Содержание

1 *Первые шаги в Android-разработке* **2**
1.1 Знакомство с курсом 2
1.1.1 Первое приложение 5
1.1.2 Интерфейс студии и структура проекта 10
1.1.3 Создание и запуск Android эмулятора 17
1.1.4 Создание стороннего эмулятора 20
1.1.5 Краткое знакомство с часто встречающимися понятиями . 20

Неделя 1

Первые шаги в Android-разработке

1.1 Знакомство с курсом

Эта специализация спроектирована не только для того, чтобы вы узнали основы разработки под Android, но еще для того, чтобы вы легко могли влиться в любой коллектив. Иначе говоря, в нашей серии курсов будет не только ванильный Android, в нем будет разбор архитектур: MVP, MVVM, Clean, новоиспеченного Life Archtecture. Будет изучение популярных библиотек, таких как Retrofit, Picasso, ButterKnife, Moxy и других, использованных на реальных проектах. Будет функционально-реактивное программирование посредством RxJava и нового Stream API. Помимо этого вы научитесь сетевому взаимодействию и хранению данных в базе, возьмете многопоточность под контроль, наведете лоск с помощью анимации и material-дизайна и узнаете многое другое.
В этом курсе мы разберём:

1. Установку и настройку Android Studio
2. Создание и работу с эмулятором
3. Основные компоненты Android приложения
4. Основные элементы интерфейса
5. View и ViewGroup реализации
6. Работу с Activity и Fragment
7. Работу с ресурсами
8. Работу с Preferences

Также будут приведены маленькие примеры кода для ознакомления с некоторыми аспектами различных механизмов. Совместно с этим, будут продемонстрированы приложения из Маркета, которые реализуют какие-либо комплексные механизмы. Для лучшего понимания реализации этих механизмов в коммерческом приложении.

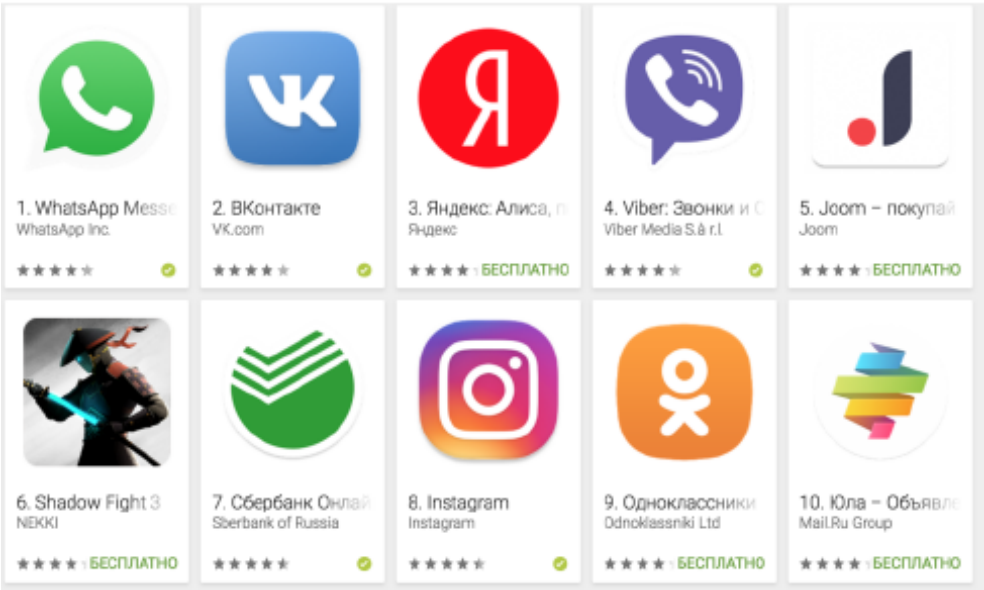


Рис. 1.1: Приложения, на которых будут разбираться разные механизмы

В 2003 году инженер-программист Энди Рубин основал Android Incorporated, которая разработала мобильную операционную систему Android, а в 2005 году эта компания была куплена Google. Первый телефон на Android (T-Mobile G1) поступил в продажу в 2008 год . В первой версии Android не было возможности воспроизведения видео, не было браузера и виртуальной клавиатуры. С каждой версией добавлялся новый функционал. Под Android программируют на Java потому что язык работает на виртуальной машине и его не нужно перекомпилировать для каждого нового устройства.

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.5%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.5%
4.1.x	Jelly Bean	16	2.2%
4.2.x		17	3.1%
4.3		18	0.9%
4.4	KitKat	19	13.8%
5.0	Lollipop	21	6.4%
5.1		22	20.8%
6.0	Marshmallow	23	30.9%
7.0	Nougat	24	17.6%
7.1		25	3.0%
8.0	Oreo	26	0.3%

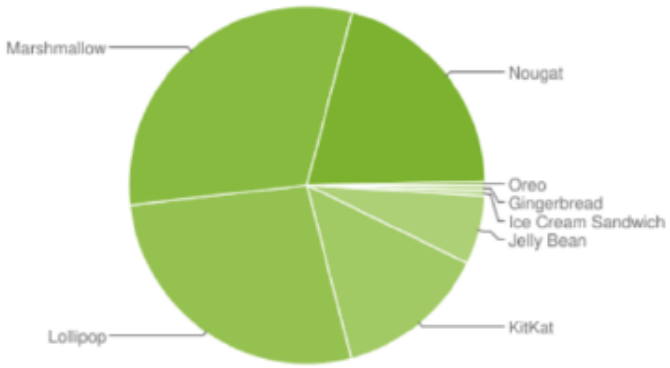


Рис. 1.2: Распределение устройств по различным версиям Android

Данные по распределению процента устройств по их API.

Весной 2017 года Google объявил об официальной поддержке Kotlin в качестве основного языка разработки под Android. Kotlin — это очень мощный JVM-язык, в котором много новых интересных возможностей, недоступных на Java. Помимо Java и Kotlin на Android можно писать и на других JVM-языках (Java Virtual Machine): Scala, Groovy и прочих.

Библиотеки поддержки:

- com.android.support:appcompat-v7:27.0.1
- com.android.support:design:27.0.1
- com.android.support:support-v13:27.0.1

Основные источники информации и ответы на типичные вопросы:

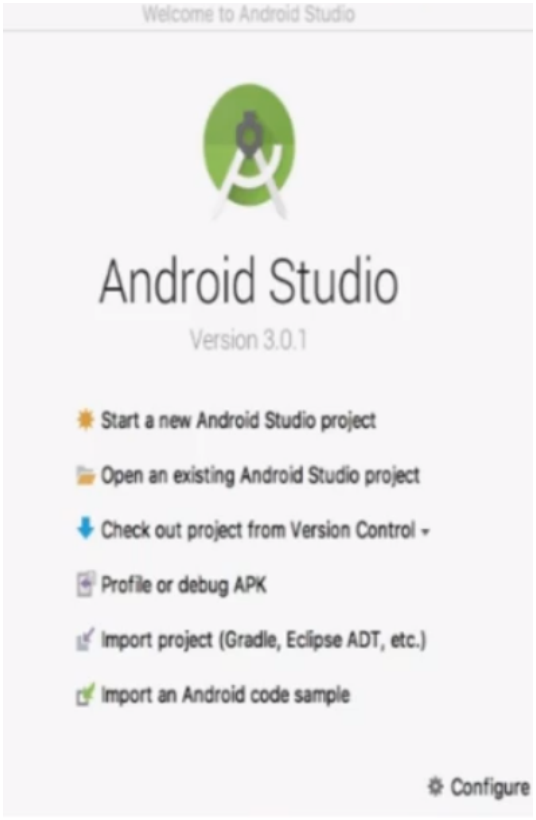
- Официальная документация - Гайды, обучение, дизайн, классы, best practices...
- Stack Overflow - 99,9% вопросов, которые у вас возникнут, уже были тут.
- Блог разработчиков Android - Новые инструменты, best practices...
- Гугл-документ с основными полезными ссылками

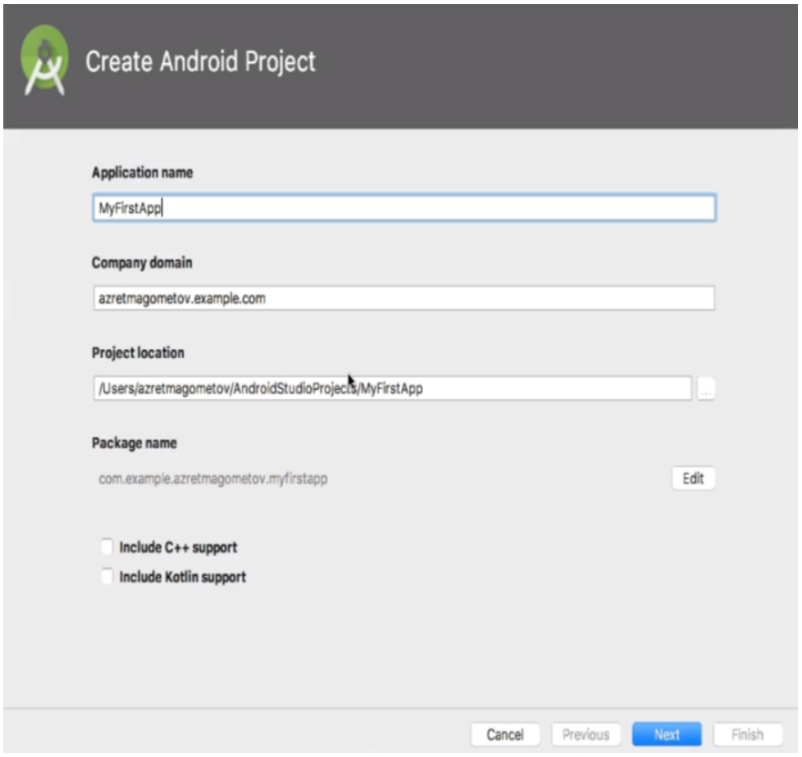
Далее выполняем всё согласно инструкции:

1. Алгоритм установки Android Studio
2. Настройка Android SDK.

