



План занятия

1. [Android](#)
2. [SDK](#)
3. [Android Studio](#)
4. [Первое приложение](#)
5. [Эмулятор](#)
6. [Activity](#)
7. [AppCompatActivity](#)
8. [UI](#)
9. [Итоги](#)



ANDROID



ANDROID

[Android](#) — операционная система (ОС), предназначенная для различных устройств:

- смартфоны, ← наша цель
- планшеты,
- часы,
- автомобильные системы,
- ноутбуки,
- и т.д.

ОС

ОС — это слой между прикладным ПО и «железом».



Прикладное ПО (далее — приложения) — это программы, которые решают задачи пользователя: обмен сообщениями, фото.

Для этих приложений ОС предоставляет сервисы. Например, отправка сообщений по сети, доступ к камере и т.д.



ОС

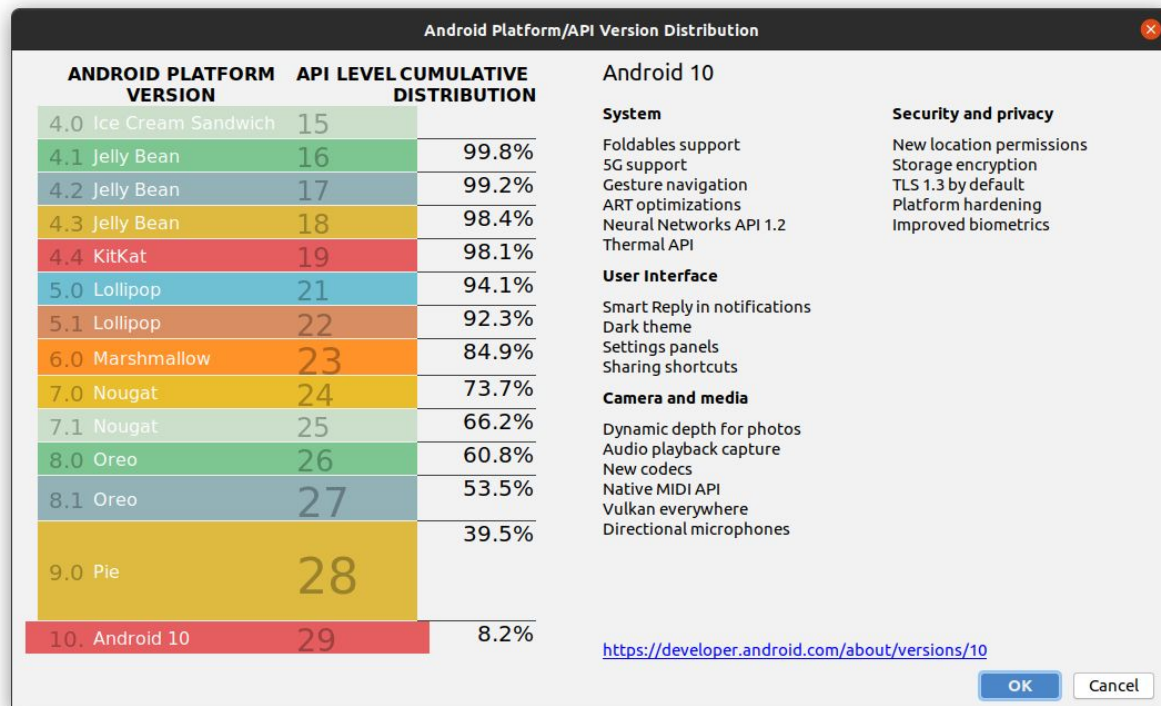
Сами приложения не могут напрямую воспользоваться устройствами, они могут только попросить Android предоставить соответствующий сервис.

Поэтому, если мы хотим разрабатывать приложения, нам нужно научиться взаимодействовать с Android для получения нужных нам сервисов.

Мы будем использовать термины Android, ОС, платформа как взаимозаменяемые, как и в официальной документации.

VERSIONS

Android не стоит на месте и каждый год выпускаются новые версии, содержащие новые возможности как для пользователей, так и для разработчиков.



Мы с вами будем использовать последнюю версию, чтобы вы знали о самых новых возможностях, но и старые тоже будем обсуждать.

важна совместимость вашего приложения с предыдущими версиями ОС



SDK



SDK

SDK (Software Development Kit) — набор библиотек и инструментов, позволяющих вам создавать приложения и использовать API платформы, в нашем случае — Android.

SDK

SDK состоит из различных компонентов:

- Android Studio (среда разработки);
- Android Emulator (эмулятор устройства);
- Android SDK Build Tools (инструменты сборки);
- Android SDK Platform (набор классов для конкретной версии платформы);
- Android SDK Platform Tools (инструменты для взаимодействия с платформой: загрузка файлов, отладка);
- Android SDK Tools (инструменты управления эмулятором, анализа приложений и их «защиты»).



