в котором переключение между фрагментами осуществляется горизонтальным свайпом. Он очень популярен, поскольку позволяет визуально расширять пространство одного экрана по горизонтали и вертикали практически до бесконечности. **Tabbed Activity** применяется для основной навигации в таких приложениях, как WhatsApp, а также при добавлении фото и переходе к личным сообщениям в Instagram.

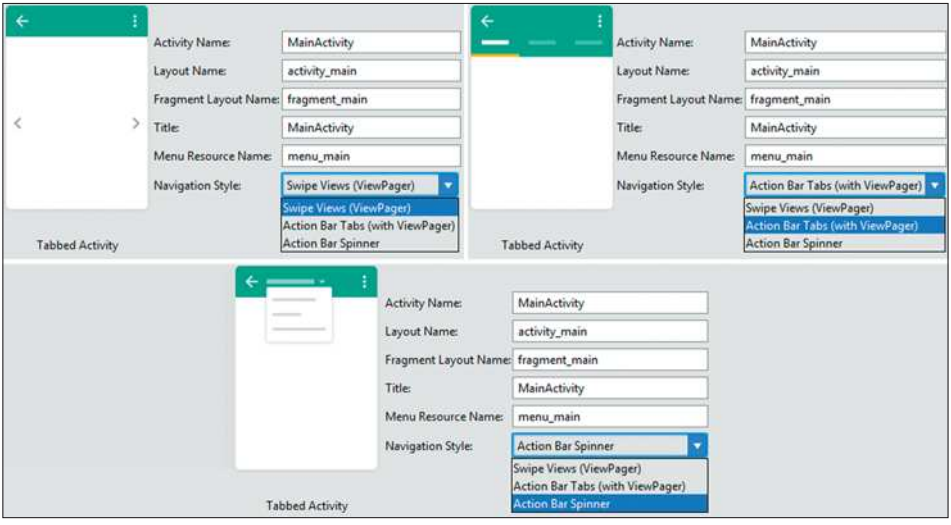




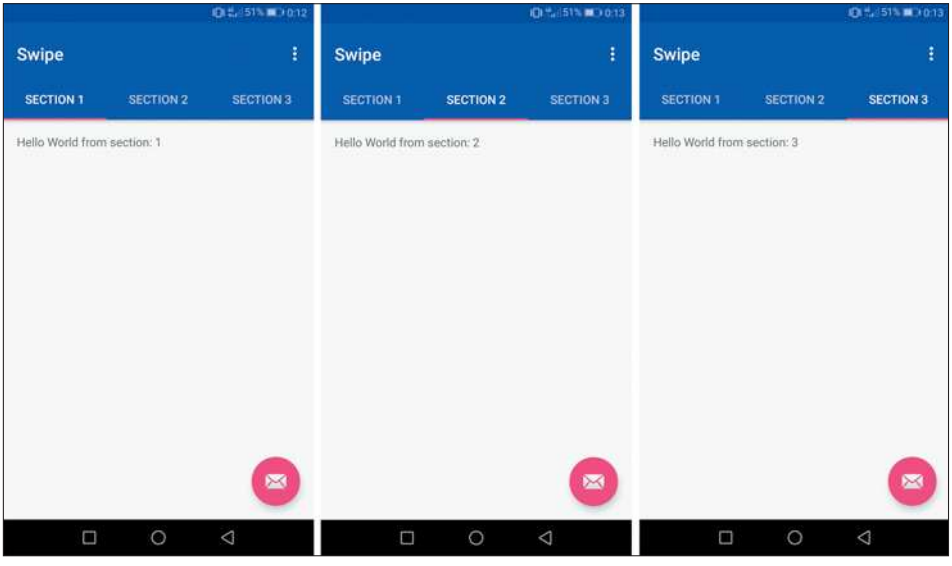
При создании проекта выбираем **Tabbed Activity** (*экран с вкладками*):



На следующем шаге выбираем имя Activity, имя первого фрагмента и стиль навигации (доступны три варианта):

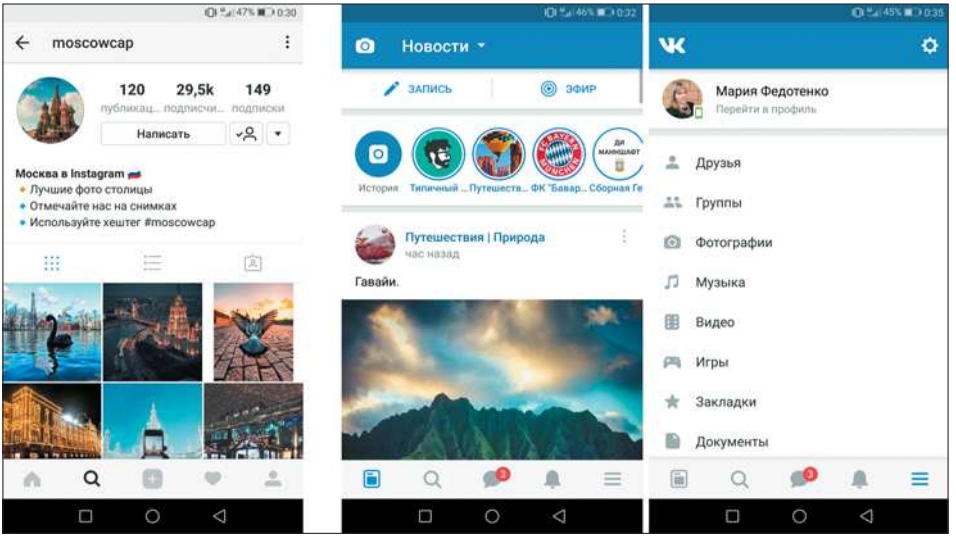


Созданное приложение имеет вид:

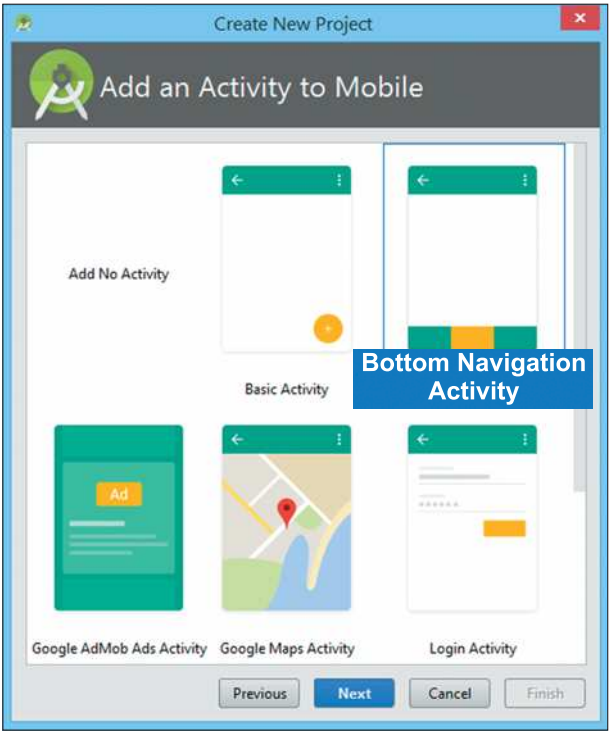


2. Воспроизвести фрагмент любого известного приложения с использованием шаблона **Bottom Navigation Activity**.
   В этом шаблоне переключение между фрагментами осуществляется с помощью нижней панели навигации.

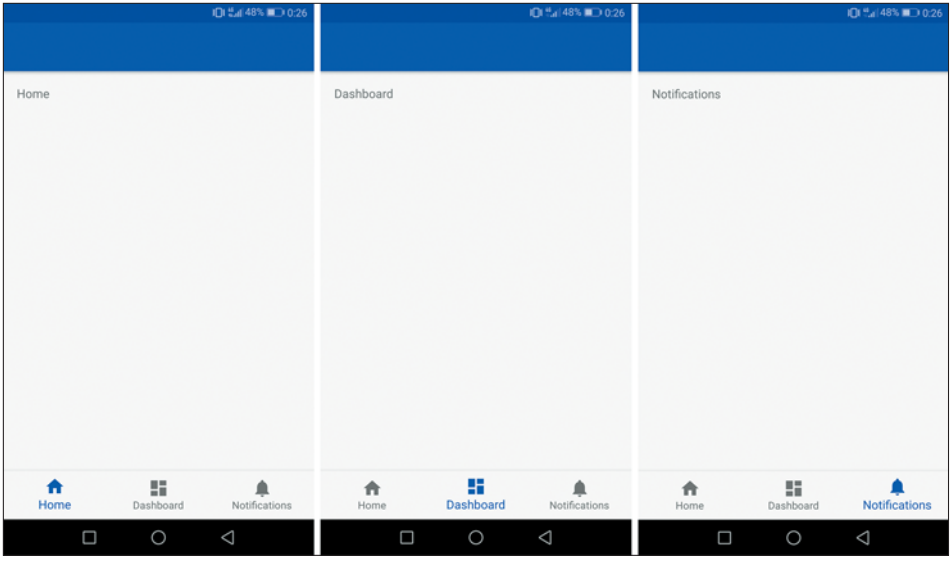
Bottom Navigation Activity применяется для основной навигации в таких приложениях, как Instagram и ВКонтакте:



При создании проекта выбираем **Bottom Navigation Activity** (*экран с нижней навигацией*).



Созданное приложение имеет вид:



Итоги главы 4

В главе 4 мы почувствовали себя дизайнерами и поняли, что иногда дизайнер должен не только подбирать цвета и рисовать красивые картинки, а быть также немного аналитиком и проектировщиком интерфейса.

Теперь мы можем:

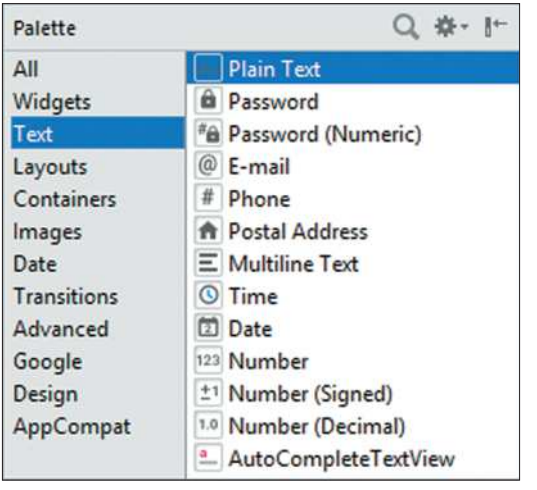
1. Оценить дизайн и юзабилити любого приложения и использовать это при разработке своих.
2. Подобрать и установить логотип приложения.
3. Добавить в приложение загрузочный экран и задать ему требуемые параметры отображения.
4. Запрограммировать анимацию элементов интерфейса.
5. Задавать единый стиль разным элементам интерфейса.
6. Настраивать и применять темы для разных Activity приложения.
7. Выстраивать навигацию по приложению с помощью меню разных типов.
8. Осуществлять навигацию по приложению через переходы между разными Activity или через переключение фрагментов внутри одной Activity.

Глава 5. Работа с текстом, изображениями и жестами

5.1. Работа с текстом

Мы уже не раз встречались с текстовым элементом **TextView** и его свойствами, преобразовывали его в **EditText** и заполняли программно.

Палитра элементов содержит группу текстовых элементов **Text**, в которой расположены элементы **TextView** и **EditText** разных типов для ввода и отображения текстового содержания самых разных видов. Для элемента уже предустановлен соответствующий набор свойств: формат ввода и отображения, требуемый тип клавиатуры и так далее.



Для работы с текстом в **Android Studio** имеется большое количество атрибутов, отвечающих за:

1. Содержание:
   - **text** — текстовое содержание элемента;
   - **autoLink** — этим свойством можно задать автоматическое распознавание различных ссылок в тексте: телефонных номеров, e-mail, URL, координат для карт Google;
   - **clickable** — активность элемента («кликабельность», возможность нажатия на него);
   - **editable** — возможность редактирования элемента пользователем;
   - **inputType** — тип вводимого значения (в данном случае определяет тип клавиатуры, которая будет отображаться при вводе данных в элемент);
   - **onClick** — событие, происходящее по нажатию на элемент.

2. Цвета:
   - **textColor** — цвет текста;
   - **background** — цвет фона текста;
   - **shadowColor** — цвет тени текста;
   - **textColorLink** — цвет подчеркивания элемента, если он подчеркивается, например при вводе данных в него.
3. Размеры:
   - **textSize** — размер текста;
   - **shadowDx** и **shadowDy** — размеры тени;
   - **shadowRadius** — радиус тени.
4. Стиль:
   - **style** — стиль элемента (задается через ресурсы или выбирается из стандартных);
   - **textAllCaps** — все ли буквы текста отображаются заглавными (как при нажатии клавиши **Caps lock**);
   - **textDirection** — направление текста (справа налево, слева направо, сверху вниз и так далее);
   - **textStyle** — стиль начертания текста (полужирный, наклонный, подчеркнутый).

5.1.1. Длинный текст

Раньше мы работали с короткими надписями в одну-две строки, но в большинстве приложений мы видим довольно длинные тексты. Как работать с ними?

Создадим новый пустой проект, и в стандартно отображающемся **TextView** вместо Hello World! пропишем известное четверостишие А. Н. Плещеева:

Травка зеленеет,
Солнышко блестит;
Ласточка с весною
В сени к нам летит.

Это уже достаточно длинный текст, поэтому для него нужно создать ресурс. В файле строковых ресурсов **strings.xml** пропишем новую строку с именем long_text:

```
<resources>
    <string name="app_name">Text</string>
    <string name="long_text">Травка зеленеет,
        Солнышко блестит;
        Ласточка с весною
        В сени к нам летит.</string>
```

Теперь вернемся к главному экрану и зададим свойству **text** нашего элемента **TextView** значение @string/long_text.

Но что мы видим? В ресурсах мы прописали текст с переносами строк, однако на экране он отображается в одну строку:

Это произошло потому, что язык XML, так же как и HTML, «признает» только свои способы форматирования текста. Вернемся к строковому ресурсу и в тех местах, где нужен перенос строки, поставим специальный управляющий символ перехода на новую строку \n (двойной перенос строки, или переход на новый абзац, — \n\n):

```
<string name="long_text">Травка зеленеет,\n
    Солнышко блестит;\n
    Ласточка с весною\n
    В сени к нам летит.</string>
```

Сразу же видим изменения:

Кроме того, весь текст или отдельные его фрагменты можно сделать полужирным, наклонным или подчеркнутым с помощью следующих тегов:

- **...** — текст, помещенный в этот тег, примет полужирное начертание (**bold** — *жирный*);
- **...** — тег для выделения текста курсивом (**italic** — *курсивный, н клонный*);
- **<u>...</u>** — тег для подчеркивания текста (**underline** — *подчеркнутый*).

Для примера выделим каждое подлежащее одним из стилей:

```
<string name="long_text">
    <u>Травка</u> зеленеет,\n
    <i>Солнышко</i> блестит;\n
    <b>Ласточка</b> с весною\n
    В сени к нам летит.</string>
```

Важно!

В режиме предпросмотра внесенных **изменений видно не будет.** Они отобразятся только при тестировании приложения на реальном устройстве или на эмуляторе.

5.1.2. ScrollView — контейнер с возможностью прокрутки

Предположим, что на экране есть еще другие элементы интерфейса и нашему тексту остается только половина пространства:

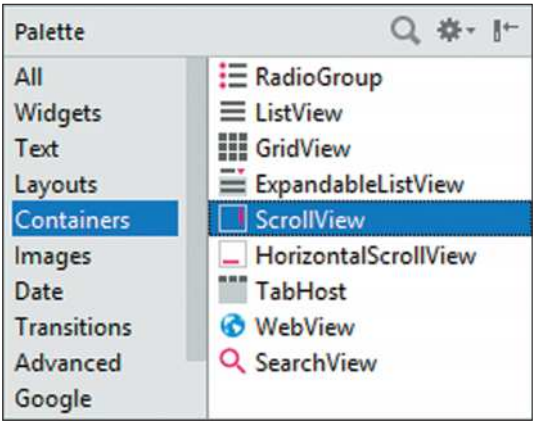
А ведь взятое нами стихотворение на самом деле не четверостишие, в нем есть еще две строфы:

С нею солнце краше
И весна милей...
Прощебечь с дороги
Нам привет скорей!

Дам тебе я зёрен,
А ты песню спой,
Что из стран далёких
Принесла с собой...

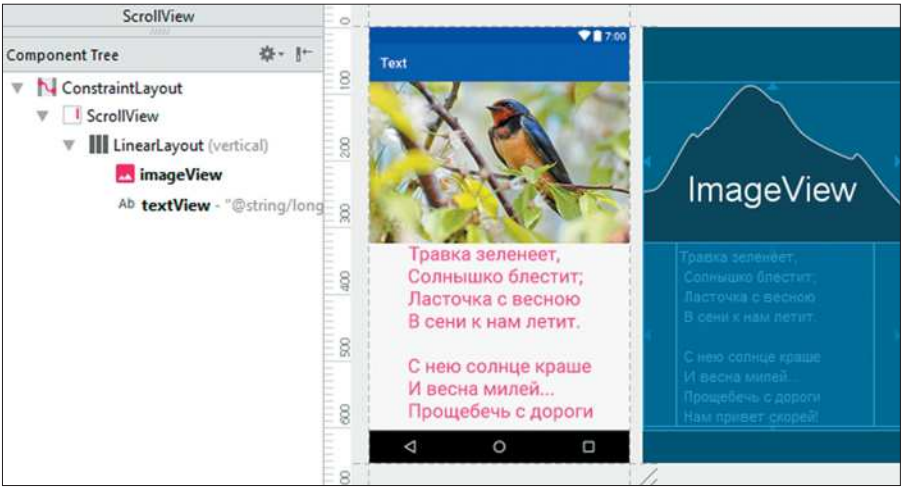
Добавим недостающие строки в **long_text**, выставив переносы строк в нужных местах. Что мы видим теперь? Текст не поместился на экране. Но что же случилось с ласточкой и как наши пользователи узнают, чем все заканчивается?

В большинстве случаев экраны в приложениях можно прокручивать. Сделаем прокручиваемым и наш текст. Для прокрутки больших по размеру элементов существует специальный контейнер — элемент **ScrollView**. В палитре элементов можно найти два типа **ScrollView** — вертикальный и горизонтальный:

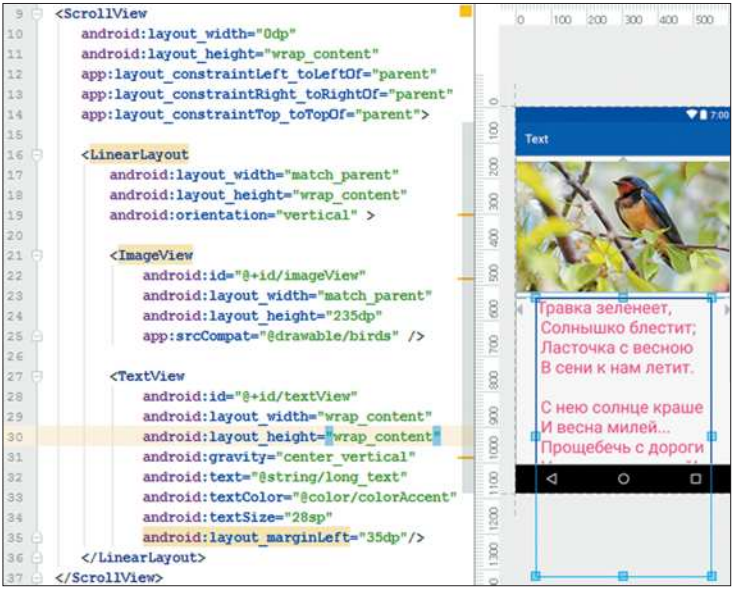


Перетащим **ScrollView** в дерево компонентов. Видим, что внутри него автоматически появился макет **LinearLayout**. Дело в том, что элемент **ScrollView** может иметь **только одного** наследника. Поэтому все, что мы захотим прокручивать, должно располагаться внутри этого наследника.

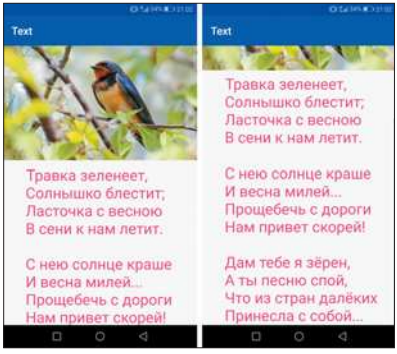
Перенесем наши **ImageView** и **TextView** внутрь **LinearLayout**:



В режиме кода **обязательно** зададим элементам, которые должны прокручиваться, высоту wrap_content. В этом случае для них будет требоваться больше места, чем располагает экран, и это создаст возможность прокрутки:



Запускаем приложение. Действительно, главный экран можно прокручивать; при этом перемещаются и текст, и изображение, а сбоку можно увидеть тонкую серую полосу прокрутки.



Задание

Сделать так, чтобы прокручивалась только нижняя часть экрана, то есть только текст.

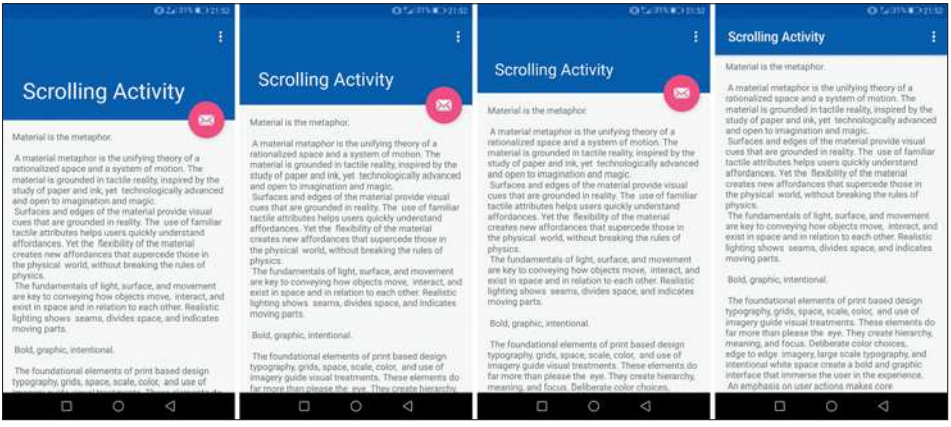
5.1.3. ScrollingActivity — прокручиваемый экран

Мы рассмотрели основные возможности **ScrollView** и теперь можем добавлять его в любые Activity наших приложений, причем как для всего экрана, так и для отдельных элементов. Но **Android Studio** была бы не **Android Studio**, если бы не имела для подобного случая своих шаблонов и заготовок.

Создаем новый проект, в качестве шаблона выбираем **Scrolling Activity**:

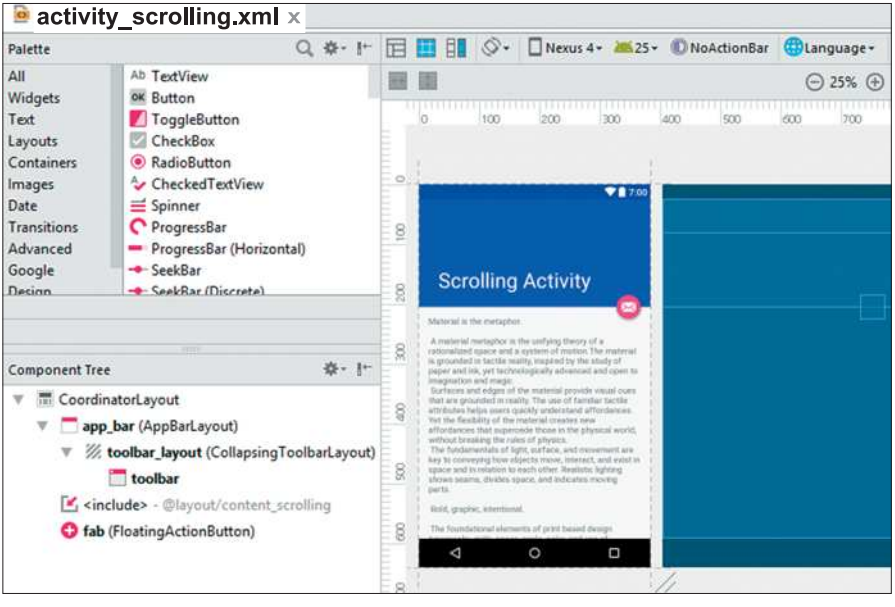
Соберем созданный проект и запустим приложение.

Принцип работы **Scrolling Activity** заключается в следующем. На экране есть длинный текст (разумеется, прокручиваемый), а над текстом расположена панель ActionBar, увеличенная по сравнению с остальными шаблонами экранов. При прокручивании страницы она меняет свою высоту и содержание до привычного нам ActionBar и снова растет при прокручивании назад:



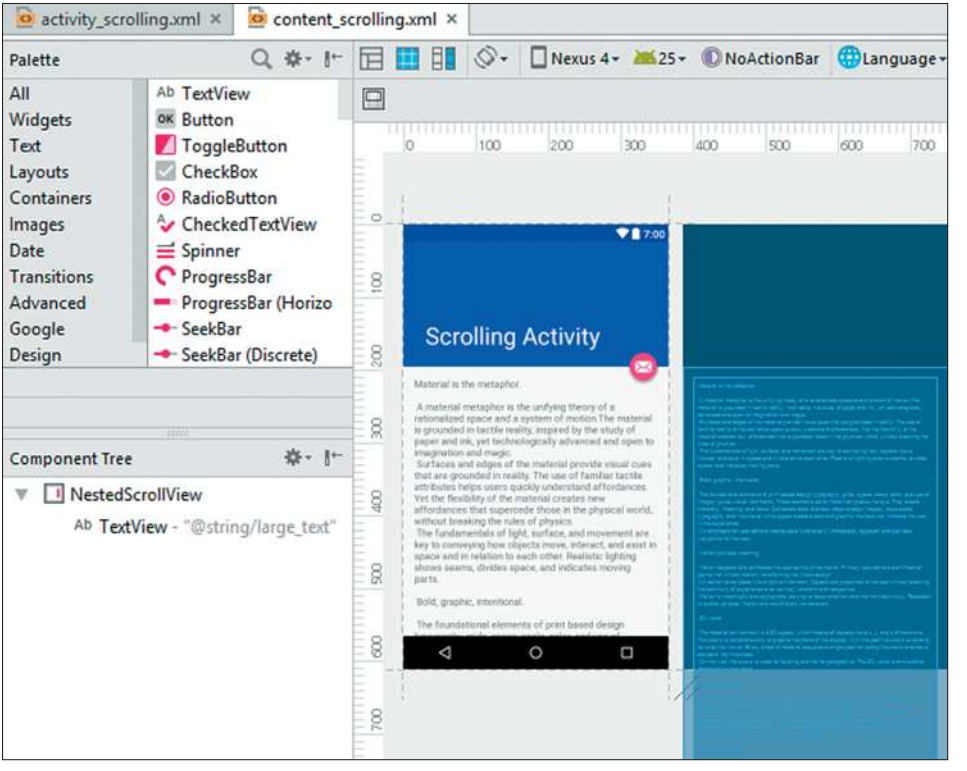
Чтобы разобраться, как это работает, рассмотрим структуру проекта. По умолчанию создано два файла xml-разметки: **activity_scrolling** и **content_scrolling**.

Файл **activity_scrolling.xml** хранит в себе информацию обо всей Activity.

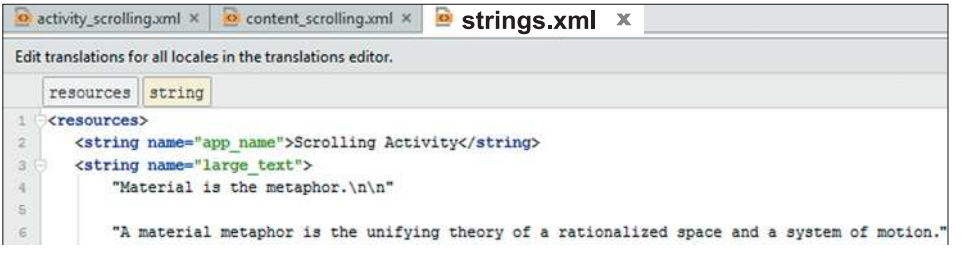


Мы видим, что панель ActionBar в данном случае представлена как целый макет **AppBarLayout**. В него входит еще один макет — **toolbar_layout**, в котором располагается сам **toolbar**. Именно такая структура и позволяет реализовать анимацию, о которой мы говорили ранее.

В файле **content_scrolling.xml** хранится разметка контента основного экрана — элемент **TextView**, помещенный в контейнер **NestedScrollView**, будет схож по своим свойствам с контейнером **ScrollView**.



Сам текст, разумеется, прописан как ресурс и хранится в файле **strings.xml**. Его оформление схоже с тем, что мы задавали тексту для **ScrollView**.



```
    "The material is grounded in tactile reality, inspired by the study of paper and ink, yet "
    "technologically advanced and open to imagination and magic.\n"
    "Surfaces and edges of the material provide visual cues that are grounded in reality. The "
    "use of familiar tactile attributes helps users quickly understand affordances. Yet the "
    "flexibility of the material creates new affordances that supercede those in the physical "
    "world, without breaking the rules of physics.\n"
    "The fundamentals of light, surface, and movement are key to conveying how objects move, "
    "interact, and exist in space and in relation to each other. Realistic lighting shows "
    "seams, divides space, and indicates moving parts.\n\n"

    "Bold, graphic, intentional.\n\n"
```

Задания

1. Заменить текст на свой, обязательно длинный, например отрывок из любимого произведения.
2. Использовать стандартное форматирование: перенос строки, подчеркивание текста, жирное начертание.
3. Установить свой фон для **AppBarLayout**. Для этого внутри **CollapsingToolbarLayout** нужно разместить элемент **ImageView**. В таком случае во время прокрутки фон ActionBar будет меняться от заданного изображения до стандартного вида.

5.2. Обработка касаний и жестов

В предыдущих уроках мы уже обрабатывали простое одиночное касание — нажатие на кнопки и другие элементы. Однако элементы установленных на наших смартфонах приложений могут по-разному реагировать на одиночное и двойное нажатия или на короткое и длинное.

Касания можно поделить на **одиночные** и **множественные**.

Одиночное касание — касание, выполненное одним пальцем. **Множественное касание** выполняется одновременно несколькими пальцами (от двух до десяти).

В этом уроке мы рассмотрим одиночное касание и относящиеся к нему жесты: двойное нажатие, долгое нажатие, скроллинг (прокручивание) и свайпинг (проведение по элементу для его сдвига).

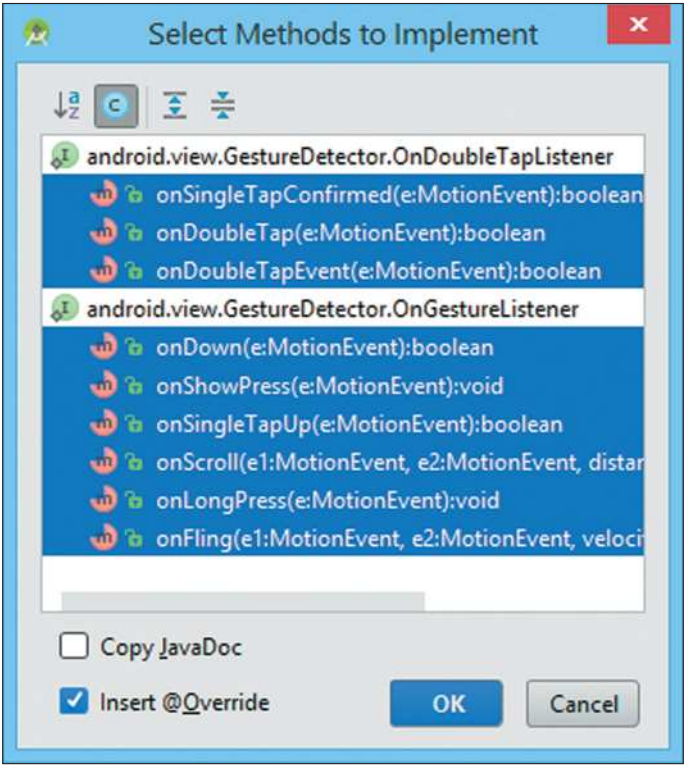
Создадим новый проект с пустой Activity и удалим из него элемент **TextView**. В этом проекте заниматься разметкой интерфейса мы не будем — нас интересует действительно пустой экран.

Перейдем в файл **MainActivity.java**. Для обработки касаний и жестов нужно импортировать два прослушивателя из класса **GestureDetector** (*детектор жестов*) — **OnGestureListener** (*прослушивателя жестов*) и **OnDoubleTapListener** (*прослушивателя двойного нажатия*):

```
activity_main.xml × | MainActivity.java ×
MainActivity
1  package com.marusyafm.gestures;
2
3  import ...
6
7  public class MainActivity extends AppCompatActivity
8          implements GestureDetector.OnGestureListener, GestureDetector.OnDoubleTapListener {
9      @Override
10     protected void onCreate(Bundle savedInstanceState) {
```

Строка, как всегда в таких случаях, подчеркнута красным, значит, нужно реализовать недостающие методы: нажать **Alt+Enter** и выбрать пункт **Implement methods**.

В открывшемся окне предлагается довольно внушительный список методов для реализации. По умолчанию выбраны все — соглашаемся с выбором, нажимаем **OK**.



Новые методы появились после метода **onCreate()**, но к ним мы вернемся чуть позже. А теперь для обнаружения жестов нужно объявить еще одну переменную типа **GestureDetectorCompat**. Назовем ее gestureDetector:

```
9  public class MainActivity extends AppCompatActivity
10                          implements GestureDetector.OnGestureListener,
11                          GestureDetector.OnDoubleTapListener {
12      private GestureDetectorCompat gestureDetector;
13
14      @Override
15      protected void onCreate(Bundle savedInstanceState) {
16          super.onCreate(savedInstanceState);
17          setContentView(R.layout.activity_main);
18
19          gestureDetector = new GestureDetectorCompat(this, this);
20          gestureDetector.setOnDoubleTapListener(this);
21      }
```

Так же как и при работе с кнопками и другими элементами, наш **gestureDetector** нуждается не только в объявлении, но и в определении (строка № 19). Дополнительно «навесим» на него прослушиватель двойного нажатия **setDoubleTapListener()**.

И в завершение подготовительных работ необходимо добавить еще один метод после метода **onCreate()** — **onTouchEvent()** (*по событию н ж тия*). Это обработчик событий, относящихся к сенсорному экрану, который будет определять сам факт нажатия и его тип:

```
22
23      @Override
24      public boolean onTouchEvent(MotionEvent event){
25          if (this.gestureDetector.onTouchEvent(event)) {
26              return true;
27          }
28          return super.onTouchEvent(event);
29      }
30
```

Теперь мы можем рассмотреть непосредственно методы обработки нажатий и жестов.

5.2.1. Двойное нажатие

Двойное нажатие обрабатывает метод **onDoubleTap()**. В нем прописывается все, что должно происходить при двойном нажатии на элемент или, как здесь, на любую точку экрана. Пусть на данном этапе это будет всплывающее сообщение о том, что было совершено двойное нажатие. Поскольку возвращаемое значение должно иметь тип **boolean**, в конце метода исправим return false на return true — иначе наш метод не «активируется»:

```
38  public boolean onDoubleTap(MotionEvent e) {
39      Toast toast1 = Toast.makeText(MainActivity.this,
40                                    "Двойное нажатие",Toast.LENGTH_SHORT);
41      toast1.setGravity(Gravity.TOP, 0, 10);
42      toast1.show();
43      return true;
44  }
```

5.2.2. Долгое нажатие

Обработка долгого нажатия прописывается в методе **onLongPress()**. В нем также пропишем выведение всплывающего сообщения:

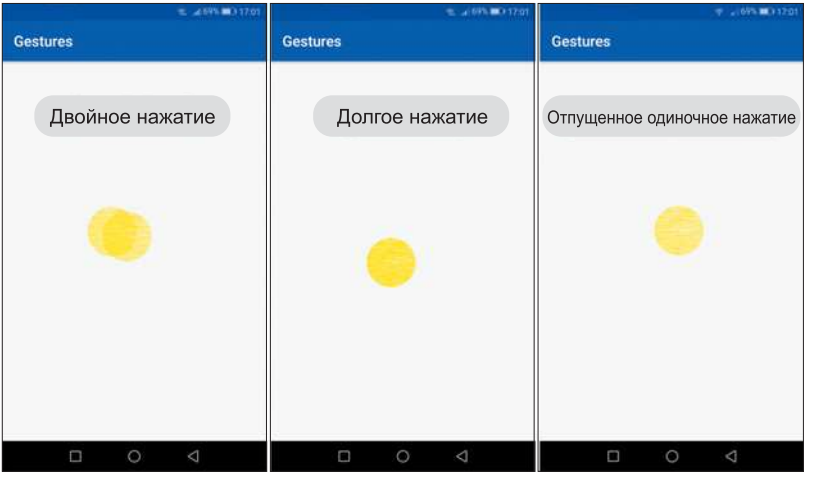
```
80  public void onLongPress(MotionEvent e) {
81      Toast toast2 = Toast.makeText(MainActivity.this,
82                                    "Долгое нажатие", Toast.LENGTH_SHORT);
83      toast2.setGravity(Gravity.TOP, 0, 10);
84      toast2.show();
85  }
```

5.2.3. Отпущенное одиночное нажатие

Существует разновидность одиночного нажатия: мы нажимаем на элемент и отпускаем его, и именно в момент отпускания должны происходить какие-то события. Для описания этих событий применяется метод **onSingleTapUp()**.

Найдем этот метод и пропишем в нем отображение всплывающего сообщения.

Запустим приложение и убедимся, что для каждого жеста выводится свое соответствующее сообщение:



5.2.4. Скроллинг и свайпинг

Итак, мы рассмотрели разные типы нажатий, но в приложениях элементы также можно прокручивать (этот жест называется **скроллинг**) и сдвигать (жест **свайпинг**). Для этих жестов в нашем проекте уже реализованы методы **onScroll()** для скроллинга и **onFling()** для свайпинга.

Найдем их и, по аналогии с предыдущими методами, пропишем в них отображение всплывающего сообщения:

5.3. Работа с камерой

Камера задействована во многих приложениях: социальные сети, мессенджеры, календари, различные приложения для заметок. В этом уроке мы не станем создавать приложение-камеру, а будем использовать системное приложение для получения изображений.

Концепция мини-приложения: на главном экране размещены кнопка и изображение. При нажатии на кнопку вызывается камера, а сделанные с ее помощью фотографии отображаются в изображении на главном экране.

Создадим проект My Camera с параметрами по умолчанию и одной Empty Activity. Для работы с камерой нужно установить приложению соответствующие разрешения. Добавим в файл манифеста **AndroidManifest.xml** разрешение на использование камеры android.permission.CAMERA и запрос на использование функций камеры android.hardware.camera:

AndroidManifest.xml

```
manifest
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      package="com.marusyafm.mycamera">
4      <uses-permission android:name="android.permission.CAMERA" />
5      <uses-feature android:name="android.hardware.camera" android:required="true" />
6
7      <application
8          android:allowBackup="true"
9          android:icon="@mipmap/ic_launcher"
10         android:label="My Camera"
11         android:roundIcon="@mipmap/ic_launcher_round"
```

Для сохранения полученных изображений в галерее нужно прописать еще одно разрешение — на добавление файлов в память устройства:

```
<uses-permission android:name=”android.permission.
WRITE_EXTERNAL_STORAGE”/>
```

На главном экране разместим два элемента: кнопку для запуска камеры и изображение (для отображения получившегося фото):

В классе **MainActivity** объявляем и определяем кнопку и изображение:

```
12 public class MainActivity extends AppCompatActivity {
13
14     private Button btnCamera;
15     private ImageView imgPhoto;
16     private final int REQUEST_CAMERA_IMAGE = 0;
17
```

```
18    @Override
19    protected void onCreate(Bundle savedInstanceState) {
20        super.onCreate(savedInstanceState);
21        setContentView(R.layout.activity_main);
22
23        btnCamera = (Button) findViewById(R.id.btnCamera);
24        imgPhoto = (ImageView) findViewById(R.id.imgResultPhoto);
25    }
```

Также для работы с камерой объявим переменную REQUEST_CAMERA_IMAGE (строка № 16). Она понадобится для дальнейшей передачи запроса на использование камеры и на обработку полученного результата. Сейчас ей можно присвоить любое числовое значение, например нуль.

В методе **onClick()** создаем намерение для обращения к камере **Intent** cameraIntent. На вход намерения передаем константу ACTION_IMAGE_CAPTURE (*действие: з хв т изобр жения*) из объекта **MediaStore**. Затем намерение **cameraIntent** передается методу **startActivityForResult()** (*з пустить Activity для получения результ т*) и запускает камеру (строки № 28–29). Метод **startActivityForResult()** используется, когда после закрытия вызываемой Activity или приложения планируется получение каких-либо данных в качестве результата работы.

```
27    public void onClick(View view) {
28        Intent cameraIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
29        startActivityForResult(cameraIntent, REQUEST_CAMERA_IMAGE);
30    }
31
32    @Override
33    protected void onActivityResult(int requestCode,int resultCode,Intent data){
34        super.onActivityResult(requestCode, resultCode, data);
35        if (requestCode == REQUEST_CAMERA_IMAGE) {
36            Bitmap imageBitmap = (Bitmap) data.getExtras().get("data");
37            imgPhoto.setImageBitmap(imageBitmap);
38        }
39    }
40 }
```

Для обработки полученных от камеры результатов запроса и данных добавляем метод **onActivityResult()** (строки № 32–39). Этот метод принимает на вход параметры:

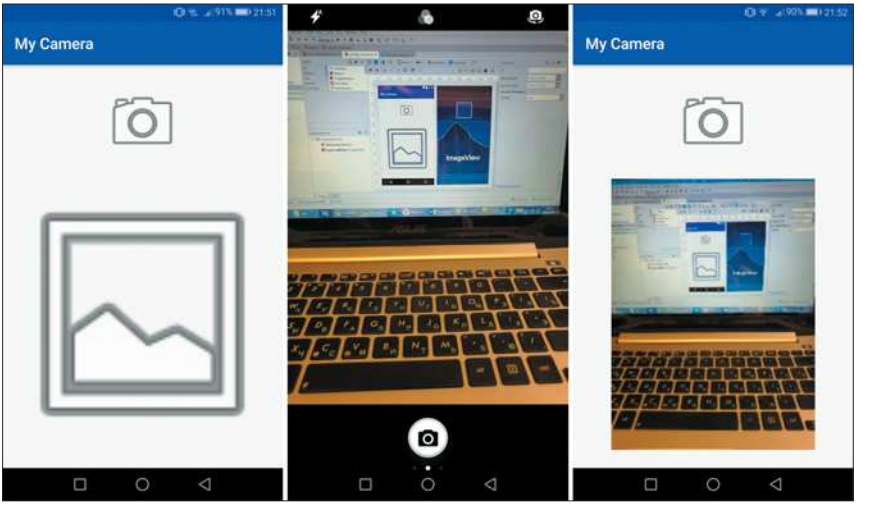
- **requestCode** (*код з прос*), который мы отправляем приложению-камере, когда вызываем ее. При этом нас интересует не процесс обработки нашего запроса самой камерой, а только результат;

- **resultCode** (*код результ т*) — принимает значения RESULT_OK и RESULT_CANCELED. Несет в себе информацию об успешности или неуспешности выполнения запроса;
- **data** (*д нные*) — данные, полученные в результате выполнения запроса, в нашем случае фото.

Если запрос выполнен успешно, получившийся снимок возвращается в параметре **data** в виде объекта **Bitmap** (строка № 36). Мы «извлекаем» этот объект с помощью методов **getExtras()** и **get(«data»)**, а затем устанавливаем его в качестве ресурса к размещенному на нашем экране элементу **ImageView** (строка № 37).