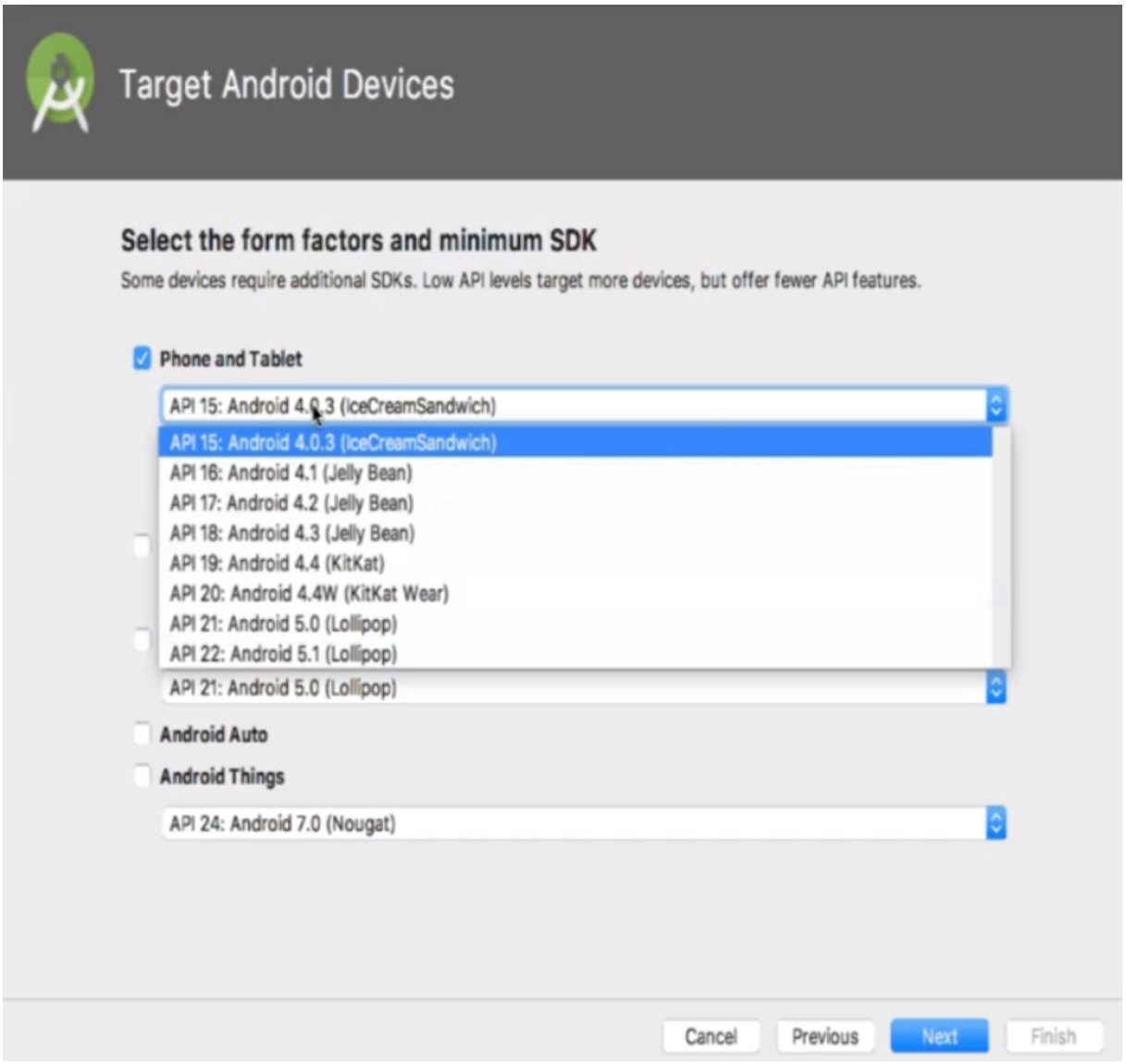
1.1.1 Первое приложение

Создание нового проекта





Для уникальности названий проектов в Android создаётся уникальное название пакета состоящее из доменной компании (или, например, адреса сайта) и названия приложения. В Google Play приложения можно опознать именно по названию пакета.
Поддержка C++ и поддержка Kotlin сейчас нам не нужны, но вообще поддержку C++ можно включить, если планируется вызвать C++ код из Java (либо использование библиотеки, написанной на C++, либо обращение к более низкому слою Android-фреймворка). Kotlin — это новый JVM-язык, который недавно получил официальную поддержку от Google. Kotlin достаточно серьезно отличается от Java и своей парадигмой, и объектно-фукнциональным стилем, и непосредственно синтаксисом. Поэтому пока не будем его включать.
Далее мы должны определиться с устройствами и версиями Android, для которых мы пишем приложение:



Каждой версии Android соответствует свой уровень API (application programming interface). Важный момент: **чем ниже минимальный уровень API, тем на большем количестве устройств может работать ваше приложение.** Почему это происходит? Смартфоны с Android далеко не всегда обновляются до последней актуальной версии API. И они остаются у людей.
Чем ниже уровень API, то меньше новых возможностей вам доступны из коробки. Каждая версия Android преподносит что-то новое, и если минимальный уровень API в котором появилась та или иная возможность. Для использования этой фичи, вам нужно будет либо проверить текущую версию OS устройства на соответствие, либо использовать функционал этой фичи из библиотеки обратной совместимости. Например, (узнать, что когда добавилось можно кликнув на "help me choose") в 6-ой версии Android (API 23) появился функционал для работы с отпечатками пальцев.Однако если я выставлю минимальный API —

19, Kitkat, то я уже не могу просто так вызвать код, чтобы использовать сенсор «отпечаток пальцев», так как приложение может быть запущено на устройстве с Android 4.4, 5.0, 5.1, то есть на OS без поддержки сенсора. И если просто вызвать код, то приложение крашнется. В этом случае есть два варианта: либо перед вызовом проверить, что текущая версия операционной системы Android ≥ 23, либо используем библиотеку обратной совместимости, которая просто использует заглушки для API ниже 23. Однако следует учитывать, что не для всех новых фич есть библиотеки обратной совместимости. Получаем, что **чем выше версия API, тем больше возможностей (фич) доступно из коробки.**
Мы будем использовать Android 4.4 KitKat (API 19), чтобы наше приложение поддерживалось 90,1% устройств.

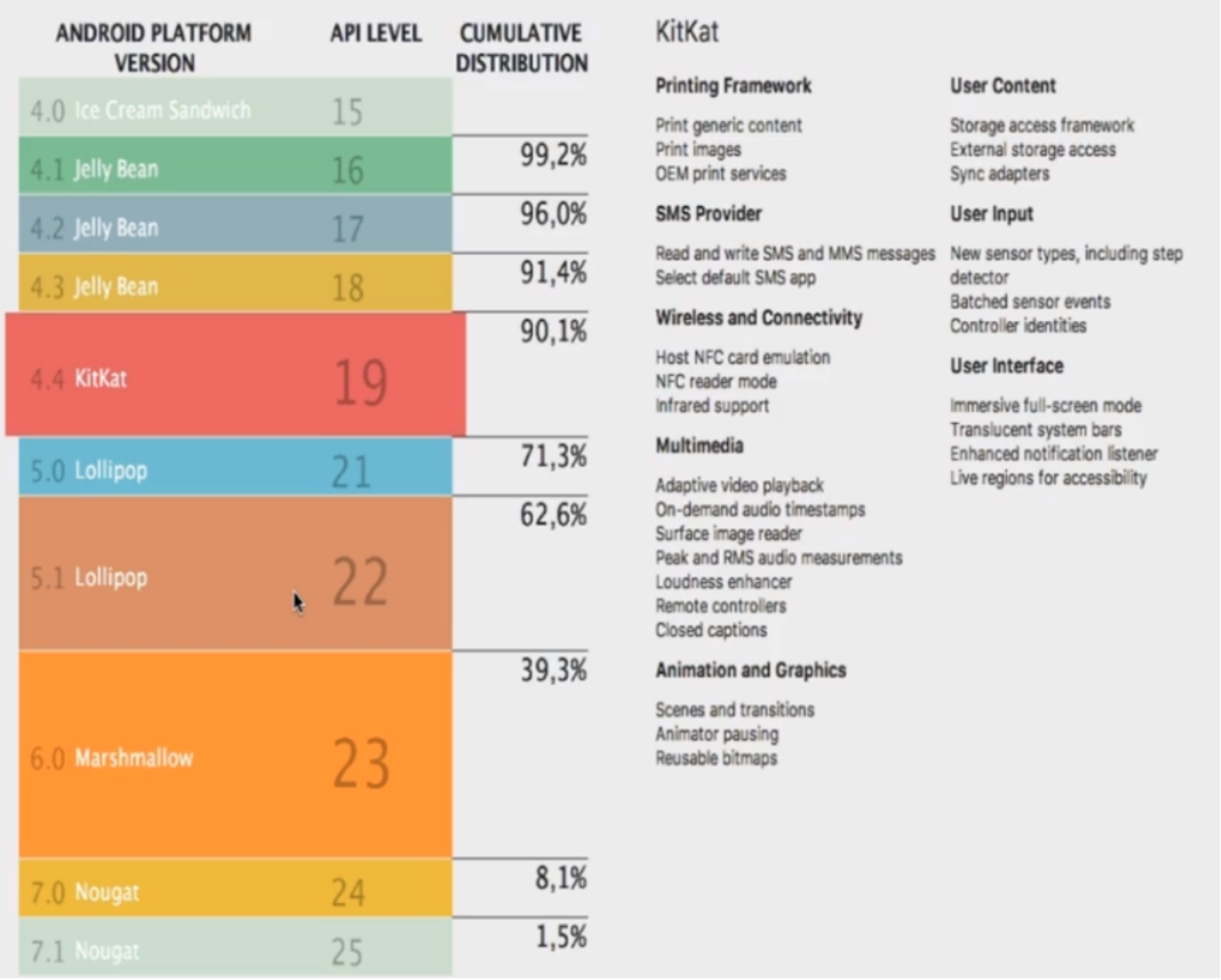


Рис. 1.3: Процент устройств, поддерживающих соответствующий API

Далее мы должны выбрать запускаемую Activity (по сути это какой-либо экран

приложения). Например, при запуске приложения у вас открывается экран авторизации — это первая Activity. Вы вбиваете логин, пароль, щелкаете вход и открывается вторая Activity — главный экран. Вы щелкаете на Настройки и открывается третья Activity — экран с настройками. Тут важно понимать одну вещь: все эти Activity — это все одна сущность, но с разной версткой интерфейса и соответствующей бизнес-логикой.