ProGuard

SDK

После установки (установка будет произведена в рамках настройки Android Studio), каждый из компонентов SDK разместится в своём каталоге:

- Android Studio — зависит от того, куда установите.
- Android Emulator — [emulator](#).
- Android SDK Build Tools — [build-tools](#).
- Android SDK Platform — [platforms](#).
- Android SDK Platform Tools — [platform-tools](#).
- Android SDK Tools — [tools](#).

На начальном этапе вы не будете напрямую запускать эти инструменты — они будут запускаться из интерфейса Android Studio, но понимание быть должно.



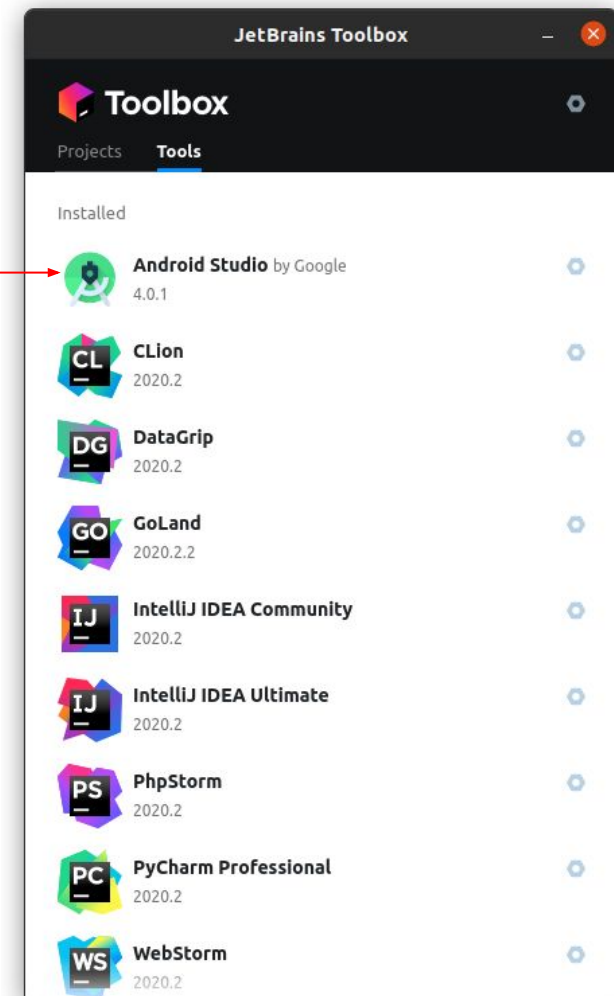
ANDROID STUDIO



ANDROID STUDIO

Разрабатывать под Android можно и в IDEA, но есть специальная IDE, построенная на платформе IDEA — [Android Studio](#). Её установка описана в материалах к ДЗ.

Именно её мы и будем использовать. Понадобится версия 4.0+.