Готово! Запускаем приложение.

Все работает — по нажатию на кнопку запускается камера, при сохранении фото отображается на главном экране.



Задания

1. Доработать приложение для записи видео.
   Запись и воспроизведение видео с камеры программируется по аналогии с получением изображения, только вместо элемента **ImageView** следует использовать элемент типа **VideoView**. Нужно объявить и определить его в классе **MainActivity**. По аналогии с REQUEST_CAMERA_IMAGE использовать переменную REQUEST_CAMERA_VIDEO.
   Метод **onClick()** изменим следующим образом:

```
Intent takeVideoIntent = new Intent(MediaStore.ACTION_VIDEO_
CAPTURE);
if (takeVideoIntent.resolveActivity(getPackageManager()) != null) {
    startActivityForResult(takeVideoIntent, REQUEST_CAMERA_ VIDEO);
```

Необходимо также внести изменения в метод **onActivityResult()**:

```
if (requestCode == REQUEST_CAMERA_VIDEO && resultCode ==
RESULT_OK) {
    Uri videoUri = intent.getData();
    videoView.setVideoURI(videoUri);
}
```

Дополнительно описывать процесс сохранения видео в галерею не нужно — файлы видео сохраняются в памяти устройства автоматически.

2. Изменить приложение, чтобы полученная фотография устанавливалась не в элементе **ImageView**, а в качестве фона главного экрана.

5.4. Приложение «Скетчбук творческой личности»

Занимаясь разработкой приложений для устройств с сенсорным экраном, было бы странно не создать приложение, позволяющее на этом экране писать и рисовать.

Концепция приложения: скетчбук. Основной экран представляет собой полотно для рисования и работы с текстом со стандартными функциями скетчбука.

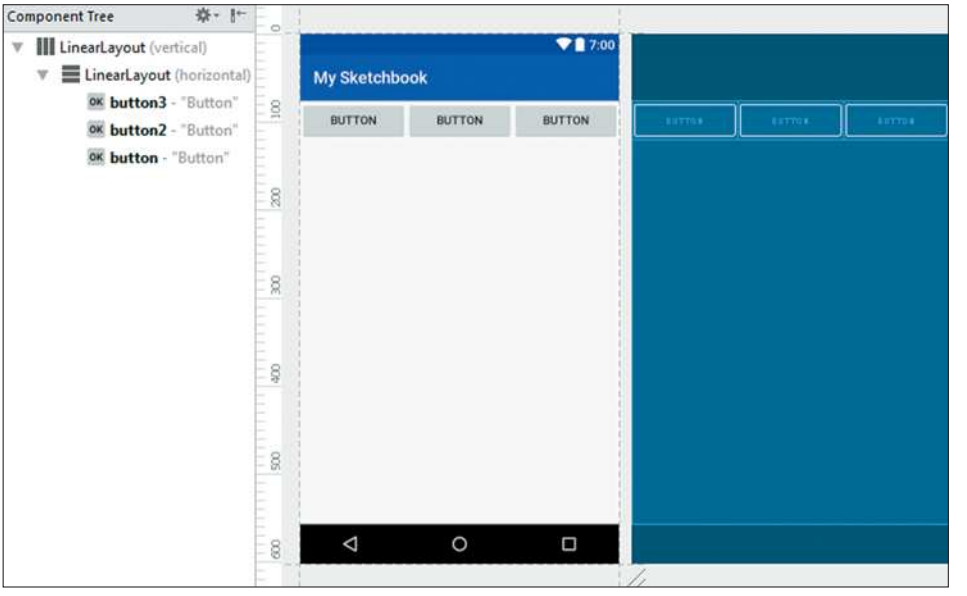
Шаг 1. Создать пустой проект My Sketchbook, удалить текстовый элемент с главного экрана.

Шаг 2. Изменить и настроить корневой элемент.

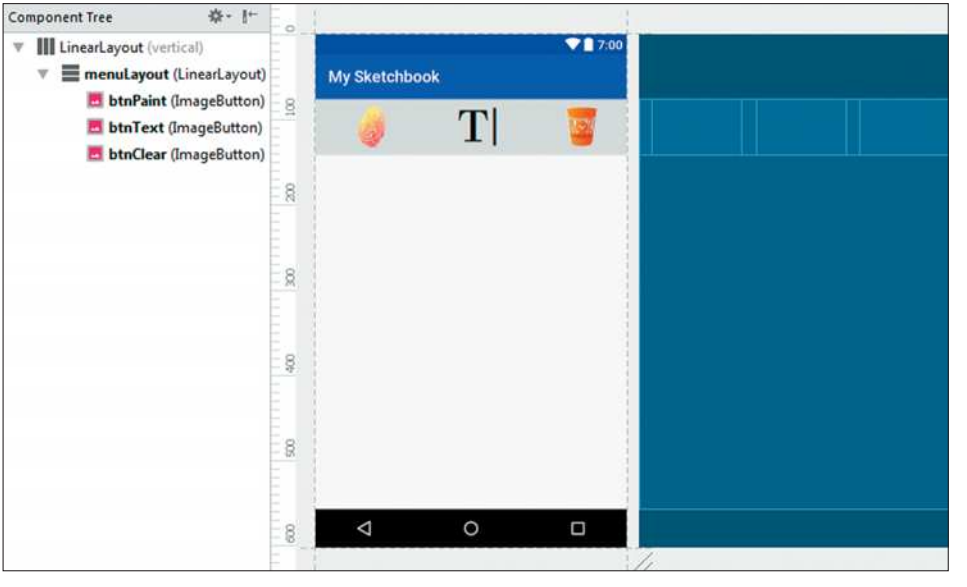
Главный экран нашего приложения будет состоять из полотна для рисования и верхнего меню. Располагаться эти элементы будут один над другим, поэтому изменим текущий корневой элемент на **вертикальный макет LinearLayout**.

Шаг 3. Добавить кнопки управления.

Для начала нам понадобятся три кнопки — для **рисования**, для **работы с текстом** и для **очистки полотна**. Эти кнопки образуют одно меню, поэтому расположим их внутри **горизонтального LinearLayout**:



Теперь, когда с общим видом разметки мы определились, изменим тип кнопок на **ImageButton**, добавим к ним изображения, зададим идентификаторы, настроим расположение относительно друг друга и зададим свойство **onClick**.



Затем перейдем в файл **MainActivity.java**, объявим и определим наши кнопки, создадим для них методы обработки событий нажатия:

activity_main.xml × MainActivity.java ×

```
1  package com.example.fedotenko_family.mysketchbook;
2
3  import ...
7
8  public class MainActivity extends AppCompatActivity {
9
10     private ImageButton btnPaint, btnText, btnClear;
11
12     @Override
13     protected void onCreate(Bundle savedInstanceState) {
14         super.onCreate(savedInstanceState);
15         setContentView(R.layout.activity_main);
16
17         btnPaint = (ImageButton) findViewById(R.id.btnPaint);
18         btnText = (ImageButton) findViewById(R.id.btnText);
19         btnClear = (ImageButton) findViewById(R.id.btnClear);
20     }
21
22     public void onBtnPaintClick(View view) {
23     }
24
25     public void onBtnTextClick(View view) {
26     }
27
28     public void onBtnClearClick(View view) {
29     }
30 }
```

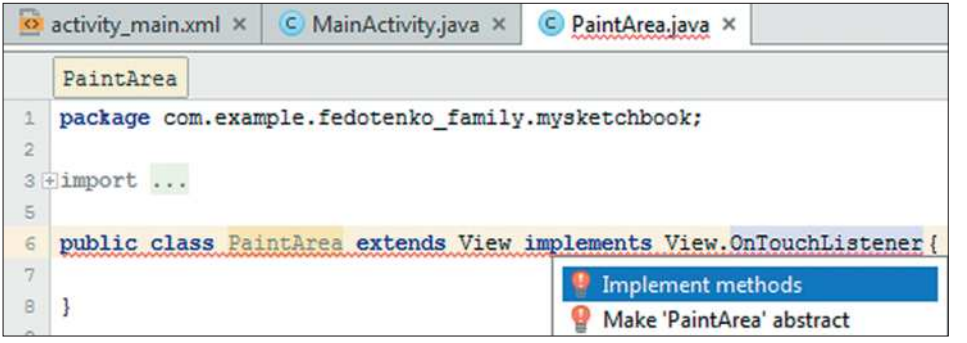
Шаг 4. Реализовать рисование пальцем.

Создадим для нашего полотна новый Java-класс **PaintArea** (в той же папке, что и класс **MainActivity**), пропишем ему расширения extends View implements View.OnTouchListener:

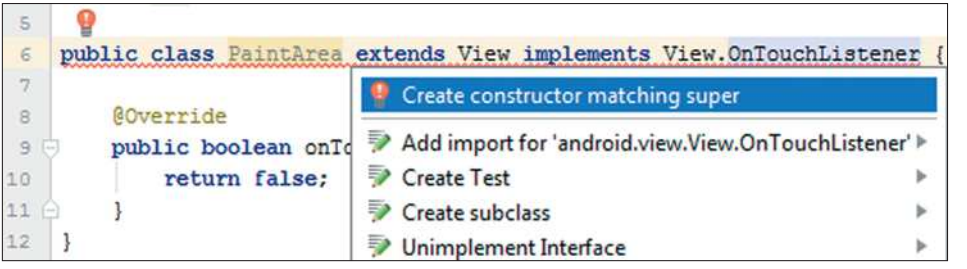
activity_main.xml × MainActivity.java × PaintArea.java ×

```
1  package com.example.fedotenko_family.mysketchbook;
2
3  import ...
5
6  public class PaintArea extends View implements View.OnTouchListener{
7
8  }
```

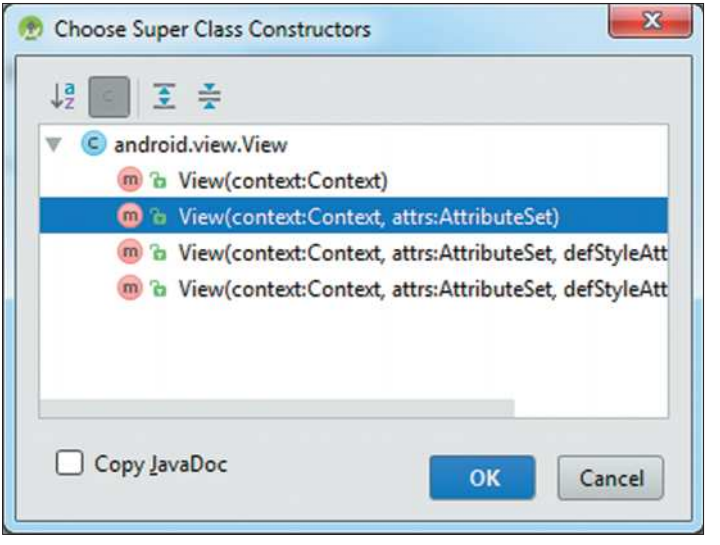
Затем нажмем **Alt+Enter** и имплементируем необходимые методы:



Снова нажмем **Alt+Enter** и выберем первый пункт — создание конструктора.



В открывшемся окне выберем вариант с двумя параметрами: контекстом и атрибутами:



Далее объявим новый объект класса **Paint** paint, зададим ему цвет (метод **setColor()**) и режим сглаживания (метод **setAntiAlias()**), соединяющий штрих и его тип (методы **setStrokeWidth()** и **setStrokeJoin()**). На всю область «навесим» прослушиватель нажатий (строка № 20):

```
13  public class PaintArea extends View implements View.OnTouchListener{
14
15      Paint paint = new Paint();
16
17      public PaintArea(Context context, @Nullable AttributeSet attrs){
18          super(context, attrs);
19
20          this.setOnTouchListener(this);
21
22          paint.setColor(Color.BLUE);
23          paint.setAntiAlias(true);
24          paint.setStrokeWidth(0.5f);
25          paint.setStrokeJoin(Paint.Join.ROUND);
26      }
27
28      @Override
29      public boolean onTouch(View v, MotionEvent event){
30          return false;
31      }
32  }
```

Рисование будет осуществляться точками, а у точки есть два параметра — ее координаты.

Пропишем точку как отдельный класс **Point**, в нем объявим переменные x и y; в дальнейшем при работе с точками будем обращаться к нему:

```
28          @Override
29          public boolean onTouch(View v, MotionEvent event) {
30              return false;
31          }
32
33          private class Point {
34              float x, y;
35          }
36      }
```

Рисовать мы будем различные фигуры, то есть набор линий, а линия — это, по сути, набор точек. В программировании с наборами элементов работают с помощью **массивов**.

Объявим массив точек (из **Point**) как объект класса **ArrayList** pointArrayList. В методе **onTouch()** пропишем, что при нажатии на экран создается объект класса **Point** — точка dot, параметрам x и y которой присваиваются соответствующие координаты нажатия. Каждая такая точка заносится в массив, а параметры dot очищаются методом **invalidate()**:

```
15    Paint paint = new Paint();
16    ArrayList<Point> pointArrayList = new ArrayList<>();
17
18    public PaintArea(Context context, @Nullable AttributeSet attrs){
19        super(context, attrs);
20
21        this.setOnTouchListener(this);
22
23        paint.setColor(Color.BLUE);
24        paint.setAntiAlias(true);
25        paint.setStrokeWidth(0.5f);
26        paint.setStrokeJoin(Paint.Join.ROUND);
27    }
28
29    @Override
30    public boolean onTouch(View v, MotionEvent event) {
31        Point dot = new Point();
32        dot.x = event.getX();
33        dot.y = event.getY();
34        pointArrayList.add(dot);
35        invalidate();
36        return true;
37    }
```

Но непосредственно «рисование» осуществляется через класс **Canvas** (*холст*). Нам нужно создать его метод **onDraw()** (*при рисовании*) и в нем прописать, что для каждой точки из текущего массива точек нужно нарисовать на «холсте» круг, в соответствующем месте и с заданными ранее параметрами paint:

```
30    @Override
31    public boolean onTouch(View v, MotionEvent event) {
32        Point dot = new Point();
33        dot.x = event.getX();
34        dot.y = event.getY();
35        pointArrayList.add(dot);
```

```
36        invalidate();
37        return true;
38    }
39
40    @Override
41    public void onDraw(Canvas canvas) {
42        for (Point point : pointArrayList) {
43            canvas.drawCircle(point.x, point.y, 10, paint);
44        }
45    }
46
47    private class Point {
```

Шаг 5. Добавить область рисования в интерфейс.

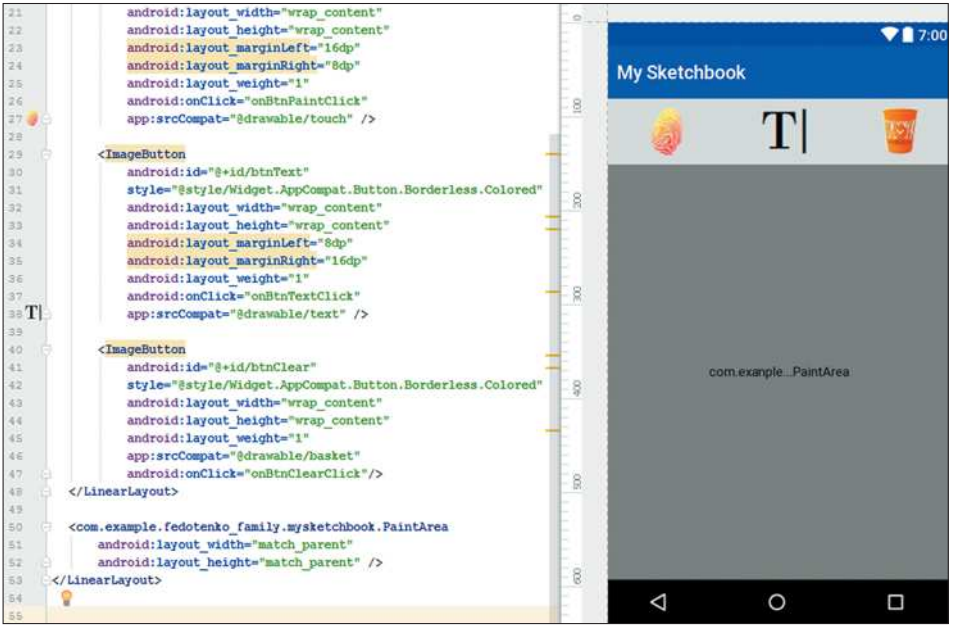
Теперь вернемся в файл xml-разметки, потому что, хотя мы и объявили полотно для рисования и определили все его параметры и методы, — в интерфейсе его все еще нет.

Поскольку мы описывали алгоритм рисования в отдельном классе, в интерфейсе этот класс можно вывести в виде объекта View и не размещать его в различных макетах и контейнерах.

Для этого после макета **горизонтальный LinearLayout**, содержащего меню, наберем название нашего класса **PaintArea**, а затем, следуя подсказкам, зададим его ширину и высоту match_parent.

```
38 T|        app:srcCompat="@drawable/text" />
39
40           <ImageButton
41               android:id="@+id/btnClear"
42               style="@style/Widget.AppCompat.Button.Borderless.Colored"
43               android:layout_width="wrap_content"
44               android:layout_height="wrap_content"
45               android:layout_weight="1"
46               app:srcCompat="@drawable/basket"
47               android:onClick="onBtnClearClick"/>
48           com.example.fedotenko_family.mysketchbook.PaintArea
49       Press Ctrl+Space to view tags from other namespaces
50         <Paint
51   </LinearLayout>
52
     Design  Text
```

Видим, что вся область экрана ниже меню превратилась в полотно для рисования:



Осталось задать элементу идентификатор:

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

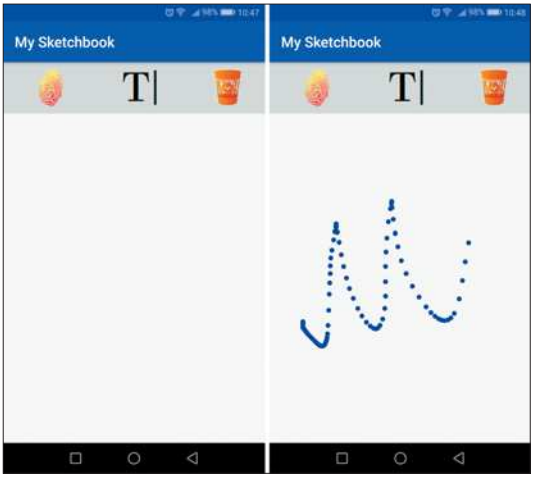
Шаг 6. Программно задать отображение области рисования.

По логике приложения, наше полотно должно отображаться при нажатии на кнопку рисования пальцем, поэтому перейдем в класс **MainActivity**, объявим и определим элемент интерфейса paintArea, в методе **onCreate()** сделаем его невидимым, а видимым только при нажатии на кнопку **btnPaint**:

```
8  public class MainActivity extends AppCompatActivity {
9
10     private ImageButton btnPaint, btnText, btnClear;
11     private PaintArea paintArea;
12
13     @Override
14     protected void onCreate(Bundle savedInstanceState) {
15         super.onCreate(savedInstanceState);
16         setContentView(R.layout.activity_main);
17
18         btnPaint = (ImageButton) findViewById(R.id.btnPaint);
19         btnText = (ImageButton) findViewById(R.id.btnText);
20         btnClear = (ImageButton) findViewById(R.id.btnClear);
```

```
21
22          paintArea = (PaintArea) findViewById(R.id.paint);
23          paintArea.setVisibility(View.INVISIBLE);
24      }
25
26      public void onBtnPaintClick(View view) {
27          paintArea.setVisibility(View.VISIBLE);
28      }
```

Теперь можно запустить и протестировать приложение. До нажатия на какую-либо кнопку не происходит ровным счетом ничего. Нажмем на кнопку рисования пальцем. Визуально экран не изменился, но теперь на нем действительно можно рисовать.



Шаг 7. Реализовать очистку экрана.

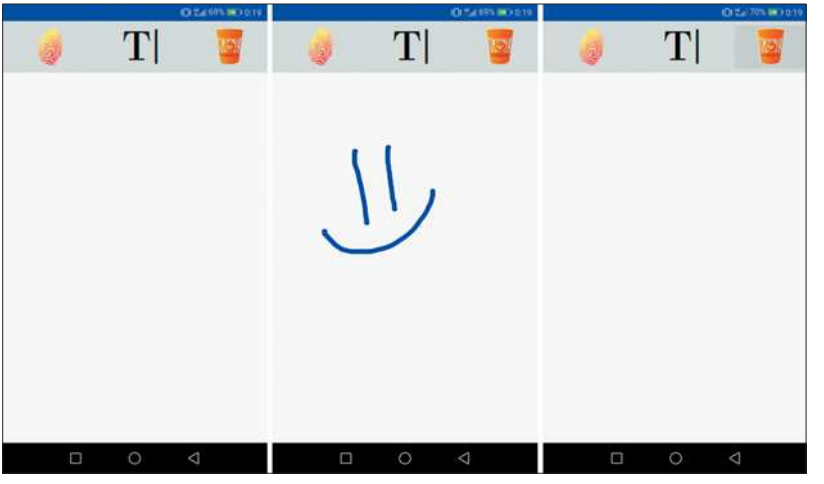
В интерфейсе мы предусмотрели кнопку очистки, а в классе **MainActivity** создали метод для ее нажатия. Добавим нужный метод **cleanPaintArea()** в классе **PaintArea** и в нем пропишем очищение массива pointArrayList и «обнуление» методом **invalidate()**:

```
47      private class Point {
48          float x, y;
49      }
50
51      public void cleanPaintArea(){
52          pointArrayList.clear();
53          invalidate();
54      }
55  }
```

Осталось применить созданный метод в классе **MainActivity** при нажатии на кнопку очистки.

```
32
33      public void onBtnClearClick(View view) {
34          paintArea.cleanPaintArea();
35      }
```

Запустим приложение. Теперь при нажатии на кнопку рисования появляется полотно для рисования, при нажатии на корзину оно очищается.



Задания

1. Активировать кнопку работы с текстом. По аналогии нужно реализовать в нашем редакторе работу с текстом. Для этого добавить на экран элемент **EditText** и прописать, чтобы он появлялся при нажатии на соответствующую кнопку, а при нажатии на корзину очищался и становился невидимым.
2. Добавить в приложение работу с камерой, чтобы в качестве фона полотна для рисования можно было установить картинку.
3. Добавить возможность изменения цветов. Например, при долгом нажатии на кнопку рисования должны появляться еще несколько кнопок для выставления разных цветов рисования.
4. Реализовать сохранение файла. Сбор всех реализованных нами элементов воедино и сохранение в файл — материал продвинутого курса. Сейчас мы можем реализовать упрощенную версию этого сохранения — добавить кнопку сохранения, а при нажатии на нее программно делать скриншот экрана.

Для того чтобы сохранять изображения, например на карту памяти устройства, нужно получить разрешение на ее использование (запись и чтение). В файле манифеста пропишем соответствующее разрешение.

```
AndroidManifest.xml × | activity_main.xml × | MainActivity.java × | PaintArea.java ×
manifest
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      package="com.example.fedotenko_family.mysketchbook">
4      <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
5
6      <application
```

В класс **MainActivity** добавим метод **makeScreenshot()** (*сделать скриншот*):

```
MainActivity.java × | AndroidManifest.xml × | activity_main.xml ×
29
30     private void makeScreenshot() {
31         Date currentDate = new Date();
32         android.text.format.DateFormat.format("yyyy-MM-dd_hh:mm:ss", currentDate);
33     }
34 }
```

Для сохранения файла нужно задать ему имя. В случае с изображениями это обычно различные комбинации с датой снимка. Поэтому объявим переменную для получения текущей даты и зададим формат ее отображения. Делать скриншот мы будем через конструкцию **try-catch**:

```
30     private void makeScreenshot() {
31         Date currentDate = new Date();
32         android.text.format.DateFormat.format("yyyy-MM-dd_hh:mm:ss",
33                                               currentDate);
34         try {
35         } catch (Throwable e) {
36             e.printStackTrace();
37         }
38     }
```

Внутри конструкции пропишем стандартный алгоритм получения скриншота: получение изображения всего окна, помещение его в кэш (участок памяти для временного хранения данных), затем сохранение как изображения **Bitmap** и очистка кэша:

```
30  private void makeScreenshot() {
31      Date currentDate = new Date();
32      android.text.format.DateFormat.format("yyyy-MM-dd_hh:mm:ss",
33                                            currentDate);
34      try {
35          View screenshot = getWindow().getDecorView().getRootView();
36          screenshot.setDrawingCacheEnabled(true);
37          Bitmap bitmap = Bitmap.createBitmap(screenshot.getDrawingCache());
38          screenshot.setDrawingCacheEnabled(false);
39      } catch (Throwable e) {
40          e.printStackTrace();
41      }
42  }
```

После этого нужно создать и сохранить в памяти соответствующий файл. В качестве пути к файлу зададим корневую папку карты памяти, имя файла — текущая дата и время, расширение — .png.

```
34      try {
35          View screenshot = getWindow().getDecorView().getRootView();
36          screenshot.setDrawingCacheEnabled(true);
37          Bitmap bitmap = Bitmap.createBitmap(screenshot.getDrawingCache());
38          screenshot.setDrawingCacheEnabled(false);
39
40          String path = Environment.getExternalStorageDirectory()
41                        .toString()+"/"+currentDate+".png";
42          File screenshotFile = new File(path);
43      } catch (Throwable e) {
44          e.printStackTrace();
45      }
```

Сохранение данных реализуется через поток **FileOutputStream**. Файл сжимается и сохраняется, а поток очищается и закрывается.

```
34      try {
35          View screenshot = getWindow().getDecorView().getRootView();
36          screenshot.setDrawingCacheEnabled(true);
37          Bitmap bitmap = Bitmap.createBitmap(screenshot.getDrawingCache());
38          screenshot.setDrawingCacheEnabled(false);
39
40          String path = Environment.getExternalStorageDirectory()
41                                .toString()+"/"+currentDate+".png";
42          File screenshotFile = new File(path);
43          FileOutputStream fileOutputStream = new FileOutputStream(screenshotFile);
44          int quality = 100;
45          bitmap.compress(Bitmap.CompressFormat.PNG, quality, fileOutputStream);
46          fileOutputStream.flush();
47          fileOutputStream.close();
48      } catch (Throwable e) {
49          e.printStackTrace();
50      }
51  }
```

Сделанный скриншот можно сразу же показать, добавив метод **showScreenshot()** (*показать скриншот*). В нем через намерение **Intent** пропишем обращение к стороннему приложению, способному открыть файл изображения:

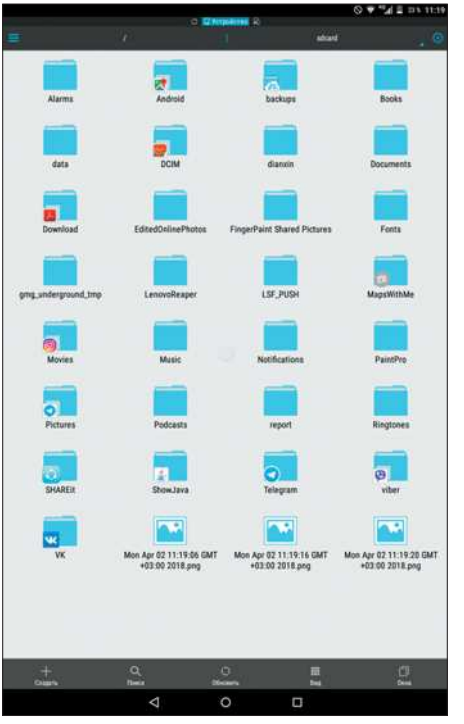
```
53  private void showScreenshot(File screenshotFile){
54      Intent intent = new Intent();
55      intent.setAction(Intent.ACTION_VIEW);
56      Uri uri = Uri.fromFile(screenshotFile);
57      intent.setDataAndType(uri, "image/*");
58      startActivity(intent);
59  }
```

Наконец, нужно добавить в интерфейс кнопку сохранения, объявить и определить ее в классе **MainActivity**, добавить метод **onClick()** для обработки ее нажатия и в нем вызвать метод **makeScreenshot()**:

```
34      try {
35          View screenshot = getWindow().getDecorView().getRootView();
36          screenshot.setDrawingCacheEnabled(true);
37          Bitmap bitmap = Bitmap.createBitmap(screenshot.getDrawingCache());
38          screenshot.setDrawingCacheEnabled(false);
39
40          String path = Environment.getExternalStorageDirectory()
41                                  .toString()+"/"+currentDate+".png";
42          File screenshotFile = new File(path);
43          FileOutputStream fileOutputStream = new FileOutputStream(screenshotFile);
44          int quality = 100;
45          bitmap.compress(Bitmap.CompressFormat.PNG, quality, fileOutputStream);
46          fileOutputStream.flush();
47          fileOutputStream.close();
48
49          showScreenshot(screenshotFile);
50      } catch (Throwable e) {
51          e.printStackTrace();
52      }
53  }
54
55  private void showScreenshot(File screenshotFile) {
56      Intent intent = new Intent();
57      intent.setAction(Intent.ACTION_VIEW);
58      Uri uri = Uri.fromFile(screenshotFile);
59      intent.setDataAndType(uri, "image/*");
60      startActivity(intent);
61  }
62
63  public void onClick(View view) {
64      makeScreenshot();
65  }
```

Метод **showScreenshot()** лучше вызвать сразу после сохранения скриншота на карте памяти устройства.

Теперь при запуске и нажатии кнопки будут отображаться сделанные скриншоты, а в корневой папке карты памяти будут появляться их файлы:



5. По аналогии с заданием 4 реализовать сохранение скриншота не всего экрана, а его части. Для этого нужно только немного изменить метод **makeScreenshot()**.

Итоги главы 5

В главе 5 мы проделали разноплановую работу и научились:

1. Форматировать текст с помощью тегов XML.
2. Работать с текстами большого объема — делать их прокручиваемыми.
3. Выполнять различные действия не только при простом одиночном нажатии на элемент или область экрана, но и при двойном нажатии, долгом нажатии, отпущенном нажатии.
4. Обрабатывать жесты — скроллинг и свайпинг.
5. Открывать камеру из наших приложений, получать изображения с нее и использовать их в нашем интерфейсе.

Глава 6. Интернет и базы данных

6.1. Интернет

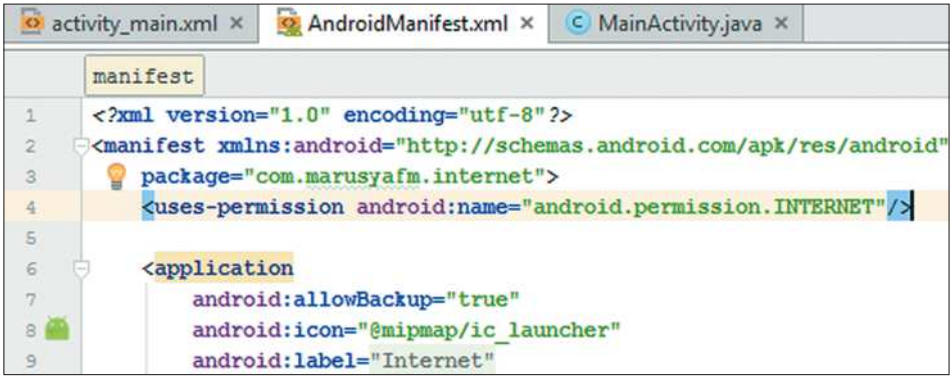
Многим приложениям для работы требуется доступ к сети Интернет. Причем некоторые методы и сервисы позволяют нам, как разработчикам, даже не прописывать полностью процесс «выхода в Интернет».

Но в то же время для соблюдения эвристик юзабилити при работе с Интернетом нам следует организовать проверку соединения и оповещение пользователя в случае, если соединение не установлено. Тогда наш пользователь будет знать, что приложение не зависло и не «сломалось», а для дальнейшей работы с ним нужно лишь обеспечить доступ к Интернету.

Концепция мини-приложения: проверка подключения к Интернету по нажатию на кнопку. Изображение на экране будет меняться в зависимости от наличия или отсутствия подключения.

Интерфейс сделаем самым простым — изображение-кнопка в центре экрана. Подготовим еще две картинки: если подключение отсутствует, то при нажатии на кнопку в качестве фона будет устанавливаться одна картинка (зададим ей **идентификатор** offline), а при наличии работающего Интернета — другая (**идентификатор** — online).

Для использования Интернета приложению нужно запросить соответствующее **разрешение**. Как мы выяснили вначале, все разрешения для приложения прописываются в файле манифеста **AndroidManifest.xml**. Открываем файл и добавляем в него тег **uses-permission** (*разрешение на использование*, строка № 4). В нем указываем **имя разрешения** (атрибут **name**) — android.permission.INTERNET (*разрешение на использование Интернет* из пространства имен *android*):



После этого разрешения добавляем еще два разрешения: на доступ к состоянию подключения сетей и Wi-Fi (строки № 5 и 6 соответственно):

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      package="com.marusyafm.internet">
4      <uses-permission android:name="android.permission.INTERNET"/>
5      <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
6      <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
```

Теперь обратимся к коду. В главном Java-классе **MainActivity** объявим и определим элементы интерфейса: **ImageButton** myButton и **ConstraintLayout** constraintLayout. Создадим обработчик события нажатия изображения-кнопки **onClick().**

За определение состояния подключения к сети и его изменения отвечает класс **ConnectivityManager** (*менеджер подключений*). Создадим экземпляр класса **connectivityManager** и получим его методом **getSystemService()** с аргументом Context.CONNECTIVITY_SERVICE — это стандартная процедура.

Еще один класс, стандартно используемый в таких случаях для описания статуса сетевого интерфейса, — **NetworkInfo.** Создадим его экземпляр networkInfo и с помощью метода **getActiveNetworkInfo()** получим текущее сетевое соединение:

```
27  public void onClick(View view) {
28      ConnectivityManager connectivityManager = (ConnectivityManager)
29                              getSystemService(Context.CONNECTIVITY_SERVICE);
30      NetworkInfo networkInfo = connectivityManager.getActiveNetworkInfo();
31  }
```

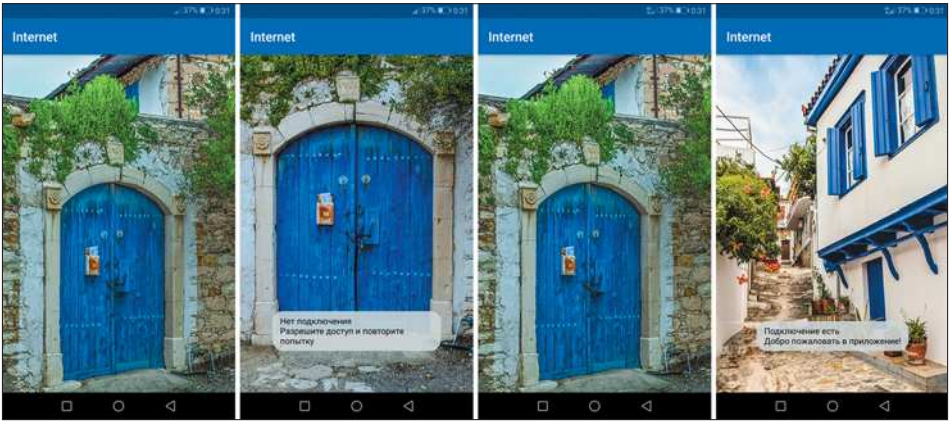
Обработку результата запроса текущего соединения организуем в конструкции **if-else:**

```
27  public void onClick(View view) {
28      ConnectivityManager connectivityManager = (ConnectivityManager)
29                              getSystemService(Context.CONNECTIVITY_SERVICE);
30      NetworkInfo networkInfo = connectivityManager.getActiveNetworkInfo();
31
32      if (networkInfo != null && networkInfo.isConnected()) {
33      } else{
34      }
```

Если соединение найдено и подключение выполнено, делаем изображение-кнопку невидимым, устанавливаем в качестве фона изображение online и выводим соответствующее всплывающее сообщение. Если нет — изображение offline и текст об отсутствии подключения:

```
27  public void onClick(View view) {
28      ConnectivityManager connectivityManager = (ConnectivityManager)
29                          getSystemService(Context.CONNECTIVITY_SERVICE);
30      NetworkInfo networkInfo = connectivityManager.getActiveNetworkInfo();
31
32      if (networkInfo != null && networkInfo.isConnected()) {
33          myButton.setVisibility(View.INVISIBLE);
34          constraintLayout.setBackground(getDrawable(R.drawable.online));
35          Toast toast1 = Toast.makeText(getApplicationContext(),
36                  "Подключение есть \nДобро пожаловать в приложение!", Toast.LENGTH_LONG);
37          toast1.show();
38      } else{
39          myButton.setVisibility(View.INVISIBLE);
40          constraintLayout.setBackground(getDrawable(R.drawable.offline));
41          Toast toast2 = Toast.makeText(getApplicationContext(),
42                  "Нет подключения \nРазрешите доступ и повторите попытку", Toast.LENGTH_LONG);
43          toast2.show();
44      }
```

Готово! Запустим приложение и «испытаем» его при включенном и выключенном Интернете. Можно использовать как Wi-Fi, так и мобильный Интернет.



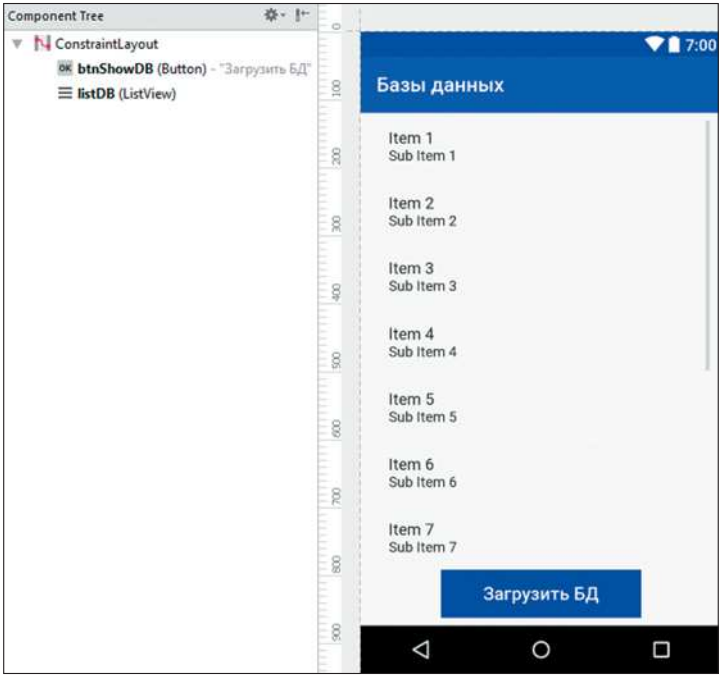
6.2. Базы данных

На примере строковых ресурсов мы уже разбирали, почему не нужно записывать абсолютно все данные в код приложения. Но тогда речь шла об отдельных текстах. Приложения же часто работают с большими объемами однотипных данных (список контактов, каталог товаров, записи с фитнес-браслета), и для работы с ними предназначены **базы данных (БД)**.

В **Android Studio** реализована поддержка БД **SQLite** — имеются все необходимые классы, интерфейсы и методы для работы с БД.

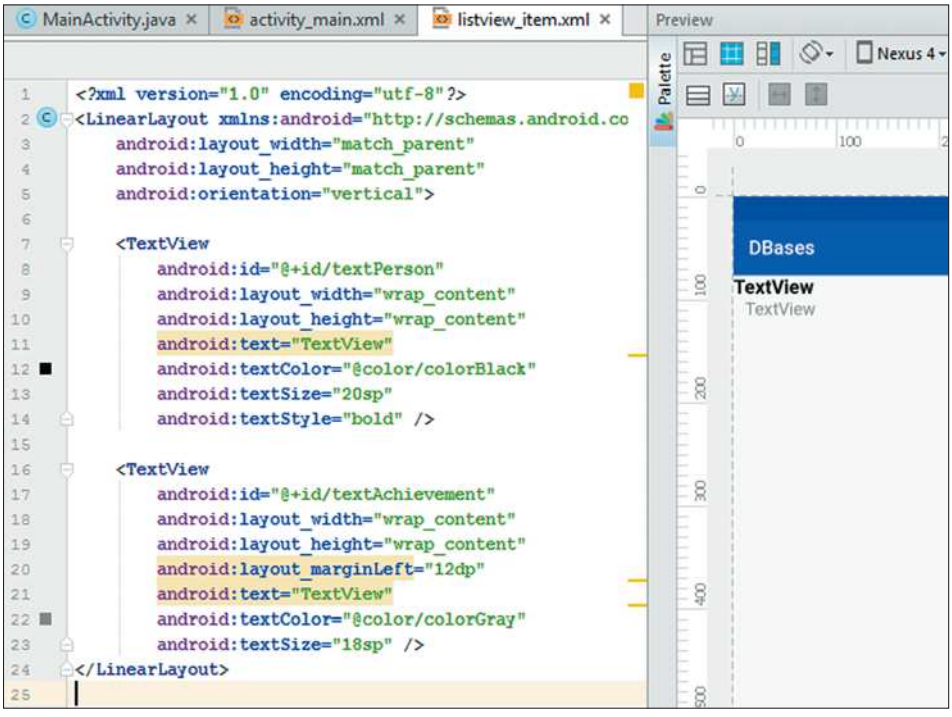
Концепция мини-приложения: на экране расположена кнопка, при нажатии на которую в контейнер **ListView** загружается содержимое базы данных. Это самая важная функция при работе с БД, поэтому с нее мы и начнем.

Создадим пустой Android Studio проект DBases. Для работы с базой данных нам понадобятся два элемента: **кнопка**, чтобы загружать БД, и **список** — контейнер **ListView**, чтобы отображать ее содержимое:

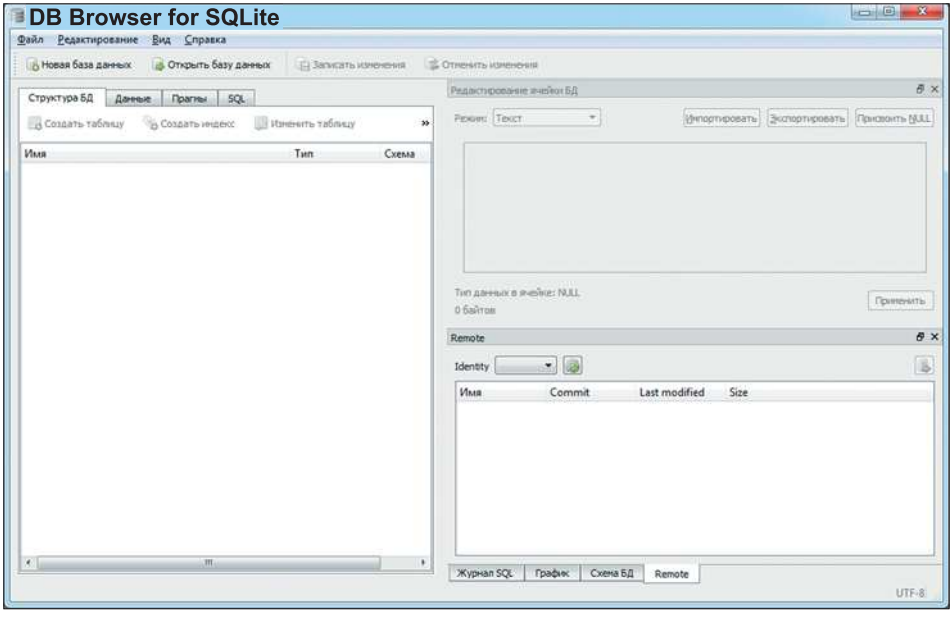


Содержимое базы данных будет отображаться не в виде таблицы, а в виде структурированного списка, поэтому нам нужно спроектировать, как будет выглядеть элемент этого списка.

Создадим новый файл xml-разметки listview_item, а в качестве корневого элемента для него выберем вертикальный **LinearLayout**. Добавим два элемента **TextView** для отображения названия элемента БД и его описания. Предположим, что в нашей БД будет храниться информация об известных личностях в IT-сфере. Тогда первому элементу **TextView** зададим идентификатор textPerson — в нем будут отображаться имена, а второму — textAchievement для отображения краткого описания, чем этот человек известен:

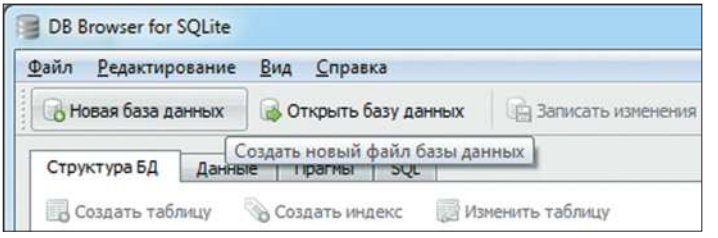


Теперь создадим саму базу данных. Это можно сделать программно, а можно с помощью различных сторонних инструментов, например **DB Browser for SQLite** — самого популярно-

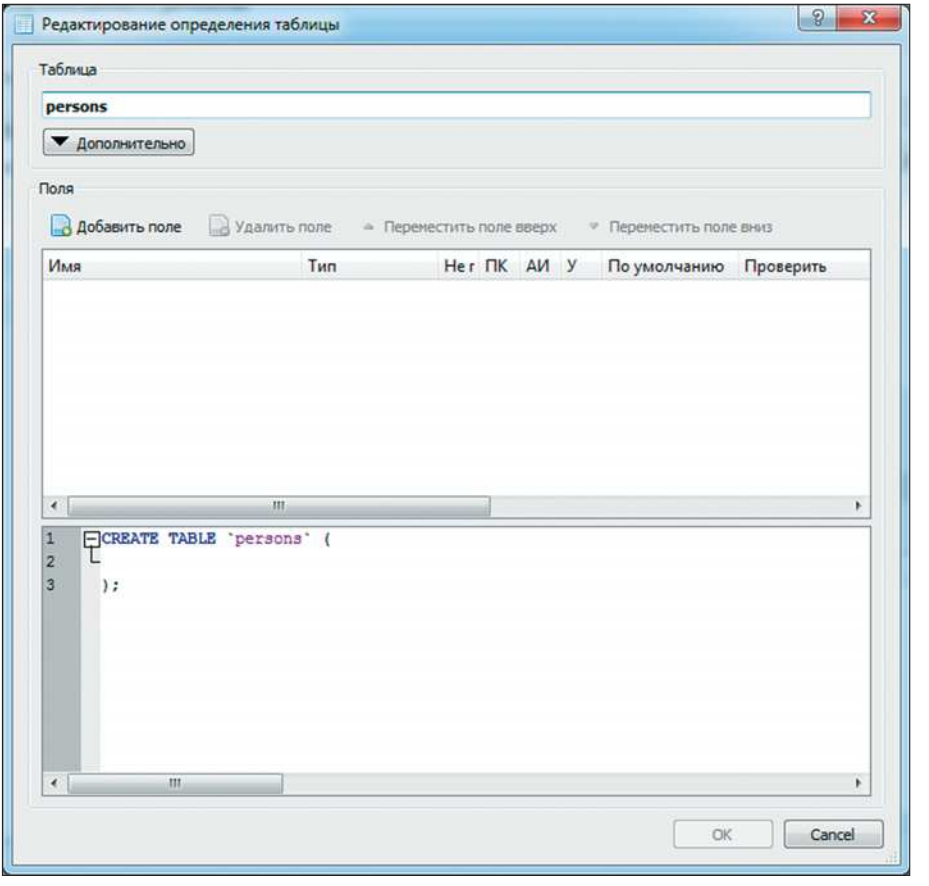


го. Скачать его можно бесплатно с официального сайта **http://sqlitebrowser.org/**. Программа русифицирована, а интерфейс дружествен и интуитивно понятен.

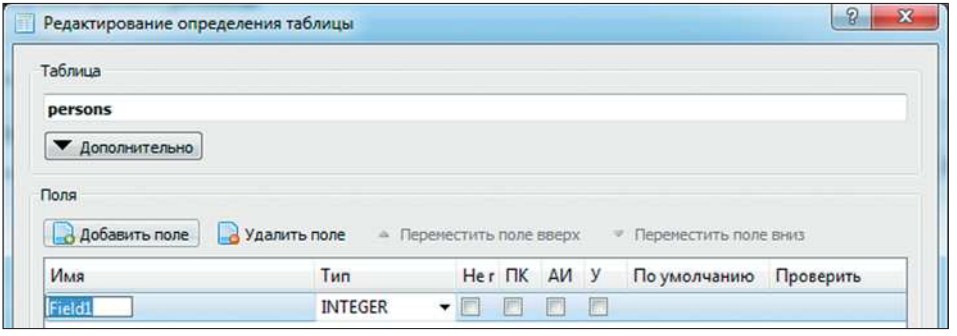
Новая база данных создается нажатием на кнопку **Новая база данных:**



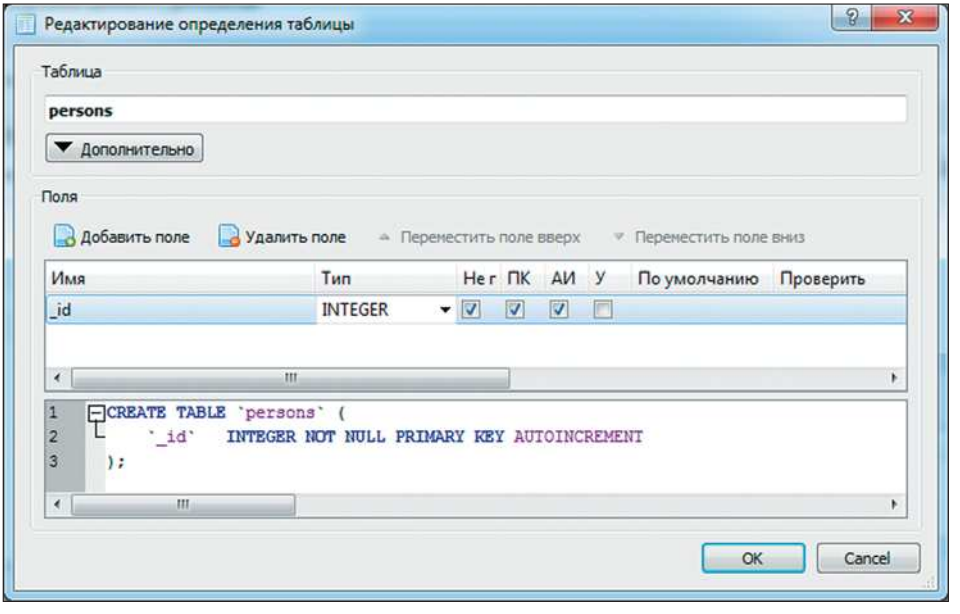
Поскольку наша база данных содержит информацию об известных IT-персонах, зададим ей имя it_geniuses.db. Расположение можно выбрать любое. Далее нажатием на кнопку **Создать таблицу** создадим новую таблицу persons:



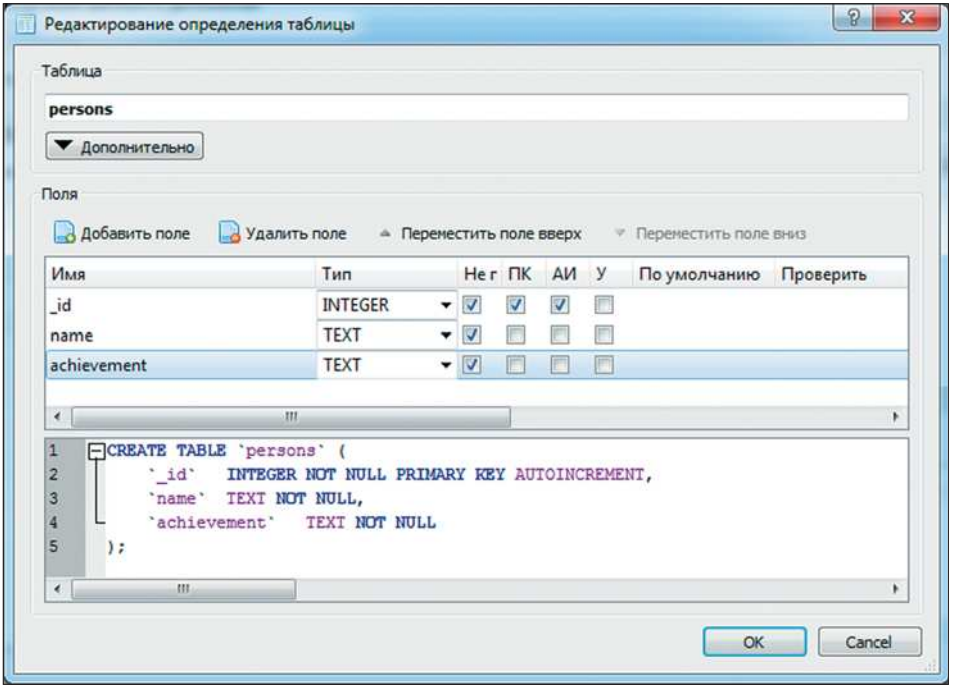
С помощью кнопки **Добавить поле** создадим в таблице новое поле:



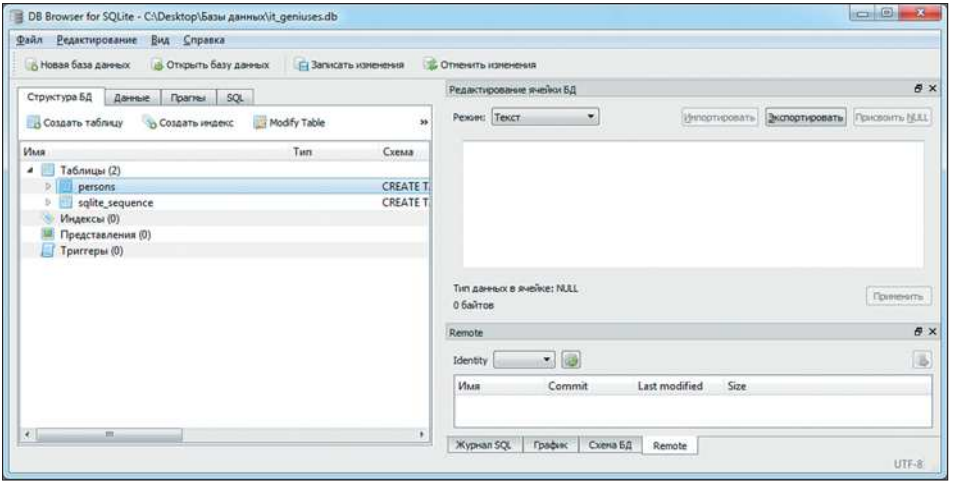
Первое поле — всегда идентификатор. В **SQLite** оно обычно именуется _id. Зададим идентификатору тип данных INTEGER, поставим галочки около полей **Не пустое**, **ПК** (первичный ключ) и **АИ** (автоматическая индексация, счетчик):



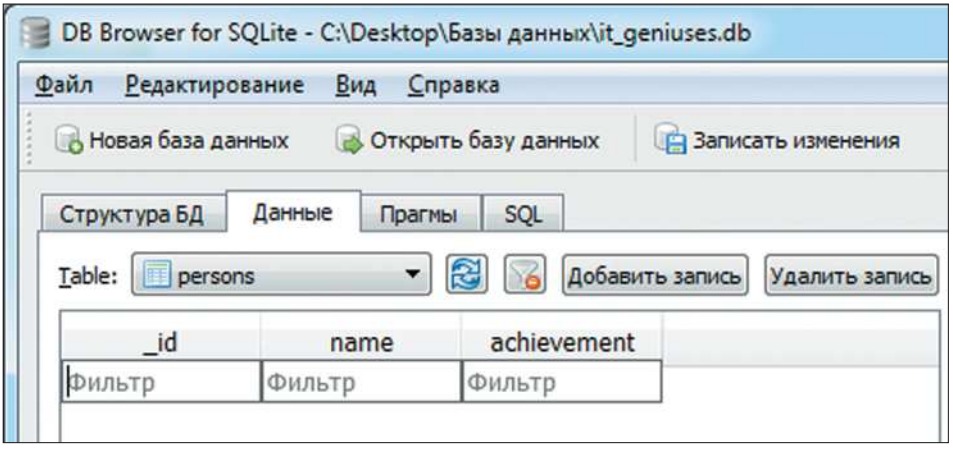
По аналогии добавляем еще два поля — name и achievement (*имя* и *достижение*) и нажимаем **OK:**



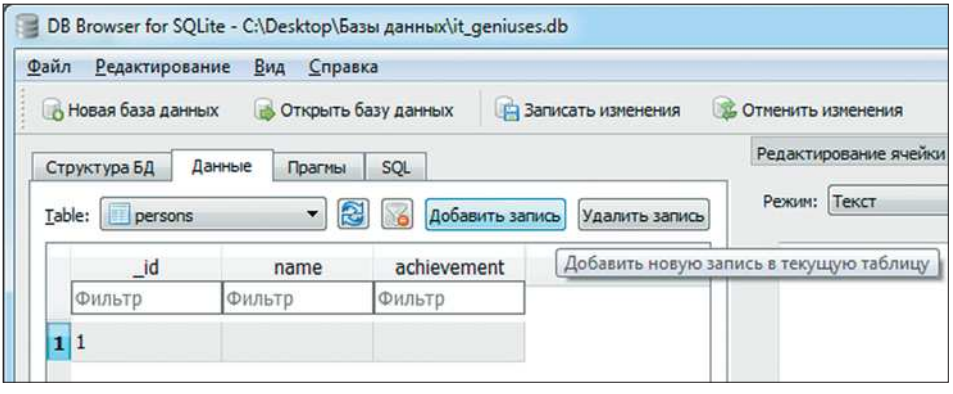
Наша таблица появилась в списке:



Теперь ее нужно заполнить. Переходим на вкладку **Данные:**



Для того чтобы добавить запись в таблицу, нажимаем кнопку **Добавить запись:**



Итак, записями нашей базы будут персоны, без которых, отчасти, мы бы сегодня (как Android-разработчики) не смогли эти записи делать:

1. **Сергей Брин** — один из основателей Google.

© Steve Jurvetson
Источник: Wikimedia

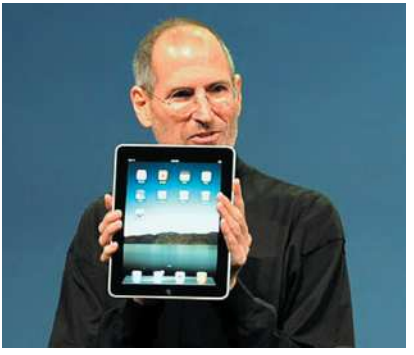
О проектах и достижениях Google можно говорить бесконечно; в частности, компания является одним из разработчиков операционной системы Android (вместе с Android Inc. и другими) и официальным разработчиком нашей **Android Studio**.

2. **Билл Гейтс**, пожалуй, не нуждается в представлении.

© World Economic Forum
Источник: Wikimedia

Большинство Android-разработчиков в мире работают на компьютерах с операционной системой Windows и пользуются многочисленными продуктами Microsoft. Да и **Android Studio** изначально создавалась только под Windows.

3. **Стив Джобс** — еще одна известнейшая личность.



© matt buchanan
Источник: Wikimedia

Хотя продукция Apple является главным конкурентом Android, следует признать, что именно благодаря этой конкуренции и происходит основное и такое качественное развитие обоих брендов.

4. **Ада Лавлейс** — первая женщина-программист.

Ада Лавлейс работала за столетие до появления привычного нам компьютера и считается не только первой женщиной-программистом, но и первым программистом в мире.

5. **Линус Торвальдс** — создатель ОС Linux.

© Alex Dawson
Источник: Wikimedia

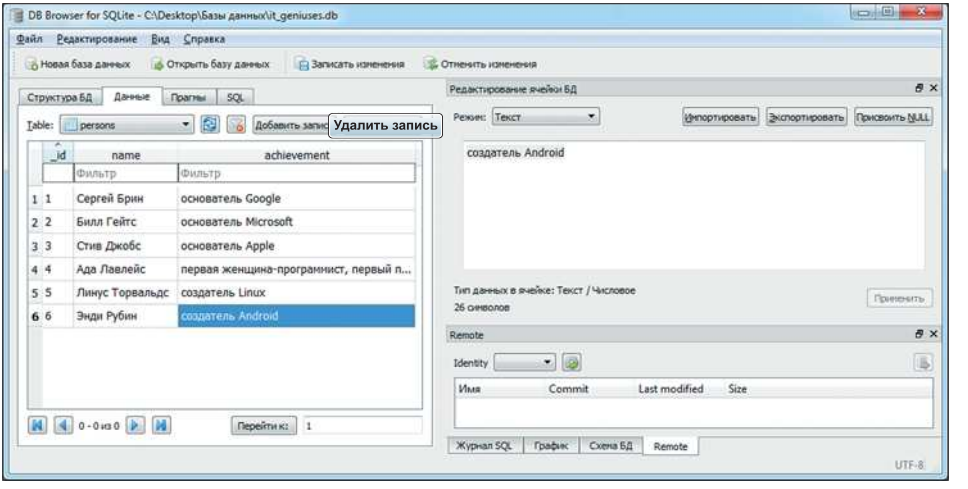
Говоря об операционных системах, несправедливо не упомянуть создателя свободной ОС Linux.

6. **Энди Рубин** — создатель Android.

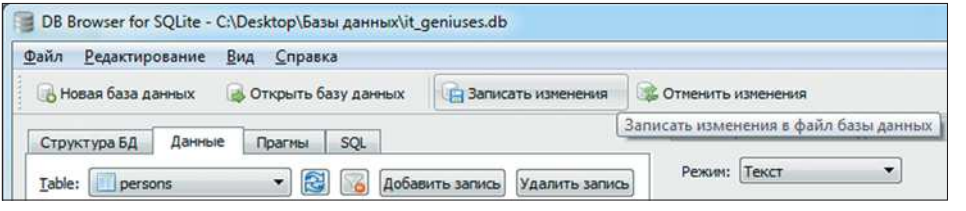
© Yoichiro Akiyama from Tokyo, Japan
Источник: Wikimedia

Он не так известен, как Стив Джобс, но для нас сейчас это одно из ключевых имен — создатель Android. Кстати, Android и назван в честь Рубина — это микс его имени Энди (Andy) и того самого зеленого дроида (droid).

Итак, добавляем несколько записей (при необходимости удаляем нажатием на кнопку **Удалить запись**):



Окончив редактирование данных в таблице, нажимаем **Записать изменения:**

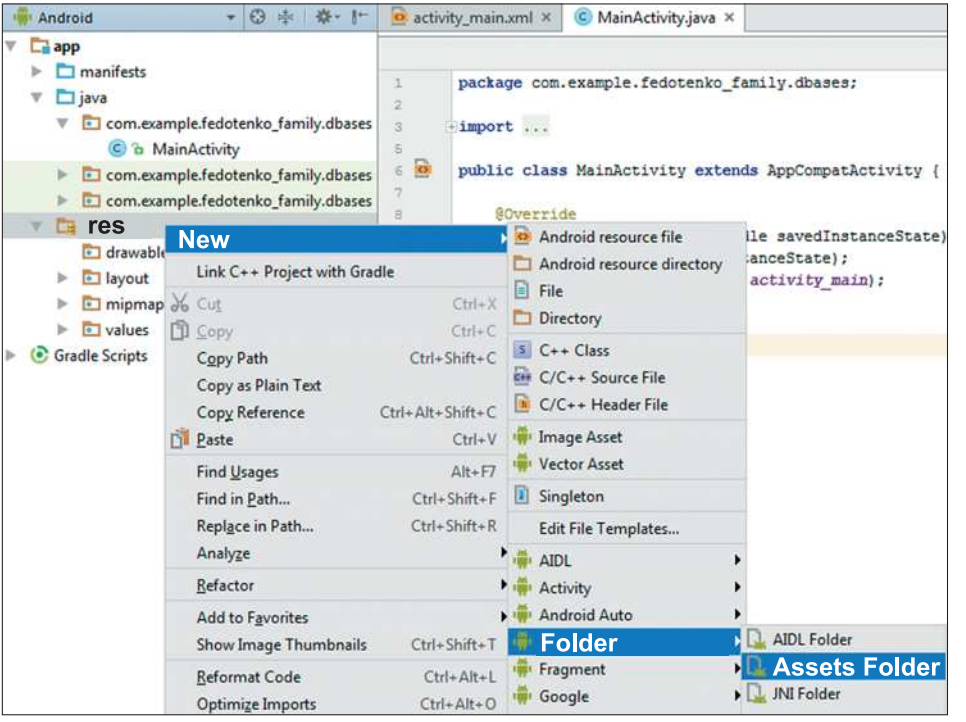


На этом подготовительная работа завершена. Вернемся в **Android Studio.**

6.2.1. Подключение БД к проекту Android Studio

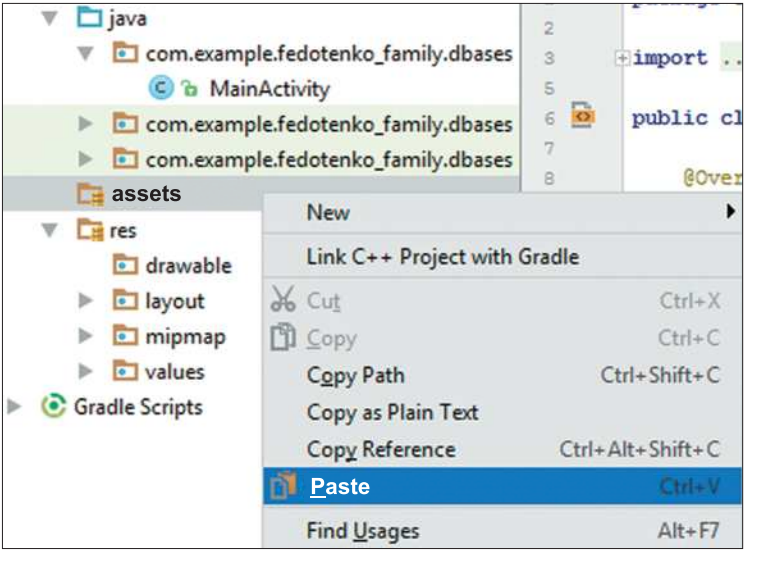
Файлы БД должны храниться в папке проекта **assets** (*ктивы*). Это папка проекта Android Studio для хранения файлов, не входящих в основной список ресурсов (файлы БД, дополнительные скрипты и так далее). В отличие от ресурсов в папке **res** таким файлам не нужно задавать идентификаторы ресурсов, для доступа к ним достаточно прописать путь.

Использование активов не является обязательным и встречается не в каждом проекте, поэтому по умолчанию такой папки в нашем проекте нет. Создадим ее нажатием правой кнопкой мыши на папке **res**, затем **New** → **Folder** → **Assets Folder:**

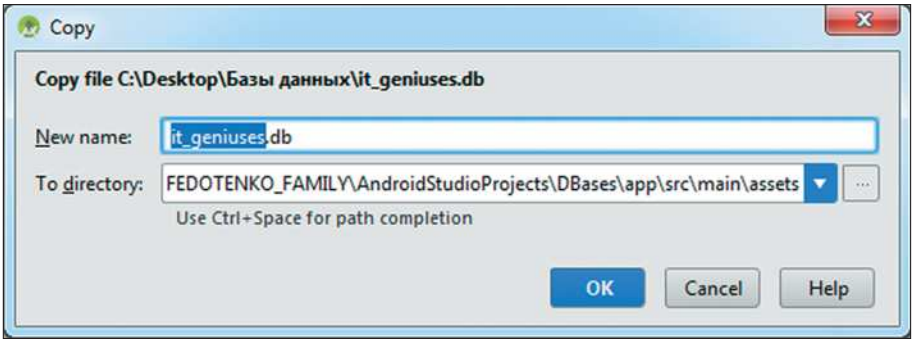


В открывшемся окне все оставляем без изменений, нажимаем **Finish.**

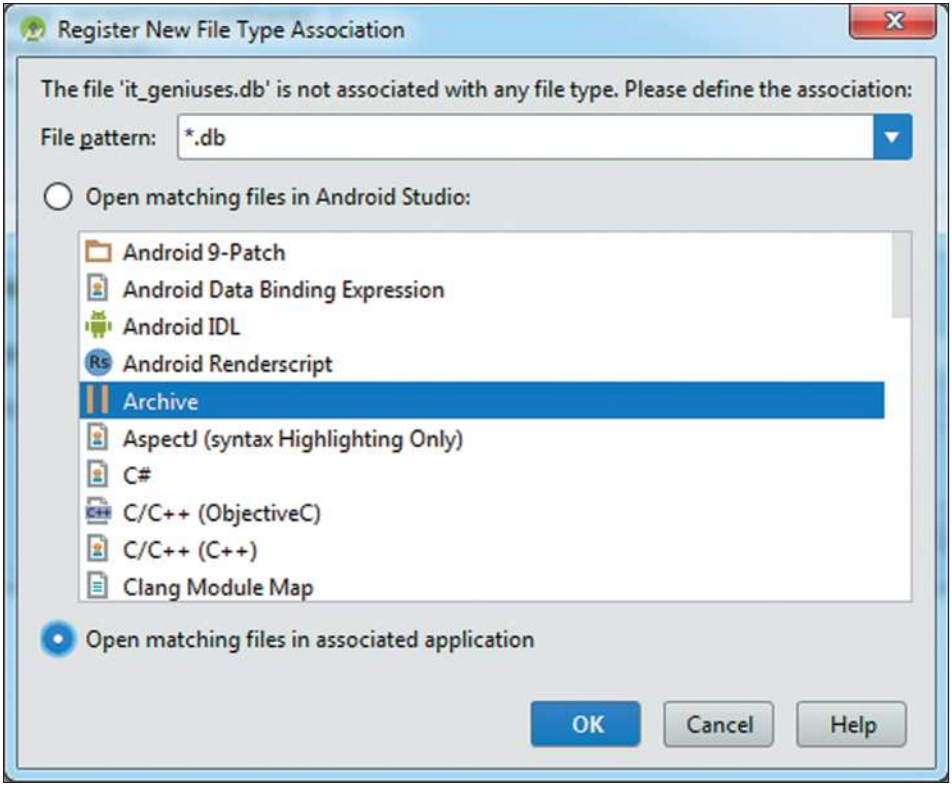
Теперь нужно найти файл нашей БД в памяти компьютера, скопировать его и вставить в проект Android Studio в только что созданную папку:



В открывшемся окне задаем имя файла (лучше оставить текущее) и нажимаем **OK:**



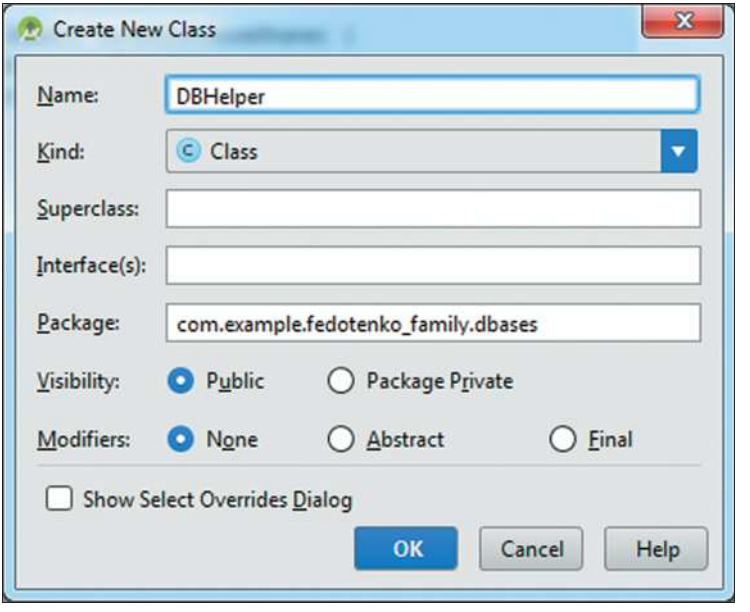
На следующем шаге обязательно выбираем второй пункт — **Open matching files in associated application** (*открыв ть соответствующие ф йлы в связ нных приложениях*). Это нужно для того, чтобы **Android Studio** восприняла файл нашей БД именно как файл стороннего приложения и не определила его как один из своих внутренних типов (архив, C#-класс, один из скриптов



и так далее). В этом случае при открытии файла БД в дальнейшем могут возникнуть проблемы с распознаванием файла и наша база данных просто не загрузится.

Файл БД появился в папке **assets**.

Далее для работы с базой данных нужно создать новый Java-класс. Назовем его DBHelper:



Расширяем его классом **SQLiteHelper**. Нажатием клавиш **Alt+Enter** реализуем все требуемые методы:

activity_main.xml × | MainActivity.java × | DBHelper.java ×

```
1  package com.example.fedotenko_family.dbases;
2
3  import android.database.sqlite.SQLiteDatabase;
4  import android.database.sqlite.SQLiteOpenHelper;
5
6  public class DBHelper extends SQLiteOpenHelper {
7
8      @Override
9      public void onCreate(SQLiteDatabase db) {
10     }
11
12     @Override
13     public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion){
14     }
15 }
```

Объявим необходимые для работы с БД переменные: DB_NAME (имя базы данных), DB_LOCATION (ее расположение), DB_VERSION (версия, нужна при обновлении БД) и контекст myContext:

```
14  public class DBHelper extends SQLiteOpenHelper {
15
16      private static String DB_NAME = "it_geniuses.db";
17      private static String DB_LOCATION;
18      private static final int DB_VERSION = 1;
19
20      private final Context myContext;
21
```

Создадим конструктор класса, в нем определим контекст (**this** — текущую Activity) и расположение БД:

```
7   public class DBHelper extends SQLiteOpenHelper {
8
9     private static String DB_NAME = "it_geniuses.db";
10    private static String DB_LOCATION;
11    private static final int DB_VERSION = 1;
12
13    private final Context myContext;
14
15    public DBHelper(Context context) {
16        super(context, DB_NAME, null, DB_VERSION);
17        this.myContext = context;
18        DB_LOCATION = context.getApplicationInfo().dataDir+"/databases/";
19    }
```

Для проверки существования файла БД добавим метод **checkDB()**, который возвращает информацию о наличии (или отсутствии) такого файла:

```
17    public DBHelper(Context context) {
18        super(context, DB_NAME, null, DB_VERSION);
19        this.myContext = context;
20        DB_LOCATION = context.getApplicationInfo().dataDir+"/databases/";
21    }
22
23    private boolean checkDB() {
24        File fileDB = new File(DB_LOCATION + DB_NAME);
25        return fileDB.exists();
26    }
27
```

Для дальнейшей синхронизации базу данных при открытии «копируют». Объявляют два потока: **InputStream** и **OutputStream.** В первый из них помещают локальную БД, а во второй — пустую. Затем побитово копируют в пустую все содержимое локальной.

```
21  public DBHelper(Context context) {
22      super(context, DB_NAME, null, DB_VERSION);
23      this.myContext = context;
24      DB_LOCATION = context.getApplicationInfo().dataDir + "/databases/";
25
26      copyDB();
27  }
28
29  private boolean checkDB() {
30      File fileDB = new File(DB_LOCATION + DB_NAME);
31      return fileDB.exists();
32  }
33
34  private void copyDB() {
35      if (!checkDB()) {
36          this.getReadableDatabase();
37          try {
38              copyDBFile();
39          } catch (IOException e) {}
40      }
41  }
42
43  private void copyDBFile() throws IOException {
44      InputStream inputStream = myContext.getAssets().open(DB_NAME);
45      OutputStream outputStream = new FileOutputStream(DB_LOCATION + DB_NAME);
46      byte[] buffer = new byte[1024];
47      int length;
48      while ((length = inputStream.read(buffer)) > 0) {
49          outputStream.write(buffer, 0, length);
50      }
51      outputStream.flush();
52      outputStream.close();
53      inputStream.close();
54  }
```

Вернемся в главный класс **MainActivity**. Объявим и определим объект класса **DBHelper**, базу данных **SQLite**, а также наши элементы интерфейса — кнопку и **ListView**.

MainActivity.java × | DBHelper.java × | activity_main.xml ×

```
1   package com.example.fedotenko_family.dbases;
2
3   import ...
8
9   public class MainActivity extends AppCompatActivity {
10
11      private DBHelper dbHelper;
12      private SQLiteDatabase database;
13
```

```
14    private Button btnShowDB;
15    private ListView listDB;
16
17    @Override
18    protected void onCreate(Bundle savedInstanceState) {
19        super.onCreate(savedInstanceState);
20        setContentView(R.layout.activity_main);
21
22        btnShowDB = (Button) findViewById(R.id.btnShowDB);
23        listDB = (ListView) findViewById(R.id.listDB);
24
25        dbHelper = new DBHelper(this);
26    }
```

Саму базу данных it_geniuses получаем из объекта класса **DBHelper** методом **getWritableDatabase()**:

```
26        dbHelper = new DBHelper(this);
27
28        try {
29            database = dbHelper.getWritableDatabase();
30        } catch (SQLException e) {
31            throw e;
32        }
33    }
```

И наконец, чтобы при нажатии на кнопку элемент **ListView** заполнялся содержанием нашей БД, пропишем обработчик события нажатия кнопки — метод **onClick()**.

Наш список IT-персон — это массив. Объявим его как объект класса **ArrayList** persons (*персоны*) (строка № 42). Но каждый элемент этого списка (то есть отдельная персона) состоит из двух (имя и достижение). Для того чтобы иметь возможность отобразить обе эти характеристики, объявим элемент массива person как объект класса **HashMap** (строка № 43). Класс **HashMap** хранит данные в хеш-таблицах в виде пар ключ–значение. В нашем случае ключом будет имя персоны, а значением — достижение, и оба будут иметь тип данных String, так как в них записан текст:

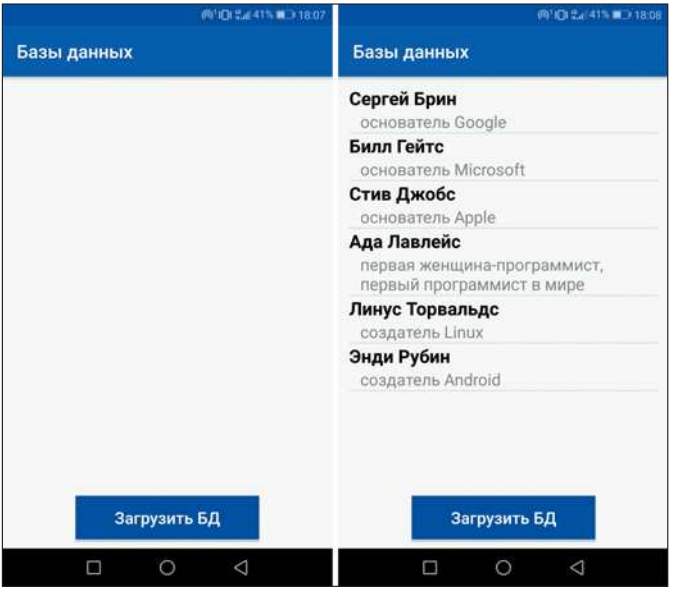
```
41  public void onClick(View view) {
42      ArrayList<HashMap<String, String>> persons = new ArrayList<>();
43      HashMap<String, String> person;
44
45      Cursor cursor = database.rawQuery("SELECT * FROM persons", null);
46      cursor.moveToFirst();
47      while (!cursor.isAfterLast()) {
48          person = new HashMap<>();
49          person.put("name",  cursor.getString(1));
50          person.put("achievement",  cursor.getString(2));
51          persons.add(person);
52          cursor.moveToNext();
53      }
54      cursor.close();
55
56      SimpleAdapter adapter = new SimpleAdapter(this, persons, R.layout.listview_item,
57              new String[]{"name", "achievement"},
58              new int[]{R.id.textPerson, R.id.textAchievement});
59      listDB.setAdapter(adapter);
60  }
```

Класс **Cursor** содержит методы для навигации по выборке. Объявим его объект cursor (строка № 45) и методом **rawQuery()** с помощью SQL-запроса SELECT * FROM persons выберем все записи из таблицы **persons**. Затем, чтобы анализировать данные по порядку, переместим курсор к первой записи методом **moveToFirst()** (*перейти к первому*, строка № 46).

И пока курсор не достиг последней записи, то есть пока не выполнится метод **isAfterLast()** (*н ходится после последнего*, строка № 47), каждому объекту класса **HashMap** person в соответствие ключу ставится полученное из БД имя (строка № 49), а каждому значению — достижение (строка № 50). Затем полученная пара записывается в массив persons (строка № 51), а курсор переходит к следующей записи методом **moveToNext()** (*перейти к следующему*, строка № 52). После этого объект cursor закрывается для освобождения памяти (строка № 54).

В строках № 56–59 объявляется объект класса **SimpleAdapter** (*простой д птер*) adapter, который ставит в соответствие полученный массив persons с предназначенным для его отображения элементом интерфейса — макетом listview_item.

На этом все. Запустим приложение и убедимся, что содержание созданной нами таблицы действительно выводится на экран по нажатию кнопки.



Это самая важная часть работы с БД. Научившись загружать данные в приложение, мы сможем разрабатывать действительно сложные проекты, с большими объемами информации. Если изменить структуру элемента списка БД, то можно создавать тесты с вариантами ответов, списки друзей и контактов, анкеты и различные справочники.

Задание

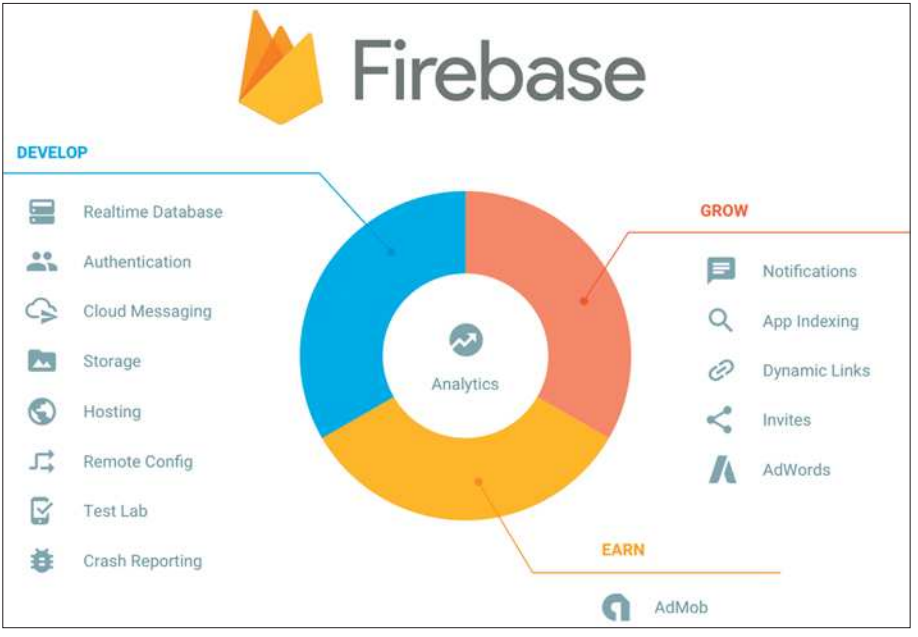
Добавить в БД еще несколько таблиц с разной структурой, а на главный экран приложения — соответствующие им кнопки. Доработать приложение, чтобы при нажатии на разные кнопки открывались соответствующие им таблицы.

Подсказка. Если изменения вносятся в БД не через приложение, то нужно менять версию БД в коде:

```
private static final int DB_VERSION = 1;
```

6.3. Инструмент Firebase

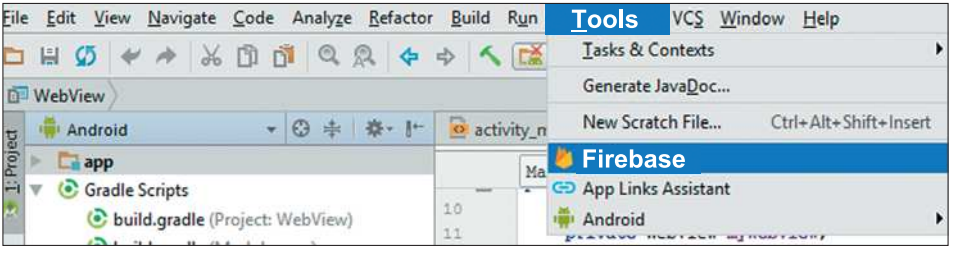
В **Android Studio**, начиная с версии 2.2.0, был встроен инструмент **Firebase:**

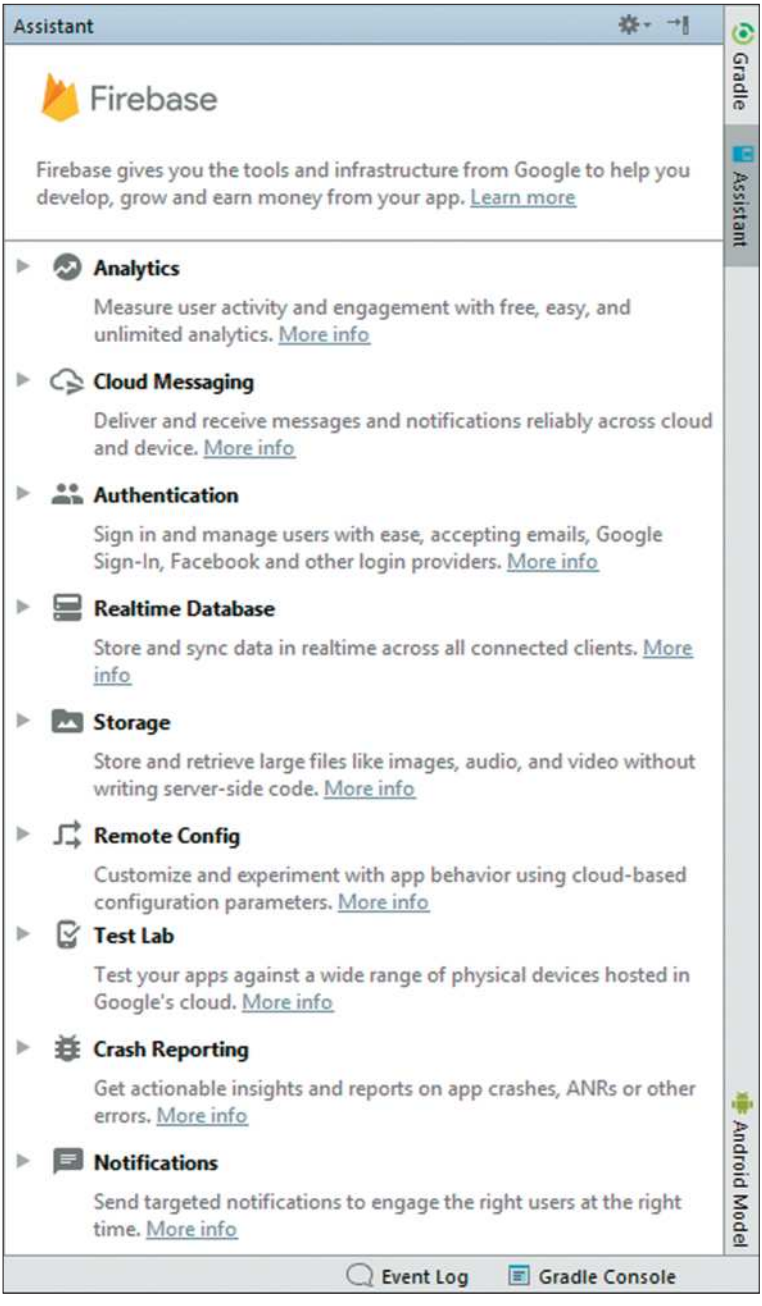


Изначально Firebase был реализован как облачная СУБД. Сейчас это популярный, доступный и очень востребованный сервис — мобильная платформа, которая при интеграции с приложением дает разработчику больше возможностей, чем просто хранение данных в синхронизируемой в реальном времени базе данных. С помощью Firebase можно подключать приложение к системам аналитики (и собирать различные данные об активности пользователей), организовать обмен сообщениями между пользователями, авторизацию в приложении (в том числе через сервисы Google, Facebook и так далее), рассылку уведомлений и многие другие функции.

Работа с Firebase осуществляется через **Консоль Firebase**, доступную по ссылке **https://firebase.google.com/**

В **Android Studio** Firebase вызывается через меню **Tools** → → **Firebase**. Все указания по работе с сервисом отображаются в окне **Assistant**.





В этом уроке с помощью Firebase мы реализуем авторизацию в приложении, а в финальном проекте создадим собственный мессенджер.

6.4. Приложение «Посторонним вход воспрещен»

Мы уже вполне можем сами реализовать авторизацию в приложении (построить интерфейс, добавить Activity и осуществить переход между ними, написать необходимые проверки) или при создании проекта выбрать шаблон **Login Activity**.

Но в первом случае нам придется писать слишком много кода самостоятельно, а во втором авторизация осуществляется только с помощью аккаунта Google, что не всегда удобно и нужно разработчикам.

Поэтому рассмотрим реализацию авторизации пользователей в приложении на примере инструмента **Firebase Authentication**.

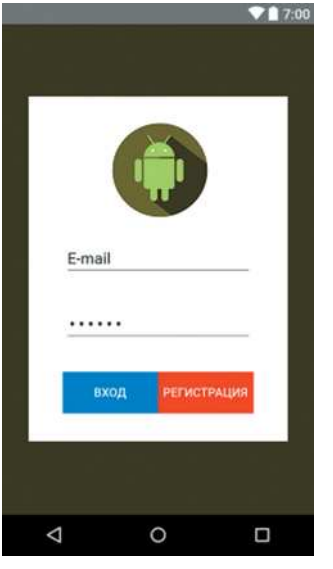
Концепция приложения: форма авторизации. Пользователь вводит e-mail и пароль, выбирает регистрацию/авторизацию, а приложение «сообщает» ему об успешном выполнении (или невозможности выполнения) данного действия.

Шаг 1. Создать проект.

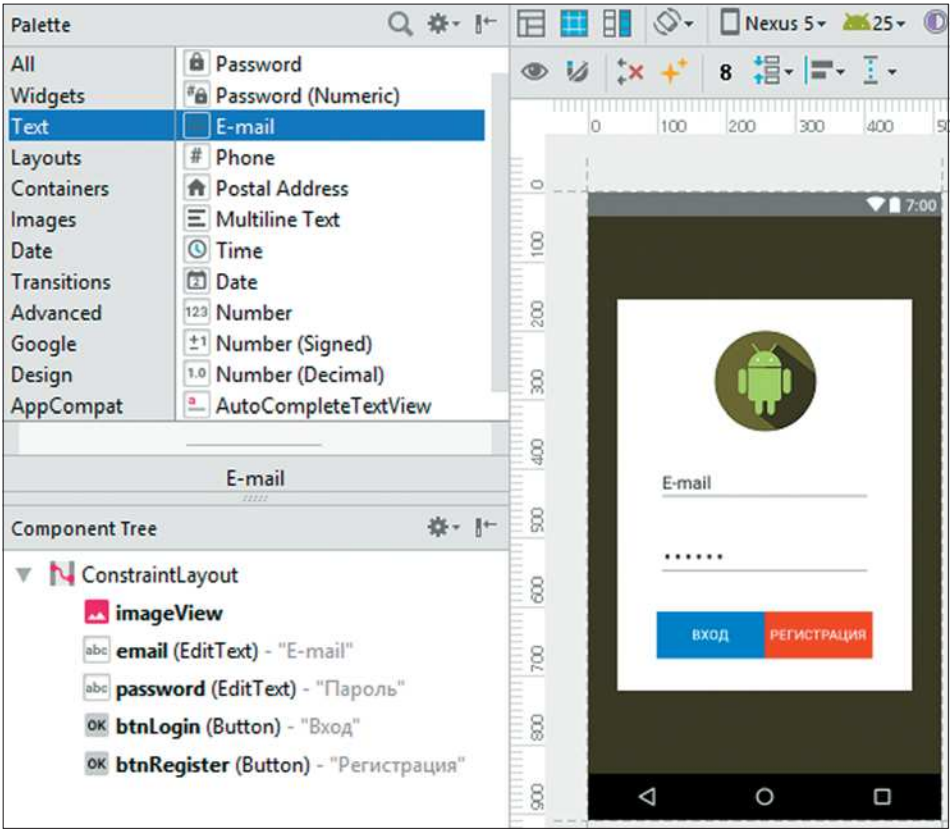
Для начала создадим проект Auth с пустой Activity. Как мы уже договорились, использовать шаблон **Login Activity** мы не станем, создадим собственную форму авторизации.

Шаг 2. Реализовать xml-разметку формы авторизации.

Начнем, как и всегда, с разметки интерфейса. Формы авторизации не отличаются большим разнообразием, их обязательные атрибуты — два поля (для ввода логина и пароля) и две кнопки (для авторизации и регистрации нового пользователя):



Для ввода логина (e-mail) и пароля в палитре элементов (группа **Text**) есть уже готовые шаблоны — **E-mail**, **Password**, **Password (Numeric)** и так далее:

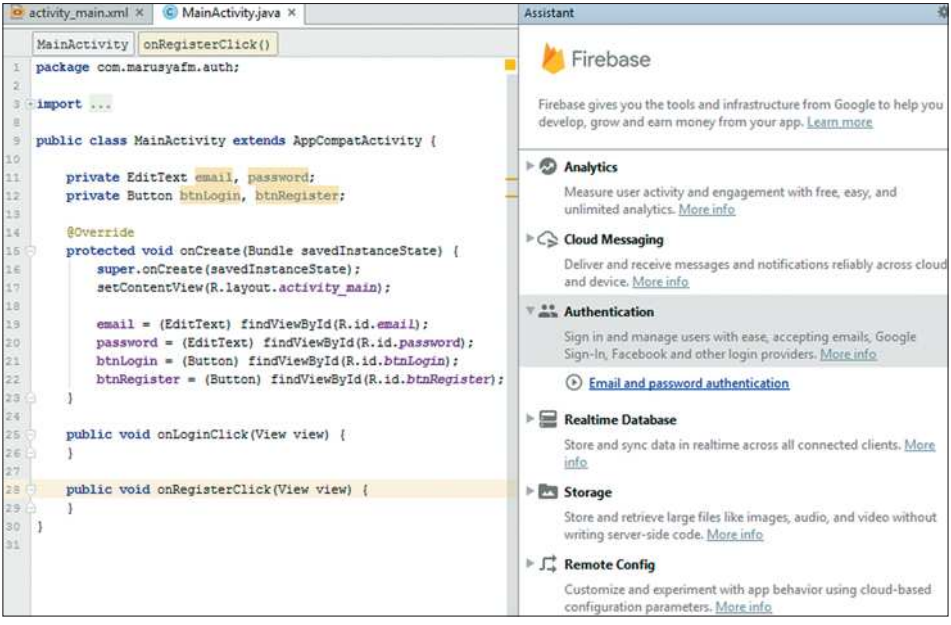


Шаг 3. Объявить и определить элементы интерфейса в главном Java-классе **MainActivity**, создать методы — обработчики событий нажатия на кнопки.

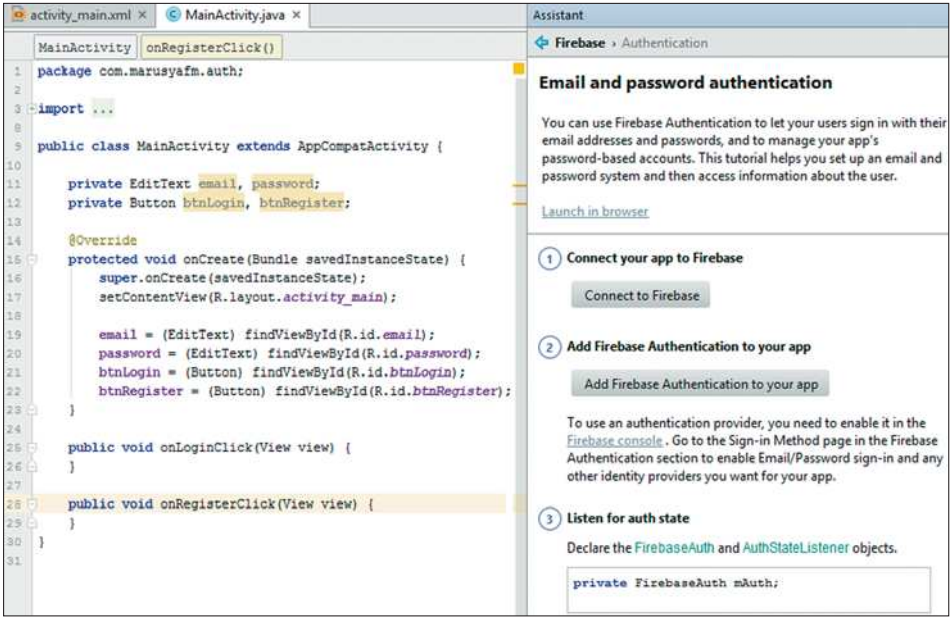
Элементы, не используемые в коде, объявлять и определять нет смысла, поэтому в расчет берем только поля для ввода и кнопки — **email**, **password**, **btnLogin**, **btnRegister**.

Шаг 4. Подключить проект к **Firebase**.

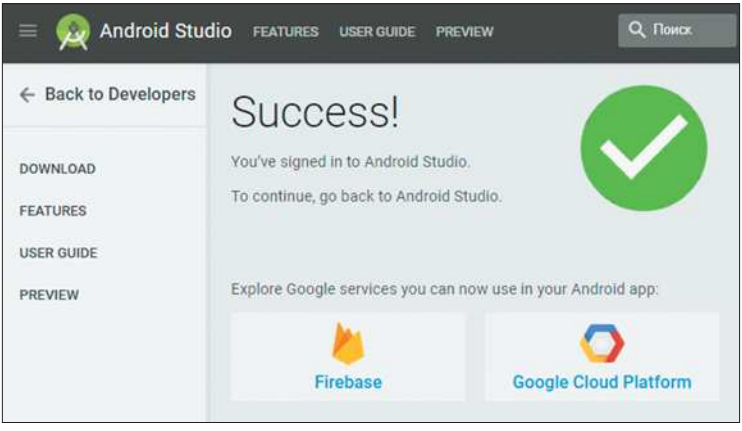
Откроем инструмент Firebase (**Tools** → **Firebase**) и в открывшемся окне **Assistant** выберем пункт **Authentication**:



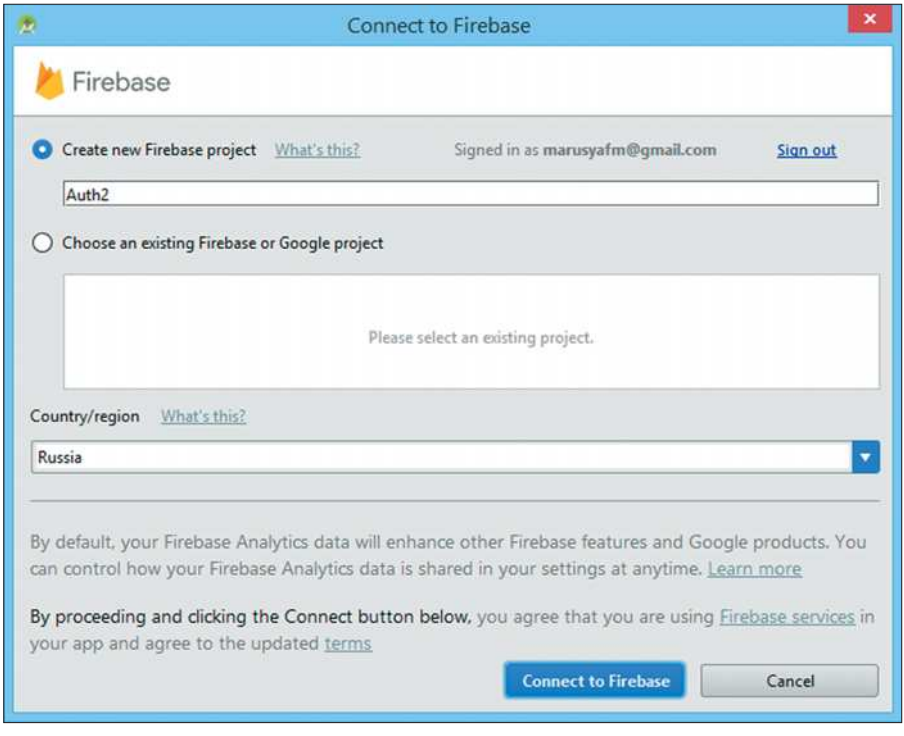
Перейдем по ссылке **E-mail and password authentication** (*утентифик ция по дресу электронной почты и п ролю*) — откроется вкладка **Authentication** (*утентифик ция*), на которой по шагам расписано, как настроить взаимодействие нашего приложения с Firebase для реализации аутентификации.



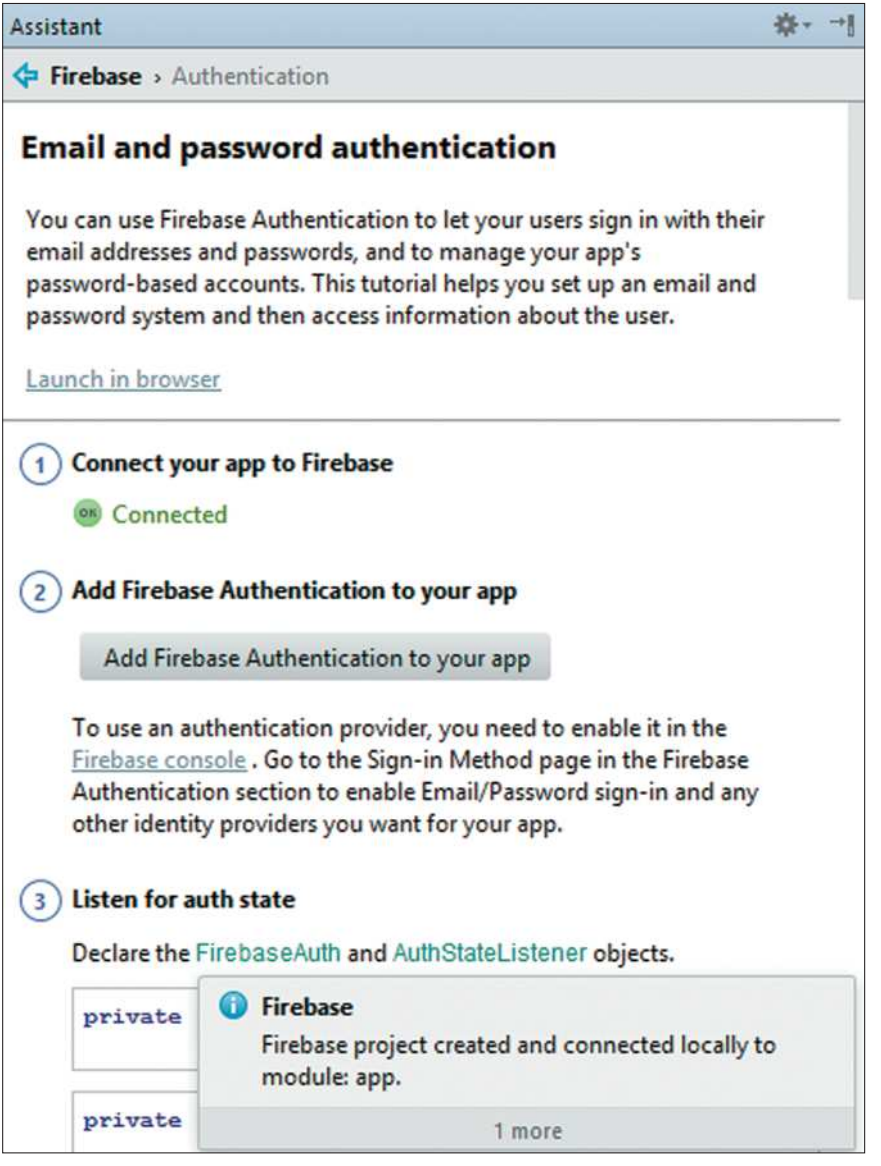
Первым действием ① нам предлагают «соединиться» с Firebase, для этого нажимаем кнопку **Connect to Firebase**. В открывшемся браузере нужно выбрать **аккаунт Google** для входа в Консоль Firebase. После завершения этого этапа на открывшейся странице выбираем Firebase и переходим в Консоль:



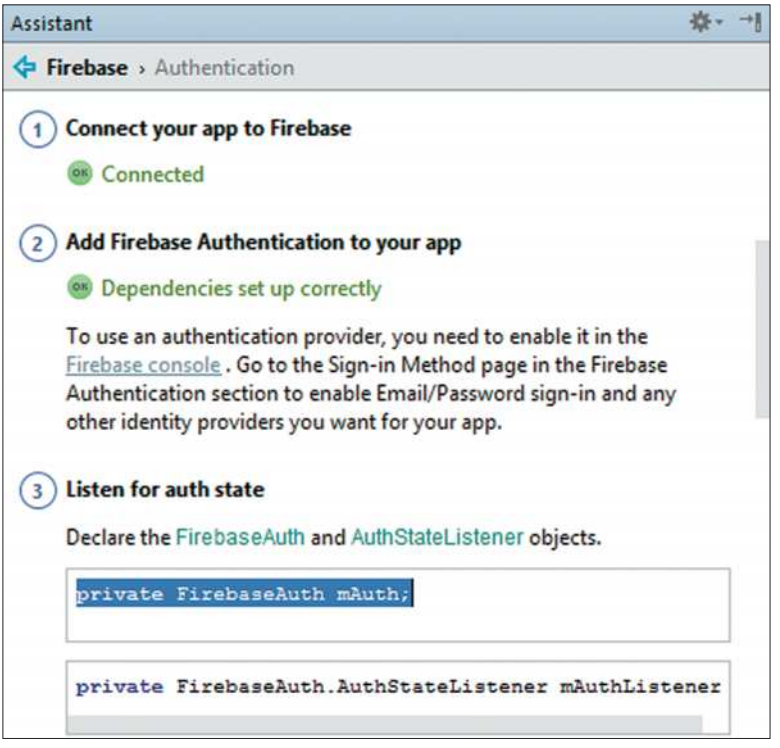
Здесь мы пока ничего делать не будем — возвращаемся в **Android Studio**. В открывшемся окне задаем параметры проекта и нажимаем кнопку **Connect to Firebase:**



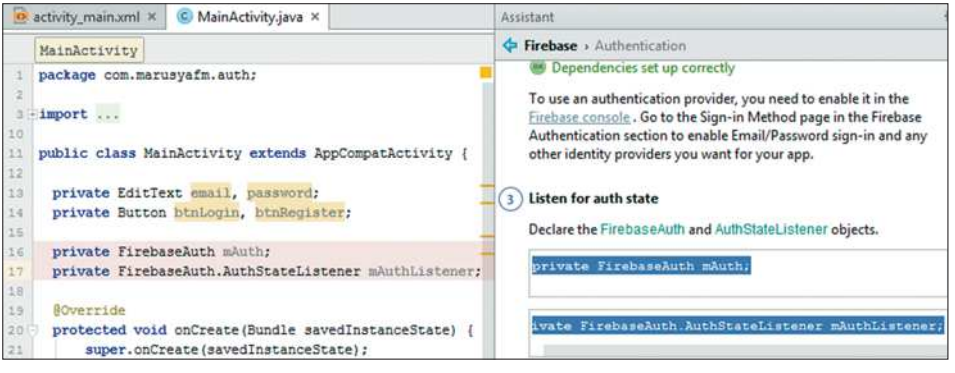
Следующее действие ② — добавить в наш проект Android Studio все необходимые зависимости и разрешения. Для этого требуется всего лишь нажать кнопку, расположенную в описании второго шага:



В открывшемся окне принимаем изменения (кнопка **Accept Changes**). После того как проект синхронизировался, можно переходить к третьему шагу в описании.

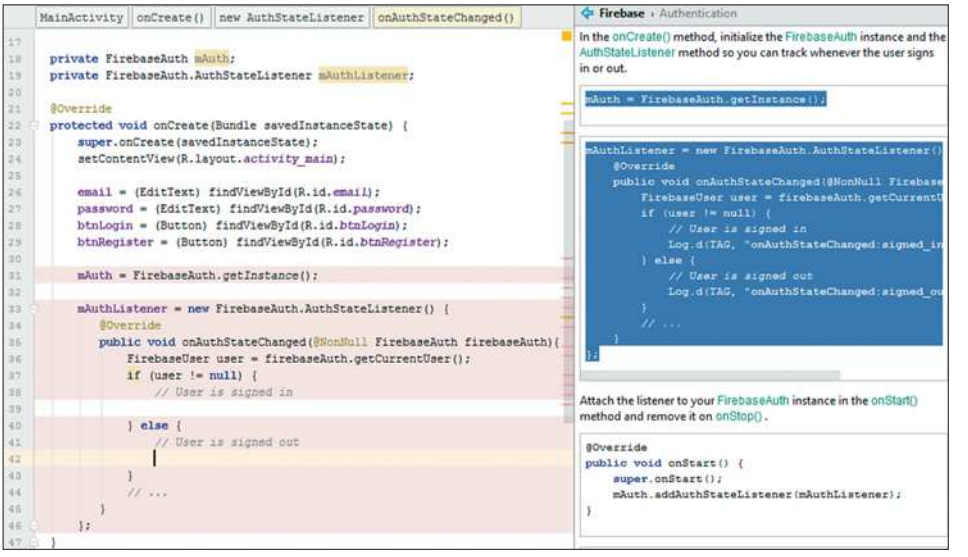


Далее нам предлагается объявить необходимые для работы переменные, и даже дан код, который нужно просто скопировать из ③ и вставить в класс **MainActivity** после уже объявленных нами ранее переменных.



Следующие два фрагмента кода для вставки предлагается разместить в методе **onCreate()**.

Не забудем периодически нажимать **Alt+Enter** для того, чтобы импортировать необходимые библиотеки. А строки вида Log.d(); удалим — логирование событий нам пока не понадобится.



Пояснение

Логирование — это ведение записи событий в специальном журнале (**log** — *журн л регистр ции*).

Какие записи вносить в этот журнал, определяет разработчик. Существует пять уровней логирования (соответственно пять методов) с разным назначением:

- **Log.e()** — логирование ошибок (**e**rror);
- **Log.w()** — логирование предупреждений (**w**arning);
- **Log.i()** — логирование общей информации (**i**nformation);
- **Log.d()** — логирование информации, полезной для отладки (**d**ebug);
- **Log.v()** — логирование подробностей (**v**erbose).

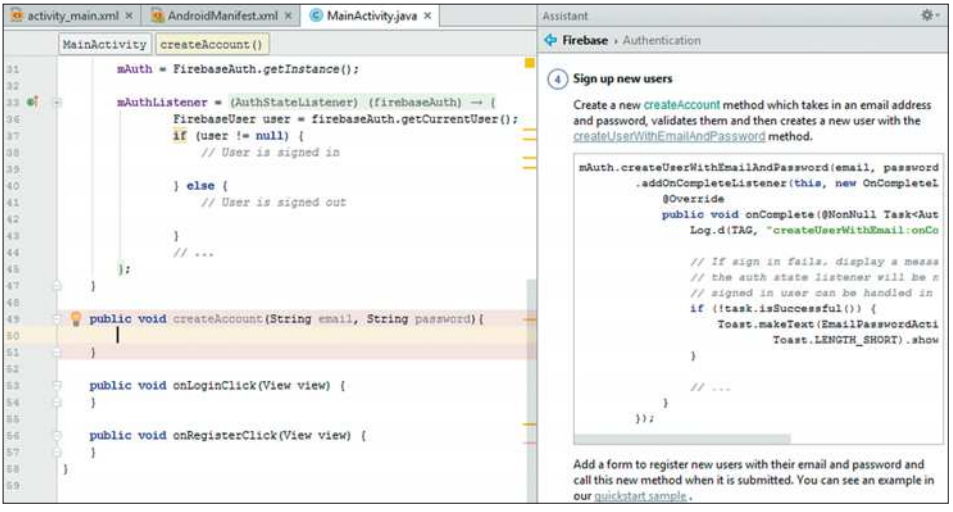
В **Android Studio** для отображения системных сообщений (и в частности журнала логов) есть специальное окно **Logcat**, в котором можно отследить все залогированные события.

Если добавить в программный код методы логирования и запустить приложение на выполнение, то потом в журнале логов можно прочитать буквально все, что происходило, и понять, что выполнено успешно, а где возникла ошибка. Например, «Стартовая страница приложения запущена», «Попытка авторизации», «Авторизация прошла успешно», «Обращение к БД», «Выгрузка данных из БД прошла успешно», «Попытка добавления записи в БД», «Ошибка сохранения записи в БД», «Завершение работы приложения».

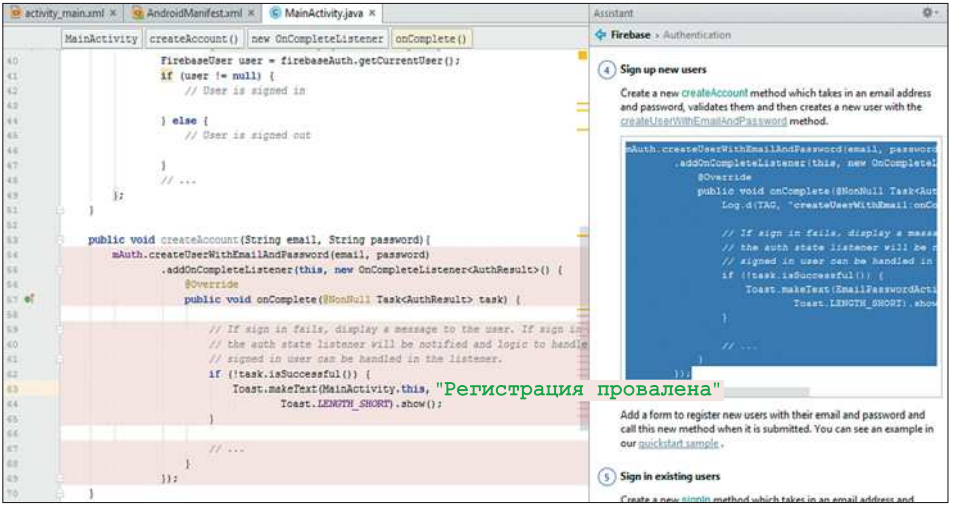
Код, предлагаемый для методов **onStart()** и **onStop()**, пока тоже опустим — на данном этапе мы обойдемся и без этих методов.

Шаг 5. Добавить методы регистрации и авторизации.

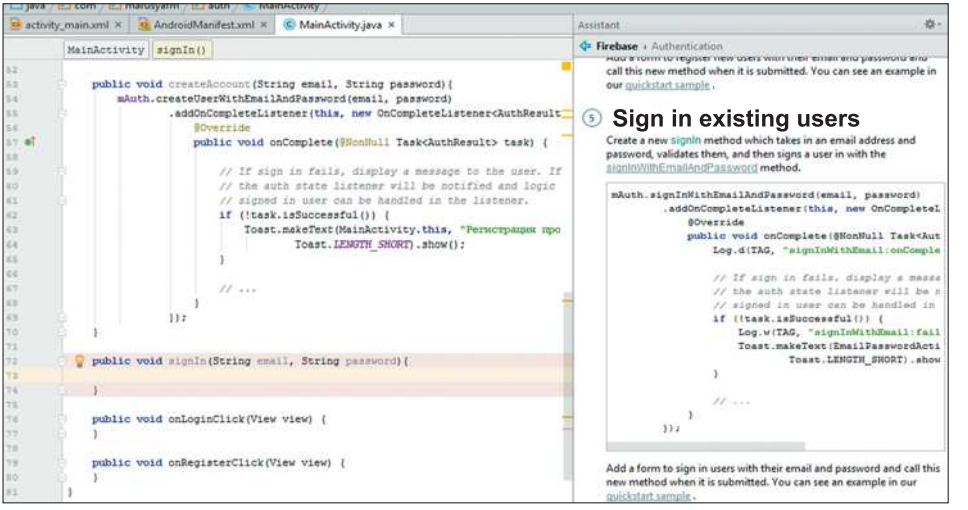
В действии ④ описания нам предлагают создать новый метод **createAccount()** для описания алгоритма регистрации нового пользователя в приложении (и соответственно создания его аккаунта):



На вход метода **createAccount** будут передаваться два параметра: e-mail и пароль. Код для данного метода также представлен. Снова удалим строки Log.d(); а текст всплывающего сообщения заменим на Регистрация провалена.



В действии ⑤, по аналогии, нужно создать еще один метод, теперь для авторизации уже зарегистрированных пользователей — **signIn()**.



Заполняется метод также предложенным кодом.

Приведем оба метода в соответствие — удалим все встречающиеся строки Log.d();, изменим текст всплывающего сообщения для неудавшейся авторизации на Авторизация провалена, в оба метода добавим условие **else** для успешной регистрации/авторизации и в них соответствующие всплывающие сообщения (строки № 62 и 74):

```
53  public void createAccount(String email, String password){
54      mAuth.createUserWithEmailAndPassword(email, password)
55          .addOnCompleteListener(this, new OnCompleteListener<AuthResult>() {
56              @Override
57              public void onComplete(@NonNull Task<AuthResult> task) {
58                  if (!task.isSuccessful()) {
59                      Toast.makeText(MainActivity.this, "Регистрация провалена",
60                              Toast.LENGTH_SHORT).show();
61                  } else
62                      Toast.makeText(MainActivity.this, "Регистрация прошла успешно",
63                              Toast.LENGTH_SHORT).show();
64              }
65          });
66  }
67
68  public void signIn(String email, String password){
69      mAuth.signInWithEmailAndPassword(email, password)
70          .addOnCompleteListener(this, new OnCompleteListener<AuthResult>() {
71              @Override
72              public void onComplete(@NonNull Task<AuthResult> task) {
```

```
                if (!task.isSuccessful()) {
                    Toast.makeText(MainActivity.this, "Авторизация провалена",
                            Toast.LENGTH_SHORT).show();
                } else
                    Toast.makeText(MainActivity.this, "Авторизация прошла успешно",
                            Toast.LENGTH_SHORT).show();
                }
            });
    }
```

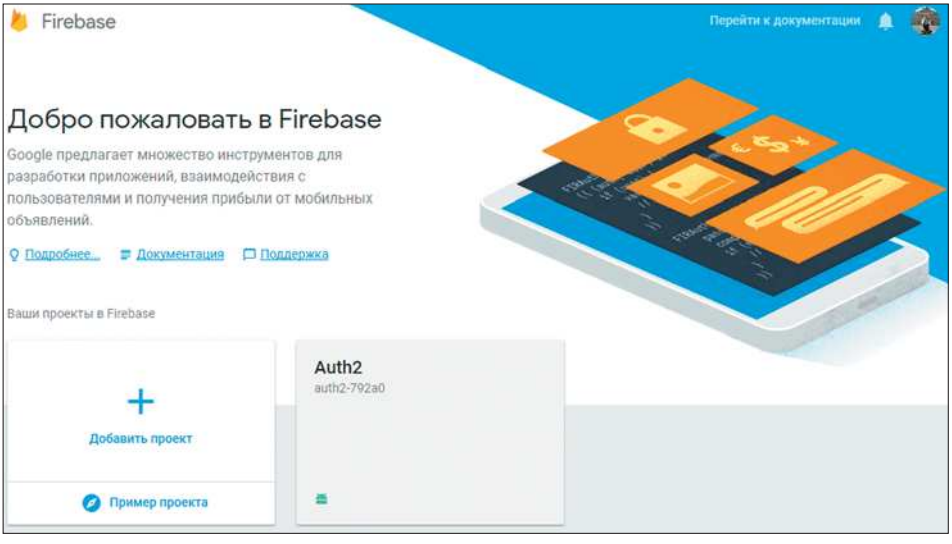
Шаг 6. Дописать методы — обработчики событий нажатия кнопок **Вход** и **Регистрация.**

Для кнопки **Регистрация** при ее нажатии обращаемся к методу **createAccount(),** передавая ему на вход содержимое заполненных пользователем полей **E-mail** и **Пароль.** Для кнопки **Вход** с теми же входными параметрами вызываем метод **signIn():**

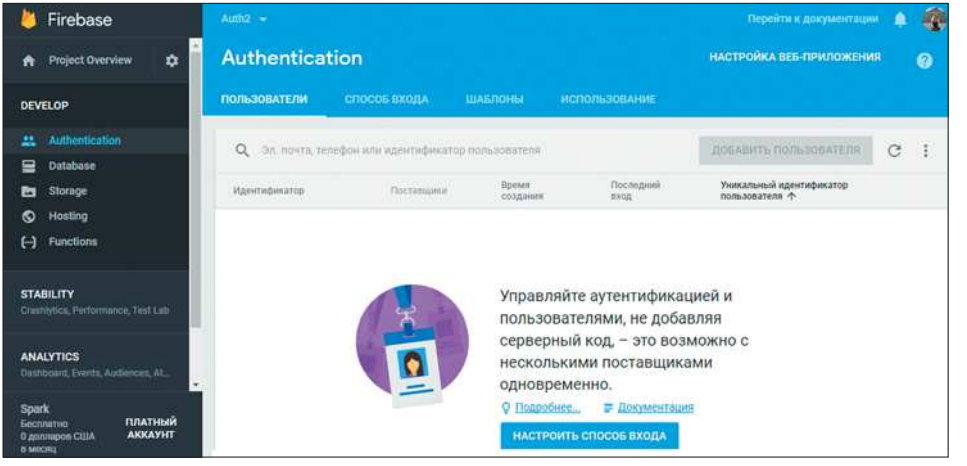
```
public void onRegisterClick(View view) {
    createAccount(email.getText().toString(), password.getText().toString());
}

public void onLoginClick(View view) {
    signIn(email.getText().toString(), password.getText().toString());
}
}
```

Шаг 7. Завершить настройку проекта в **Firebase.**

Снова откроем браузер, где уже произведен вход в **Консоль Firebase**, и выберем созданный нами в начале урока проект.

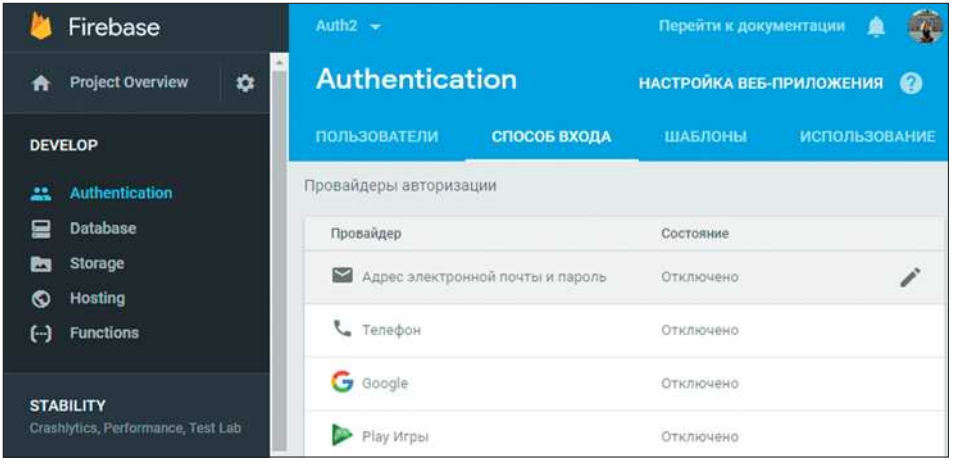


В левом боковом меню раскроем вкладку **DEVELOP**, а в ней пункт **Authentication:**



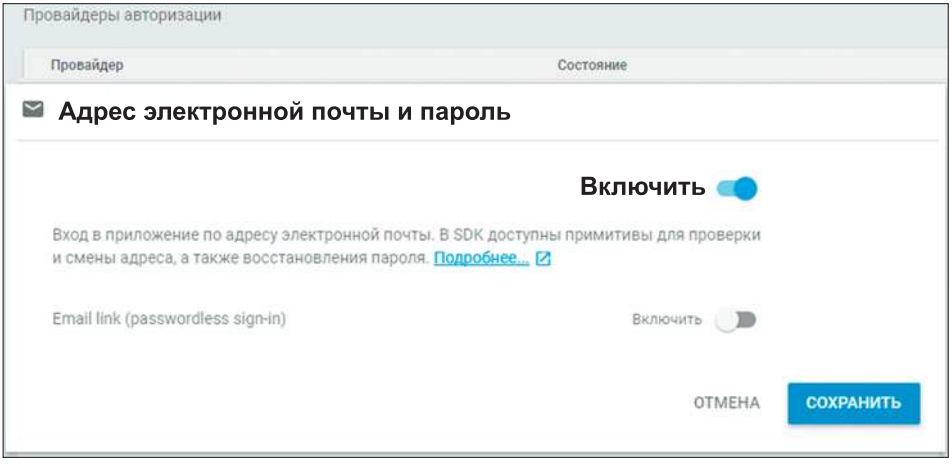
На вкладке **Пользователи** будут отображаться зарегистрированные в приложении пользователи. Их можно будет добавлять, удалять, изменять их данные, но пока это все недоступно (даже кнопка **Добавить пользователя** неактивна), потому что не настроен **способ входа.**

Чтобы это изменить, перейдем на соседнюю вкладку **Способ входа. Firebase** предоставляет нам возможность осуществлять аутентификацию пользователей через многие известные сервисы, например Google и Facebook, но сейчас нас интересует **вход по адресу электронной почты и паролю.** Выберем соответствующий пункт в списке и нажмем на «карандаш» рядом с ним.



В открывшемся окне активируем переключатель **Включить** для входа в приложение по адресу электронной почты и сохра-

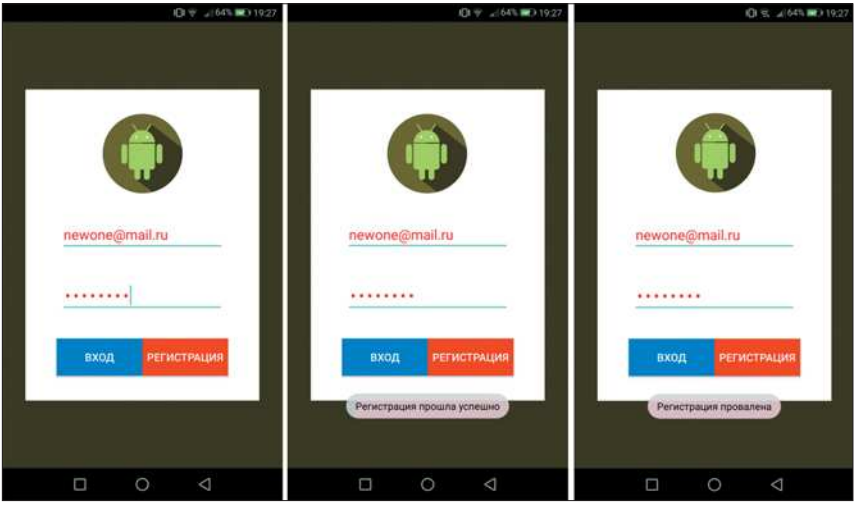
няем изменения. Включать вторую опцию (вход без пароля, по ссылке, направляемой на электронную почту) пока не будем — разберем сначала классический вариант аутентификации.



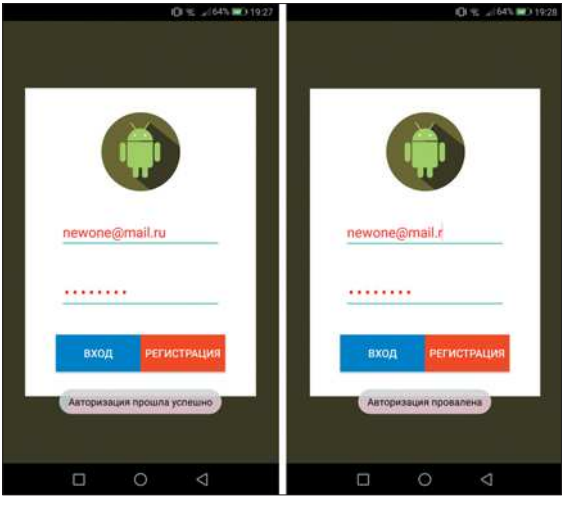
На этом дополнительные приготовления окончены.

Шаг 8. Вернуться в **Android Studio**, собрать и запустить проект.

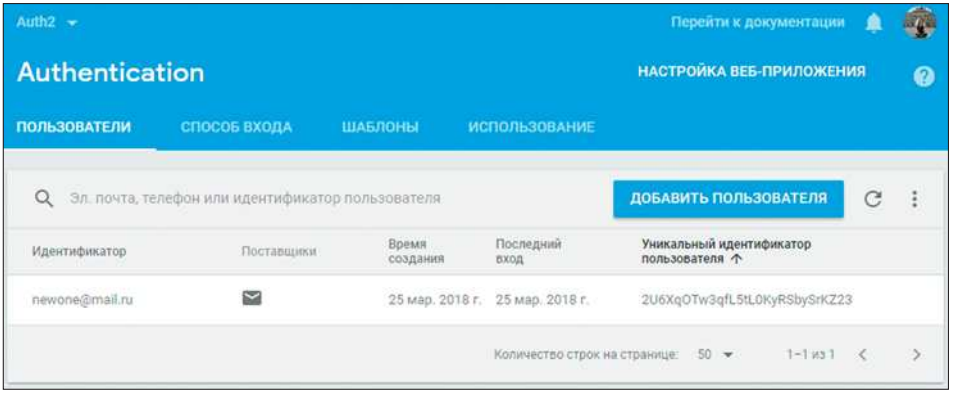
Теперь наше приложение работает так: если корректно заполнить поля **E-mail** и **Пароль** и нажать кнопку **Регистрация** — появится сообщение об успешной регистрации нового пользователя. Если же нажать кнопку снова, приложение оповестит нас о провале регистрации, ведь такой пользователь уже существует (вывести соответствующее сообщение мы можем, добавив дополнительную проверку).



Если ввести данные уже зарегистрированного пользователя и нажать кнопку **Вход**, появится сообщение об успешной авторизации. Если изменить в этих данных хотя бы один символ — получим провал авторизации:



Все изменения в списке пользователей тут же отображаются в **Консоли Firebase:**



Задания

1. При нажатии кнопки регистрации добавить поле подтверждения пароля и прописать в коде проверку совпадения введенных паролей. Для этого можно использовать знакомый нам метод **getText()**.
2. Добавить в приложение еще одну Activity, наполнить ее информацией и при успешной авторизации реализовать переход к ней.
3. При регистрации/авторизации добавить проверку подключения к Интернету.

Итоги главы 6

Навыки, полученные при прохождении уроков главы 6, помогут нам в дальнейшем разрабатывать действительно серьезные приложения — с большими объемами данных, работающие online и взаимодействующие со сторонними сервисами.

Мы научились:

1. Определять подключение к сети Интернет, уведомлять пользователя о его наличии (отсутствии) и в зависимости от этого выполнять различные действия.
2. Создавать базы данных, подключать их к приложению и отображать их содержимое в интерфейсе.
3. Подключать свой проект к сервису **Firebase** и реализовывать авторизацию в приложении с его помощью.

Глава 7. Сторонние приложения и встроенные инструменты

7.1. Работа со сторонними приложениями

Как мы уже убедились на примере камеры, разрабатывая мобильные приложения, совершенно необязательно разрабатывать самостоятельно все 100% их функций и каждый раз «изобретать велосипед». Наоборот, гораздо разумнее и экономнее (по времени) для выполнения некоторых действий использовать сторонние приложения, особенно стандартные и системные. В этом уроке мы рассмотрим обращения к некоторым из них.

7.1.1. Телефон

Иногда в приложениях указывают телефонные номера, но, чтобы на них звонить, их не нужно запоминать или копировать. Обычно разработчики все учитывают, и при нажатии на номер открывается стандартное приложение — телефон.

Стандартные приложения, такие как телефон, вызываются из других приложений с помощью намерения **Intent**, которое описывается в методе–обработчике события нажатия соответствующего элемента (например, кнопки):