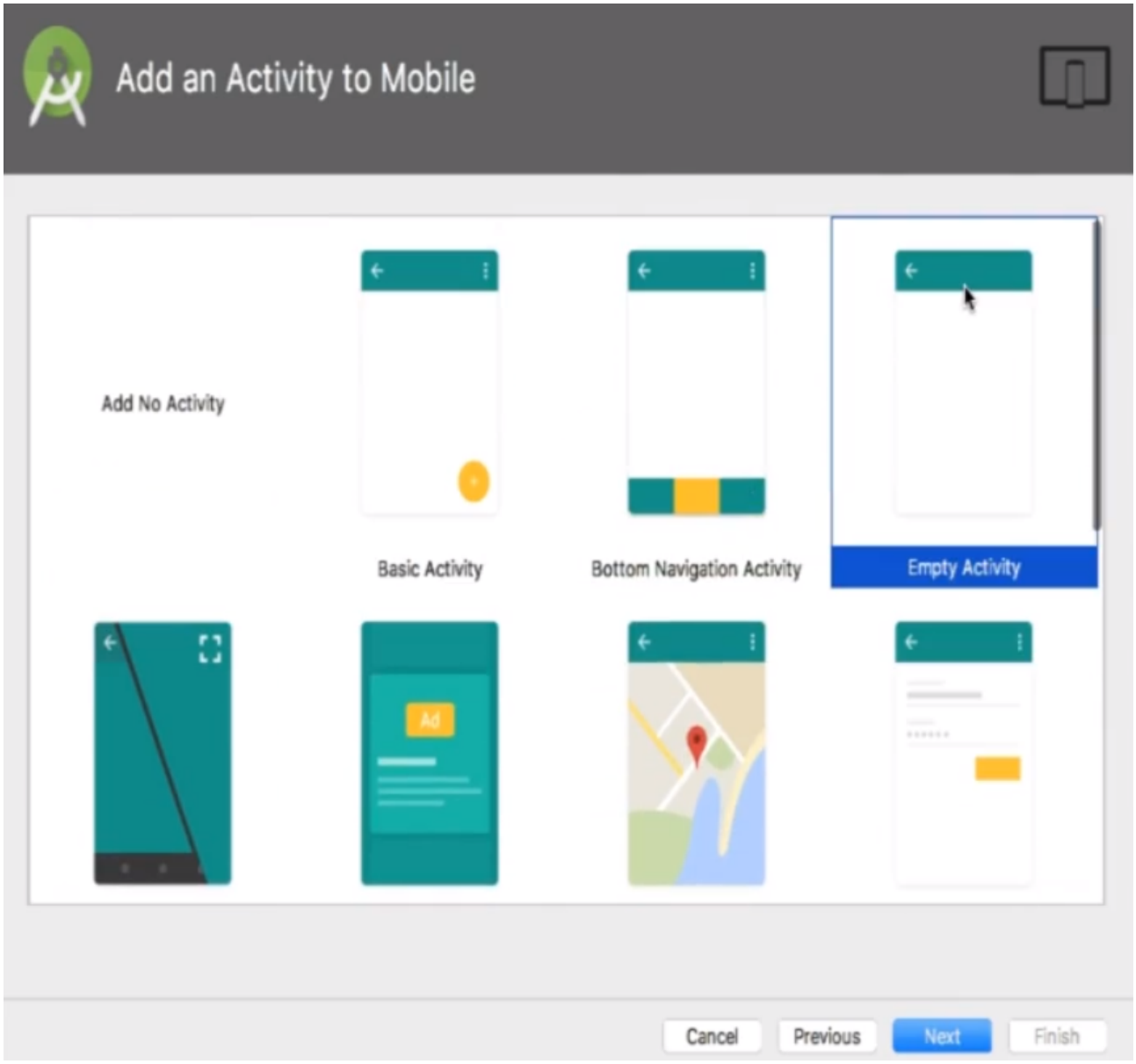


Рис. 1.4: Выбор типа Activity

Выбираем Empty Activity, щелкаем Next. Здесь выбираем название Activity, оно должно быть опознаваемым и обоснованным. Например, не стоит называть Actitivy своим именем. И еще: хорошим тоном считается использовать название класса, наследующегося от компонента Android, название этого самого компонента. В нашем случае это Activity, LaunchActivity. Галочку «Сгенерировать

разметку» оставляем, чтобы не создавать ее самим. Галочку обратной совместимости оставляем, что именно она делает, я расскажу чуть попозже. Нажимаем Next. Studio качает необходимые файлы. Щелкаем Finish и ждём запуска студии (это может занять час времени и требовать стабильного подключения к интернету).

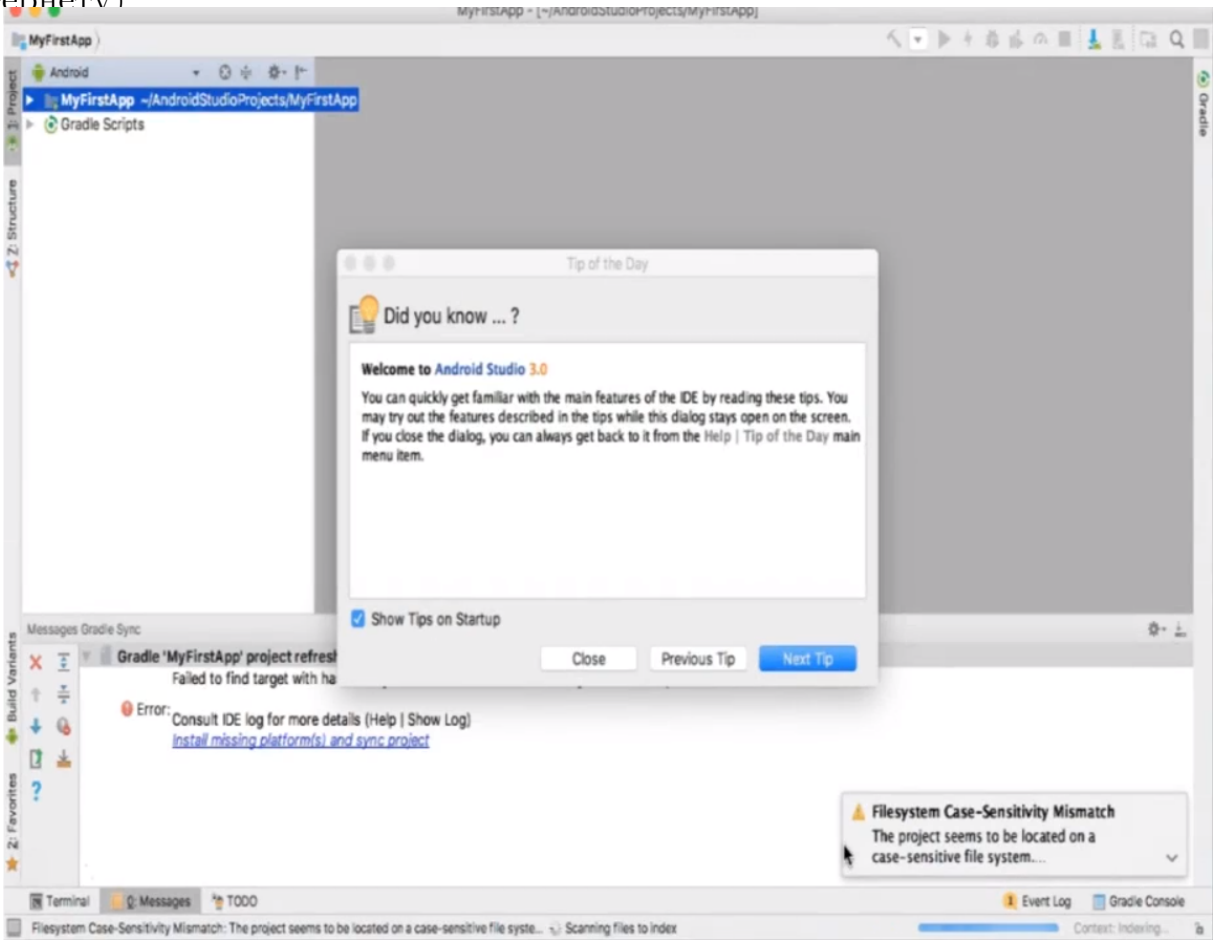
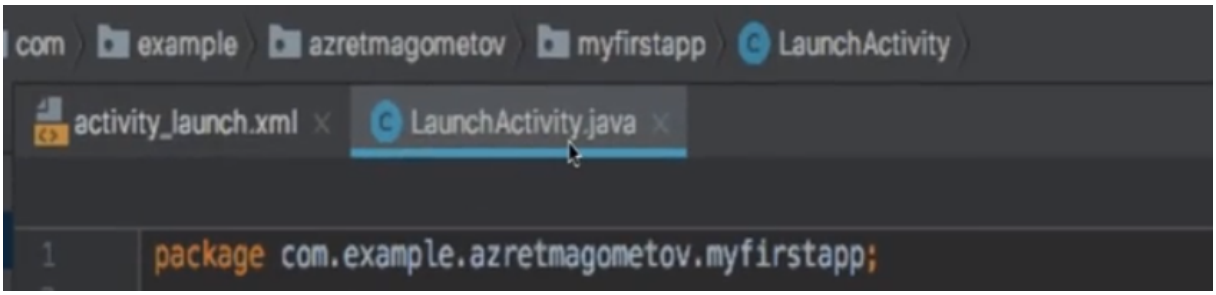


Рис. 1.5: Запуск Android Studio

Закрываем окно подсказок. И видим, что студия выдаёт ошибку "Failed to find target with hash...". Кликаем на "Install missing platform(s) and sync project принимаем лицензионное соглашение. Next. Ждём загрузки. Finish. Студия снова выдаёт ошибку, на этот раз нажимаем "Install Build Tools 26.0.2 and sync project". Всё устанавливается и происходит долгий первый запуск студии.