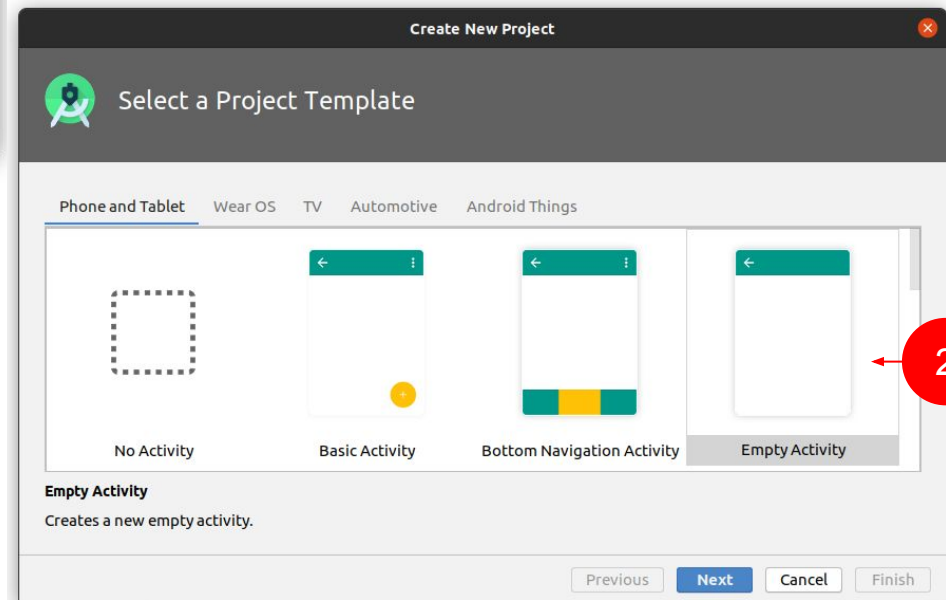
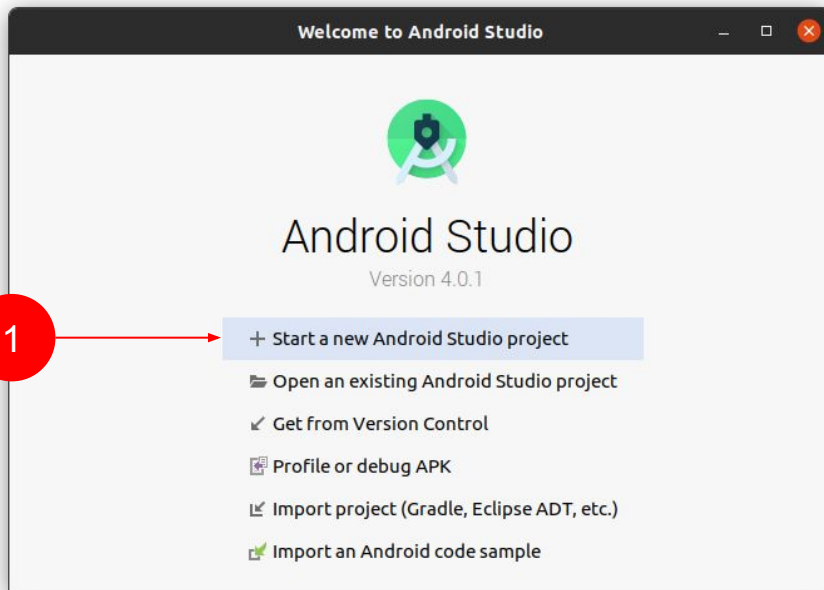
ПЕРВОЕ ПРИЛОЖЕНИЕ

ПОДХОД

Мы будем использовать практико-ориентированный подход, т.е. показывать, что и как делается, и в процессе объяснять необходимую теорию.

Давайте создадим наше первое приложение, которое могло бы хоть что-то отображать на экране. Но это будет не просто «Hello World», это будет «Deep Dive Hello World». Мы достаточно глубоко будем погружаться в некоторые аспекты, чтобы у вас сложилось понимание того, как оно на самом деле устроено.

СОЗДАНИЕ ПРОЕКТА



ACTIVITY

«An activity is a single, focused thing that the user can do. Almost all activities interact with the user, so the Activity class takes care of creating a window for you in which you can place your UI.» — [официальная документация](#).

«Activity* — это сфокусированная единица того, что может делать пользователь. Почти все activity взаимодействуют с пользователем, поэтому класс Activity отвечает за создание для вас окна, в котором вы можете разместить ваш UI» — вольный перевод.

Примечание*: мы не будем переводить названия ключевых терминов (Activity — Активность), потому так почти никто не делает, и коллеги будут вас переспрашивать, что такое «Активность»?

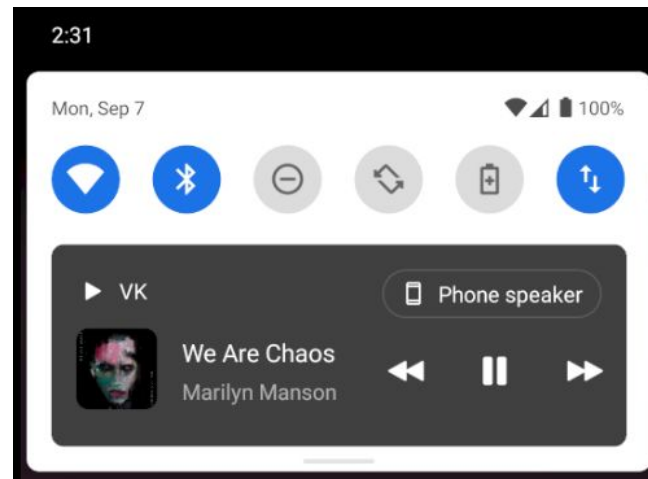
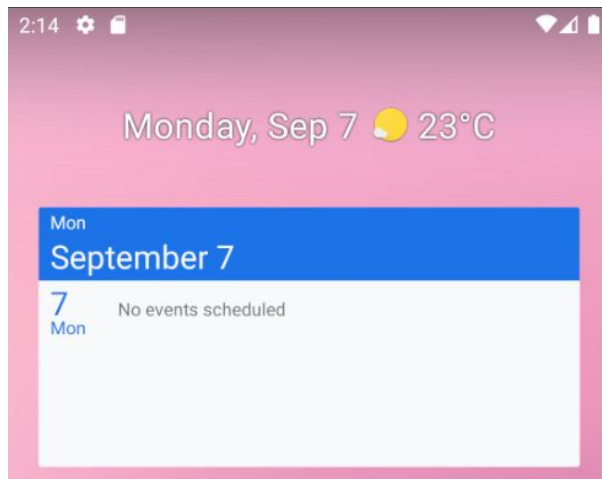
ENTRY POINT