```
Intent myIntent;
myIntent = new Intent(Intent.ACTION_DIAL, Uri.parse
(“tel:+7(926)888-88-88”));
startActivity(myIntent);
```

Тип приложения, которое должно быть открыто, выявляется из формата написания значения параметра **Uri** методом **parse()** (*н лизиров ть, р спозн ть*). Для вызова телефона параметр **Uri** записывается в формате tel:XXX, где вместо XXX прописывается набор символов, который будет автоматически набран при открытии телефона.

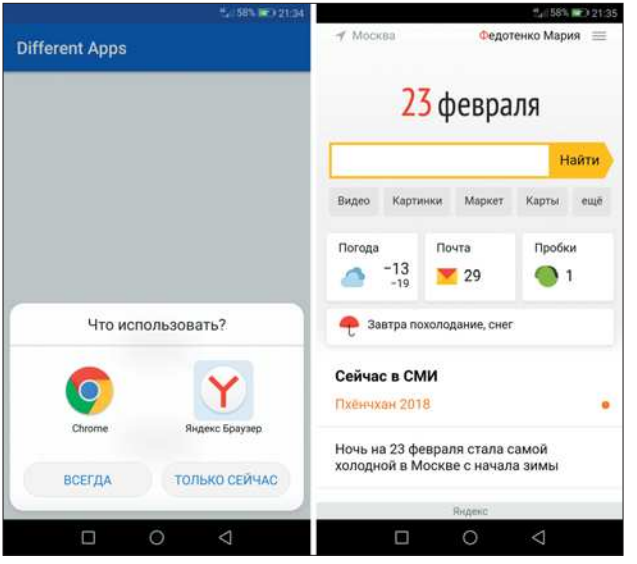
Пользователю останется только нажать кнопку вызова.

7.1.2. Браузер

Если нужно открыть некоторую ссылку, браузер также вызывается через намерение. Ссылка прописывается в формате https://XXX:

```
Intent myIntent;
myIntent = new Intent(Intent.ACTION_VIEW,
Uri.parse("https://yandex.ru"));
startActivity(myIntent);
```

В этом случае приложение будет искать сторонние приложения, способные открыть ссылку, и, если их будет найдено несколько, предложит выбор:



7.1.3. Электронная почта

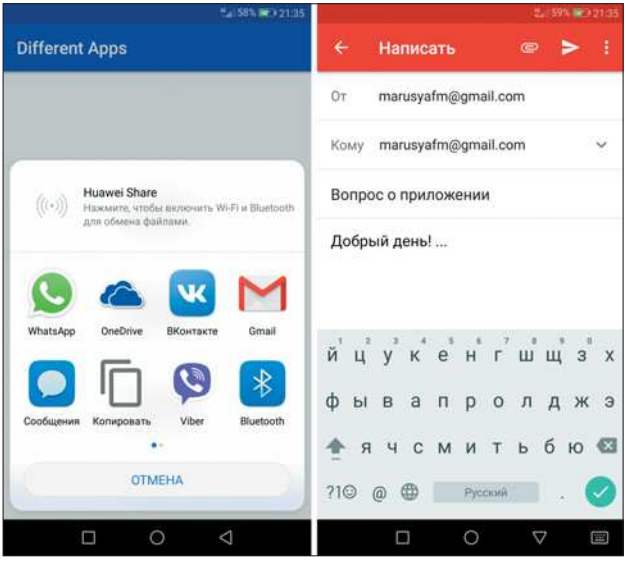
Некоторые приложения используют электронную почту, например для обратной связи с разработчиком или в интернет-магазине для уточнения деталей заказа.

В этом случае намерению задают параметр ACTION_SEND (*действие: отправить*), а в качестве дополнительных параметров можно задать e-mail получателя, тему письма, шаблон текста письма и так далее:

```
Intent myIntent;
myIntent = new Intent(Intent.ACTION_SEND);
myIntent.setType(«text/*»);
myIntent.putExtra(Intent.EXTRA_EMAIL, «marusyafm@gmail.com»);
```

```
myIntent.putExtra(Intent.EXTRA_SUBJECT, «Вопрос о приложении»);
myIntent.putExtra(Intent.EXTRA_TEXT, «Добрый день! ...»);
startActivity(Intent.createChooser (myIntent, «Написать разработчику»));
```

Пользователю будет предложен выбор почтового клиента или мессенджера, который откроется уже с заполненными параметрами письма.



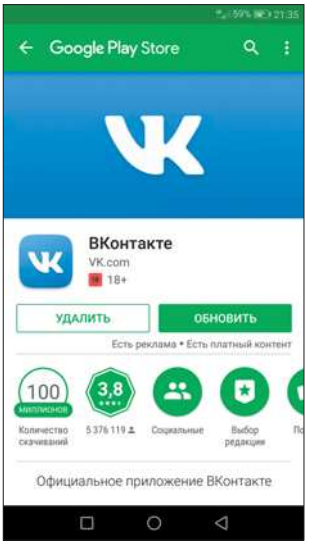
7.1.4. Магазин приложений Google Play

В приложении можно рекомендовать другие приложения или, например, попросить нашего пользователя оставить отзыв. Для этого нужно вызвать магазин приложений Google Play:

```
Intent myIntent;
myIntent = new Intent(Intent.ACTION_VIEW,
Uri.parse(“market://details?id=com.vkontakte.android”));
startActivity(myIntent);
```

Существует два способа записи ссылки для перехода в магазин приложений:

1) market://details?id=XXX — открывает заданное приложение в приложении Google Play;
2) http://play.google.com/store/apps/details?id=XXX — предоставляет пользователю выбор — открыть в приложении Google Play или в браузере.



7.1.5. Фонарик

Большинство устройств может работать в качестве фонарика. Причем в современных смартфонах и планшетах фонарик — это не специально встроенный для этих целей светодиод или лампочка. Для того чтобы «включить свет», нужна... камера! Приложения-фонарики обращаются к камере смартфона, но не открывают ее, а только запускают одну из ее функций — вспышку.

Значит, для превращения телефона в фонарик нужно получить доступ к камере устройства. Для этого в файл манифеста прописываются разрешения на использование камеры, ее стандартных функций и вспышки.

```
<uses-permission android:name="android.permission.CAMERA"/>
<uses-permission android:name="android.permission.FLASHLIGHT"/>
<uses-feature android:name="android.hardware.camera"/>
<uses-feature android:name="android.hardware.camera.flash"/>
```

С точки зрения интерфейса нам нужна только одна кнопка на главном экране, например элемент **ImageButton**. Установим для нее идентификатор btnFlashlight и обработчик события нажатия onClick().

В **MainActivity.java** объявим переменную btnFlashlight для нашей кнопки. Нам понадобится еще несколько переменных:

- для использования камеры private Camera myCamera;
- для установки параметров камеры private Camera.Parameters myParameters;
- для определения наличия вспышки у камеры устройства (true — имеет, false — не имеет) private boolean hasFlash;

В методе **onCreate()** определяем кнопку, а также выясняем, есть ли вспышка у устройства, на котором запущено приложение. Если есть, открываем камеру методом **open()** (*открыть*) и запрашиваем ее параметры методом **getParameters()** (*получить параметры*). Если нет, можно вывести всплывающее сообщение об отсутствии вспышки:

```
hasFlash = getApplicationContext().getPackageManager()
        .hasSystemFeature(PackageManager.FEATURE_CAMERA_FLASH);
if (hasFlash) {

    myCamera = Camera.open();
    myParameters = myCamera.getParameters();
} else {
    Toast myToast = Toast.makeText(getApplicationContext(),
            "У камеры нет вспышки", Toast.LENGTH_SHORT);
    myToast.show();
}
```

В методе нажатия кнопки **onClick()** мы снова получаем параметры камеры, устанавливаем состояние вспышки FLASH_MODE_TORCH (*режим вспышки — фонарик*, то есть включена), применяем заданные нами параметры к камере и запускаем ее методом **startPreview()** (*начать предпросмотр*):

```
myParameters = myCamera.getParameters();
myParameters.setFlashMode(Camera.Parameters.FLASH_MODE_TORCH);
myCamera.setParameters(myParameters);
myCamera.startPreview();
```

Теперь можно запустить приложение. Да будет свет!

Задания

1. Добавить возможность выключения фонарика. Сейчас наш фонарик включается при нажатии на кнопку, а выключается только при выходе из приложения (на некоторых устройствах не выключается вовсе). Это явная недоработка, исправим ее, внеся в Java-класс **MainActivity** ряд дополнений и изменений. Объявим новую переменную:

   ```
   private boolean flashlightIsOn;
   ```

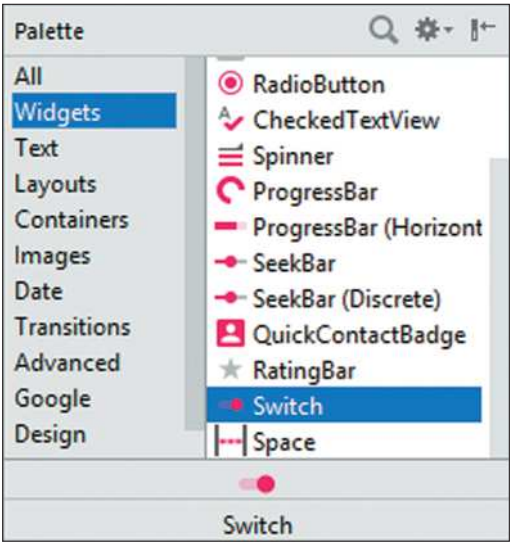
   которая понадобится для установки статуса вспышки (true — включена, false — выключена).
   В методе **onClick()** запуск камеры и вспышки помещаем в конструкцию if-else. Если вспышка выключена (в нашем алгоритме «не включена»), будет выполняться код, который мы написали раньше для включения, а иначе для вспышки устанавливается значение «выключена» и работа камеры приостанавливается.

```
39  public void onClick(View view) {
40      if (!flashlightIsOn) {
41          myParameters = myCamera.getParameters();
42          myParameters.setFlashMode(Camera.Parameters.FLASH_MODE_TORCH);
43          myCamera.setParameters(myParameters);
44          myCamera.startPreview();
45          flashlightIsOn = true;
46      } else {
47          myParameters.setFlashMode(Camera.Parameters.FLASH_MODE_OFF);
48          myCamera.setParameters(myParameters);
49          myCamera.stopPreview();
50          flashlightIsOn = false;
51      }
52  }
```

   Теперь при нажатии на кнопку фонарик включится, а при повторном нажатии выключится и будет снова включаться-выключаться по нашей команде.
2. Добавить всплывающие сообщения Фонарик включен и Фонарик выключен при включении и выключении соответственно.
3. Заменить изображение-кнопку на другой элемент управления — **переключатель**.

7.1.6. Switch — переключатель

В палитре элементов Android Studio есть элемент **Switch** (*переключ тель*):



Этот элемент имеет два положения: включено и выключено. Пользователь может перетаскивать ползунок переключателя или просто нажимать на него для перевода в другое состояние.

В xml-разметке основными атрибутами переключателя являются:

- **android:checked** — статус переключателя. Принимает значения true (включен) и false (выключен);
- **android:textOff** — текст в выключенном состоянии;
- **android:textOn** — текст во включенном состоянии;
- **android:text** — общий текст, отображающийся для «всего переключателя» возле него, вне зависимости от состояния.

В Java-классе переключатель объявляется и определяется как элемент типа **Switch** (*переключ тель*). Текст меняется методами **setText()** (*уст новить общий текст*), **setTextOn()** (*уст новить текст для включенного состояния*) и **setTextOff()** (*уст новить текст для выключенного состояния*). Статус переключателя определяется методом **isChecked()**.

Для выполнения задания нужно перераспределить уже написанный нами код для управления фонариком, чтобы им можно было управлять с помощью переключателя.

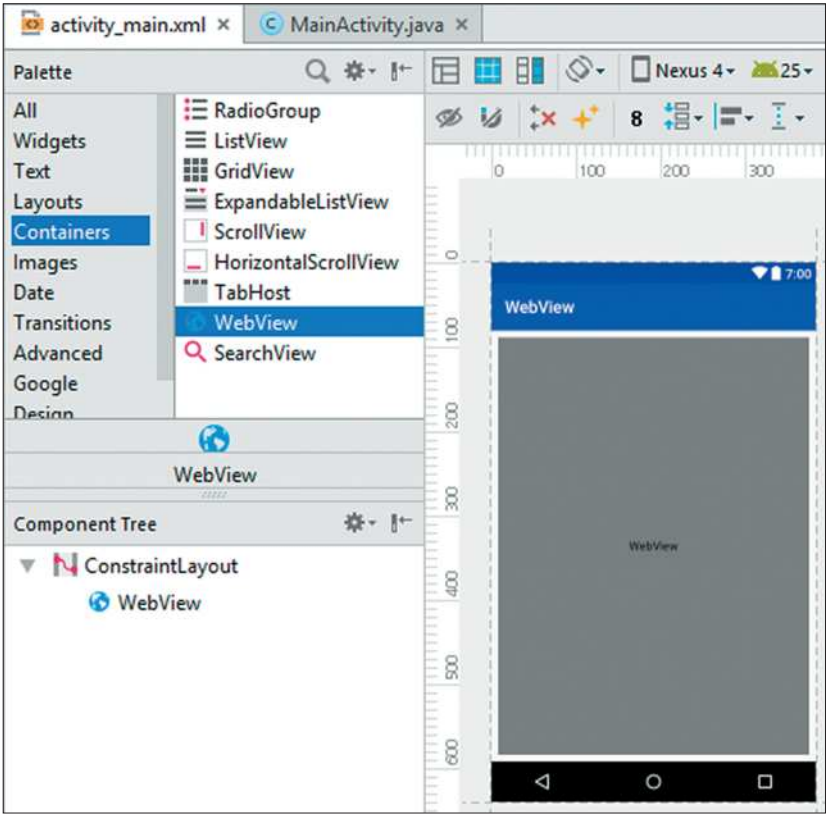
7.2. Конвертация сайта в мобильное приложение

Для того чтобы прочитать статью, ссылка на которую помещена в приложении, или получить информацию с сайта компании, для которой разработано приложение, уже необязательно открывать браузер — мобильные приложения могут выполнить это «своими силами». В **Android Studio** для этого предназначен элемент **WebView**.

WebView (*элемент для отобр жения веб-содержимого*) — это элемент-контейнер, с помощью которого можно встраивать в мобильные приложения веб-страницы или их части. Он позволяет переходить по ссылкам, не обращаясь к браузеру, или даже создать собственный браузер.

Разберемся, как это можно использовать. Создадим проект WebView с параметрами по умолчанию и одной Empty activity.

В **activity_main.xml** в режиме дизайна перетащим на главный экран из палитры элементов (группа **Containers**) элемент **WebView**:



Зададим ему идентификатор webView и «впишем» его в родительский макет — установим длину и ширину match_parent, чтобы все края совпадали с краями родительского элемента:

```
<WebView
    android:id="@+id/webView"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

Теперь, поскольку наше приложение собирается выходить в Интернет для загрузки веб-страницы, нужно прописать ему соответствующее разрешение.

Открываем файл **AndroidManifest.xml** и добавляем в него разрешение на использование Интернета android.permission.INTERNET (строка № 4):

activity_main.xml × AndroidManifest.xml ×

manifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.marusyafm.webview">
    <uses-permission android:name="android.permission.INTERNET"></uses-permission>

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="WebView"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
```

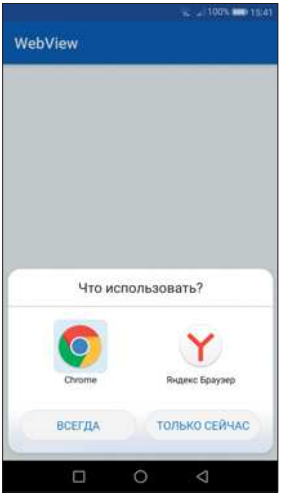
Осталось написать Java-код для наполнения **WebView** и отображения его содержимого на экране. Переходим в файл **MainActivity.java**.

Объявляем и определяем наш элемент **WebView**. С помощью метода **loadUrl()** (*з грузить url стр ницы*, то есть *ссылку н нее*) задаем адрес страницы, которую хотим загрузить, например **http://www.yandex.ru**:

```
public class MainActivity extends AppCompatActivity {

    private WebView myWebView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```
15          super.onCreate(savedInstanceState);
16          setContentView(R.layout.activity_main);
17
18          myWebView = (WebView)findViewById(R.id.webView);
19          myWebView.loadUrl("http://www.yandex.ru");
20
21      }
22  }
```

Запускаем приложение и видим, что оно пытается открыть ссылку, но для этого запрашивает браузер:



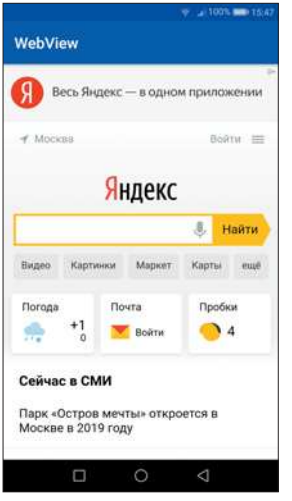
Но это не то, что нам нужно, — мы хотим открывать страницу прямо в приложении. Для этого добавим еще несколько строк в метод **onCreate()** (строки № 20–23):

```
11      private WebView myWebView;
12
13      @Override
14      protected void onCreate(Bundle savedInstanceState) {
15          super.onCreate(savedInstanceState);
16          setContentView(R.layout.activity_main);
17
18          myWebView = (WebView)findViewById(R.id.webView);
19          myWebView.loadUrl("http://www.yandex.ru");
20          myWebView.setWebViewClient(new WebViewClient());
21
22          WebSettings myWebSettings = myWebView.getSettings();
23          myWebSettings.setJavaScriptEnabled(true);
24      }
25  }
```

Метод **setWebViewClient()** (*уст новить веб-клиент*) позволит приложению самостоятельно загрузить выбранную ссылку, не обращаясь для этого к браузеру.

А строки № 22 и 23 «включают распознавание» языка JavaScript, то есть делают его понятным для нашего приложения, которое по умолчанию ориентируется на Java. Таким образом, после «включения» JavaScript нам будут доступны все действия с веб-страницами, как и в обычном браузере.

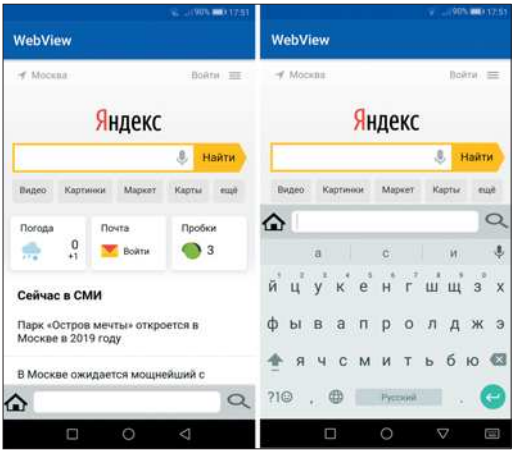
Проверяем. Веб-страница открывается непосредственно в нашем приложении и функционирует в «нормальном» режиме: мы можем нажать на любую кнопку, перейти по любой ссылке и ввести любой запрос в поисковую строку — результаты отобразятся в нашем приложении:



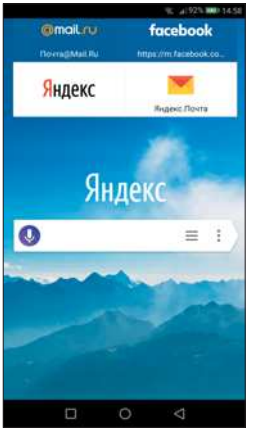
Задания

1. Добавить в приложение область поиска.
   Сейчас мы отобразили всего одну страницу в **WebView** и убедились, что она полноценно работает в качестве «поисковика», то есть мы создали что-то, отдаленно напоминающее браузер. И теперь ничто не мешает нам создать **собственный браузер!**
   В первую очередь нашему приложению не хватает **строки поиска** (сейчас в нем есть область поиска только потому, что в качестве стартовой страницы выбрана поисковая система, а не обычный сайт).
   Ее можно реализовать, добавив на главный экран горизонтальный **LinearLayout**, разместив в нем следующие элементы:

1) **Кнопку «Домой».** Добавить для нее обработчик события нажатия и прописать в нем доступ к стартовой странице (по аналогии с тем, что мы ранее написали в методе **onCreate()**).
2) **Текстовое поле EditText для ввода поискового запроса.**
3) **Кнопку «Поиск».** Ее обработчик события нажатия пишется по аналогии с методом **onCreate()**. Однако в методе **loadUrl()** задается уже не конкретная ссылка, а значение, полученное из поля **EditText**, например, с помощью метода **getText()**.



Теперь нашему приложению не требуется сайт-поисковик в качестве стартовой страницы — можно установить любой сайт или разработать свою стартовую страницу, как это сделано в большинстве браузеров.



2. Обеспечить корректную навигацию.
Сейчас поиск и навигация по страницам работают, но при нажатии кнопки **Назад** приложение закрывается, тогда как в браузере положено возвращаться на предыдущую страницу.

Для исправления ситуации добавим в класс **MainActivity** новый метод **onBackPressed()** (*по н ж тию н кнопку* — имеется в виду системная кнопка **Назад**). В этом методе пропишем, что если в **WebView** заложена возможность «пойти назад» (то есть мы находимся не на начальной странице), то следует «идти назад» (перейти к предыдущей отображенной странице). В противном случае выполнять действие, предусмотренное для кнопки по умолчанию (стандартно это закрытие приложения):

```
26      @Override
27      public void onBackPressed() {
28          if(myWebView.canGoBack()) {
29              myWebView.goBack();
30          } else {
31              super.onBackPressed();
32          }
33      }
```

3. Проанализировать несколько браузеров для Android и по их образу усовершенствовать дизайн и расширить функционал нашего браузера.

7.3. Чтение QR-кодов

Штрих-код (*штриховой код*, метка) наносится на продукты, товары или их упаковку и предназначен для считывания различными специальными устройствами (например, на кассе в магазине). Обычно представляет собой последовательность черных и белых вертикальных полос и цифр.

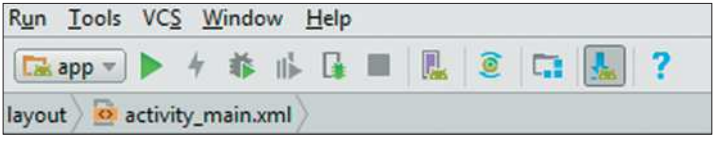
QR-код (Quick Response Code — *код быстрого ре гиров ния*) — двумерный (матричный) штрих-код, считав который с помощью специальных мобильных приложений, можно получить быстрый доступ к более подробной информации об объекте, характеристикам товара или контактам человека. QR-коды обычно содержат ссылки на сайты, визитные карточки или просто текстовые сообщения.

Раньше мы встречали различные штрих-коды в основном на продуктах в магазине, знали, что в них содержится информация о производителе и самом продукте, но ничего с этой информацией не делали. Сейчас штрих-коды и особенно QR-коды можно встретить практически везде — в музеях, магазинах, на сайтах, в рекламе на улице.

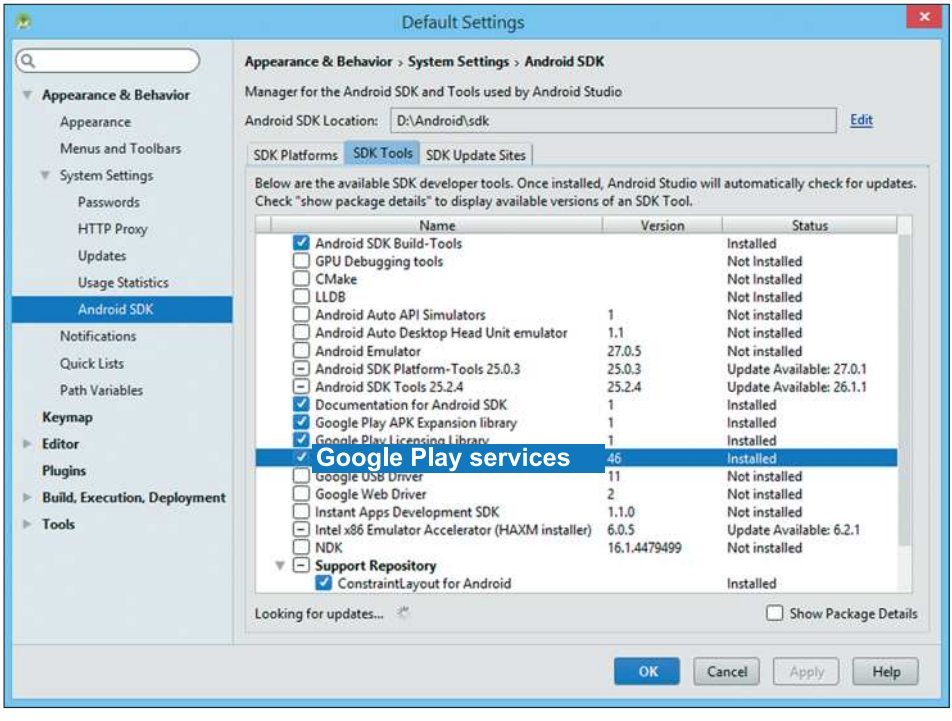
Для того чтобы сгенерировать QR-код, создано множество сайтов и приложений. Но на практике гораздо чаще приходится считывать коды, поэтому создадим приложение именно для их распознавания, то есть **сканирования QR-кодов.**

Назовем Android Studio проект QR Code Scanner. В качестве **Minimum SDK** (на втором шаге) выберем **API** не меньше 21: Android 5.0 (Lollipop), чтобы иметь возможность подключить и использовать все необходимые библиотеки. Остальные параметры оставим по умолчанию, в том числе выбор Empty activity.

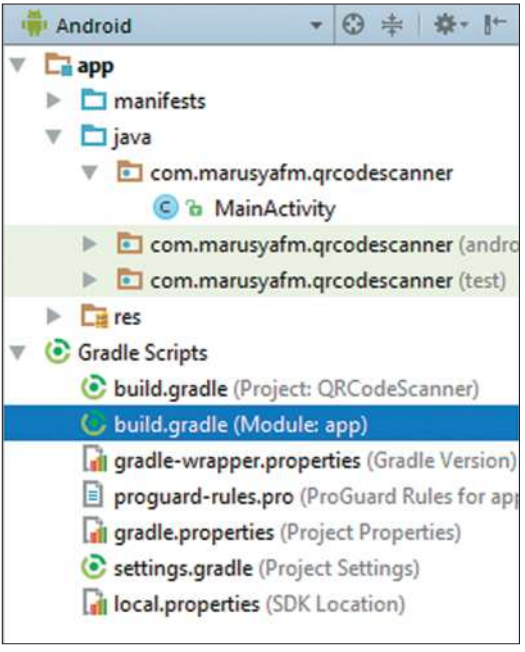
Для работы с QR-кодами нам понадобится доступ к **Google Play services** (*сервис м Google Play*) и библиотеке **Zxing Library**. С помощью панели быстрого доступа откроем **Android SDK Manager** нажатием на соответствующий значок:



В открывшемся окне перейдем на вкладку **SDK Tools** и убедимся, что **Google Play services** установлены и имеют версию не ниже 26. Если нет — выберем соответствующий пункт в списке и установим обновление (кнопка **Apply**):



Для подключения библиотеки **ZXing**[1] **Library**, которая нужна для считывания QR- и штрих-кодов, откроем файл **build.gradle** в разделе **Gradle Scripts**. В этом разделе два файла с таким именем, но нас интересует второй — с пометкой **(Module: app)**.



В открывшемся файле найдем раздел **dependencies** (*зависимости*), добавим в него строку об использовании соответствующей встроенной Android-библиотеки (строка № 30) и синхронизируем проект нажатием на ссылку **Sync Now** (*синхронизировать сейчас*):

[1] **ZXing** — сокращение от **Zebra Crossing** — пешеходный переход типа «зебра» с чередованием полос, как и на штрих-кодах.

```
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
        exclude group: 'com.android.support', module: 'support-annotations'
    })
    compile 'com.android.support:appcompat-v7:25.1.0'
    compile 'com.android.support.constraint:constraint-layout:1.0.2'
    testCompile 'junit:junit:4.12'
    compile 'com.journeyapps:zxing-android-embedded:3.4.0'
}
```

Теперь можно вернуться к дизайну и коду нашего приложения. В файле **activity_main.xml** размещаем по центру экрана кнопку **Сканировать QR-код** и задаем ее свойства, а в классе **MainActivity** создаем обработчик ее нажатия — метод **onClick()**:

activity_main.xml × MainActivity.java ×

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="h
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.marusyafm.qrcodescanner.MainActivity">

    <Button
        android:id="@+id/btnScanner"
        android:layout_width="280dp"
        android:layout_height="70dp"
        android:background="@color/colorPrimaryDark"
        android:onClick="onClick"
        android:text="@string/qr_button_name"
        android:textColor="@android:color/background_light"
        android:textSize="20sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</android.support.constraint.ConstraintLayout>
```

В классе **MainActivity** объявляем и определяем нашу кнопку:

```
public class MainActivity extends AppCompatActivity {

    private Button btnScanner;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```
15          super.onCreate(savedInstanceState);
16          setContentView(R.layout.activity_main);
17
18          btnScanner = (Button) findViewById(R.id.btnScanner);
19      }
20
21      public void onClick(View view) {
22
23      }
24  }
```

Кнопка будет запускать сканер штрих-кодов. Для использования стандартного сканера нужно импортировать в наш проект из библиотеки **ZXing** классы **IntentIntegrator** (*н мерение-интегр тор*, объекты которого интегрированы в наш проект из библиотеки **ZXing**) и **IntentResult** (*н мерение-результ т*, его объекты будут описывать результат этой интеграции):

```
import com.google.zxing.integration.android.IntentIntegrator;
import com.google.zxing.integration.android.IntentResult;
```

Далее объявляем и определяем сканер как объект класса **IntentIntegrator:**

```
10  public class MainActivity extends AppCompatActivity {
11
12      private Button btnScanner;
13      private IntentIntegrator qrCodeScanner;
14
15
16      @Override
17      protected void onCreate(Bundle savedInstanceState) {
18          super.onCreate(savedInstanceState);
19          setContentView(R.layout.activity_main);
20
21          btnScanner = (Button) findViewById(R.id.btnScanner);
22          qrCodeScanner = new IntentIntegrator(this);
23      }
```

В обработчике события нажатия кнопки пропишем запуск сканера с помощью метода **initiateScan()** (*инициир ть ск ниров ние*):

```
25          public void onClick(View view) {
26              qrCodeScanner.initiateScan();
27          }
```

Для обработки результатов, полученных от сканера по завершении процесса сканирования, и отображения этих результатов в на-

шем приложении напишем еще один метод — **onActivityResult()**. Он уже знаком нам по работе с камерой и другими сторонними приложениями:

```
30    @Override
31    protected void onActivityResult(int requestCode, int resultCode, Intent data){
32
33    }
```

Для дальнейших действий с результатом, полученным от сканера, нам понадобятся: объект класса **IntentResult** result для анализа возвращенных параметров и строковая переменная myResult. В нее мы запишем полученные от сканера *д нные*, используя метод **getContents()** (*получить содержимое*).

```
31    @Override
32    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
33        IntentResult result = IntentIntegrator.parseActivityResult(requestCode, resultCode, data);
34        final String myResult = result.getContents();
35    }
```

Обработку результата пропишем в форме условного оператора **if-else**, проверяя, вернул ли сканер какие-либо данные:

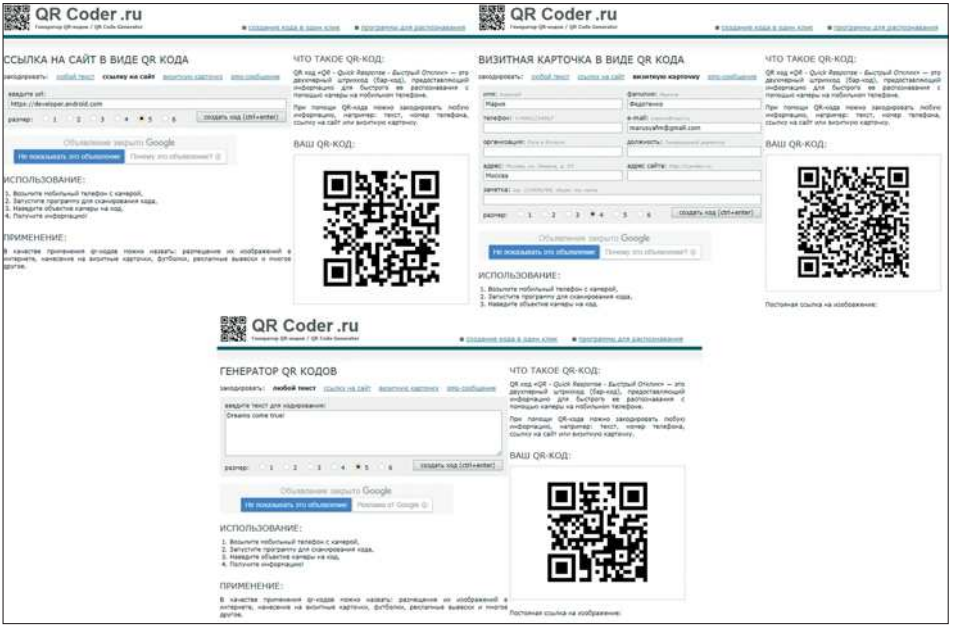
```
31    @Override
32    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
33        IntentResult result = IntentIntegrator.parseActivityResult(requestCode, resultCode, data);
34        final String myResult = result.getContents();
35
36        if (result.getContents() == null) {
37
38        } else {
39
40        }
41    }
```

Если запрос выполнен успешно, сканер сработал, но не вернул данные, записанные в QR-коде[1], пусть наше приложение выдает всплывающее сообщение с текстом «QR-код не распознан». Иначе, то есть сканер нашел QR-код, просканировал его и получил данные, приложение должно отображать результат сканирования в диалоговом окне:

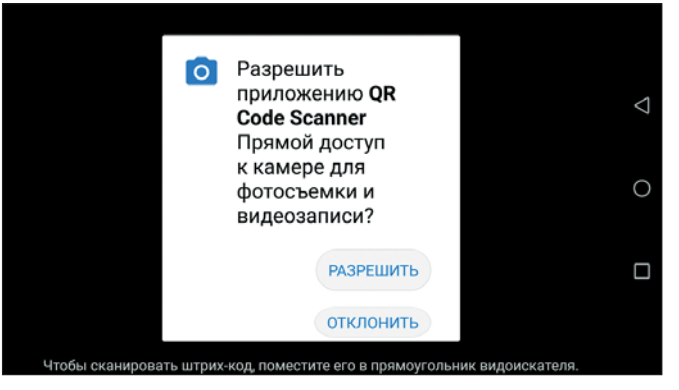
```
40        if (result.getContents() == null) {
41            Toast.makeText(this, "QR-код не распознан", Toast.LENGTH_LONG).show();
42        } else {
43            AlertDialog.Builder builder = new AlertDialog.Builder(this);
44            builder.setTitle("Результат сканирования:");
45            builder.setMessage(result.getContents());
46            AlertDialog myAlert = builder.create();
47            myAlert.show();
48        }
49    }
```

[1] Это может произойти, например, если QR-код не был найден или был «смазан» и нечитабелен.

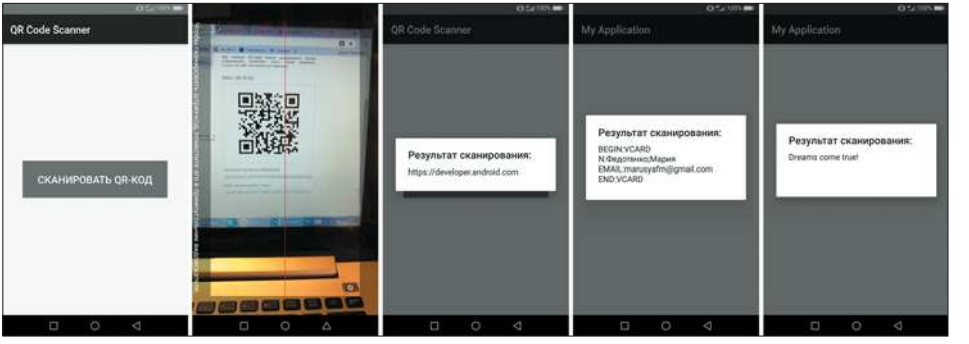
Готово! Теперь наш сканер можно опробовать, но для этого нужны примеры QR-кодов. Их можно найти в Интернете, на различных товарах и продуктах, а также сгенерировать самим, воспользовавшись одним из многочисленных сайтов, например **http://qrcoder.ru**/ Здесь можно сгенерировать коды для ссылки на сайт, визитной карточки или простого текстового сообщения:



Запускаем приложение (в данном случае лучше тестировать его на реальном смартфоне). При первом запуске видим запрос на разрешение использовать камеру. Разрешаем:



При нажатии на кнопку запускается сканер. При наведении на QR-код и его распознавании сканер издает звуковой сигнал и возвращает приложение на главный экран, где уже отображены результаты сканирования в зависимости от типа содержимого QR-кода (ссылка на сайт, визитка, произвольное сообщение).



Задания

1. Добавить в диалоговое окно две кнопки. Одна из них открывает сайт, ссылка на который содержится в QR-коде, а другая закрывает диалоговое окно.

   Для этого добавим в построитель диалогового окна методы **setNeutralButton()** для кнопки **Открыть сайт** и **setPositiveButton()** для кнопки **OK**.

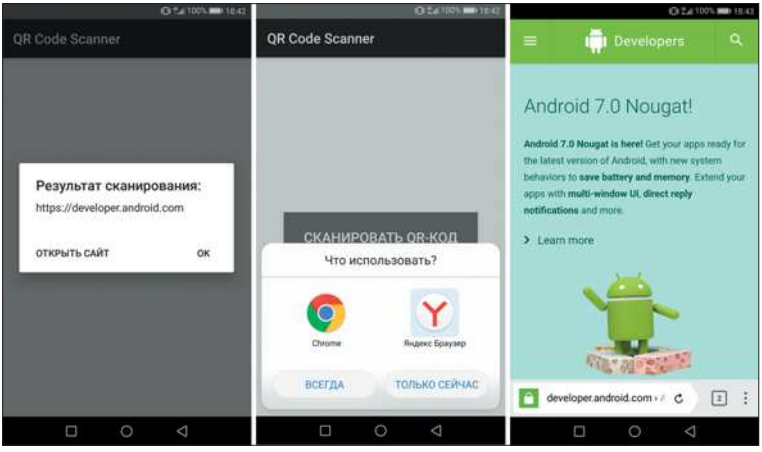
   Как закрывать диалоговое окно, мы уже знаем, а для кнопки перехода на сайт подойдет следующий обработчик события нажатия:

```
46      builder.setNeutralButton("Открыть сайт", new DialogInterface.OnClickListener() {
47          @Override
48          public void onClick(DialogInterface dialog, int which) {
49              Intent browserIntent = new Intent(Intent.ACTION_VIEW, Uri.parse(myResult));
50              startActivity(browserIntent);
51          }
52      });
```

   в котором мы анализируем полученные от сканера данные (являются ли они ссылкой)[1] и, если являются, инициируем запрос браузера.

   Запускаем приложение. Теперь при сканировании кода с заложенной ссылкой на сайт приложение запрашивает браузер, в котором откроется соответствующий сайт.

[1] Метод **Uri.parse(myResult)** анализирует (**parse** — *н лизиров ть*), содержит ли значение переменной myResult все обязательные части URI (Uniform Resource Identifier — унифицированного идентификатора ресурса).

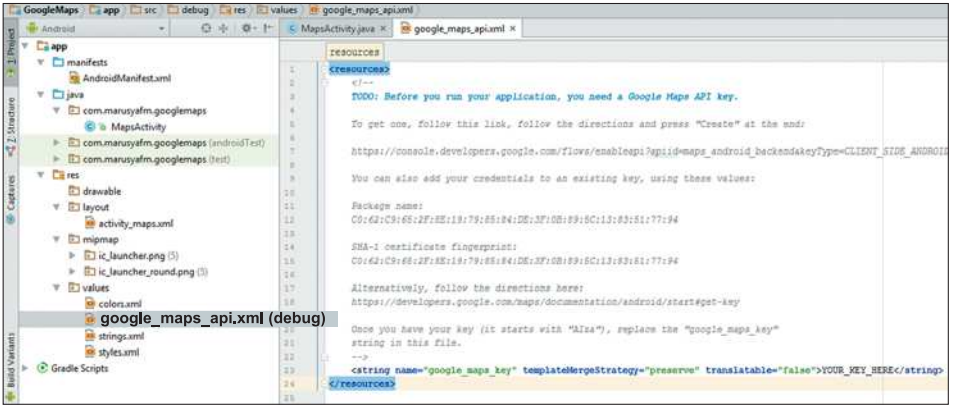


7.4. Работа с картами Google

В приложениях часто используются карты — отметить посещение, показать расположение компании, определить местоположение пользователя и подсказать ему ближайшие кафе или парк.

Для этого в **Android Studio** есть встроенные инструменты для работы с картами Google maps и шаблон **Google Maps Activity**. Выберем именно этот шаблон при создании нового проекта:

Главное изменение в структуре открывшегося проекта — новый файл **google_maps_api.xml** в папке ресурсов **values** (открывается по умолчанию).

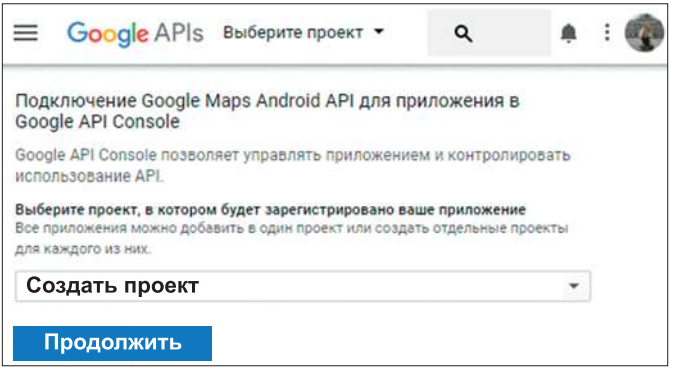


В этом файле мы пропишем ключ **API**. В комментариях уже содержатся подсказки, как его получить.

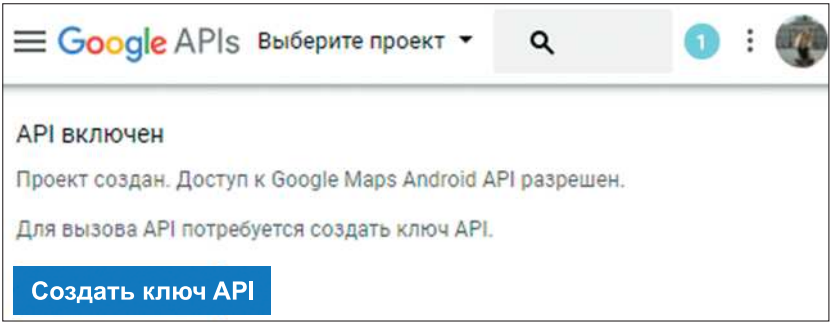
Пояснение

Google Maps API key — ключ для проверки подлинности — генерируемая совокупность проверочных кодов. Создан в рамках реализации политики конфиденциальности Google. Используется для получения некоторых данных о приложении и его пользователях без возможности их редактировать и без доступа к персональным данным.