1.1.2 Интерфейс студии и структура проекта

Итак, прямо по центру — это окно, где мы непосредственно пишем наш код. Сейчас тут открыт файл LaunchActivity, и это, как вы можете заметить, java-файл. В java-файл классах мы пишем бизнес-логику нашего приложения.



В начале у нас в LaunchActivity.java появился код:

Листинг 1.1: LaunchActivity.java

```
package com.example.azretmagometov.myfirstapp;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
//стандартно для всех
public class LaunchActivity extends AppCompatActivity {
//наша LaunchActivity наследуется от AppCompatActivity.
//Из-за обратной совместимости работает везде одинаково
//При написании Android-приложений мы наследуемся компонентов
//android, которые вшиты в телефоны и добавляем свою логику
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    //вызов метода родителя
    setContentView(R.layout.activity_auth);
    //задаём layout экрана
  }
}
```

после завершения проекта, после сборки мы получим apk, в котором не будет оригинальных компонентов Android. В apk будут компоненты, которые мы написали сами, или добавили через библиотеки. Смотрите, наша LaunchActivity наследуется от AppCompatActivity. И само собой, AppCompatActivity в конечном итоге будет наследоваться от обычной activity. Давайте проследим этот момент. Нажимаем Ctrl (или Command если вы на Mac) и щелкаем по AppCompatActivity - мы перешли в класс AppCompatActivity и видим, что оно наследуется от FragmentActivity и т.д. имеем:
LaunchActivity ⇒ AppCompatActivity ⇒ FragmentActivity ⇒
BaseFragmentActivity16 ⇒ BaseFragmentActivity14 ⇒ SupportActivity ⇒ Activity.

Саму Activity мы не можем просто так посмотреть. Поэтому кликаем на неё. принимаем соглашение JetBrainsDecompiler и видим сам класс Activity без исходников, потому что код вшит в телефон. Android Studio предлагает скачать Android API 26 Platform. Щёлкаем Download и качаем код Android 8.0. Но надо понимать, что activity у разных API могут чем-то отличаться и вести себя по-разному на разных моделях Android. После скачивания вернёмся к LaunchActivity (1.1). Слова AppCompat, Compat, Support значат что работает поддержка обратной совместимости и на всех устройствах Activity должна работать максимально одинаково (т.к. класс взят из библиотеки поддержки).
При создании Activity (т.е. нашего экрана), вызывается метод onCreate, который создает экран. super.onCreate означает, что вызывается метод родителя onCreate(). И в компонентах Android чаще всего необходимо вызывать методы родителей, потому что они, в конце концов, упираются в какую-то низкоуровневую логику, наподобие сохранения state'а, отправки сообщений, очистки и тому подобное. С помощью Ctrl (или Command) можно посмотреть код супер-метода onCreate(). Видим, что в super методе есть какой-то код, после которого опять же вызывается super метод. И это означает, что в вашем текущем коде тоже нужно вызывать super метод. Конкретно сейчас если попытаться убрать super метод, то при запуске Activity вылетит с ошибкой, потому что мы убиваем ее логику. Через Ctrl нажимаем на activity_auth:

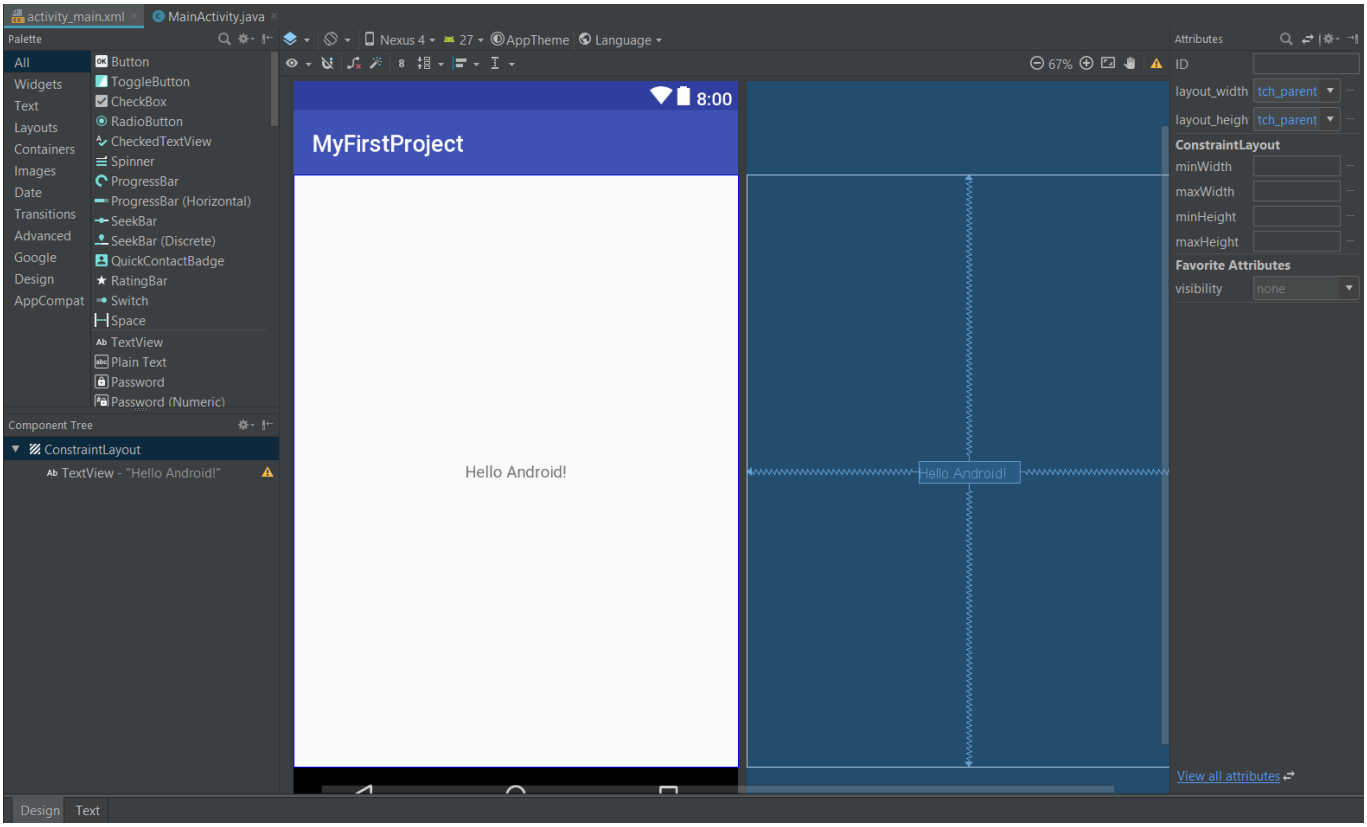
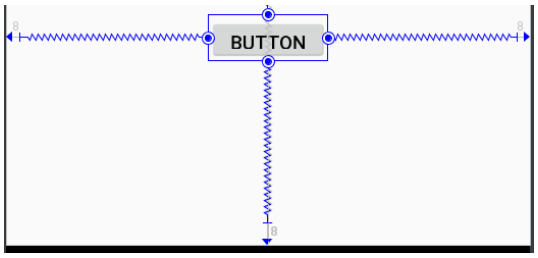


Рис. 1.6: вкладка Design у activity_main.xml Файла

Сверху слева находится **Palette (палитра)** т.е. список элементов интерфейса, которые можно добавить на верстку. Она отфильтрована по типам: работа с текстом, виджеты, и т.д.
Под палитрой видим **Component Tree (дерево элементов)** - дерево компонентов, из которых состоит текущая верстка. Здесь мы видим, что корневым является ConstraintLayout, и в нем находится textView с текстом «Hello World!».
Справа мы видим превью экрана, именно так и выглядит, будет выглядеть наш экран на устройстве.
Сверху **панель управления превью**, можно выбрать девайс, на котором будет запущено приложение, ну и проверить предпросмотр, повернуть телефон при желании. Ниже видим ещё инструменты. Они используются только для работы с ConstraintLayout, про это я расскажу чуть попозже.
Правее находится **Attributes(панель атрибутов)**. Этот список атрибутов на самом деле неполный. Внизу находится ссылка View all attributes. Щелкаем и видим уже весь список атрибутов. Что еще вам нужно знать? Атрибуты для каждого элемента свои. То есть если я сейчас щелкну на ConstraintLayout, то атрибуты поменяются.
Давайте сейчас попробуем поменять текст на кнопке. Видим атрибут Hello World, text, точнее, со значением «Hello World!». Пишем в text: «Hello Android!». Видим, что текст поменялся.
Важный момент - расположение элементов на верстке зависит от того, в каком контейнере они располагаются. В нашем случае контейнер — это ConstraintLayout, и в ConstraintLayout элементы мы можем располагать как угодно, важно их к чему-нибудь привязать с помощью Constraint (пружинки в четыре стороны от "Hello, Android!"). Важный момент - расположение элементов на верстке зависит от того, в каком контейнере они располагаются.



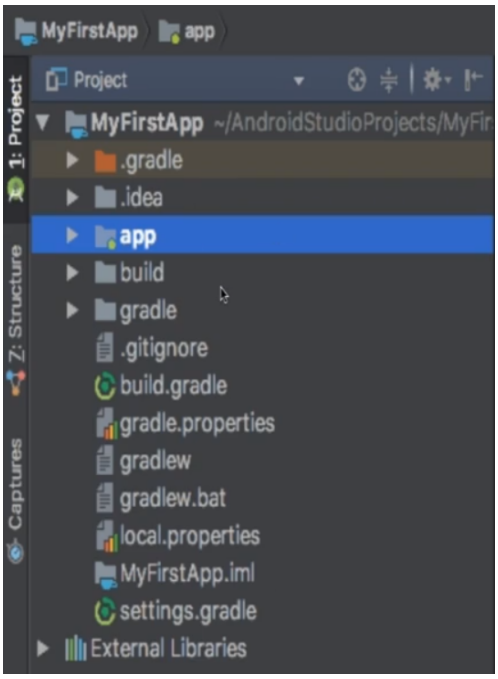
В нашем случае контейнер — это ConstraintLayout, и в ConstraintLayout элементы мы можем располагать как угодно, важно их к чему-нибудь привязать с помощью Constraint (пружинки в четыре стороны от "Hello, Android!"). Теперь мы нажимаем на белые точечки по краям синей рамки и привязываем кнопку к краям и надписи "Hello, Android".

Таким образом привязываем с четырёх сторон нашу кнопочку. Заметим, что в поле её атрибутов есть TextView(унаследованный от textView). Поменяем Button на, например, Click. Надпись изменилась.
Всё это время мы находились во вкладке Design (снизу). Но есть ещё вкладка Text - это xml разметка нашего Activity. Получается, что Studio позволяет нам создавать разметку либо с помощью дизайна, вкладки Design, когда мы просто перетягиваем элементы и растягиваем мышкой, либо напрямую через XML.

Откроем получившуюся разметку:

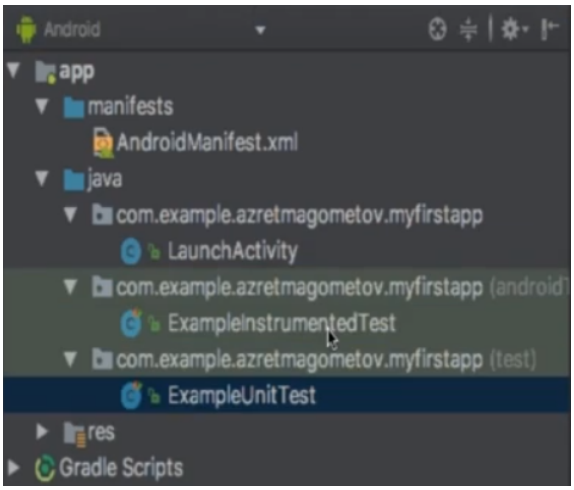
```
<?xml version="1.0" encoding="utf-8"?>
  <android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/
    android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context="jzargo.myfirstproject.MainActivity">
  <TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginBottom="8dp"
    android:text="Hello Android!"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
  <Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="34dp"
    android:layout_marginBottom="8dp"
    android:layout_marginEnd="8dp"
    android:layout_marginStart="8dp"
    android:layout_marginTop="60dp"
    android:text="Click"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/textView"
    app:layout_constraintVertical_bias="0.0" />

</android.support.constraint.ConstraintLayout>
```

Закроем файл разметки (activity_xml) и наша активити (LaunchActivity.java). Теперь вопрос: как найти наши файлы?



Слева, как вы могли заметить, находится панель со структурой нашего проекта. И эта панель, она поддерживает несколько различных представлений. Прямо сейчас используется представление Android. Это представление оставляет только те файлы и папки, которые важны прямо здесь и сейчас для работы с Android-кодом. Но мы можем переключиться на Project, и тогда структура файлов и папок будет такой же, какой она является на вашем жестком диске. Но опять же это плодит очень много ненужных файлов, слишком много папок, поэтому мы переключимся обратно на Android и разберем структуру здесь. Итак, изначально мы видим две папки: app и Gradle Scripts.

app — это модуль вашего приложения. **Модуль** — это отдельная папка со своими ресурсами и кодом, которые используются на конкретном типе устройств. Смотрите, при генерации, при создании проекта мы ставили галочку на «Планшеты и телефоны». Поэтому в нашем приложении сгенерировался только один модуль. Если бы мы поставили еще одну галочку, например, на часах, то у нас было бы в приложении два модуля: app для телефонов и планшетов, как сейчас, и wear для умных часов. Код и ресурсы на умных часах, отличаются от кода и ресурсов, которые использовались бы в приложении. Но если у них были бы какие-то схожие, смежные, смежный функционал, действительно, смежный функционал, то его, в свою очередь, можно было бы выделить в отдельный модуль, который использовали бы и модуль app, и модуль wear. Примерно так же работают подключаемые библиотеки: они просто добавляют еще один модуль к вашему приложению (т.е. библиотеки, исходный код которых вы встраиваете в проект и можете менять).



Вернёмся к представлению Android и увидим это.
AndroidManifest — это файл, в котором находится основная информация о вашем приложении. В первой папке наш LaunchActivity. Во второй - инструмент тесты. А в третьей - юнит тесты.

Листинг 1.2: AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/
  res/android"
  package="com.example.azretmagometov.myfirstapp">
  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".LaunchActivity">
    <intent-filter>
      <action android:name="android.intent.action.MAIN" /
        >
      <category android:name="android.intent.category.
        LAUNCHER" />
    </intent-filter>
    </activity>
  </application>
</manifest>
```

Видим: иконку и круглую иконку (в более поздних версиях) для запуска приложения с главного экрана, тема и активити, которые будут использоваться в приложении. В intent-filter написано, что она — запускаемая Activity, то есть при нажатии на иконку на экране телефона приложение запустится именно с этой Activity. Подробная информация о файле AndroidManifest. В папке java:

- в первой подпапке видим нашу LaunchActivity

- во второй (android) - **инструментальные тесты** - тестирование взаимодействия коммпонентов андроид и поэтому они запускаются как обычное приложение - на эмуляторе или смартфоне.
- В третьей (test) - **юнит-тесты** - проверка java-классов и тестирование бизнес-логики приложения. Они запускаются прямо с компьютера.

В папке res находятся **ресурсы** приложения - это всё, что не является java кодом и не планируется качать с интернета(строки, цвета, константы, разметка экрана, картинки и т.д.).

- Папка drawable хранятся изображения. По умолчанию там лежат иконки (в виде векторной графики) запуска в виде нескольких слоёв.
- В layout находится наш файл разметки (вёрстки) activity_launch.xml
- В mipmap хранится иконка с главного экрана. Инструкция по построению изображения
- В values значения строк, цветов и стилей (несколько атрибутов, объединённых в один, например, одинаковые кнопки)

Теперь рассмотрим Gradle Scripts. Там нас интересует build.gradle (app).

```
apply plugin: 'com.android.application'
android {
  compileSdkVersion 26//уровень API сборки проекта
  defaultConfig {
    applicationId"com.example.azretmagometov.
      myfirstproject"
    minSdkVersion 19 //минимальная поддерживаемая версия SDK
    targetSdkVersion 26 //максимальная поддерживаемая версия
    versionCode 1//версия самого приложения. Для обновлений
    versionName "1.0"//номер версии, видный пользователю
    testInstrumentationRunner "android.support.test.
      runner.AndroidJUnitRunner"
  }
  buildTypes {
    release {
      minifyEnabled false
      proguardFiles getDefaultProguardFile('proguard-
        android.txt'), 'proguard-rules.pro'
    }
  }
}
dependencies {
```