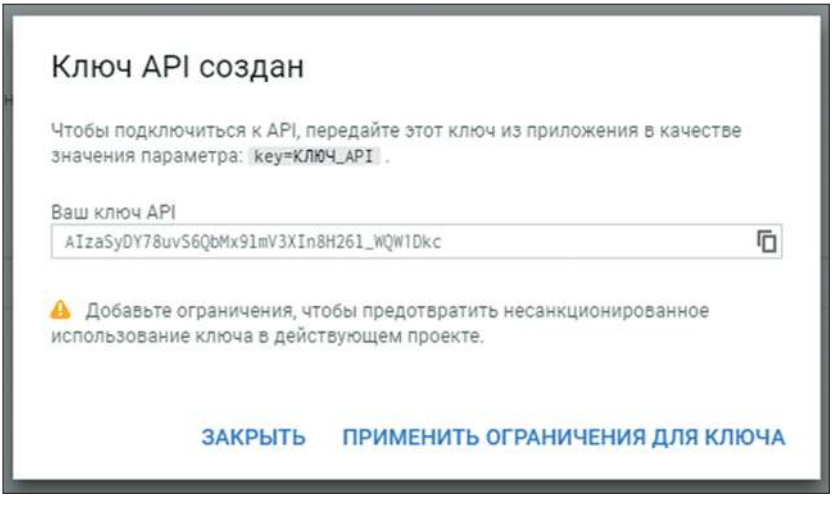
В строке № 7 кода есть ссылка, которую нужно скопировать и вставить в адресную строку браузера. Откроется страница Google APIs:



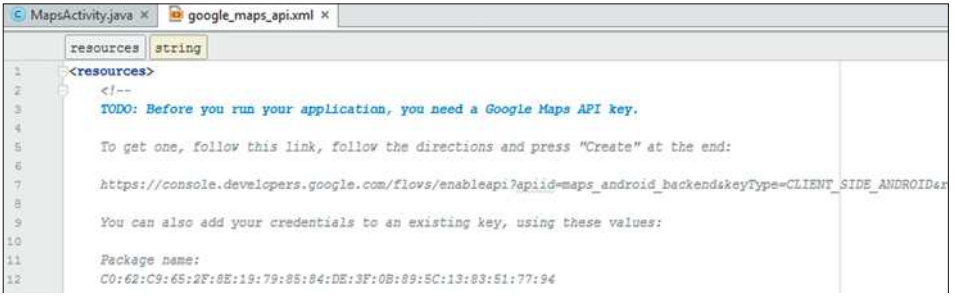
Выберем пункт Создать проект и нажмем кнопку **Продолжить.** Когда проект будет создан, нужно создать и сам ключ, нажав соответствующую кнопку:



После создания ключ будет выведен на экран:



Полученное значение нужно скопировать и вставить в файл **google_maps_api.xml** вместо текста YOUR_KEY_HERE (строка № 23):



```
    SHA-1 certificate fingerprint:
    C0:62:C9:65:2F:8E:19:79:85:84:DE:3F:0B:89:5C:13:83:51:77:94

    Alternatively, follow the directions here:
    https://developers.google.com/maps/documentation/android/start#get-key

    Once you have your key (it starts with "AIza"), replace the "google_maps_key"
    string in this file.
    -->
    <string name="google_maps_key" templateMergeStrategy="preserve" translatable="false">YOUR_KEY_HERE</string>
</resources>
```

На этом подготовительная работа завершена.

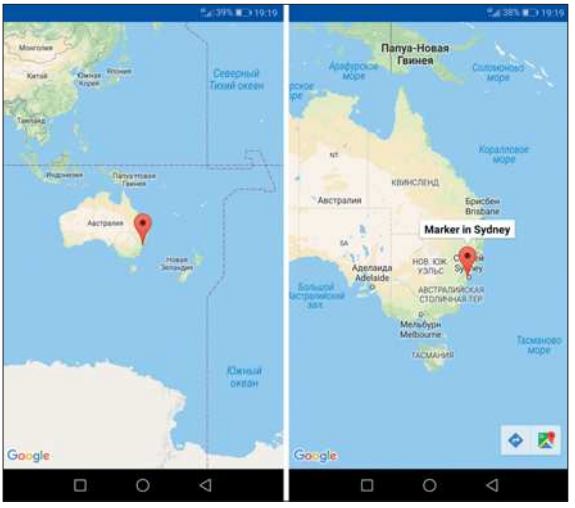
Перейдем к Java-классу **MapsActivity**. В нем, помимо метода **onCreate()**, уже прописан по умолчанию еще один метод — **onMapReady()** (*по готовности к рты*). К нему дан комментарий о том, что в этом методе прописываются все манипуляции с картой, которые нужно совершить при ее открытии, то есть когда она будет готова к работе.

По умолчанию на карту добавлен маркер в Сиднее (Австралия), но могут быть добавлены и другие маркеры, прослушиватели и обработчики событий, настройки камеры. Также говорится о том, что если сервисы Google Play не установлены на устройстве, то они будут запрошены. Использование карты будет доступно только после их установки и возврата к приложению:

```
    /**
     * Manipulates the map once available.
     * This callback is triggered when the map is ready to be used.
     * This is where we can add markers or lines, add listeners or move the camera. In this case,
     * we just add a marker near Sydney, Australia.
     * If Google Play services is not installed on the device, the user will be prompted to install
     * it inside the SupportMapFragment. This method will only be triggered once the user has
     * installed Google Play services and returned to the app.
     */
    @Override
    public void onMapReady(GoogleMap googleMap) {
        mMap = googleMap;

        // Add a marker in Sydney and move the camera
        LatLng sydney = new LatLng(-34, 151);
        mMap.addMarker(new MarkerOptions().position(sydney).title("Marker in Sydney"));
        mMap.moveCamera(CameraUpdateFactory.newLatLng(sydney));
    }
}
```

Сейчас в методе объявлена карта Google mMap и точка sydney с заданными координатами (широтой и долготой), соответствующими Сиднею. Затем на карту методом **addMarker()** (*доб вить м ркер*) добавляется маркер с позицией sydney и названием Marker in Sydney (*м ркер в Сиднее*). Метод **moveCamera()** (*переместить к меру*) отвечает за то, чтобы при открытии карты «камера» была направлена именно на наш объект.

Запустим приложение и убедимся в этом:

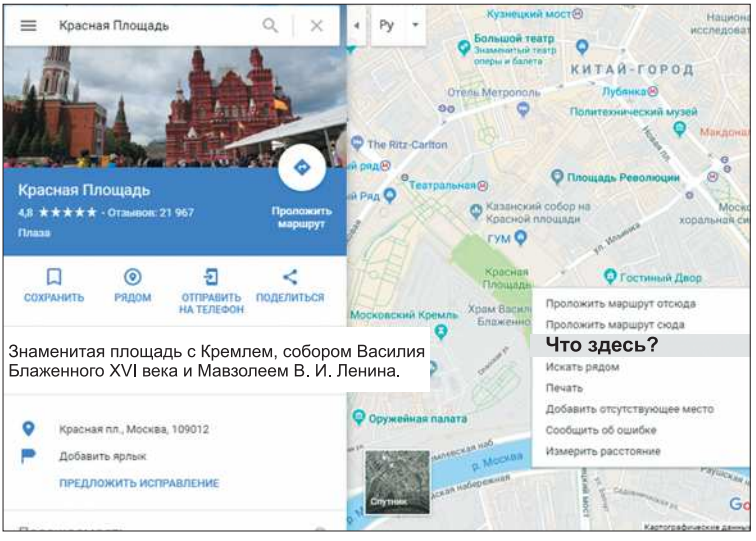


Открывается карта, на которой маркером отмечен Сидней, в левом нижнем углу логотип Google, а в правом — стандартные кнопки: **Открыть объект с помощью Google maps** и **Поделиться.**

Рассмотрим, какие возможности работы с картами предоставляет нам **Android Studio.**

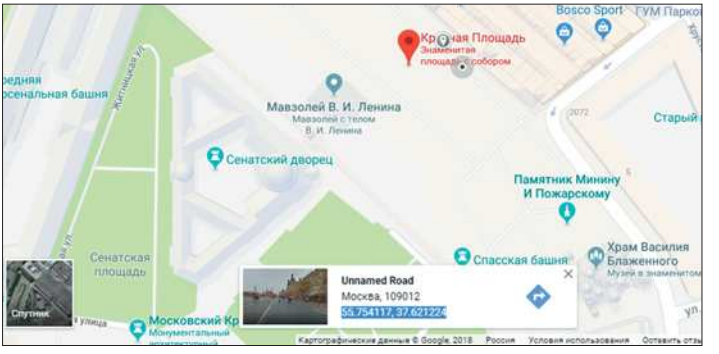
7.4.1. Установка маркера

Чтобы установить маркер в выбранном месте, откроем карты Google в браузере и найдем это место. Нажмем на него правой



кнопкой мыши и выберем пункт Что здесь? (для примера выбрана Красная площадь в Москве).

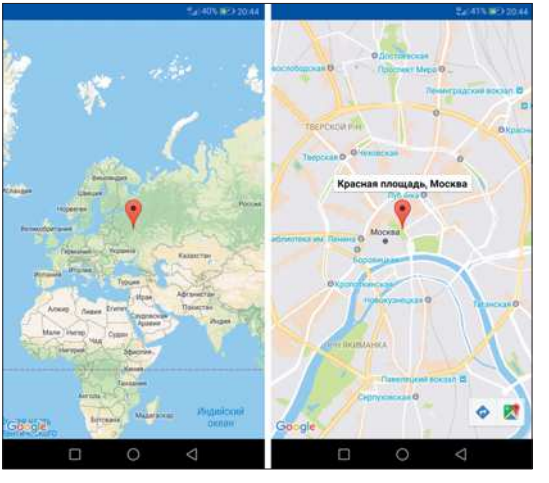
В нижней части экрана появится виджет с описанием местоположения выбранного объекта и координатами, которые нас и интересуют. Скопируем их.



Возвращаемся к нашему проекту Android Studio и заменяем текущие координаты на скопированные (строка № 41), название метки sydney на moscow, а название на Красная площадь, Москва.

```
37    @Override
38    public void onMapReady(GoogleMap googleMap) {
39        mMap = googleMap;
40
41        LatLng moscow = new LatLng(55.753688, 37.622037);
42        mMap.addMarker(new MarkerOptions().position(moscow).title("Красная площадь, Москва"));
43        mMap.moveCamera(CameraUpdateFactory.newLatLng(moscow));
44    }
45 }
```

Теперь при запуске приложения мы увидим маркер на новом месте:

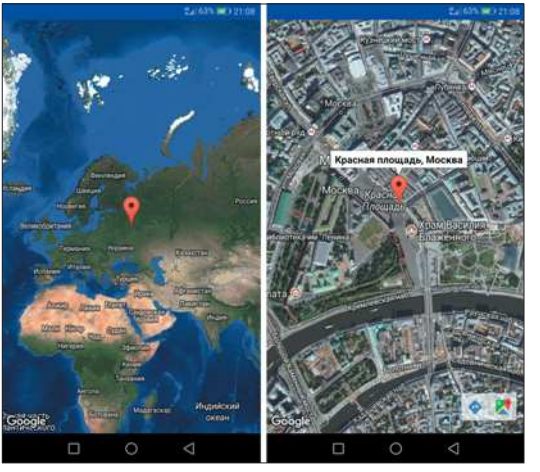


7.4.2. Изменение типа и настроек карты

Тип карты задается с помощью метода **setMapType()** (*уст новить тип к рты*):

```
mMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);
```

Этот метод принимает на вход значения MAP_TYPE_HYBRID (*тип к рты — гибрид*), MAP_TYPE_SATELLITE (*тип к рты — спутник*), MAP_TYPE_NORMAL и так далее по аналогии с web-версией карт Google. Например, наше приложение с картой типа гибрид будет выглядеть следующим образом:



Тип карты можно задать также в xml-разметке (атрибут **mapType**), как и другие параметры: **cameraTargetLat** (широта), **cameraTargetLng** (долгота), **cameraTilt** (наклон — обычно карта отображается под прямым углом, как будто мы смотрим на нее сверху, но может отображаться и под наклоном), **cameraZoom** (значение для приближения по умолчанию), **uiCompass** (включена ли кнопка компаса, как на Google maps), а также включены ли **uiRotateGestures** (жесты вращения карты), **uiScrollGestures** (жесты прокручивания карты), **uiTiltGestures** (жесты изменения наклона карты), **uiZoomControls** (кнопки приближения/отдаления), **uiZoomGestures** (жесты приближения/отдаления):

MapsActivity.java × | activity_maps.xml × | AndroidManifest.xml × | google

```
1  <fragment xmlns:android="http://schemas.android.com/apk/res/android"
2      xmlns:map="http://schemas.android.com/apk/res-auto"
3      xmlns:tools="http://schemas.android.com/tools"
4      android:id="@+id/map"
5      android:name="com.google.android.gms.maps.SupportMapFragment"
```

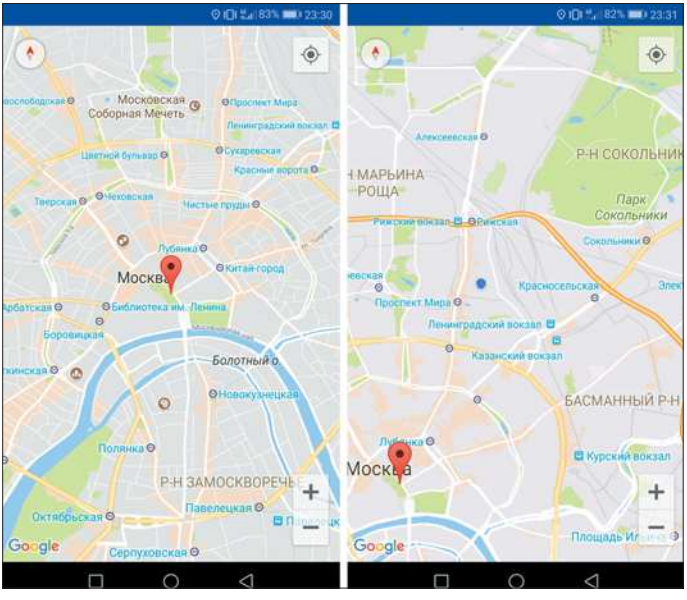
```
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    map:cameraTargetLat="55.753688"
    map:cameraTargetLng="37.622037"
    map:cameraTilt="30"
    map:cameraZoom="13"
    map:mapType="normal"
    map:uiCompass="true"
    map:uiRotateGestures="true"
    map:uiScrollGestures="true"
    map:uiTiltGestures="true"
    map:uiZoomControls="true"
    map:uiZoomGestures="true"
    tools:context="com.marusyafm.googlemaps.MapsActivity" />

```

7.4.3. Определение текущего местоположения

Для определения текущего местоположения используется метод **setMyLocationEnabled() (*уст новить мое местоположение включенным*):**

```
mMap.setMyLocationEnabled(true);
```

Теперь в приложении появилась кнопка нахождения текущего местоположения и она работает — местоположение указывается синим значком по центру экрана:



7.5. Приложение «Вокруг света за 80 дней»

В этом курсе мы не занимаемся разработкой игр, но создание викторины — отличный способ поработать с картами Google и другими приложениями.

Концепция приложения: приложение-викторина. При запуске приложения открывается карта Google, на которую нанесены точки (маркеры). Каждая точка — переход к вопросу о месте, в котором она расположена. Цель: ответить правильно на все вопросы и «собрать» все точки на карте.

Шаг 1. Создать Android Studio проект, в качестве стартовой Activity выбрать **Google Maps Activity**.

Шаг 2. Для работы с картами Google получить **ключ API** (с помощью Google APIs) и прописать его в файле **google_maps_api.xml**.

В Google APIs для каждого нового приложения можно создавать новый проект, а можно использовать текущий. Но речь идет именно о проекте, **ключ нужно каждый раз создавать заново**. Если скопировать сам ключ API из другого приложения — работать не будет.

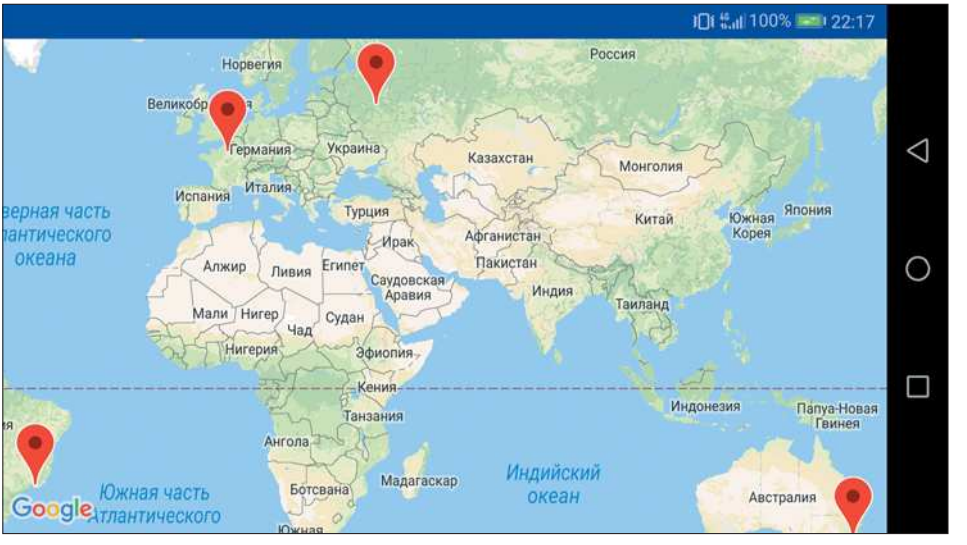
Шаг 3. Добавить на карту несколько маркеров.

В файле **MapsActivity.java** найдем метод **onMapReady()**, в нем пропишем несколько новых точек и для них установим на карте маркеры. Пусть это будут Москва (moscow), Париж (paris) и Рио-де-Жанейро (rio):

```
37    @Override
38    public void onMapReady(GoogleMap googleMap) {
39        mMap = googleMap;
40
41        // Add a marker in Sydney and move the camera
42        LatLng sydney = new LatLng(-34, 151);
43        LatLng moscow = new LatLng(55.755814, 37.617635);
44        LatLng paris = new LatLng(48.856651, 2.351691);
45        LatLng rio = new LatLng(-22.801122, -43.336894);
46
47        mMap.addMarker(new MarkerOptions().position(sydney).title("Sydney"));
48        mMap.addMarker(new MarkerOptions().position(moscow).title("Moscow"));
```

```
49          mMap.addMarker(new MarkerOptions().position(paris).title("Paris"));
50          mMap.addMarker(new MarkerOptions().position(rio).title("Rio"));
51
52          mMap.moveCamera(CameraUpdateFactory.newLatLng(sydney));
53      }
```

Запустим приложение и убедимся, что на карте действительно можно одновременно устанавливать несколько маркеров:

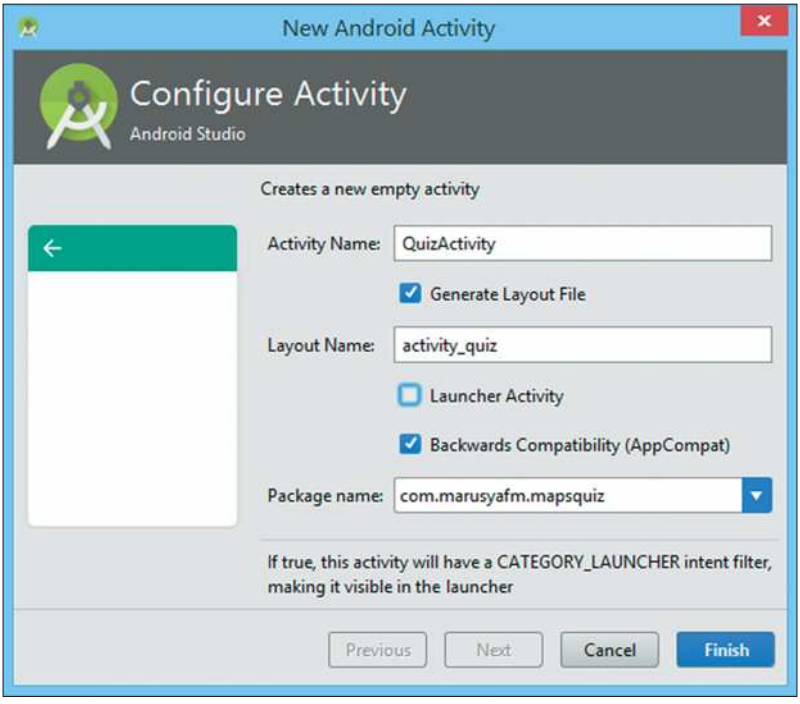


Шаг 4. Добавить Activity для вопросов викторины.

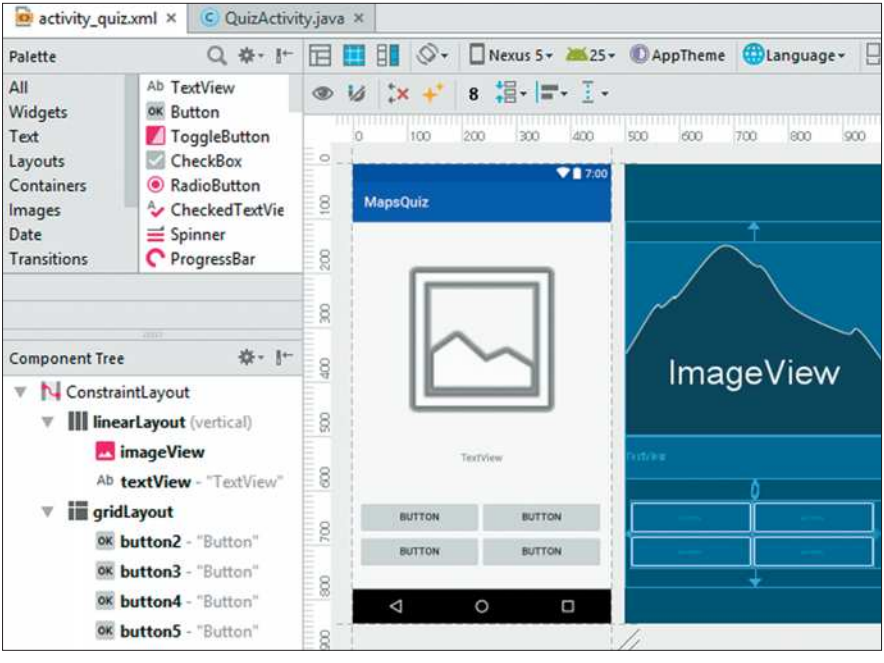
По логике приложения, при нажатии на маркер должен осуществиться переход к вопросу о месте, к которому маркер привязан. Чтобы не слишком усложнять код главной Activity, оставим в ней только работу с картами, а для работы с вопросами организуем вторую Activity.

При создании новой Activity зададим ей имя QuizActivity, обязательно установим галочки **Generate Layout File** (его имя можно оставить по умолчанию) и **Backwards Compatbility**, а галочку **Launcher Activity**, наоборот, обязательно снимем, потому что создаем не главную Activity.

Когда новая Activity будет создана и все изменения вступят в силу, начнем с ее xml-разметки.

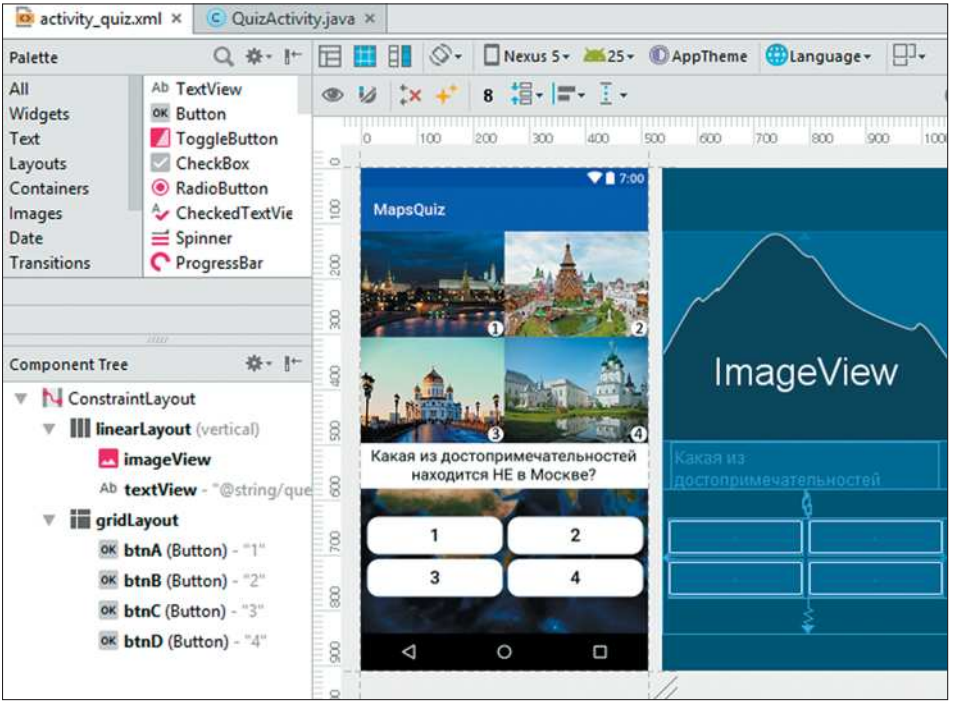


Шаг 5. Создать xml-разметку для **QuizActivity.**
Откроем файл **activity_quiz.xml.**



Для наглядности в вопросы добавим картинки. Для создания области вопроса разместим в верхней части экрана **вертикальный LinearLayout**, а в нем картинку и текст вопроса. В нижней части экрана будет располагаться область выбора варианта ответа. Разместим здесь макет **GridLayout** и четыре кнопки для выбора одного из вариантов ответа.

Зададим элементам интерфейса необходимые свойства: идентификаторы, ресурсы, размеры, расположение относительно друг друга. Для кнопок, как всегда, обязательно свойство **onClick**.



Шаг 6. Объявить и определить элементы QuizActivity в файле **QuizActivity.java**.

Шаг 7. Прописать обработчики событий нажатия кнопок с вариантами ответов **(btnA, btnB, btnC, btnD)**.

Пусть при выборе неверного ответа кнопка становится красной, а верного — зеленой, а остальные кнопки красными. Для этого используем метод **setBackgroundColor()**. Не забудем добавить ресурсы для новых цветов в файл **colors.xml**

Для кнопки с верным ответом также реализуем диалоговое окно с комментарием по ответу.

```
44        AlertDialog.Builder builder = new AlertDialog.Builder(QuizActivity.this);
45        builder.setTitle("Верно!")
46             .setMessage("Это Ростовский кремль")
47             .setIcon(R.drawable.smile)
48             .setCancelable(false)
49             .setPositiveButton("Продолжить",
50                     (dialog, id) → {
52                         // позже здесь пропишем действия при нажатии на кнопку
53                     });
55        AlertDialog alert = builder.create();
56        alert.show();
```

Шаг 8. Реализовать переход между Activity.

Итак, мы уже создали и настроили Activity с вопросами викторины, но главной Activity остается карта. Поэтому, чтобы увидеть вопросы, а после ответа вернуться к карте, нужно обязательно реализовать переходы.

Вернемся к файлу **MapsActivity.java**. Чтобы переход к викторине происходил при нажатии на маркер на карте, маркерам нужно добавить прослушиватель. Используем метод **setOnMarkerClickListener()** (*установить прослушиватель нажатий на маркер*):

```
37    @Override
38    public void onMapReady(GoogleMap googleMap) {
39        mMap = googleMap;
40
41        // Add a marker in Sydney and move the camera
42        LatLng sydney = new LatLng(-34, 151);
43        LatLng moscow = new LatLng(55.755814, 37.617635);
44        LatLng paris = new LatLng(48.856651, 2.351691);
45        LatLng rio = new LatLng(-22.801122, -43.336894);
46
47        mMap.addMarker(new MarkerOptions().position(sydney).title("Sydney"));
48        mMap.addMarker(new MarkerOptions().position(moscow).title("Moscow"));
49        mMap.addMarker(new MarkerOptions().position(paris).title("Paris"));
50        mMap.addMarker(new MarkerOptions().position(rio).title("Rio"));
51
52        mMap.moveCamera(CameraUpdateFactory.newLatLng(sydney));
53
54        mMap.setOnMarkerClickListener();
55    }
56 }
```

В параметрах метода начинаем прописывать:

```
new GoogleMap.OnMarkerClickListener()
```

и, следуя подсказкам **Android Studio**, добавляем метод **onMarkerClick()** (*по н ж тию н м ркер*):

```
58      mMap.setOnMarkerClickListener(new GoogleMap.OnMarkerClickListener() {
59          @Override
60          public boolean onMarkerClick(Marker markerName) {
61              if(markerName.getTitle().equals("Moscow")) {
62                  Intent intent = new Intent(MapsActivity.this, QuizActivity.class);
63                  startActivity(intent);
64              }
65              return true;
66          }
67      });
```

Внутри метода пропишем, что если имя нажатого маркера — Moscow (для сравнения используем метод **equals()**), то с помощью уже знакомого нам намерения **Intent** должен происходить переход к Activity с вопросами.

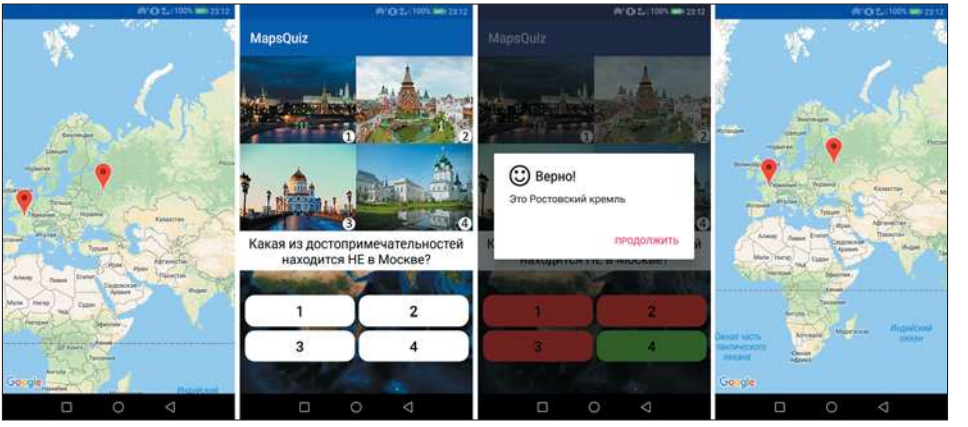
Отлично, теперь при нажатии на маркер Москвы мы увидим вопрос о Москве и даже сможем на него ответить и узнать результат. Но как вернуться обратно к карте?

Сейчас у нас ничего не происходит при нажатии на кнопку «Продолжить» в диалоговом окне. В построителе окна найдем метод нажатия на кнопку, а в нем пропишем метод закрытия диалогового окна и затем переход к классу **MapsActivity** (снова через намерение **Intent**):

```
44      AlertDialog.Builder builder = new AlertDialog.Builder(QuizActivity.this);
45      builder.setTitle("Верно!")
46          .setMessage("Это Ростовский кремль")
47          .setIcon(R.drawable.smile)
48          .setCancelable(false)
49          .setPositiveButton("Продолжить",
50                  (dialog, id) -> {
52                      dialog.cancel();
53                      Intent intent = new Intent(QuizActivity.this, MapsActivity.class);
54                      startActivity(intent);
55                  });
57      AlertDialog alert = builder.create();
58      alert.show();
```

Шаг 9. Собрать и запустить приложение.

Мы сделали это! При нажатии на московский маркер приложение отображает викторину, а при правильном ответе и нажатии на кнопку продолжения осуществляется переход обратно к карте.



Задания

Мы выстроили логику приложения. Сейчас оно максимально простое (всего с одним вопросом), поэтому существует масса возможностей по его доработке, например:

1. Добавить еще вопросы в викторину и комментарии к неправильным ответам (сейчас только к правильному). При большом количестве вопросов и постоянном обновлении карты можно для обеих Activity использовать фрагменты.
2. После получения правильного ответа можно выполнить переход к сайтам с некоторой справочной информацией (внутри приложения, с помощью **WebView**) или реализовать таким образом подсказки.
3. После получения правильного ответа и возврата к карте маркер может исчезать или становиться зеленым.
   Вот один из способов задания маркеру цвета:

```
mMap.addMarker(new
MarkerOptions().position(moscow).icon(BitmapDescriptor
Factory.defaultMarker(BitmapDescriptorFactory.HUE_GREEN)).title
(“Marker in Moscow”));
```

4. Подключить к приложению базу данных, хранить в ней вопросы и ответы, а уже из нее отображать в приложение. Примерная структура БД в этом случае: таблица вопросов (идентификатор вопроса, текст вопроса) и таблица ответов (идентификатор ответа, текст ответа, идентификатор вопроса, идентификатор правильного ответа).
5. Добавить приложению стартовый экран.

Итоги главы 7

При разработке приложений не нужно усложнять себе жизнь и изобретать велосипед. Можно, нужно и удобно для реализации некоторых функций обращаться к сторонним приложениям. В этом мы убедились в уроках главы 7 и научились:

1. Вызывать сторонние приложения: телефон, браузер, электронную почту, магазин приложений Google Play.
2. Включать фонарик на устройстве, который оказался вспышкой камеры, а ее вызывать мы уже умели.
3. Конвертировать сайт в мобильное приложение с помощью контейнера **WebView**, чтобы избежать работы с большим количеством текста и любого другого контента, который уже существует в виде сайта.
4. Считывать и анализировать QR-коды.
5. Подключать в приложение карты Google, определять местоположение, устанавливать маркеры и реализовывать различные переходы при нажатии на них.

Глава 8. Итоговый проект «Общалка». Уровень: продвинутый Android-разработчик

Вот мы и подошли к завершению нашего путешествия по основам Android-разработки. Но если мы все это время говорили об основах, почему в названии главы фигурирует продвинутый Android-разработчик? Потому что, разбирая основы Android-разработки на примерах простеньких приложений, мы собирали навыки, как фрагменты пазла, и теперь, соединив их, вполне можем собрать цельную и сложную картину.

Не верится? А как насчет создания собственного мессенджера вроде WhatsApp или Facebook Messenger? Да, конечно, у этих приложений большая команда разработчиков... Но спроектировать интерфейс и воспроизвести основную функцию — обмен сообщениями — мы уже вполне можем.

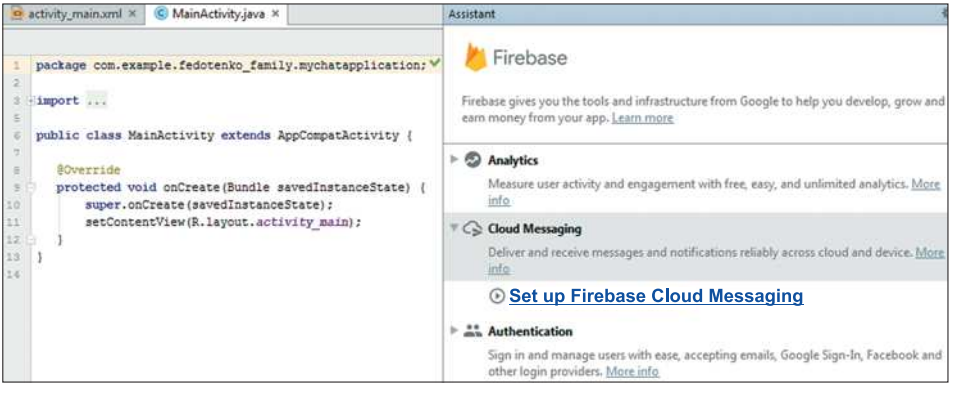
Концепция приложения: приложение для обмена сообщениями.

Итак, в качестве итогового проекта мы создадим простой чат, который в дальнейшем легко трансформируется в мессенджер.

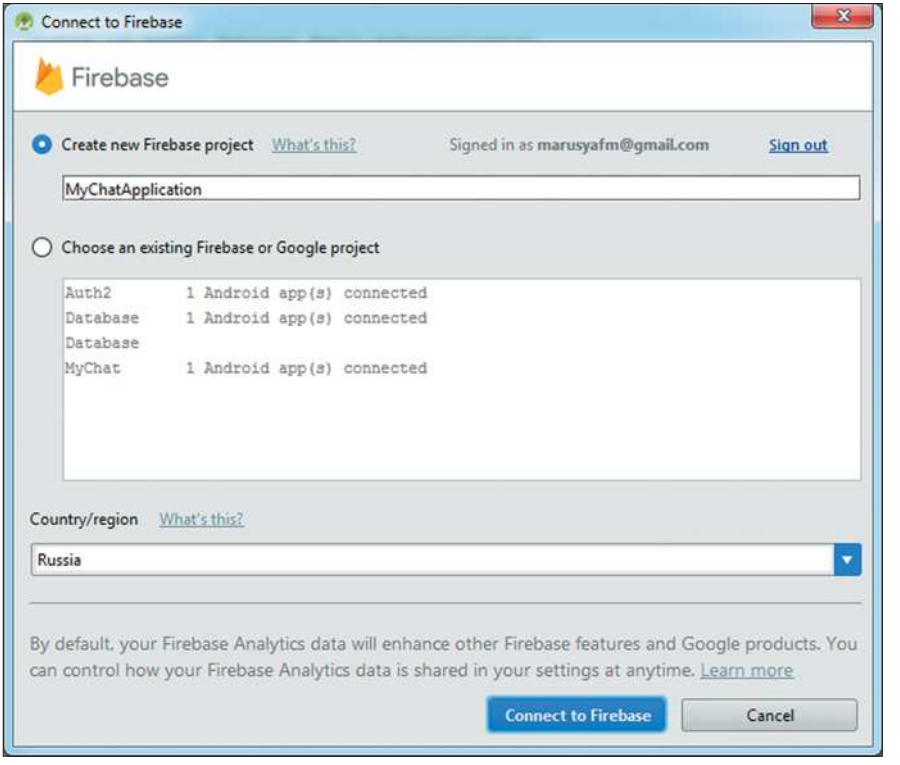
Шаг 1. Создать пустой проект My Chat Application.

Шаг 2. Подключить проект к Firebase.

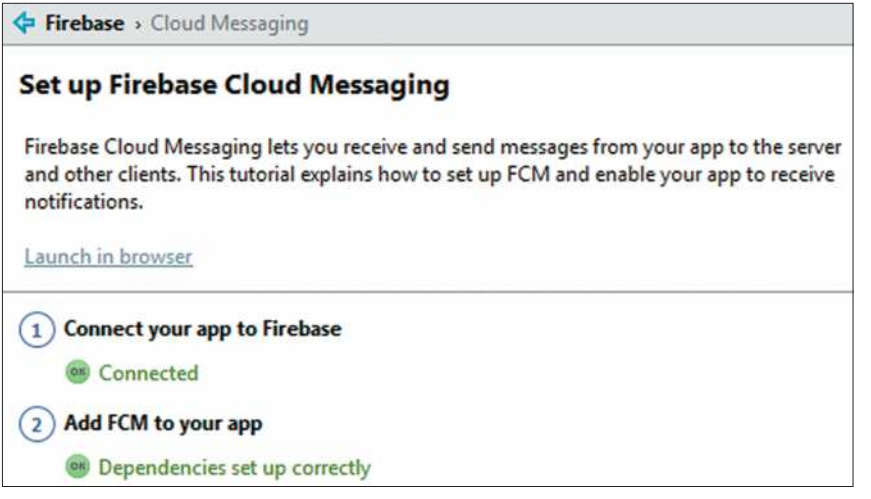
Написать полный код приложения для обмена сообщениями нам было бы пока сложновато, но инструмент Firebase помогает в этом и многим крупным компаниям. Последуем их примеру и освоим еще одну возможность, представляемую Firebase, — **Cloud Messaging** (*обл чный обмен сообщениями*). Запустим Firebase в нашем проекте, выберем соответствующий пункт в **Assistant** и нажмем ссылку **Set up Firebase Cloud Messaging** (*уст новить обл чный обмен сообщениями Firebase*):



В открывшейся вкладке **Cloud Messaging** выполним действия ① и ②. При подключении проекта к Firebase обязательно выберем создание нового проекта:

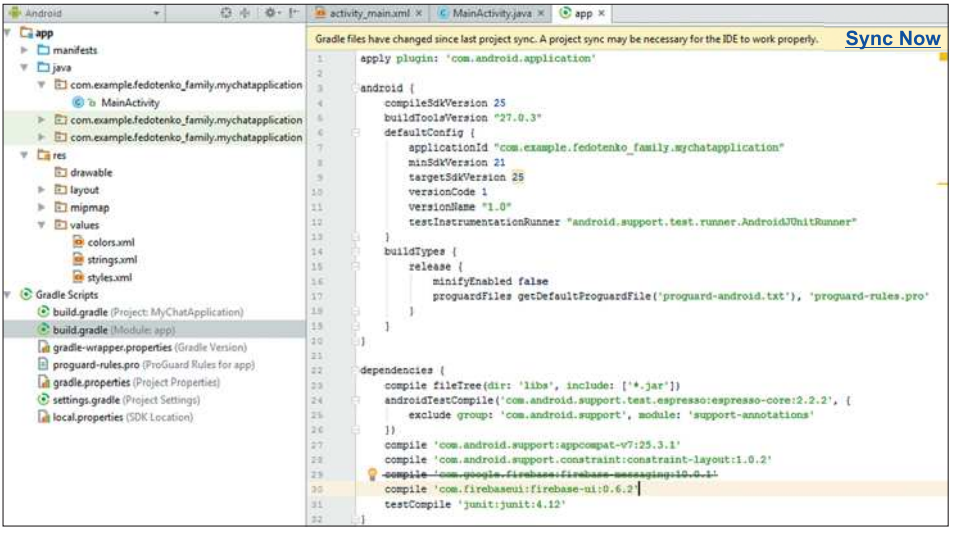


Когда оба действия выполнены, соответствующие пункты станут зелеными. На этом работа с **Assistant** завершена.



Шаг 3. Прописать дополнительные разрешения.

Теперь найдем в структуре проекта файл **buld.gradle (Module: app)** и в нем заменим compile ‘com.google.firebase:firebase-messaging: 10.0.1’ (строка № 29) на compile ‘com.firebaseui:firebase-ui:0.6.2’ (строка № 30):



После этого следует обязательно синхронизировать проект нажатием ссылки **Sync Now**.