```
    implementation fileTree(dir:'libs',include: ['*.jar'])
    implementation'com.android.support:appcompat-v7:26.1.0'
    implementation 'com.android.support.constraint:
      constraint-layout:1.0.2'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'com.android.support.test:
      runner:1.0.1'
    androidTestImplementation 'com.android.support.test.
      espresso:espresso-core:3.0.1'
}
```

В скобках пишется, к какому уровню (папок) принадлежит данный файл - в режиме просмотра Project он лежит соответственно в папке app, отвечающей за Android (для часов был бы wear и т.д.). В build.gradle (app) находятся инструкции для сборки нашего проекта.
В блоке dependencies хранятся библиотеки, которые мы добавляем в проект.
Всё со структурой. Итак, в файле Java у нас хранятся Java-файлы, в ресурсах хранятся ресурсы. Ресурсы делятся на подтипы. В gradle-скриптах хранятся build.gradle файлы. Самый важный из них — Module: app.

Полезные материалы: о директории java и директориях res и assets

1.1.3 Создание и запуск Android эмулятора

В Android Studio откроем меню ⇒ Tools ⇒ Android ⇒ AVD manager (Android Virtual Device Manager) ⇒ Create Vitual Device и тут выбираем нужную конфигурацию (или заранее собранный preset т.е. существующий телефон)

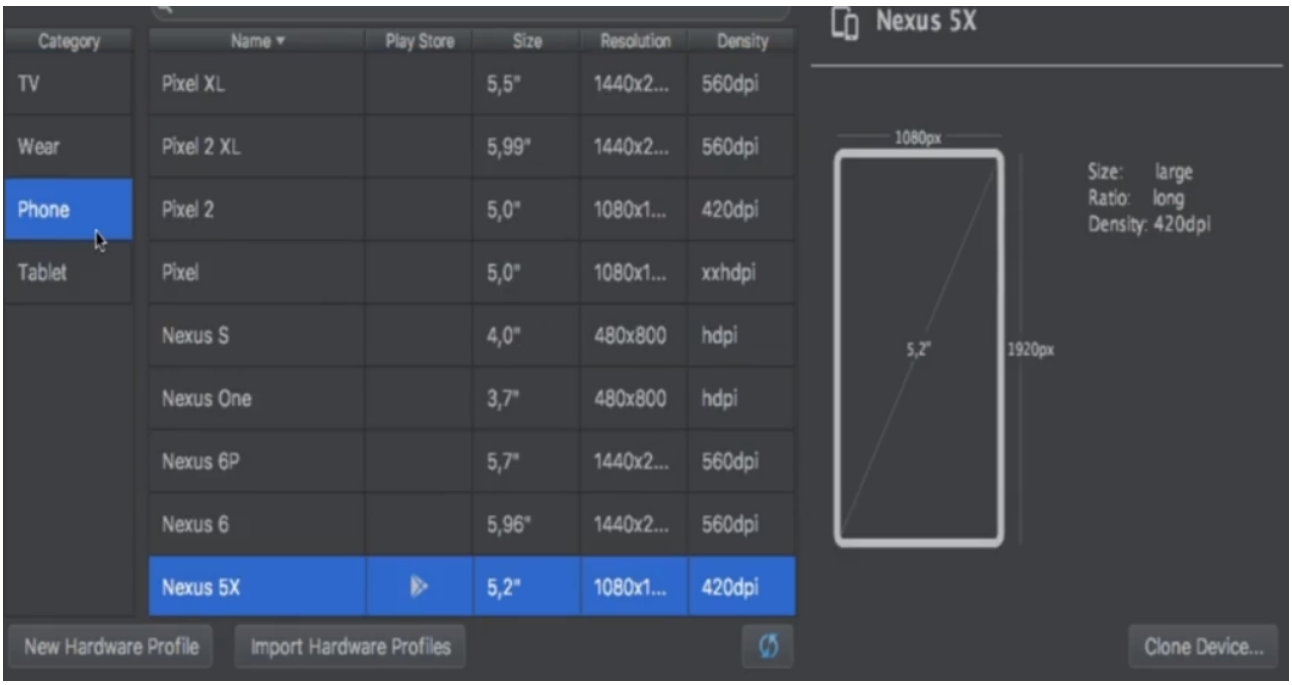


Рис. 1.7: Выбор эмулятора из готовых пресетов

Обращать внимание надо на Size (размер диагонали в дюймах), Resolution (разрешение экрана в пикселях) и Density (dpi=dots per inch т.е. точек на дюйм). Подробнее про разрешение здесь. Если ни один пресет не устраивает, то можно создать свою комбинацию нажав на new hardware profile.
Мы же выберем Pixel 5,0 дюйма 1080×1920. Жмём Next. Выбираем операционную систему Oreo т.е. API 26 (если она не установлена то закружаем через Download. Это может занять много времени). Можно и другую, главное помнить, что если мы для приложения выбирали какой-то уровень API, то наш эмулятор должен быть уровнем не ниже.
Насчёт архитектуры (ABI): Архитектура x86 либо ARM. Если у вас процессор от Intel и операционная система Windows, то эмулятор (и если процессор поддерживает технологию VTx), то эмулятор будет работать очень быстро засчет ускорения. Если у вас AMD процессор и вы работаете на Windows, то штатный эмулятор, который предлагает Android Studio, будет работать очень медленно. В этом случае вам нужно воспользоваться либо реальным устройством, либо сторонним эмулятором. Откуда брать сторонние эмуляторы, я расскажу попозже. Кроме того x86 рано или поздно упираются в 10-й уровень API Android 2.3.3. У ARM есть даже образы первого Android, уровень API 3, Android 1,5.
В последнем столбце Target выделено, что некоторые из образов имеют в скобках Google API. Это Google Play сервисы. Соответственно, даже на них можно будет запустить Play market Google Play и установить какие-то приложения и другие возможности Google Play Services. Оставляем Oreo, щелкаем Next.

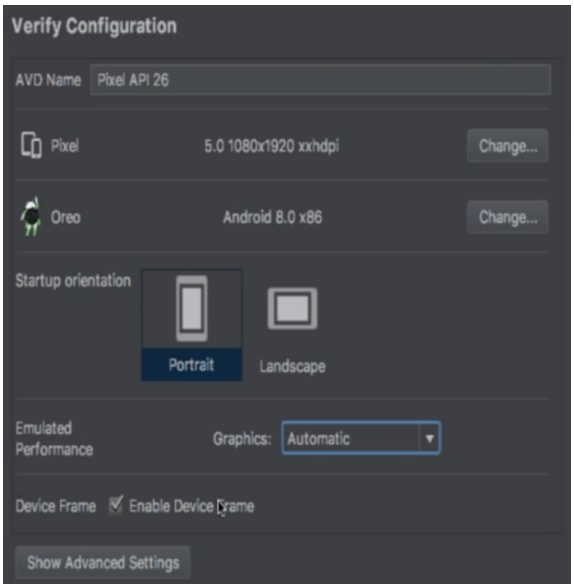
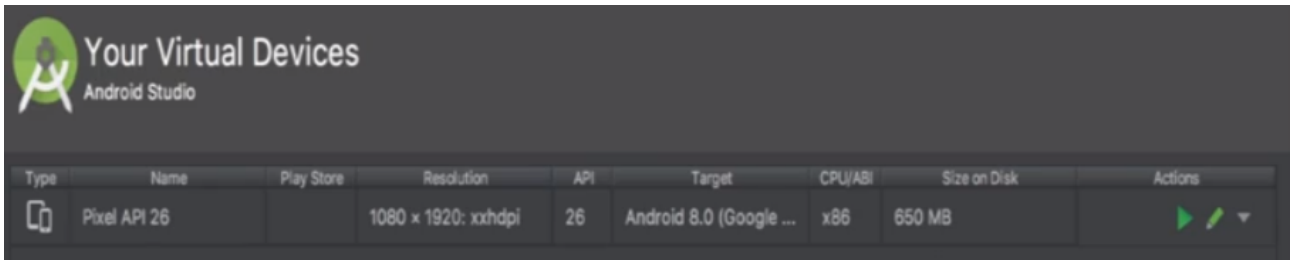


Рис. 1.8: Конфигурация эмулятора

Открывается окно с итогами. Graphics Automatic можно не трогать. Эмулятор будет сам решать, на чем ему будет работать лучше, быстрее: либо на железном ускорении, либо на софтовом ускорении. Дальше галочка Enable Device Frame означает, что в эмуляторе будет такая декоративная окантовка в виде той оболочки, которую мы выбрали. Лучше отключим. Кликаем на Show Advanced Settings, и здесь настройки эмулятора, но самая важная настройка — это Enable keyboard input, то есть включение возможности вбивать текст в эмулятор через соответственно клавиатуру компьютера, а не набирать его в самом эмуляторе через виртуальную клавиатуру. Оставляем галочку, щелкаем Finish.
Эмулятор создался. И мы видим такое окошко:



Запускаем эмулятор (зелёный треугольник) и видим экран:

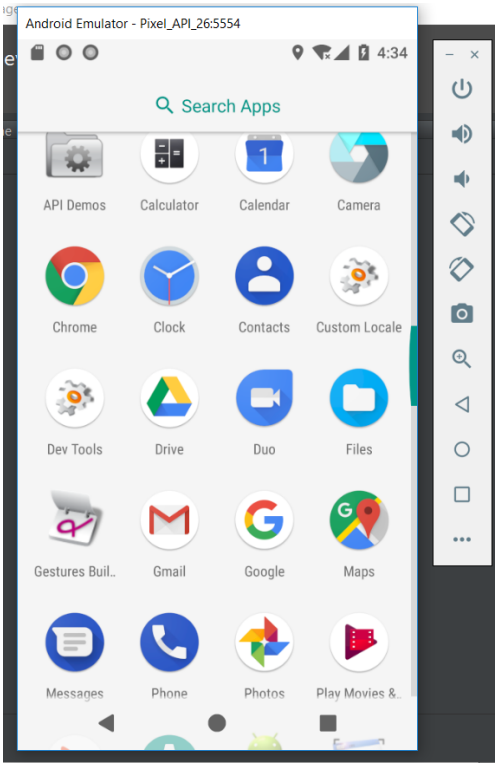


Рис. 1.9: Работающий эмулятор

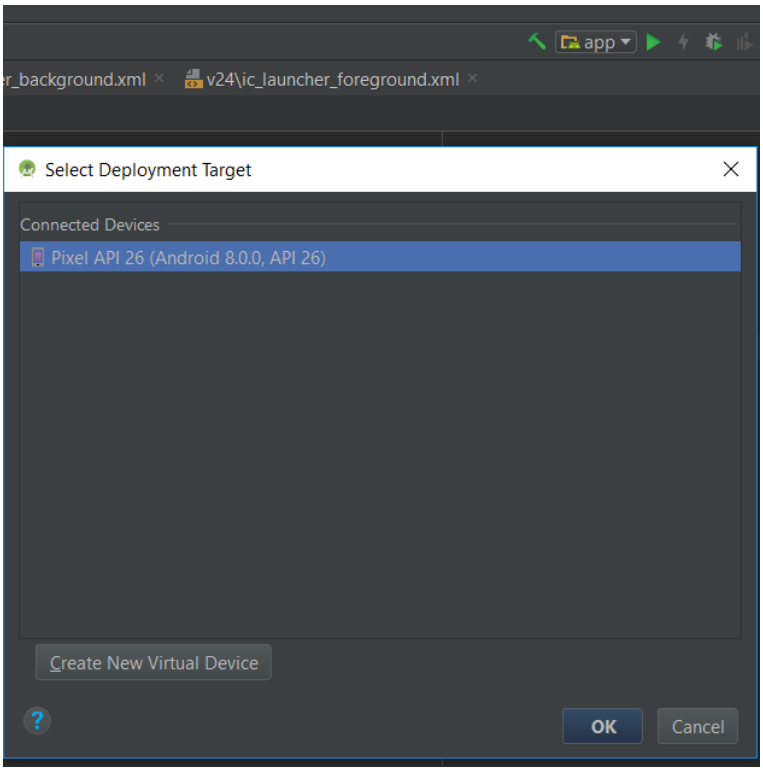


Рис. 1.10: Окно выбора эмулятора

Теперь давайте запустим на нём наш проект. Переключаемся обратно в Android Studio и щёлкаем на Run (зелёный треугольник в правом верхнем углу). И всё работает как на макете. Можно попробовать изменить размер текста дописав в activity_launch.xml в разделе TextView

```
android:textSize="40sp"
```

Снова щёлкаем на Run (Shift+F10) и видим изменения. Android Studio 3 поддерживает instant run т.е. при изменении .xml не требуется повторная установка приложения. Проверим изменения дописав ещё

```
android:textStyle="bold"
android:textColor="@color/colorAccent"
```

И щёлкаем на Instant run (Apply Changes, на Ctrl+F10). И через пару секунд всё поменялось. Это куда быстрее чем сборка, установка и запуск с нуля. Но у этого

метода есть минусы, например, если у вас очень много классов и очень много зависимостей, то изменения в одном классе не могут примениться к изменениям в других классах. Поэтому если у вас тяжелый проект, то лучше не пользоваться Instant Run, а железно запускать заново.

1.1.4 Создание стороннего эмулятора

Если ваш процессор от AMD либо от Intel, но не поддерживает VT-x, то, скорее всего, вы заметите, что штатный эмулятор, который предлагает нам Android Studio, работает очень медленно. Тогда качаете эмулятор от Genymoution (для этого надо просто зарегистрироваться на их сайте). Но в этом эмуляторе есть ограничения (на работу с камерой, геоолокацией и т.д.). Этот эмулятор работает в среде VirtualBox, поэтому если его у вас нет, качаете вместе с VirtualBox. Установка такая же как и у любого приложения для Windows (через Wizard). После установки открываем сам Virtual device creation wizard, логинимся через аккаунт. который создали на сайте, кликаем Add и аналогично созданию эмулятора в AVD Manager выбираем себе нужный пресет нужной версии Android и нужной модели и качаете образ.

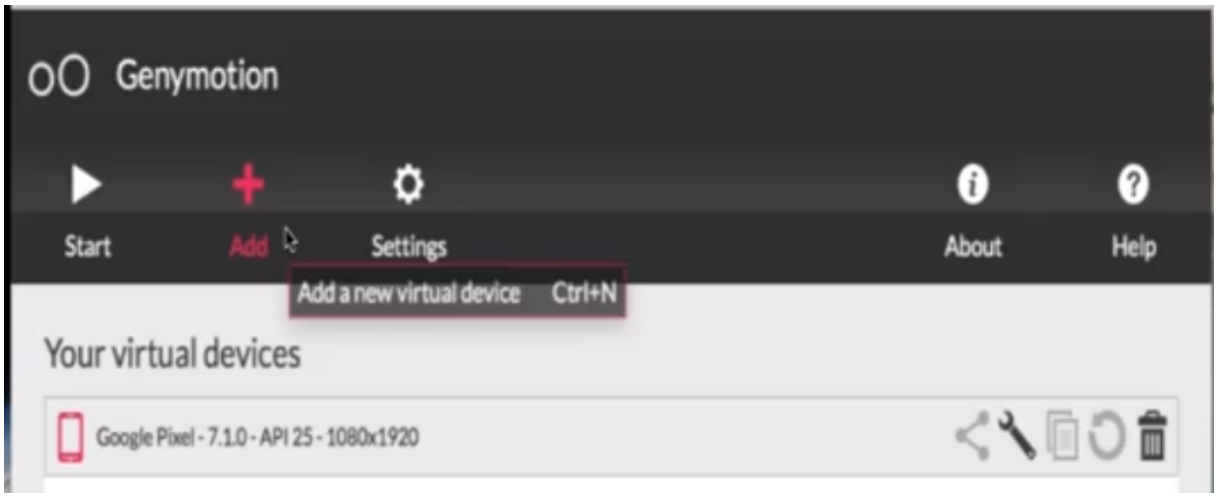


Рис. 1.11: Добавление эмулятора от Genymoution

После запуска эмулятора переходим в студию и так же в Run выбираем нужный эмулятор и включаем Instant Run. Всё запускается.
Но если по какой-то причине вам нужен другой, то можно скачать эмулятор от Microsoft, в котором можно разрабатывать в среде Visual Studio на платформе Xamarin, используя язык C#, но этим мы в данном курсе заниматься не будем.

1.1.5 Краткое знакомство с часто встречающимися понятиями

Для начала про работу с ресурсами. Каждый раз, когда вы добавляете новый ресурс в структуру ваших ресурсов, для нее генерируется уникальный идентификатор. Для примера заходим в activity_launch, видим код нашей Activity. И

Studio подсказывает нам, что текст у нас захардкожен (записан в самом xml коде), что неплохо было бы вместо текста Hello Android! использовать ссылку на строковый ресурс.Давайте воспользуемся этим советом, потому что это правильно.

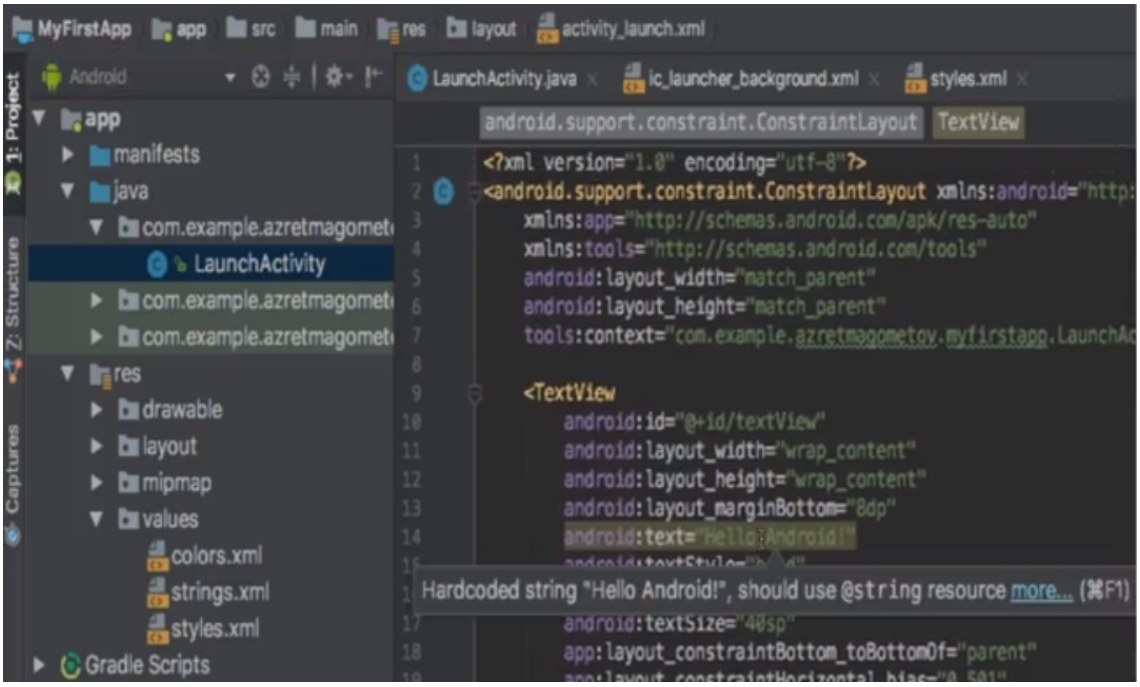


Рис. 1.12: Ошибка при использовании строки, а не строкового ресурса

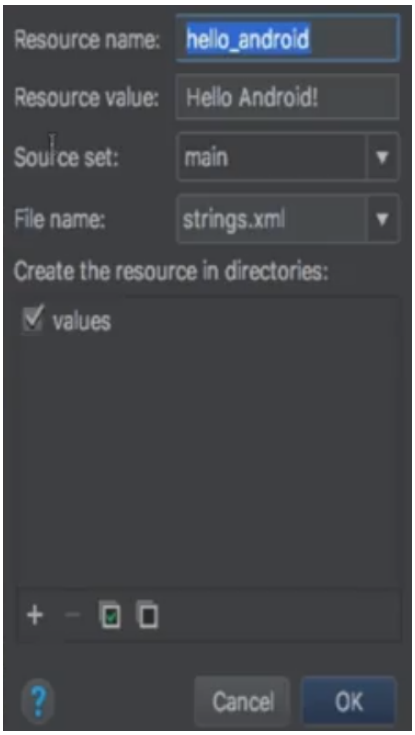


Рис. 1.13: Добавление строкового ресурса

Правильно хранить все строковые ресурсы в одном файле strings, вместо того чтобы хардкодить их, а потом при изменении искать по всем файлам, переименовывать, переделывать, переписывать. Плюс если вы вдруг решите перевести свое приложение с английского на французский, вам достаточно будет перевести только файл strings, так как все строки находятся в нем. Для того, чтобы это исправить наводим на значение атрибута текст(Hello Android) нажимаем Alt+Enter и выбираем в появившемся контекстном меню Extract string resource. Вбиваем название и значение ресурса. Ok.

Теперь вместо захардкоженного Hello Android! мы видим ссылку на этот ресурс. Причем ссылка вида: тип ресурса, @string и название ресурса Hello Android! Давайте перейдем в файл strings щелкнув на strings, либо зажав Ctrl (Command) и щелкнуть прямо на ресурс.

Теперь мы можем в strings менять содержание строки и оно изменится везде, где использовалось. И все переиспользованные ресурсы крайне желательно именно хранить в ресурсах.
К ресурсам из java файлов обращаться можно строкой