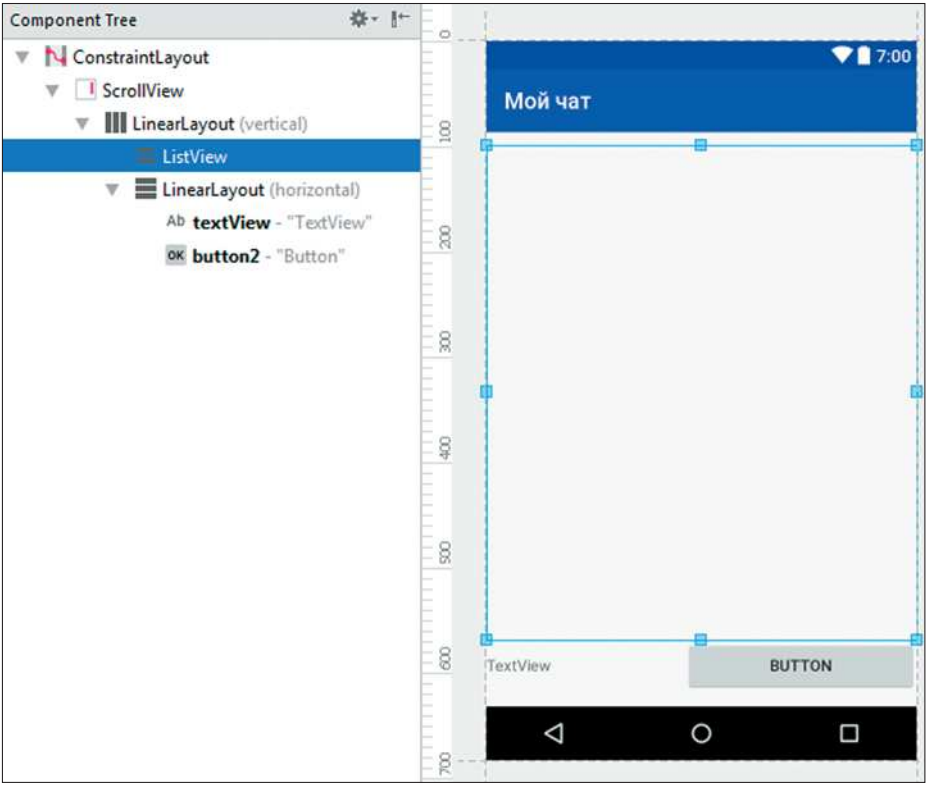
При синхронизации могут отобразиться различные предупреждения (не ошибки) — их можно скрыть:

Шаг 4. Спроектировать интерфейс приложения.

Открываем файл **activity_main.xml** и удаляем из него текст.

Экран в мессенджерах обычно прокручиваемый, поэтому добавим контейнер **ScrollView**. Как мы помним, **ScrollView** может иметь только один элемент-наследник, поэтому внутри него размещаем **вертикальный LinearLayout**.

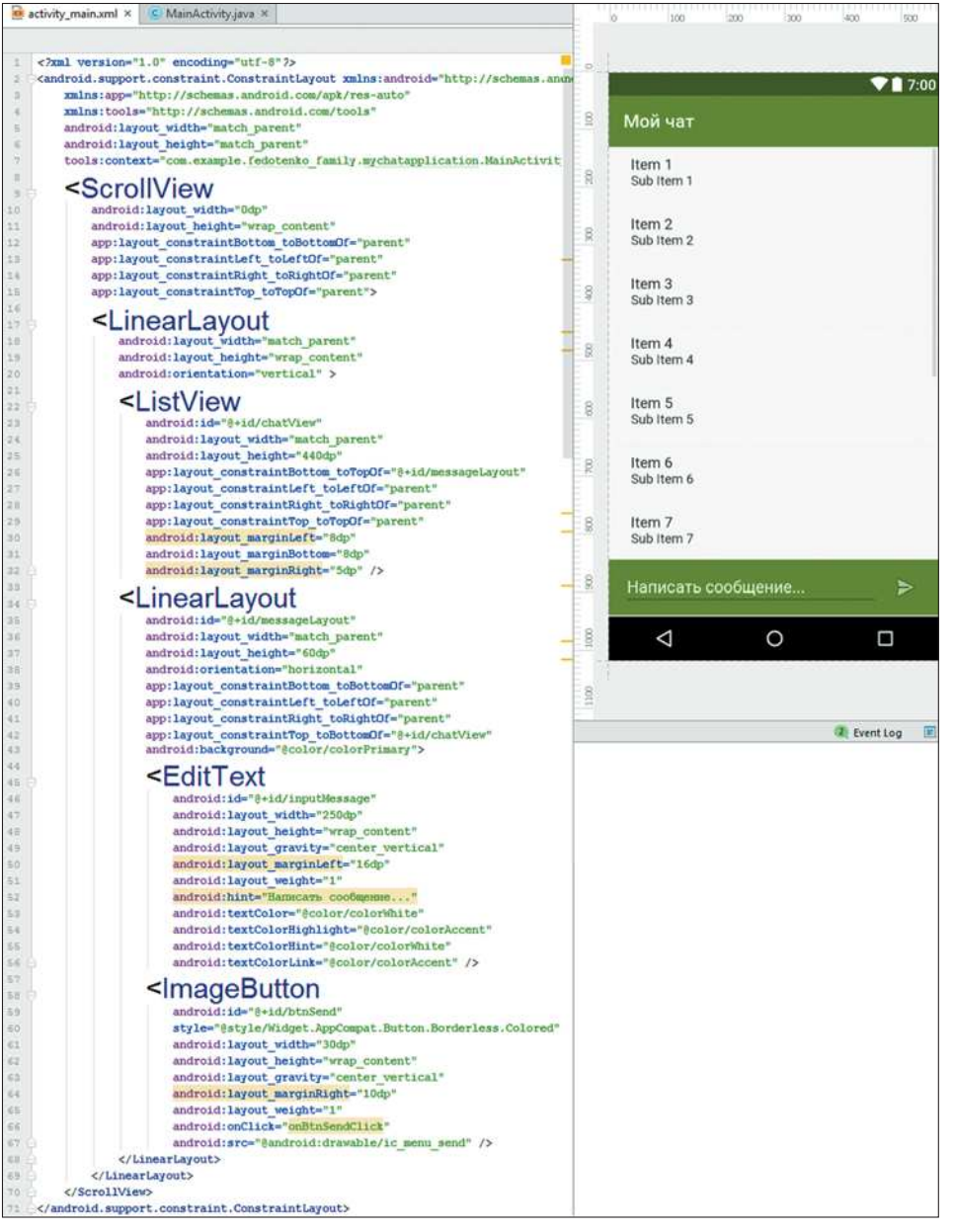
Область, в которой отображаются сообщения, это, по сути, список, поэтому первым внутри **LinearLayout** разместим элемент **ListView**. Еще нам нужна область написания и отправки сообщения — это текстовый элемент и кнопка, «завернутые» в **горизонтальный LinearLayout**:



Шаг 5. Настроить свойства элементов.

Как и во всех предыдущих уроках, элементам нужно задать идентификаторы и расположение относительно друг друга, настроить шрифты и цвета.

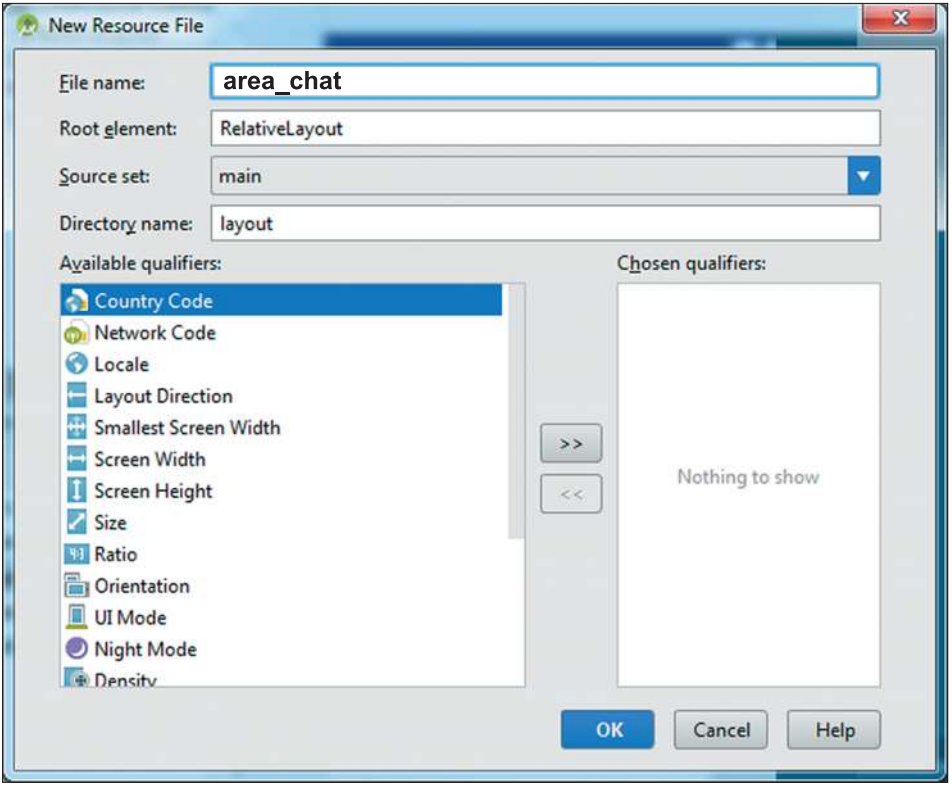
Текстовый элемент изменим на **EditText** (для того чтобы вводить в него сообщения), а кнопку — на **ImageButton** и зададим ей стандартное изображение для кнопки **Отправить** из коллекции.


activity_main.xml
MainActivity.java
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.andr
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context="com.example.fedotenko_family.mychatapplication.MainActivit
<ScrollView
android:layout_width="0dp"
android:layout_height="wrap_content"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toTopOf="parent">
<LinearLayout
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:orientation="vertical" >
<ListView
android:id="@+id/chatView"
android:layout_width="match_parent"
android:layout_height="440dp"
app:layout_constraintBottom_toTopOf="@+id/messageLayout"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toTopOf="parent"
android:layout_marginLeft="8dp"
android:layout_marginBottom="8dp"
android:layout_marginRight="5dp" />
<LinearLayout
android:id="@+id/messageLayout"
android:layout_width="match_parent"
android:layout_height="60dp"
android:orientation="horizontal"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toBottomOf="@+id/chatView"
android:background="@color/colorPrimary">
<EditText
android:id="@+id/inputMessage"
android:layout_width="250dp"
android:layout_height="wrap_content"
android:layout_gravity="center_vertical"
android:layout_marginLeft="16dp"
android:layout_weight="1"
android:hint="Написать сообщение..."
android:textColor="@color/colorWhite"
android:textColorHighlight="@color/colorAccent"
android:textColorHint="@color/colorWhite"
android:textColorLink="@color/colorAccent" />
<ImageButton
android:id="@+id/btnSend"
style="@style/Widget.AppCompat.Button.Borderless.Colored"
android:layout_width="30dp"
android:layout_height="wrap_content"
android:layout_gravity="center_vertical"
android:layout_marginRight="10dp"
android:layout_weight="1"
android:onClick="onBtnSendClick"
android:src="@android:drawable/ic_menu_send" />
</LinearLayout>
</LinearLayout>
</ScrollView>
</android.support.constraint.ConstraintLayout>
7:00
Мой чат
Item 1
Sub Item 1
Item 2
Sub Item 2
Item 3
Sub Item 3
Item 4
Sub Item 4
Item 5
Sub Item 5
Item 6
Sub Item 6
Item 7
Sub Item 7
Написать сообщение...
Event Log

Шаг 6. Область сообщения.

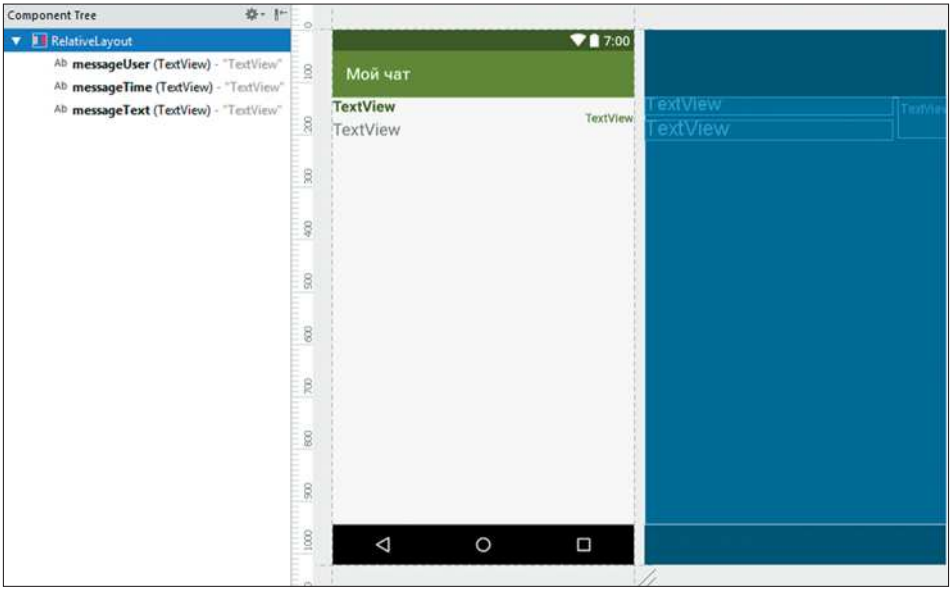
Мы уже выделили под все сообщения элемент **ListView**, но нужно еще настроить вид одного сообщения в этом списке.

Создадим для сообщений отдельный xml-файл, поместим его в ту же папку **layout** и назовем area_chat. Корневым элементом выберем **RelativeLayout**:



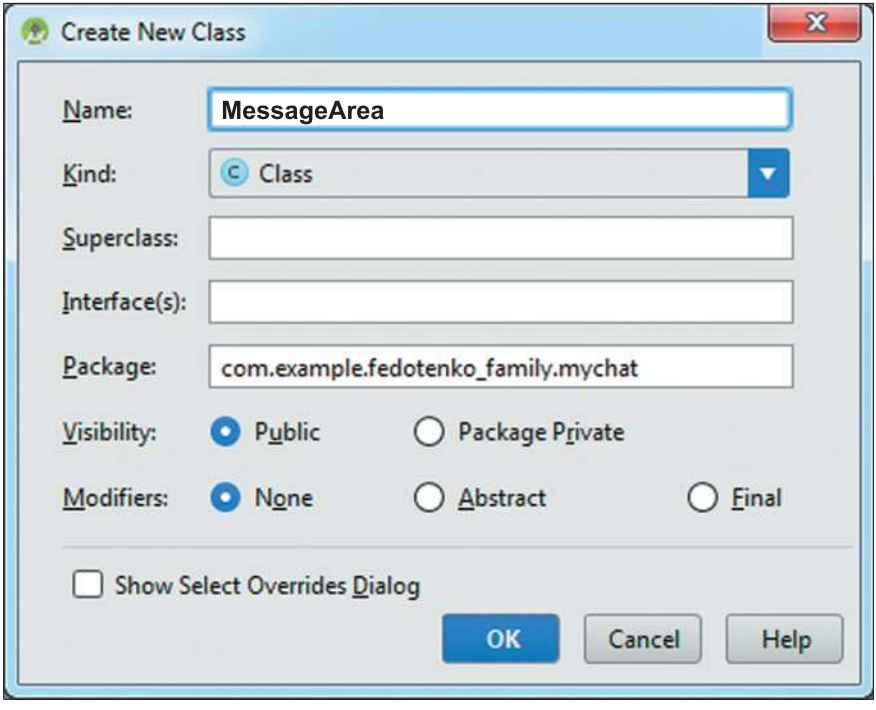
По умолчанию у сообщения в чате три основных параметра: **имя пользователя**, отправившего сообщение, **текст сообщения** и **время отправки** сообщения.

Добавим три текстовых элемента, зададим им идентификаторы messageUser (расположим элемент в левом верхнем углу), messageText (под предыдущим элементом) и messageTime (справа, по высоте обоих элементов):



Шаг 7. Создать новый класс для сообщений.

Для определения параметров и методов сообщений создадим новый Java-класс MessageArea:



В созданном классе объявим и определим параметры сообщения: messageText (текст), messageUser (пользователь) и messageTime (время отправки):

MessageArea.java × | MainActivity.java × | area_chat.xml × | activity_

MessageArea | setMessageUser()

```
1  package com.example.fedotenko_family.mychatapplication;
2
3  import java.util.Date;
4
5  public class MessageArea {
6      private String messageText;
7      private String messageUser;
8      private long messageTime;
9
10     public MessageArea(String messageText, String messageUser){
11         this.messageText = messageText;
12         this.messageUser = messageUser;
13         messageTime = new Date().getTime();
14     }
```

Добавим необходимые методы get и set для каждого из параметров, чтобы иметь возможность как получать их значения, так и изменять их. Методы, возвращающие значение некоего свойства класса, начинаются на **get** (*получить*) и называются **геттеры**. Методы, устанавливающие новое значение, начинаются на **set** (*уст новить*) и называются **сеттеры**.

```
10     public MessageArea(String messageText, String messageUser){
11         this.messageText = messageText;
12         this.messageUser = messageUser;
13         messageTime = new Date().getTime();
14     }
15
16     public MessageArea(){
17     }
18
19     public String getMessageText() {
20         return messageText;
21     }
22
```

```
23    public void setMessageText(String messageText) {
24        this.messageText = messageText;
25    }
26
27    public String getMessageUser() {
28        return messageUser;
29    }
30
31    public void setMessageUser(String messageUser) {
32        this.messageUser = messageUser;
33    }
34
35    public long getMessageTime() {
36        return messageTime;
37    }
38
39    public void setMessageTime(long messageTime) {
40        this.messageTime = messageTime;
41    }
42 }
```

Шаг 8. Дописать главный Java-класс.

Перейдем в файл **MainActivity.java**.

Объявим и определим основные элементы интерфейса — кнопку для отправки сообщения, список для отображения сообщений и текст для ввода сообщения:

```
20  public class MainActivity extends AppCompatActivity {
21
22      private ImageButton btnSend;
23      private ListView chatView;
24      private EditText inputMessage;
25
26      private static int SIGN_IN_REQUEST_CODE = 1;
27      private FirebaseListAdapter<MessageArea> adapter;
28
29      @Override
30      protected void onCreate(Bundle savedInstanceState) {
31          super.onCreate(savedInstanceState);
32          setContentView(R.layout.activity_main);
33
34          btnSend = (ImageButton)findViewById(R.id.btnSend);
35          chatView = (ListView)findViewById(R.id.chatView);
36          inputMessage = (EditText) findViewById(R.id.inputMessage);
37
```

```
38          if (FirebaseAuth.getInstance().getCurrentUser() == null) {
39              startActivityForResult(AuthUI.getInstance()
40                      .createSignInIntentBuilder()
41                      .build(), SIGN_IN_REQUEST_CODE);
42          } else {
43              showChat();
44          }
45      }
```

Здесь нам понадобятся еще две переменные: SIGN_IN_REQUEST_CODE со значением 1 (код результата выполнения запроса при авторизации) и adapter — объект класса **FirebaseListAdapter** для работы с областью сообщений.

В методе **onCreate()** допишем конструкцию **if-else** для определения текущего пользователя: если текущий пользователь не найден — запускать процесс авторизации, иначе — отображать чат.

Android Studio выделяет метод **showChat()** красным, потому что мы его еще не написали. Создадим метод **showChat()** и внутри него обратимся к адаптеру. На вход адаптера передаем:

- класс, который работает с сообщениями (MessageArea.class);
- элемент интерфейса, в котором должен отображаться элемент списка — сообщение чата (area_chat).

Затем напишем метод **populateView()**, в котором методы из класса **MessageArea** «соединятся» с элементами интерфейса, а именно с текстовыми элементами, составляющими сообщение чата:

```
47  private void showChat() {
48      adapter = new FirebaseListAdapter<MessageArea>(this, MessageArea.class, R.layout.area_chat,
49                                                     FirebaseDatabase.getInstance().getReference()){
50          @Override
51          protected void populateView(View v, MessageArea model, int position) {
52              TextView messageText = (TextView)v.findViewById(R.id.messageText);
53              TextView messageUser = (TextView)v.findViewById(R.id.messageUser);
54              TextView messageTime = (TextView)v.findViewById(R.id.messageTime);
55
56              messageText.setText(model.getMessageText());
57              messageUser.setText(model.getMessageUser());
58              messageTime.setText(DateFormat.format("HH:mm", model.getMessageTime()));
59          }
60      };
61      chatView.setAdapter(adapter);
```

В строке № 57 задается формат даты. Можно отображать полную дату — с годом, месяцем, числом и так далее, но в мессенджерах обычно отображаются только часы и минуты.

Шаг 9. Прописать обработку авторизации в приложении.

Мы уже реализовывали авторизацию в приложении с помощью Firebase, но есть более простой способ — подключить стан-

дартную авторизацию. Впоследствии мы это сможем изменить, но сейчас, для примера, чем проще, тем лучше.

Создадим метод **onActivityResult()**, передавая ему на вход параметры requestCode (код запроса), resultCode (код результата) и data (сопровождающие данные). В этом методе реализуем проверку: если код запроса совпадает с кодом авторизации и если результат запроса положительный — авторизация прошла успешно, можно вывести соответствующее всплывающее сообщение, иначе — авторизация провалена, о чем нужно сообщить пользователю:

```
63  @Override
64  protected void onActivityResult(int requestCode, int resultCode, Intent data) {
65      super.onActivityResult(requestCode, resultCode, data);
66      if (requestCode == SIGN_IN_REQUEST_CODE){
67          if (resultCode == RESULT_OK) {
68              Toast.makeText(this, "Вход выполнен", Toast.LENGTH_SHORT).show();
69              showChat();
70          } else {
71              Toast.makeText(this, "Авторизация провалена", Toast.LENGTH_SHORT).show();
72              finish();
73          }
74      }
75  }
```

Шаг 10. Реализовать отправку сообщения.

Сообщение должно отправляться по нажатию на кнопку, поэтому добавим метод **onBtnSendClick()**. В нем пропишем отправку сообщения, точнее сохранение его в базу данных Firebase для дальнейшего отображения. На данном этапе мы отправляем в БД текст сообщения и e-mail отправившего его пользователя. Впоследствии набор этих параметров можно будет изменить.

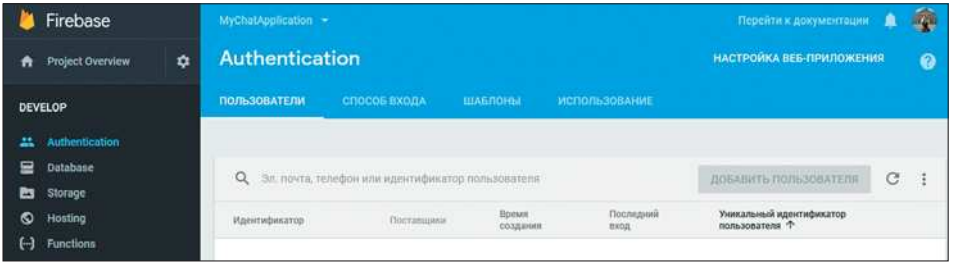
```
77  public void onBtnSendClick(View view) {
78      FirebaseDatabase.getInstance()
79              .getReference()
80              .push()
81              .setValue(new MessageArea(inputMessage.getText().toString(),
82                      FirebaseAuth.getInstance().getCurrentUser().getEmail()));
83      inputMessage.setText("");
84  }
```

После отправки сообщения текст в поле для ввода должен сбрасываться (строка № 83).

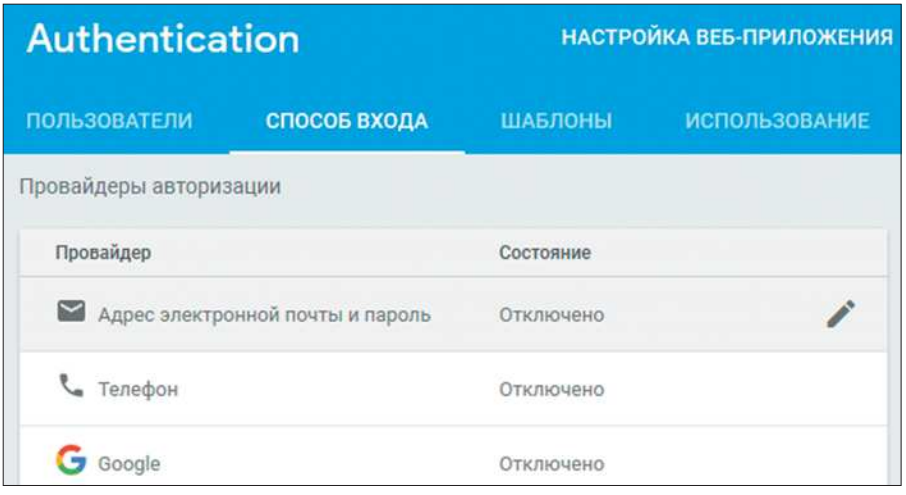
Шаг 11. Включить авторизацию через Консоль Firebase.

Если сейчас попытаться собрать и запустить приложение — будет выдана ошибка, потому что в Консоли Firebase по умолчанию не включен ни один из способов авторизации.

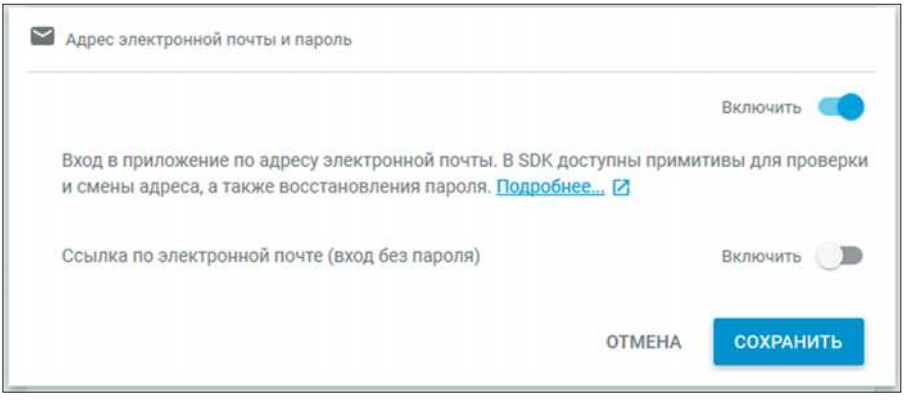
Перейдем в Консоль, на вкладку **Authentication**:



Здесь перейдем на вкладку **Способ входа**, где выберем первый способ авторизации (по адресу электронной почты и паролю) и перейдем к его редактированию.



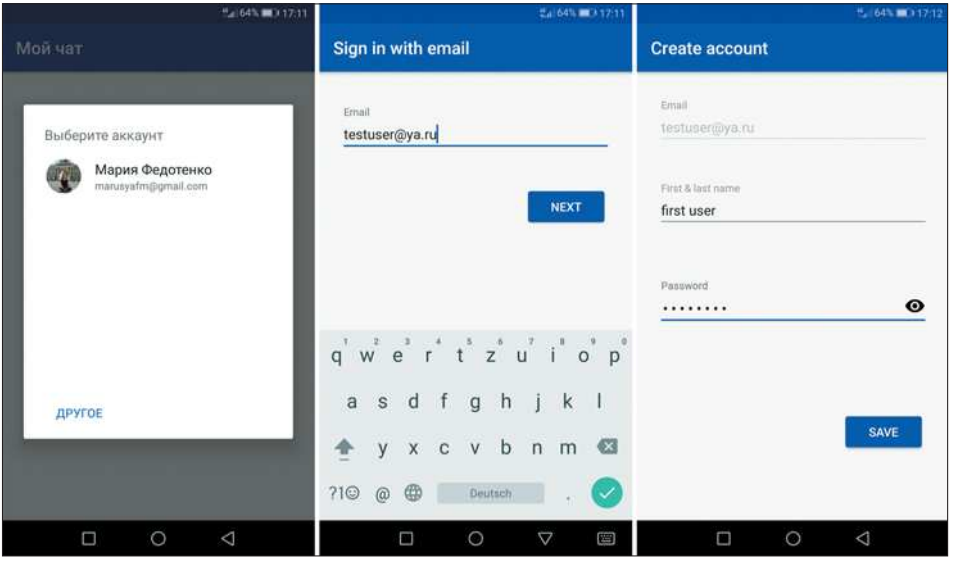
В открывшемся окне включим выбранный способ авторизации и сохраним изменения:



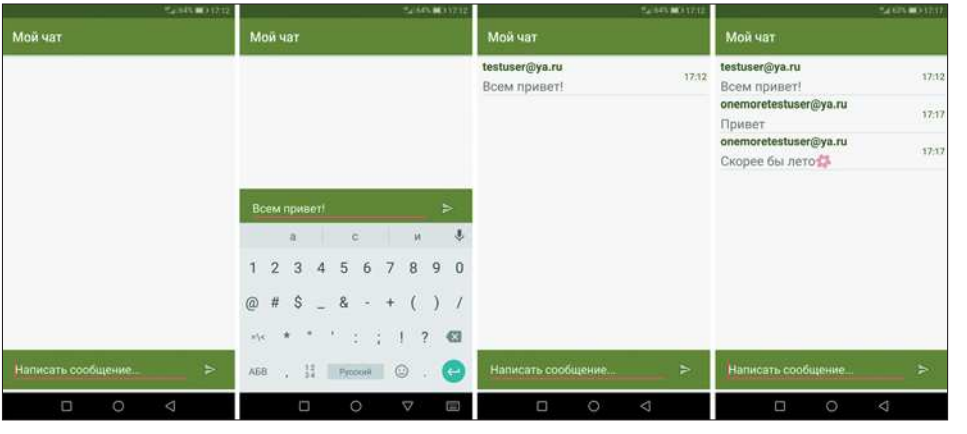
Все готово! Теперь наше приложение будет работать.

Шаг 12. Собрать проект и запустить приложение.

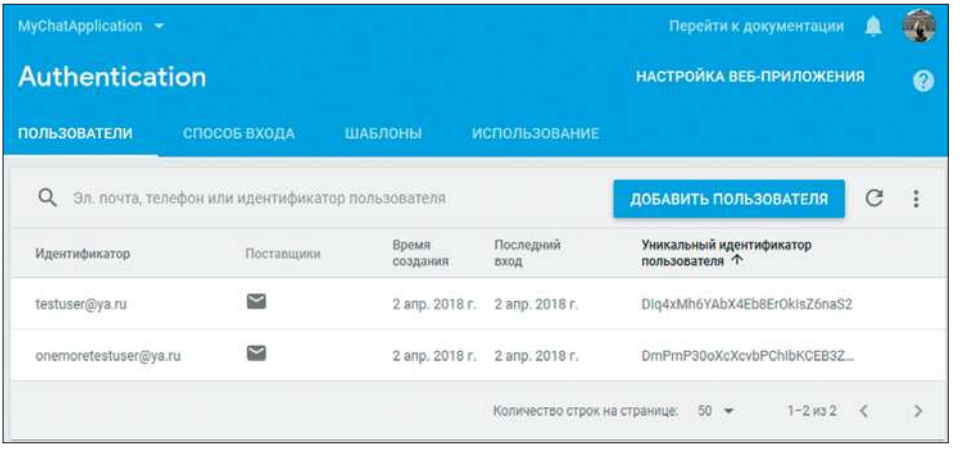
При запуске приложения в первую очередь откроется форма авторизации (стандартная, предоставленная Google). Здесь нужно создать нового пользователя — ввести для него адрес электронной почты, имя и пароль.



После успешного завершения авторизации откроется главный экран нашего чата. Теперь можно зарегистрировать еще нескольких пользователей и начинать общаться — чат будет доступен всем зарегистрированным аккаунтам.



Просматривать зарегистрированных пользователей и управлять ими можно через Консоль Firebase:



Итоги главы 8

Поскольку это финальный проект, он рассчитан на полет фантазии. Поэтому вместо заданий и выводов рассмотрим **возможности**.

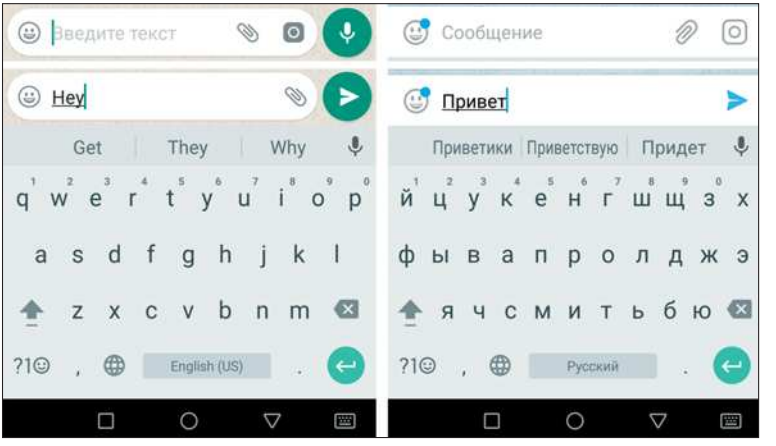
Итак, для данного приложения мы уже вполне **можем:**

1. Реализовать собственную авторизацию.
   Мы уже умеем это делать, и помимо оригинального дизайна, используя собственную форму авторизации, можем запросить необходимые данные пользователя по своему усмотрению, чтобы потом отображать их в приложении.
2. Реализовать выход из приложения (и аккаунта).
   Для этого можно создать меню любого рассмотренного нами вида, добавить в него пункт **Выход** и в обработчике события его нажатия прописать выход из аккаунта, а при необходимости — из приложения (метод **finish()**):

```
AuthUI.getInstance().signOut(this)
  .addOnCompleteListener(new OnCompleteListener<Void>() {
      @Override
      public void onComplete(@NonNull Task<Void> task) {
      Toast.makeText(MainActivity.this, “Выход выполнен”,
Toast.LENGTH_SHORT).show();
          finish();
      }
  });
```

3. Изменить дизайн по своему усмотрению.
   Мы уже знаем все основы проектирования интерфейсов и поэтому можем как воспроизвести дизайн известных приложений, так и создать свой, совершенно новый.

4. В большинстве мессенджеров кнопка **Отправить** появляется только тогда, когда пользователь начинает набирать сообщение:



Мы уже вполне можем реализовать проверку на пустоту поля: когда поле пустое, сделаем кнопку невидимой, а при появлении в нем символов — отобразим.

5. Реализовать различные способы оповещения пользователей.

Нам уже не составит труда отображать уведомление при появлении нового сообщения в чате или открывать диалоговое окно при попытке пользователя выйти из приложения.

6. Реализовать проверку подключения к Интернету при запуске приложения и, например, при отправке сообщения. Пользователь должен знать, что происходит с приложением.
7. Добавить профиль пользователя с информацией о нем и список пользователей, которым можно отправлять сообщения.

На сайте Firebase есть некоторые полезные рекомендации по дальнейшей работе с приложением для общения: как сделать диалог с конкретным пользователем, добавить тему беседы и так далее[1].

[1] Обратите внимание, что данные, передаваемые нами из приложения в **Firebase**, будут храниться не на российских серверах. Это не имеет особого значения сейчас — при обучении и разработке небольшого приложения для тренировки навыков. Но в дальнейшем при разработке серьезного мессенджера для большой аудитории это нужно будет учесть.

Это только примерный (и далеко не полный) список наших уже имеющихся возможностей для доработки получившегося приложения! Сделайте его удобным, запоминающимся своим неповторимым стилем и вызывающим желание вернуться.

Заключение

Поздравляем! Теперь вы смело можете называть себя **Android-разработчиками**!

И хотя вам еще многому предстоит научиться, полученных знаний уже вполне хватит, чтобы хотя бы частично воспроизвести практически любое приложение, которое приходит на ум.

Не останавливайтесь! Реализуйте любые идеи, даже самые смелые, и, возможно, именно ваши приложения будут знать и любить миллионы пользователей.

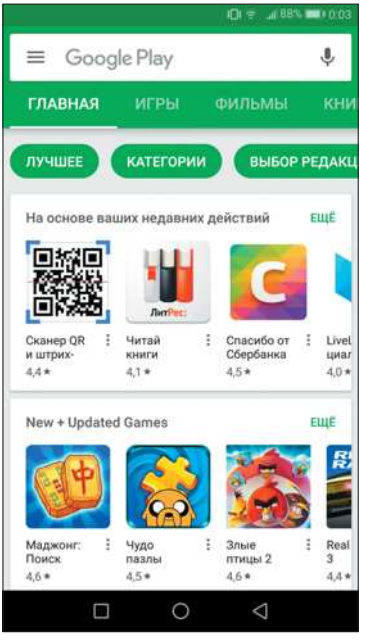
Приложения

Магазины приложений

Мы уже научились создавать полноценные работающие приложения и собирать их в файлы APK, а значит, можем делиться ими со своими знакомыми и всячески распространять их.

Но для того чтобы наши приложения мог оценить весь мир, их нужно загрузить в **магазин приложений**.

Google Play



Первое, что приходит на ум при упоминании магазина приложений (для Android), — Google Play.

Это официальная платформа для распространения мобильных приложений от Google, поэтому в первую очередь все Android-разработчики ориентируются именно на Google Play. На сегодняшний день на Google Play выложено более 1,5 млн приложений.

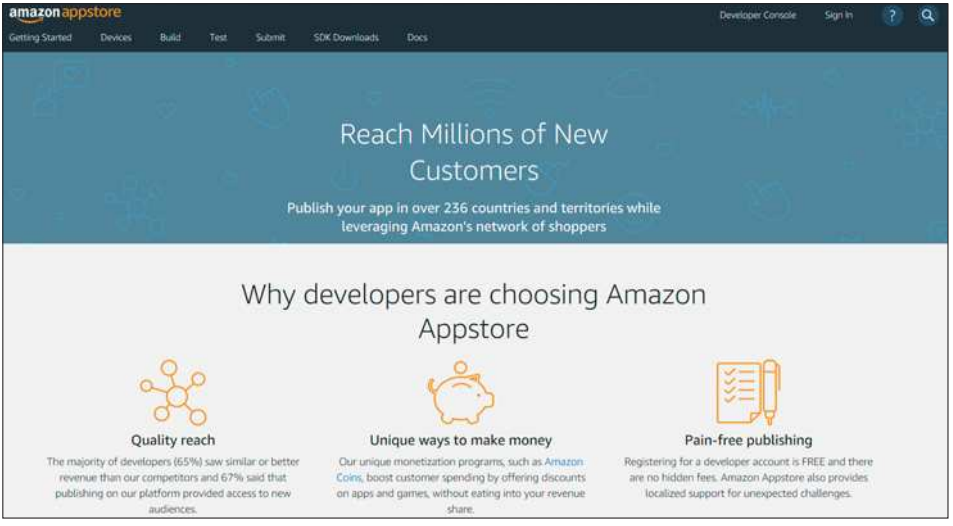
Но, как и все, что связано с компанией Google, взаимодействие Google Play с разработчиками тщательно регламентировано и охраняется законодательством США. Для того чтобы выложить свои разработки в этом магазине, необходимо пройти серьезную процедуру регистрации. Нужно создать аккаунт разработчика, внести регистрационный взнос (около $ 25 США), ввести большое количество персональных данных и платежных реквизитов (для дальнейшего перевода средств, полученных от продажи приложений через магазин). Поэтому лучше поручить это человеку, достигшему совершеннолетия.

После создания аккаунта разработчика можно выкладывать приложение. Для этого понадобится еще один сервис от Google — Google Play Console. Загружается файл apk, заполняется краткое и полное описание приложения, добавляются скриншоты и логотип, задаются основные свойства приложения (тип, категория, возрастные ограничения), указывается контактная информация разработчика. Затем выбирается тип распространения (платное или бесплатное).

Через несколько часов, в течение которых Google выполняет всевозможные проверки, наше приложение будет доступно в поиске.

AmazonAppStore

Один из самых популярных аналогов Google Play — **Amazon AppStore** от компании Amazon:

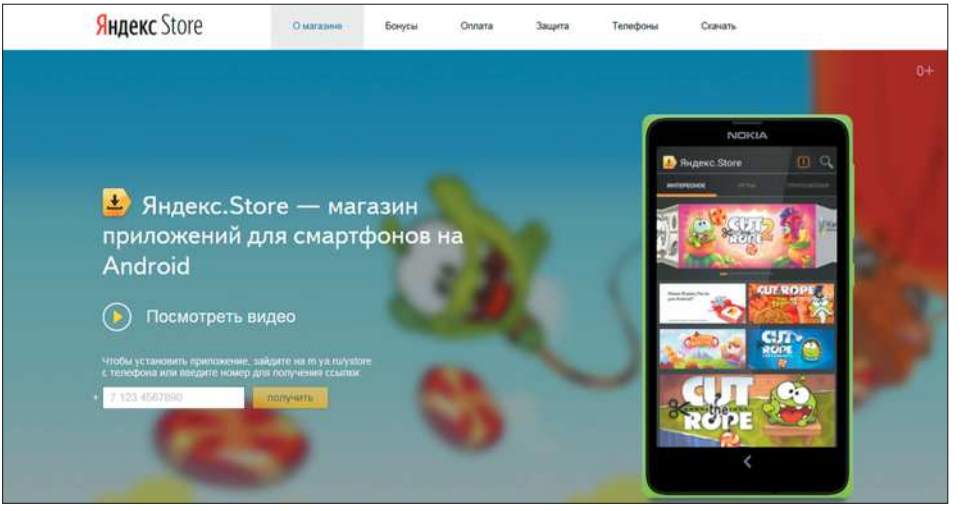


Более полумиллиона приложений, отсутствие регистрационного сбора при создании аккаунта разработчика, грамотная система продвижения и постоянные акции для привлечения новых пользователей — вот что такое Amazon AppStore.

Однако если на Google Play можно выложить практически любое приложение (проверки здесь в большей степени направлены на соблюдение законодательства и «исполнение» приложения, чем на его содержание), то перед попаданием в Amazon AppStore приложения проходят очень строгие проверки «по всем направлениям». Это гарантирует пользователям получение только действительно качественного продукта.

Яндекс.Store

Этот российский аналог Google Play от компании Яндекс на сегодняшний день насчитывает уже более сотни тысяч приложений и ориентирован в первую очередь на Россию и русскоязычных пользователей.

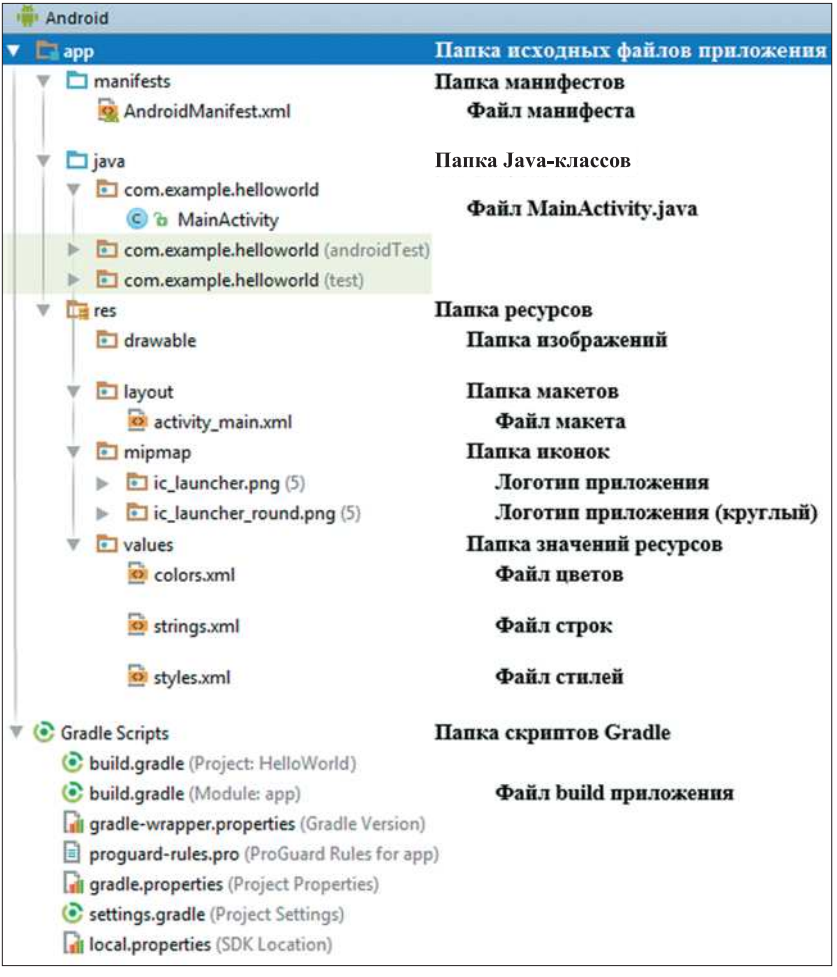


Интерфейс сервиса по умолчанию полностью на русском языке, загружаемые приложения проверяются российскими специалистами и антивирусом. Создание аккаунта разработчика и распространение приложений — бесплатно.