```
getString(R.string.hello_android);
```

Каждый раз, когда вы добавляете новый ресурс (в папку resources) для него генерируется уникальный идентификатор, уникальный указатель. Этот идентификатор имеет численный тип, и он хранится в классе R в своем собственном подмножестве. В нашем случае это string.
Создадим другой ресурс и посмотрим на поведение. Заходим в res⇒ values ⇒ colors.xml и пишем

```
<color name="red"#FF0000</color>
```

Возвращаемся к launch_activity.java и обращаемся к этому цвету java-кодом:

```
getColor(R.color.red);
```

Процесс следующий: добавляем ресурс, студия для него сама генерирует указатель, который находится в классе R в соответствующем подмножестве.
Варианты R:

- color - цвета
- string - строки
- anim - анимации, attr - атрибуты, style - стили
- bool - булевы значенияб integer - целочисленные значения
- drawable - картинки, mipmap - иконка запуска приложения
- layout - соответственно, layout-ы, о которых мы говорили
- styleable — это добавляемые атрибуты для своих собственных view-элементов

Рис. 1.14: контекстное меню возможных типов ресурсов

Заметим, что с getColor() у нас возникает ошибка:

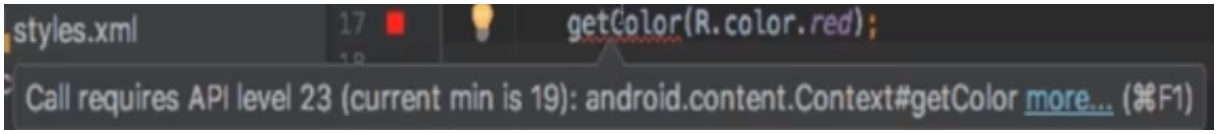


Рис. 1.15: Ошибка несоответствия API

Студия подчеркивает getColor(), потому что Call requires API level 23 (именно про это и говорилось в выборе минимального API). То есть мы не можем просто вызвать getColor(), потому что конкретно этот метод появился в 23-м API. А у нас минимальное API 19. Поэтому не исключено, что приложение будет работать на 19-22 API, и вызов этого метода вызовет ошибку. Можем нажать Alt + Enter, и Surround with — стандартный метод на проверку:

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
  getColor(R.color.red);
}
```

Теперь этот метод запустится только если версия SDK не меньше, чем M - Murchmello(23 API). Либо, как я также говорил, мы можем воспользоваться методом из библиотеки поддержки. И выглядит он так: ContextCompat, (Compat указывает на то, что класс из библиотеки поддержки). getColor() this вместо Context и R.color. red. То есть этот метод не ругается на то, что у нас версия API неправильная. И мы плавно переходим к следующему понятию, следующему термину — это **Context**.

```
ContextCompat.getColor(this, R.color.red);
```

Заметим, что перед **this** появится серая надпись "context:"
Context — это глобальная точка ресурса, это такой очень мощный, много умеющий объект. Он может буквально все: доступ к ресурсам, создание view-элементов прямо в коде, доступ к возможностям телефона и т.д. Поэтому конкретно этот метод просит использовать Context. Но мы вместо Context передаем this потому что activity (а this здесь указывает именно именно на Activity) является одним из наследников контекста. То есть если мы будем щелкать на extends, в конце упремся в контекст. Поэтому во всех методах, которые принимают контекст, можно передавать Activity (либо this, либо activity.this). В нашем случае this можно заменить на LaunchActivity.this т.е. Activity может передавать себя вместо контекста — методы, которые требуют контекста.
Теперь у нас есть Java-код (вся логика в этом коде). И отдельно XML-разметка. Вопрос: как обратиться к View-элементам — к кнопке и к тексту, которые у меня определены в XML-разметке из Java-кода? Все достаточно просто: когда мы определяем View-элемент в разметке и если планируется обращаться к нему из Java-кода, мы обязательно указываем атрибут android:id :

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

Надпись "@+id" означает, что мы добавляем новое id. Мы можем у объектов писать любые id и потом по этим id к ним обращаться. Переименовываем в .xml файле кнопку и текст:

```
<TextView
  android:id="@+id/tv_text"
  ...
<Button
  android:id="@+id/btn_click"
  ...
  app:layout_constraintTop_toBottomOf="@+id/tv_text" />
```

Теперь из .java класса обратимся к элементам:

```
Button mBtnClick = findViewById(R.id.btn_click);
```

Во-первых, мы определяем тип View-элемента. У нас это кнопка. Пишем: тип класса — Button, название — nBtnClick, опять же любое, что нам нужно. И называем метод findViewById, findViewById — вот он, — в который мы передаем указатель на нашу кнопку, передаем id, которое определяли в XML-layout. И, как, вы наверное, уже догадались, указатель на кнопку тоже хранится в R-классе: R.id.btn_click.
К слову Android Studio 3 поддерживает SmartCast, то есть раньше, когда мы обращались к View-элементам, нужно было еще кастовать найденный элемент к нужному типу. Сейчас это не нужно. Раньше findViewById возвращал просто View, а View — это класс-родитель для всех View-элементов, для всех элементов интерфейса. Сейчас он возвращает public <T extends View>View и этот метод определенного AppCompat Activity.
Стандартно в Java переменные, определённые в методе, видны только внутри этого метода. Для того, чтобы сделать mBtnClick глобальной (видной по всей activity), сделаем её переменной поля (а не метода): в начале (перед @Override) напишем

```
private Button  mBtnClick;
...//уже внутри метода onCreate
mBtnClick = findViewById(R.id.btn_click);
```

Т.е. вне всех методов в теле класса мы её объявили. И уберём слово Button в месте, где изначально определяли нашу кнопку. Теперь мы хотим проверить, что эта кнопка та и что она работает. Добавим для неё функционал.
Для **определения нажатия** в Android используется стандартный Java-механизм с **listener**'ами, то есть со слушателями. И выглядит это так: nBtnClick.setOnClickListener (new) — Ctrl, пробел — View.OnClickListener. Что происходит? У класса View, который является родителем всех View-элементов элементов интерфейса имеется поле OnClickListener, и setOnClickListener(), метод setOnClickListener() записывает текущий OnClickListener, аргумент, в это поле. И при нажатии на кнопку

выполняется вызов метода onClick() этого listener. То есть если убрать всю мишуру, то при нажатии на эту кнопку выполнится вот этот метод. Давайте что-нибудь попробуем сделать. Мы можем изменить текст на TextView-элементе, то есть при нажатии на эту кнопку поменяется этот текст.
Сначала в res ⇒ strings.xml создадим строку, на которую будем менять старую:

```
<string name="new_text">this text is new</string>
```

Теперь изменяем наш LaunchActivity.java вот так:

Листинг 1.3: LaunchActivity.java

```
package com.example.azretmagometov.myfirstapp;
import android.os.Build;
import android.support.v4.content.ContextCompat;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class LaunchActivity extends AppCompatActivity {
  private Button  mBtnClick;//объявили кнопку снаружи
  private TextView mText;    //объявили текст снаружи

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

    final String newString = getString(R.string.new_text)
       ;
    mText = findViewById(R.id.tv_text);
    mBtnClick = findViewById(R.id.btn_click);
    mBtnClick.setOnClickListener(new
                View.OnClickListener(){
      @Override
      //при нажатии на кнопку выполняется метод onClick()
      public void onClick(View view){
        mText.setText(newString);
      }//заранее задали в strings.xml
    });
  }
}
```

Один маленький лайфхак с шорткатами: получить строку

```
final String newString = getString(R.string.new_text);
```

можно было так: вводим "getString(R.string.new_text);" , выделяем это и нажимаем Ctrl+Alt+v и получаем "String string = " перед выделенным. Название меняем на newString и отмечаем флажок "final?" т.к. в TextView мы загоняем текст, который мы строку текста, которую мы забрали из ресурса: newString. И так как мы обращаемся к переменной из коллбэка — этот метод onClick() называется коллбэком, — то эта переменная обязательно должна быть final, то есть при компиляции кода JVM должна четко знать, что эта переменная не изменится, пока мы обращаемся к ней.
Нажимаем Run, переключаемся на эмулятор и (если наше приложение в нём обновилось), видим, что при клике текст поменяется на новый.
Теперь про **Bundle**, который мы видим в строчке

```
protected void onCreate(Bundle savedInstanceState) {
```

Bundle — это своего рода обертка над массивом данных, и он используется для передачи данных либо для сохранения и считывания каких-либо значений:

```
Bundle bundle = new Bundle(); //Создадим Bundle
bundle.putString("KEY","SAD");
//вызвали метод по ключу и значению
```

значения в Bundle хранятся с помощью механизма KeyValue, то есть так же, как они хранятся в HashMap. Помимо строкового типа Bundle поддерживает byte, char, float, integer, integer array (T.e. все примитивные типы, их массивы и строки). Еще он поддерживает **Serializable и Parcelable** — это специальный тип, который позволяет объекту класса сохранять свое состояние. То есть если наш класс помечен как Serializable либо Parcelable, то он его можно сериализовать и передавать в этом сериализованном виде в, допустим, в другой Activity. Bundle используются для передачи данных еще в другой Activity. И потом в другом Activity просто развернуть (десериализовать) этот файл и получить его в том же состоянии, в котором мы его оставили. Разница между Serializable и Parcelable в том, что первый — это стандартный Java-механизм, и он использует рефлексию для работы. Из-за этого он ощутимо медленнее. Parcelable — это механизм, который появился в Android. В нем мы уже сами прописываем, как что сериализовать и десериализовать. То есть мы можем сказать какие поля класса должны быть сериализованы. Или при десериализации, чтобы эти поля класса принимали вот это значение, а не то, которое они имеют на данный момент.