И это далеко не полный список — только самые популярные магазины. Утверждать можно одно — на сегодняшний день существует множество различных путей распространения мобильных приложений, и какой из них выбрать — решать вам.

Девять шагов к идеальному приложению

Спортивное ориентирование по проекту Android Studio



Структура проекта Android Studio

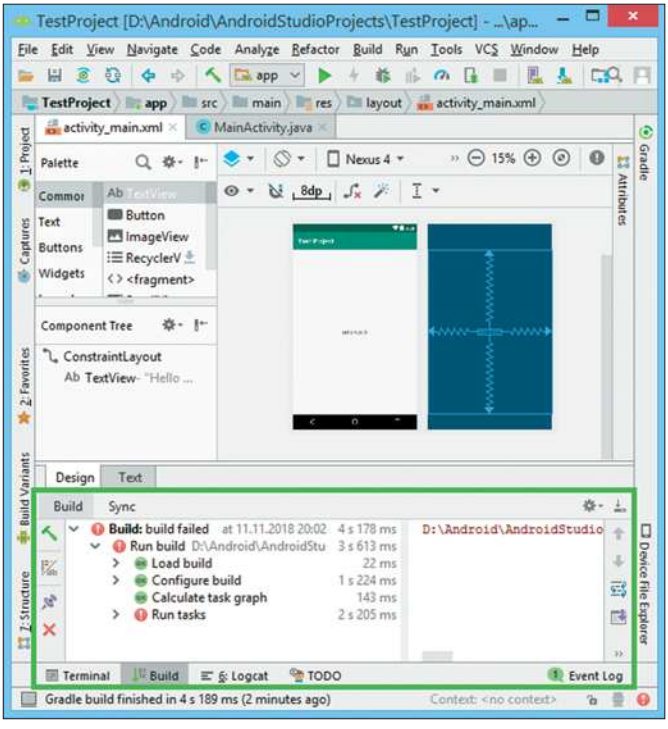
Папка/файл	Название	Что внутри?
app	**Папка исходных файлов приложения**	Все файлы, относящиеся к приложению
manifests	**Папка манифестов**	Файлы манифестов, содержащие основную настроечную информацию о приложении
AndroidManifest	**Файл манифеста**	В файле манифеста можно изменить имя приложения, версию, логотип, основную тему, структуру, назначить стартовый экран и т. д.

Окончание табл.

Папка/файл	Название	Что внутри?
java	**Папка классов Java**	Все классы Java, относящиеся к работе приложения
MainActivity	**Файл MainActivity.java**	Файл Java-класса главной Activity приложения. В нем описывается все, что должно происходить при загрузке Activity, а также при нажатии на ее элементы и т. д.
res	**Папка ресурсов**	Все ресурсы приложения: изображения, разметка интерфейса, цвета, строковые ресурсы, стили
drawable	**Папка изображений**	Все картинки, которые планируется использовать в интерфейсе приложения
layout	**Папка макетов**	Разметка интерфейса всех экранов приложения, описанная на языке XML
activity_main.xml	**Файл макета**	Описание интерфейса главной Activity на языке разметки XML
mipmap	**Папка иконок**	Логотип приложения и его круглая вариация
ic launcher	**Папка логотипа приложения**	Логотип приложения во всех необходимых системе Android разрешениях
ic_launcher_round	**Папка круглого логотипа приложения**	Копия логотипа приложения в круглой вариации
values	**Папка значений ресурсов**	Описания ресурсов приложения (цветов, строк, стилей) на языке XML для более удобного доступа к ним
colors.xml	**Файл цветов**	В файле можно изменять основные цвета приложения и добавлять новые
strings.xmll	**Файл строк**	В файле можно сохранять длинные текстовые содержания элементов и различные заголовки
styles.xml	**Файл стилей**	В файле можно изменять установленные для компонентов приложения стили, а также прописывать новые стили
Gradle Scripts	**Папка скриптов Gradle**	Описанные на языке Groovy скрипты системы сборки Gradle, с помощью которой можно собирать и запускать наши приложения
build.gradle	**Файл build приложения**	В файле можно изменить минимальную и требуемую версии SDK для приложения, а также прописать поддержку необходимых библиотек и сервисов

Исправление типичных ошибок

При работе в **Android Studio** мы как программисты иногда допускаем ошибки. Уследить за всеми компонентами проекта довольно сложно, поэтому случается не заметить ошибку как в Java-коде, так и в xml-разметке, файлах ресурсов или скриптах gradle. Но при сборке проекта **Android Studio** все это анализирует и, если находит ошибку, сообщает о ней в окне сборки **Build**.



Окно **Build** разделено на две части. В **левой части** показывается, на каком этапе сборки была обнаружена ошибка, а в **правой** — выводится информация об этой ошибке: в каком файле проекта и на какой строке она найдена, в чем заключается и что можно сделать для ее исправления.

Но ошибаться можем не только мы, но и сама **Android Studio**. В связи с этим в данном приложении собраны некоторые рекомендации, как работать с возникающими ошибками и как их избегать.

1. Проверять подключение к Интернету.

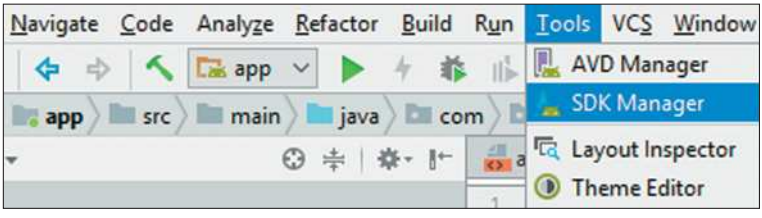
Создавать приложения можно и без доступа к Интернету. Однако при открытии проекта **Android Studio** выполняет его сборку и, если обнаруживает недостающие компоненты, пытается сразу

же загрузить их. Подключение к Интернету требуется также при внесении изменений в скрипты gradle, например при подключении новых библиотек.

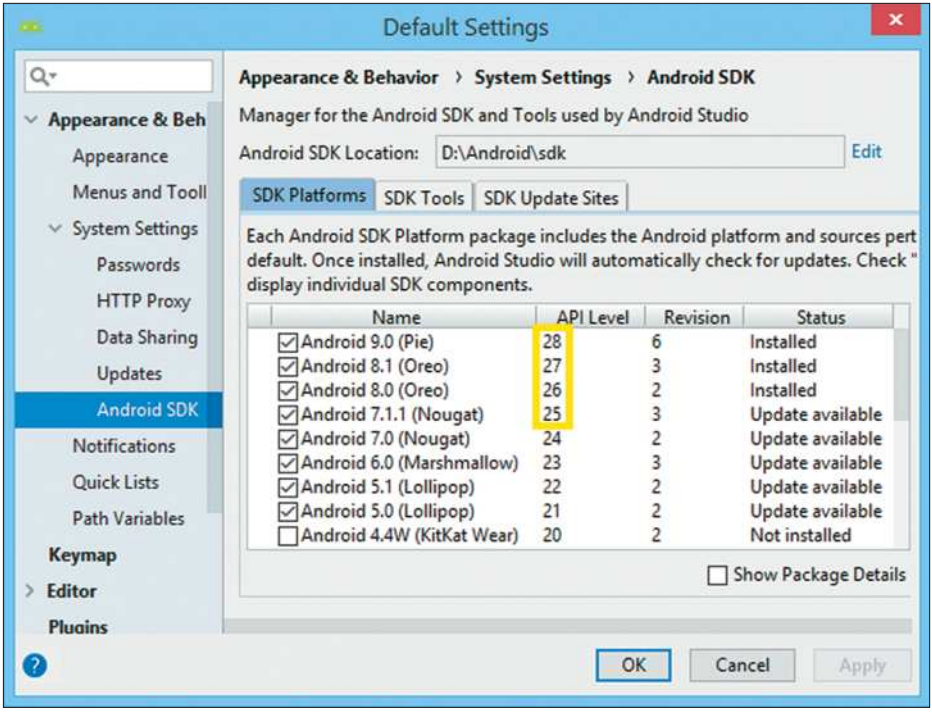
Поэтому, начиная работу с **Android Studio**, желательно иметь активное подключение к Интернету.

2. Проверять подключение всех необходимых модулей в модуле SDK Manager.

Это лучше всего сделать сразу после установки **Android Studio**. Запустим SDK Manager:

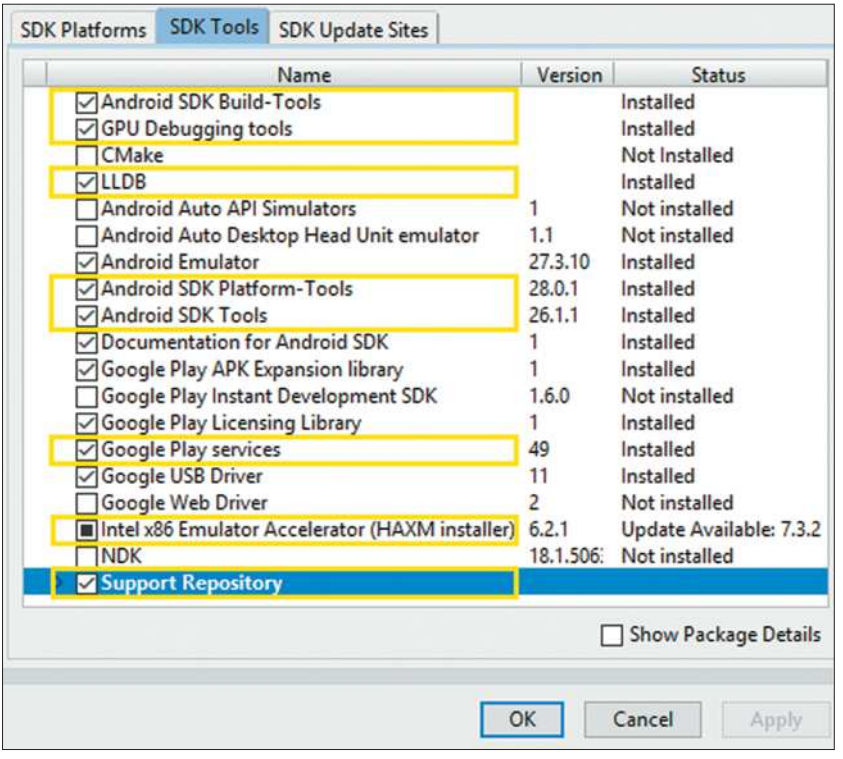


Первой откроется вкладка **SDK Platforms**. Для предотвращения возможных ошибок, связанных с конфликтом версий, лучше сразу установить платформы с уровнем API **(API Level) 25** и выше:



После выбора нужных платформ следует нажать кнопку **Apply** и дождаться окончания установки.

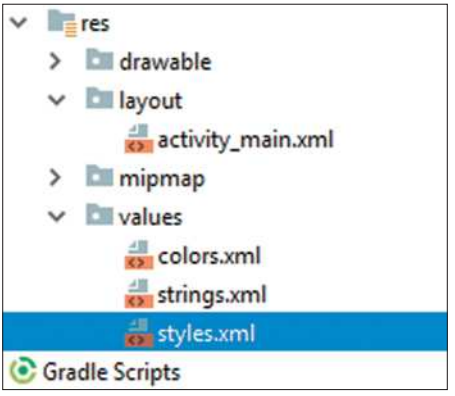
Теперь перейдем на вторую вкладку — **SDK Tools**. Здесь нужно установить инструменты, выделенные на рисунке желтым:



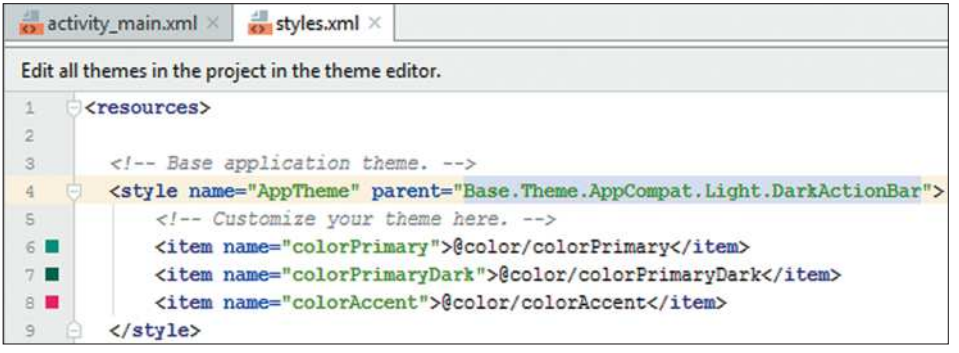
Для установки нужно снова поставить все необходимые галочки и нажать кнопку **Apply**.

3. Ошибка: не отображается разметка интерфейса в режиме предварительного просмотра и макета:

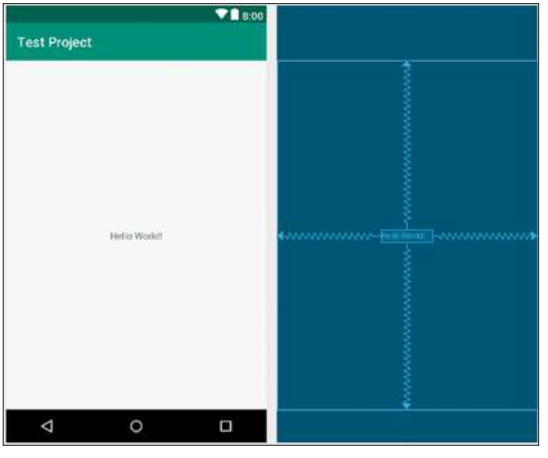
Эта ошибка может быть вызвана конфликтом стилей. Для ее исправления откроем файл стилей **styles.xml:**



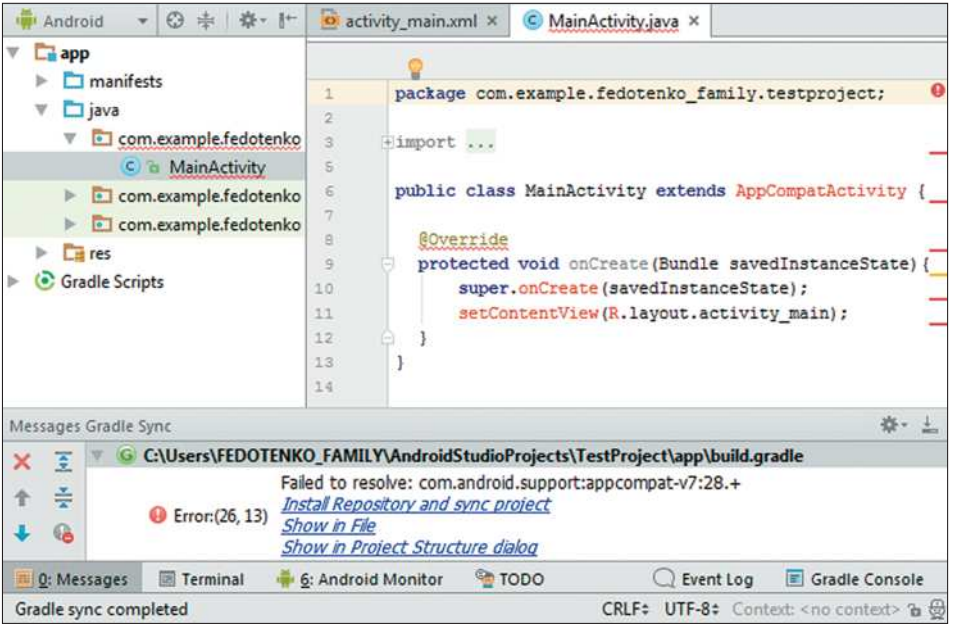
В файле найдем главную тему и изменим ее атрибут **Parent** (*родительск я тем*). Можно прописать любую тему, а можно добавить к текущей префикс Base:



В результате интерфейс приложения отображается как надо.

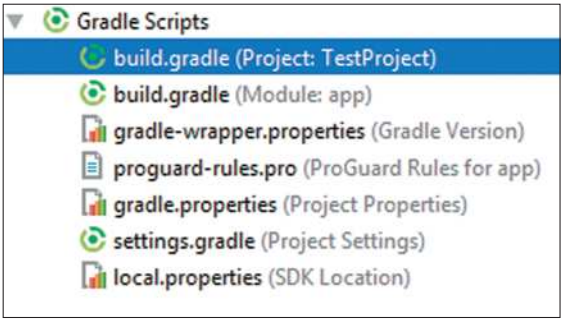


4. Ошибка: 26,13, а также 25,13, 27,13 или 28,13.



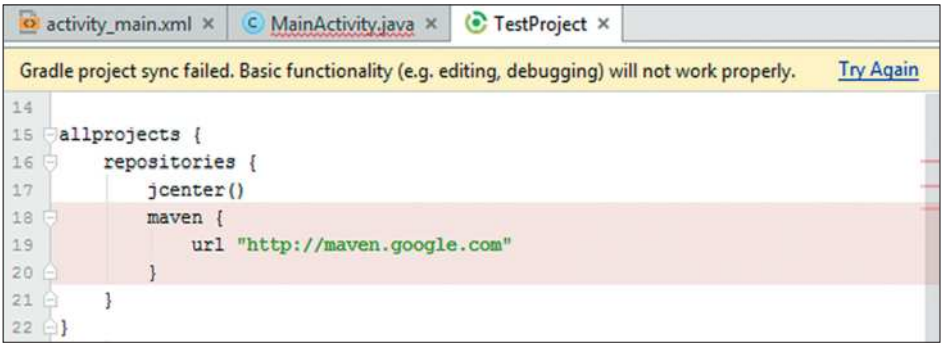
Эта ошибка обычно обусловлена нехваткой некоторых библиотек. Для ее устранения можно нажать на ссылку **Install Repository and sync project**, но она не всегда работает. Можно начать искать, какой именно библиотеки не хватает, и установить ее вручную. Но есть более быстрый способ.

Откроем файл **build.gradle:**



В нем после строки № 17 пропишем следующий код:

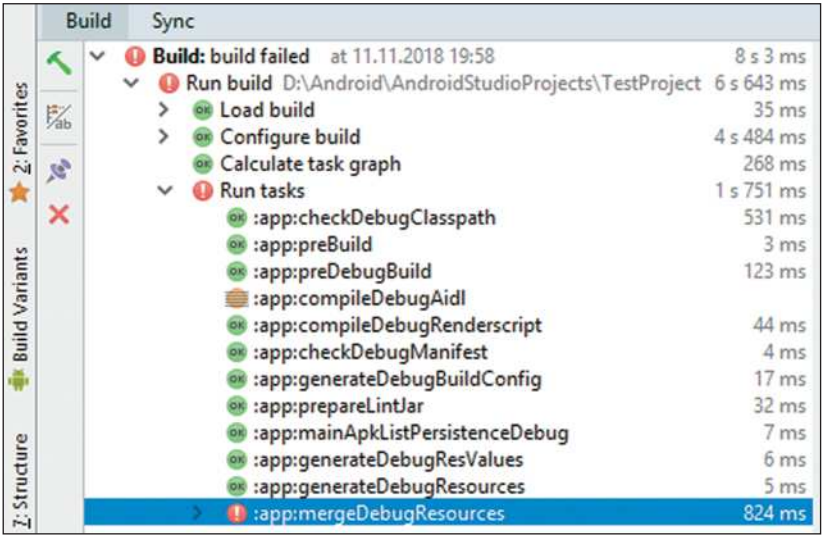
```
maven {
  url "http://maven.google.com"
}
```



Этот код указывает **Android Studio** скачать недостающие библиотеки с официального сайта.

Далее следует обязательно нажать на ссылку **Try Again** (*попробовать снова*), чтобы проект «собрался» заново и продолжил работу уже без ошибок.

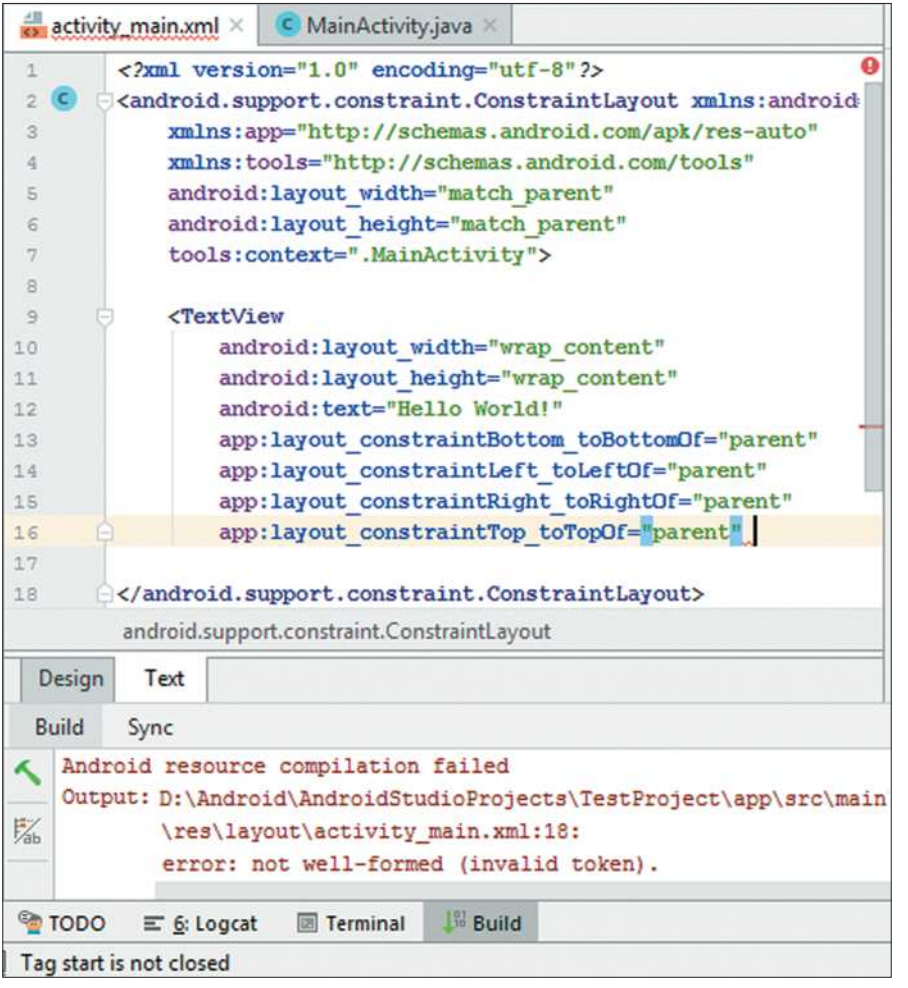
5. Ошибка в :appMergeDebugResources.



Эта ошибка появляется, когда что-то не так с изображениями, которые мы добавили в проект. Чтобы ее устранить, откроем папку проекта **drawable** и внимательно рассмотрим имена всех изображений. Имя изображения должно **начинаться с маленькой латинской буквы, не содержать букв кириллицы**, а само изображение должно иметь расширение **.png**.

Как мы видим, имя изображения IMG_20180607_144601.jpg не соответствует сразу двум пунктам этих требований, поэтому и возникла ошибка.

6. Все остальные ошибки Android Studio подробно описывает. Например, в приведенном ниже случае **Android Studio** пишет, что ошибка в файле **activity_main.xml** в строке № 18 (см. пункт **Output:**), а чуть ниже написано **Tag start is not closed** (*открывающий тег не закрыт*).



Открываем файл **activity_main.xml**, находим строку № 18 и видим, что тег <TextView действительно не закрыт. Добавляем закрывающий тег />, и ошибка устранена.

Теперь, даже если ошибки будут периодически возникать, они не будут пугать нас или сбивать с толку.

Успехов!

Глоссарий

Здесь собраны основные термины и названия, встречающиеся в книге. Обращайтесь к этому разделу, чтобы освежить их в памяти.

Android Studio — официальная среда разработки мобильных приложений для устройств с операционной системой Android от компании Google. В ней мы и разрабатываем наши мобильные приложения.

ActionBar (*п нель действий*) — «шапка» приложения. В ней отображается имя приложения и кнопки вызова меню.

Activity (*ктивн я обл сть экр н*) — каждый отдельно взятый экран приложения.

activity_main.xml — файл проекта, в котором на языке разметки XML описан интерфейс главного экрана наших приложений: элементы интерфейса, их расположение и свойства.

activity_main_drawer.xml — файл проекта, в котором на языке XML описана структура меню-шторки: названия, порядок и свойства его пунктов.

Alert Dialog (*ди логовое окно*) — один из способов оповещения пользователей — всплывающее окно, в котором запрашивается у пользователя ответ на некоторый вопрос. Обычно содержит вопрос и кнопки с предлагаемыми вариантами ответа.

Alt+Enter — «волшебное» сочетание клавиш, которое помогает понять и исправить ошибки, допущенные нами при наборе кода.

Android SDK (Software Development Kit) — набор инструментов для разработки программного обеспечения для операционной системы Android. Включает в себя эмуляторы устройств, документацию и все необходимые для разработки пакеты.

Android Studio проект — все файлы приложения, необходимые для его разработки. Хранятся в служебных папках, имеющих сложную структуру, и распознаются **Android Studio** как единый проект.

AndroidManifest.xml (файл манифеста) — файл проекта, в котором на языке разметки XML описана основная информация о приложении, необходимая для его распознавания операционной системой смартфона или магазином приложений (название приложения, иконка, разрешения, основные компоненты приложения).

AnimationUtils (*средств ним ции*) — Java-класс, в котором описаны все методы реализации анимации элементов интерфейса.

APK (Android Package Kit) — исполняемый (установочный) файл, который используется для передачи, установки и распространения наших приложений.

app (*приложение*) — папка проекта, в которой хранятся все файлы приложения.

assets (*ктивы*) — папка проекта для хранения файлов, не входящих в основной список ресурсов (файлы БД, дополнительные скрипты и так далее).

AVD Manager (Android Virtual Device Manager) — менеджер виртуальных Android-устройств.

Bottom Navigation Activity (*экр н с нижней н виг цией*) — шаблон интерфейса, в котором переключение между фрагментами осуществляется с помощью нижней панели навигации.

builder (*построитель, конструктор, сборщик*) — объект Java, в котором задаются основные параметры уведомлений, необходимые для их «построения» и отображения.

Canvas (*холст*) — Java-класс, в котором прописаны основные методы, необходимые для реализации возможности рисования пальцем на экране.

commit() (*з фиксиров ть*) — метод Java, который вызывается, чтобы внесенные изменения вступили в силу.

Component Tree (*дерево компонентов*) — панель Android Studio, на которой отображаются названия и идентификаторы всех элементов, расположенных в данный момент на экране.

ConnectivityManager (*менеджер подключений*) — Java-класс, в котором прописаны основные методы для получения информации о текущих подключениях.

ConstraintLayout (*м кет привязок*) — интерфейс (макет), внутри которого элементы располагаются с привязкой к расположению других элементов или родительского элемента.

create() (*созд ть*) — метод Java, который вызывается, чтобы создать объект (например, уведомление) со всеми параметрами, прописанными в построителе.

Design и **Text** (*диз йн и код*) — два режима работы с файлами xml-разметки; вкладки для переключения между этими режимами.

dp — пиксели, независимые от плотности. Абстрактная единица измерения, используется для того, чтобы сделать интерфейс приложения адаптивным и чтобы его элементы выглядели одинаково на всех устройствах (экранах с разными разрешениями).

drawable (*рисунки*) — папка проекта, в которой хранятся все изображения, используемые в данном проекте.

EditText (*ред ктируемый текст*) — элемент интерфейса, в который можно вводить текст.

Empty Activity (*пустой экр н*) — шаблон интерфейса, в котором нет предустановленных элементов интерфейса и прописанных по умолчанию методов.

findViewById() (*н йти элемент по идентифик тору*) — метод Java, который вызывается для поиска элемента интерфейса по его уникальному идентификатору.

finish() (*з вершить*) — метод Java, который вызывается для завершения работы текущей Activity или всего приложения.

Firebase — мобильная платформа, которая при интеграции с приложением дает разработчику возможность хранить данные в базе данных, синхронизируемой в реальном времени, подключать приложение к системам аналитики, организовывать обмен сообщениями между пользователями, авторизацию в приложении, рассылку уведомлений и многие другие функции.

float — тип данных Java для работы с дробными числами (числами с плавающей точкой).

FloatingActionButton (*пл в ющ я кнопк действия*) — элемент интерфейса, плавающая кнопка в правом нижнем углу экрана, предназначенная для того, чтобы самые важные функции приложения всегда находились «под рукой» у пользователя.

Fragment (*фр гмент*) — Java-класс, в котором описаны основные методы работы с фрагментами (элементами-контейнерами, которые можно переключать внутри одной Activity).

FragmentManager (менеджер фрагментов) — Java-класс, в котором прописаны методы управления и переключения фрагментов.

FrameLayout (*р мочный м кет*) — элемент интерфейса (макет), внутри которого элементы располагаются поверх других элементов.

GestureDetector (*детектор жестов*) — Java-класс, в котором описаны основные методы обработки касаний и жестов.

getActiveNetworkInfo() (*получить информ цию об ктивной сети*) — метод Java, который вызывается для получения информации о текущем сетевом соединении.

getId() (*получить идентифик тор*) — метод Java, который вызывается для получения идентификатора заданного элемента.

getText() (*получить текст*) — метод Java, который вызывается для «извлечения» текстового содержания элемента интерфейса, прописанного в свойстве **text** элемента.

Google Maps Activity (*экр н для р боты с к рт ми Google*) — шаблон интерфейса для работы с картами Google.

Google Play — официальный магазин приложений от Google.

Gradle — система автоматической сборки. **Android Studio** выполняет сборку приложения с помощью системы Gradle в фоновом режиме, выполняя Gradle-скрипты без необходимости нашего вмешательства как разработчиков.

Gradle Scripts (*скрипты Gradle*) — папка проекта, в которой хранятся написанные на языке Groovy скрипты системы автоматической сборки Gradle.

GridLayout (*м кет-сетк*) — элемент интерфейса (макет), внутри которого элементы располагаются в прямоугольной сетке, похожей на таблицу.

Handler (*обр ботчик*) — Java-класс, в котором прописаны основные методы для создания и управления фоновыми потоками, выполняющимися в любое заданное время.

ID (*идентифик тор*) — уникальный идентификатор элемента для последующего обращения к этому элементу. Идентификатор должен напоминать о типе и содержании элемента.

Image Asset Studio — встроенный инструмент Android Studio, который предназначен для простого и быстрого создания различных значков: для логотипа приложения, для пунктов меню, кнопок и так далее.

ImageButton (*изобр жение-кнопк*) — элемент интерфейса, представляющий собой «кликабельную» картинку (картинку, на которую можно нажать, как на кнопку).

ImageView (*элемент-изобр жение*) — элемент интерфейса, предназначенный для отображения картинки.

initiateScan() (*инициров ть ск ниров ние*) — метод Java, который вызывается для запуска встроенного сканера и начала сканирования.

inputType (*тип ввод*) — свойство элементов с редактируемым текстом, тип вызываемой клавиатуры. Этот атрибут отвечает за то, какая клавиатура будет отображаться при заполнении элемента (цифровая, текстовая и так далее).

int — тип данных Java для работы с целыми числами.

Intent (*н мерение*) — Java-класс, объекты которого представляют собой абстрактное описание одной выполняемой операции. С помощью этой операции можно запросить выполнение некоторого действия у другой Activity или даже стороннего приложения.

isChecked() (*пост влен ли г лочк*) — метод Java, вызываемый для того, чтобы определить, поставлена ли выбранная галочка (чекбокс) или выбран ли пункт меню.

Java — объектно-ориентированный язык программирования. На нем мы пишем код, чтобы интерфейс приложения стал «живым».

java — папка проекта, в которой хранится исходный программный код нашего приложения — все Java-классы, необходимые для разработки и тестирования.

Landscape variation (*горизонт льный в ри нт*) — копия Activity, создаваемая для горизонтальной ориентации экрана.

Layout (*м кет*) — элемент интерфейса для компоновки элементов внутри него.

layout (*м кет*) — папка проекта, в которой хранится вся xml-разметка интерфейса.

Layout resource file (*ф йл ресурсов м кет*) — новый файл xml-разметки, создаваемый для описания новых элементов интерфейса.

LinearLayout (horizontal) (*горизонт льный линейный м кет*) — элемент интерфейса (макет), внутри которого элементы располагаются друг за другом по горизонтали.

LinearLayout (vertical) (*вертик льный линейный м кет*) — элемент интерфейса (макет), внутри которого элементы располагаются друг за другом по вертикали.

ListView (*элемент-список*) — элемент интерфейса, предназначенный для хранения и отображения списков.

loadUrl() (*з грузить ссылку*) — метод Java, который вызывается для загрузки содержимого некоторой ссылки (обычно web-страницы).

Login Activity (*экр н вториз ции*) — шаблон интерфейса для создания страницы авторизации в приложении.

main.xml — файл проекта, в котором на языке разметки XML хранится описание главного меню.

MainActivity (*гл вный экр н*) — Java-класс, в котором описано все, что относится к главному экрану приложения.

makeText() (*«сдел ть», то есть з д ть текст*) — метод Java, который вызывается, чтобы заполнить всплывающее сообщение текстом.

manifests (*м нифесты*) — папка проекта, в которой хранятся файлы манифеста.

Navigation Drawer Activity (*экр н–построитель н виг ции*) — шаблон интерфейса для создания экрана с тремя видами меню — главным, меню-шторкой и контекстным.

Notification (*уведомление*) — сообщение, которое выводится за пределами самого приложения для привлечения внимания пользователя (сначала в виде значков в области уведомлений, а затем полная версия на панели уведомлений).

onBackPressed() (*при н ж тии н кнопку **Назад***) — метод Java, который вызывается для активации возможности нажатия на системную кнопку **Назад**.

onClick() (*при н ж тии*) — метод Java, в котором описывается обработка нажатия на кнопку (или любой другой элемент интерфейса).

onCreate() (*при созд нии*) — метод Java, в котором описывается все, что должно происходить при переходе к данному экрану.

onDoubleTap() (*по двойному н ж тию*) — метод Java, в котором прописываются методы обработки двойного нажатия.

OnDoubleTapListener (*прослушив тель двойного н ж тия*) — Java-класс, в котором прописаны основные методы распознавания двойного нажатия.

onFling() (*при «см хив нии»*) — метод Java, в котором прописываются методы обработки жеста «свайпинг».

OnGestureListener (*прослушив тель жестов*) — Java-класс, в котором прописаны основные методы распознавания жестов.

onLongPress() (*при долгом н ж тии*) — метод Java, в котором прописывается обработка долгого нажатия.

onNavigationItemSelected() (*по выбору пункт н виг ционного меню*) — метод Java, в котором прописывается обработка выбранного пункта меню.

onScroll() (*при прокручив нии*) — метод Java, в котором прописываются методы обработки жеста «скроллинг» (прокрутка).

onTouchEvent() (*по событию н ж тия*) — метод Java, в котором прописывается обработка событий, относящихся к сенсорному экрану (определение самого факта нажатия и его типа).

Palette (*п литр элементов*) — панель Android Studio. В ее левой части перечислены названия типов элементов (все элементы, виджеты, текстовые элементы, шаблоны, изображения), а в правой — все элементы выбранного типа. Для того чтобы добавить элемент на экран, достаточно перетащить его из палитры элементов.

Portrait Variation (*вертик льный в ри нт*) — копия Activity, создаваемая для вертикальной ориентации экрана.

ProgressBar (*индик тор з грузки*) — элемент интерфейса для отображения хода загрузки или выполнения какого-либо процесса. Бывает круговой и горизонтальный.

Properties (Attributes) (*п нель свойств*) — панель Android Studio, в которой отображаются все свойства выбранного элемента.

QR-код (*код быстрого ре гиров ния*) — двумерный (матричный) штрих-код, считав который с помощью специальных мобильных приложений, можно получить быстрый доступ к более подробной информации об объекте.

res (*ресурсы*) — папка проекта, в которой хранятся все картинки (папка **drawable**), файлы xml-разметки (папка **layout**), служебные иконки (папка **mipmap**) и описания всех заданных в проекте значений различных переменных (папка **values**).

Resources (*ресурсы*) — библиотека ресурсов. Если определить и прописать изображение или какой-нибудь текст как ресурс, в дальнейшем к нему можно будет неоднократно обращаться, указав его уникальный идентификатор id, а не переписывая текст каждый раз заново.

ScrollingActivity (*прокручив емый экр н*) — шаблон интерфейса для создания экрана с длинным прокручиваемым текстом.

ScrollView (*прокручив емый элемент*) — элемент интерфейса с возможностью прокрутки.

setBackgroundColor() (*уст новить цвет фон*) — метод Java, который вызывается для изменения цвета фона элемента.

setColor() (*уст новить цвет*) — метод Java, который вызывается для изменения цвета элемента.

setText() (*уст новить текст*) — метод Java, который вызывается для изменения текстового содержания элемента.

setTitle() (*уст новить н зв ние*) — метод Java, который вызывается для изменения названия элемента (уведомления, пункта меню).

setVisibility() (*уст новить видимость*) — метод Java, который вызывается для изменения типа видимости элемента (видимый, невидимый, скрытый).

show() (*пок з ть*) — метод Java, который вызывается для отображения уведомления после установки всех его параметров.

SoundPool (*звуковой пул, объединение*) — Java-класс, в котором описаны основные методы для реализации возможности загрузки, воспроизведения и работы со звуками в приложении.

sp — пиксели, независимые от масштабирования. Единицы измерения, предназначенные для работы с текстом, для наиболее корректного отображения шрифтов. Значение задается пользователем и остается неизменным для любого разрешения экрана.

Splash Screen (*з ст вк*) — загрузочный экран приложения, то есть экран-заставка, который демонстрируется пользователю во время загрузки приложения.

SQLiteHelper (*помощник для р боты с SQLite*) — Java-класс, в котором описаны основные методы для реализации работы с базами данных SQLite.

startActivity() (*з пустить Activity*) — метод Java, который вызывается для запуска заданной Activity.

String — тип данных языка Java для работы со строками (текстом).

Switch (*переключ тель*) — условный оператор языка Java, который в отличие от if и if-else применяется для известного списка значений, а также предусматривает ситуацию по умолчанию (default).

Switch (*переключ тель*) — элемент интерфейса, который имеет два режима — включен и выключен. Служит для переключения между этими параметрами.

Tabbed Activity (*экр н с вкл дк ми*) — шаблон интерфейса, в котором переключение между фрагментами осуществляется горизонтальным свайпом.

TaskStackBuilder (*построитель стек з д ч*) — Java-класс, в котором описаны методы для реализации своего рода списка, в который будут заноситься задачи по мере их выполнения.

TextView (*текстовый элемент*) — элемент интерфейса для отображения на экране текста.

this (*этот*) — ключевое слово, указатель на текущий объект данного класса в Java.

Toast (*уведомление о тосте*) — всплывающее сообщение, обеспечивающее простейшую обратную связь с пользователем.

toString() (*в строку*) — метод Java, который вызывается для «конвертации» извлеченного текста в строку.

uses-permission (*р зрешение н использов ние*) — тег манифеста, который запрашивает разрешение на использование служебных программ и сторонних приложений (камера, Интернет и так далее).

WebView (*элемент для отобр жения веб-содержимого*) — элемент интерфейса, с помощью которого можно встраивать в мобильные приложения веб-страницы или их части.

XML — расширяемый язык разметки, созданный для описания данных. В **Android Studio** на языке XML описываются элементы интерфейса (дизайн) приложения, а также некоторые ресурсы (цвета, строки, стили).

ZXing Library — библиотека, необходимая для считывания QR- и штрих-кодов.

Главное меню — меню, которое вызывается обычно из правого верхнего угла ActionBar. На сегодняшний день в него обычно выносят такие пункты, как «Настройки», «О приложении», «Оценить приложение» и так далее, то есть моменты, которые несут в себе информацию о приложении, но напрямую не относятся к его основному содержанию.

Исключение — возникшая во время выполнения программы нештатная ситуация (ошибка).

Контекстное меню — меню, которое может вызываться по нажатию на любой элемент приложения, как правило, долгим нажатием. Представляет собой аналог контекстного меню, которое мы вызываем нажатием правой кнопкой мыши при работе за компьютером.

Меню-шторка — основное меню приложения. Оно разворачивается на большую часть экрана и вызывается либо свайпом от края экрана, либо нажатием на кнопку с тремя полосками, которая обычно располагается в левом верхнем углу в ActionBar приложения.

Стиль (в разработке мобильных приложений) — набор настроек отдельно взятого элемента приложения, которые можно отнести к его дизайну (цвет, шрифт, цвет фона и так далее).

Тема (в разработке мобильных приложений) — стиль, который применяется ко всему приложению или, как минимум, ко всей Activity.

Эмулятор — виртуальное устройство, имитирующее работу смартфона, на котором можно запустить и протестировать наши приложения.

Юзабилити (удобство использования) — способность приложения быть удобным для пользователя, практичным, эстетичным, простым и интуитивно понятным в использовании, позволять в кратчайшее время достигать желаемого результата и при этом обеспечивать пользователю чувство удовлетворенности от использования именно этого приложения.

Минимальные системные требования определяются соответствующими требованиями программ Adobe Reader версии не ниже 11-й либо Adobe Digital Editions версии не ниже 4.5 для платформ Windows, Mac OS, Android и iOS; экран 10"

Электронное издание для дополнительного образования

Серия: «Школа юного программиста»

Федотенко Мария Александровна

РАЗРАБОТКА МОБИЛЬНЫХ ПРИЛОЖЕНИЙ
ПЕРВЫЕ ШАГИ

Под редакцией В. В. Тарапаты

Для детей старшего школьного возраста

Ведущий редактор *Т. Г. Хохлова*
Художники *В. А. Прокудин, Я. В. Соловцова*
Технический редактор *Т. Ю. Федорова*
Корректор *И. Н. Панкова*
Компьютерная верстка: *Е. Г. Ивлева*

Подписано к использованию 22.01.19.
Формат 155×225 мм

Издательство «Лаборатория знаний»
125167, Москва, проезд Аэропорта, д. 3
Телефон: (499) 157-5272
e-mail: info@pilotLZ.ru, http://www.pilotLZ.ru

Автор книги, Федотенко Мария Александровна, профессиональный Android-разработчик, имеет 5-летний опыт работы в ведущих IT-компаниях России и Германии.

В настоящее время преподает программирование и разработку мобильных приложений в МПГУ.

Активная участница движения «Women in Tech».

Программирование – это грамотность XXI века!

Книги новой серии «Школа юного программиста» издательства «Лаборатория знаний» построены на методике пошагового обучения программированию. Следуя этой методике, любой желающий, от школьника до студента вуза, сможет научиться писать программы, разрабатывать мобильные приложения и компьютерные игры и даже освоить технологии машинного обучения и нейросетей.

В серию войдут следующие учебные пособия:

- **«Учимся вместе со Scratch: программирование, игры, робототехника» (5–6 классы)**
- **«Scratch 2.0: от новичка к продвинутому пользователю. Пособие для подготовки к Scratch-Олимпиаде» (1–11 классы)**
- **«Творческие задания в среде Scratch. Рабочая тетрадь для 5–6 классов»**
- **«Scratch 2.0: творческие работы на вырост. Рабочая тетрадь для 7–8 классов»**
- **«Создаем игры с Kodu Game Lab» (4–5 классы)**
- **«Python для начинающих – от основ до ООП и приложений» (7 класс)**
- **«Олимпиадное программирование на Python» (7–8 классы)**
- **«C# – новый учебный курс программирования от основ до продвинутого уровня» (8–9 классы)**
- **«Разработка мобильных приложений. Первые шаги» (8–9 классы)**
- **«Web-разработка: создай свой идеальный сайт. Обучаемся тонкостям HTML, HTML5, CSS3, SQL, PHP, JavaScript» (8–10 классы)**
- **«Основы искусственного интеллекта и нейросетей» (10–11 классы, студенты) и другие.**