В отличие от приложений, который мы разрабатывали до этого (которые начинали работу с `main`), в Android точкой входа в приложение (т.е. откуда приложение может начать свою работу) может быть один из 4-х типов компонентов:

- Activity — экран с пользовательским интерфейсом.
- Service — фоновая задача.
- Broadcast Receiver — получение событий от системы. Например, включение устройства, наступление определённой даты и т.д.
- Content Provider — предоставление данных другим приложениям.

ЭКРАН

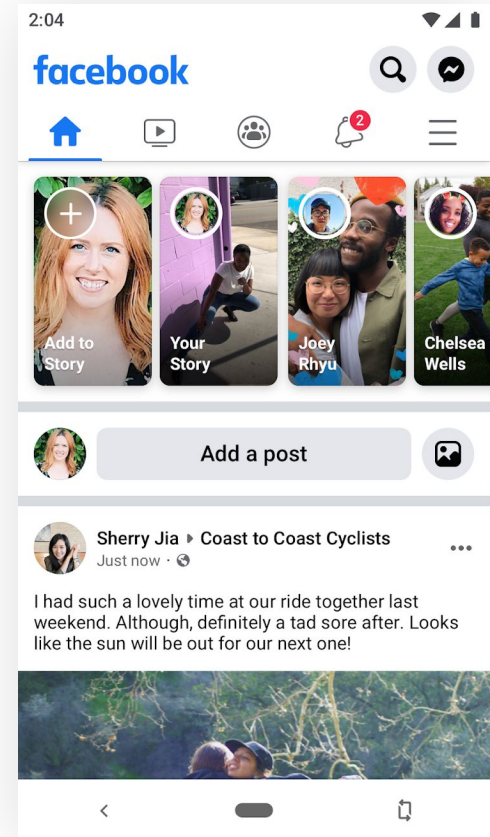
Нужно отметить, что когда мы будем говорить «Экран» в терминах UI, то не будем иметь в виду виджеты рабочего стола или управляющие элементы в верхней панели:



ACTIVITY

В большинстве случаев Activity занимает целиком весь экран, т.е. очень высокоуровнево её (Activity) можно представлять как отдельный экран нашего приложения, с которым взаимодействует пользователь (например, экран просмотра ленты новостей*): ● —————→

Примечание*: сопоставление экран = Activity не всегда верно, т.к. Android предоставляет и другие объекты для организации экранов.



play.google.com

CONFIGURATION

Create New Project

Configure Your Project

Name: NMedia

Package name: ru.netology.nmedia

Save location: /home/coursar/AndroidStudioProjects/NMedia

Language: Kotlin

Minimum SDK: API 29: Android 10.0 (Q)

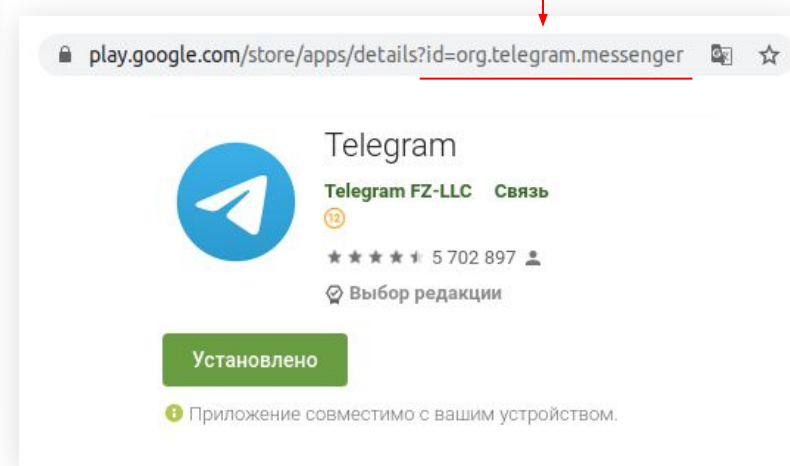
Your app will run on approximately 8.2% of devices. [Help me choose](#)

☐ Use legacy android.support libraries

Empty Activity
Creates a new empty activity.

Previous Next Cancel Finish

уникальный id в
Google Play



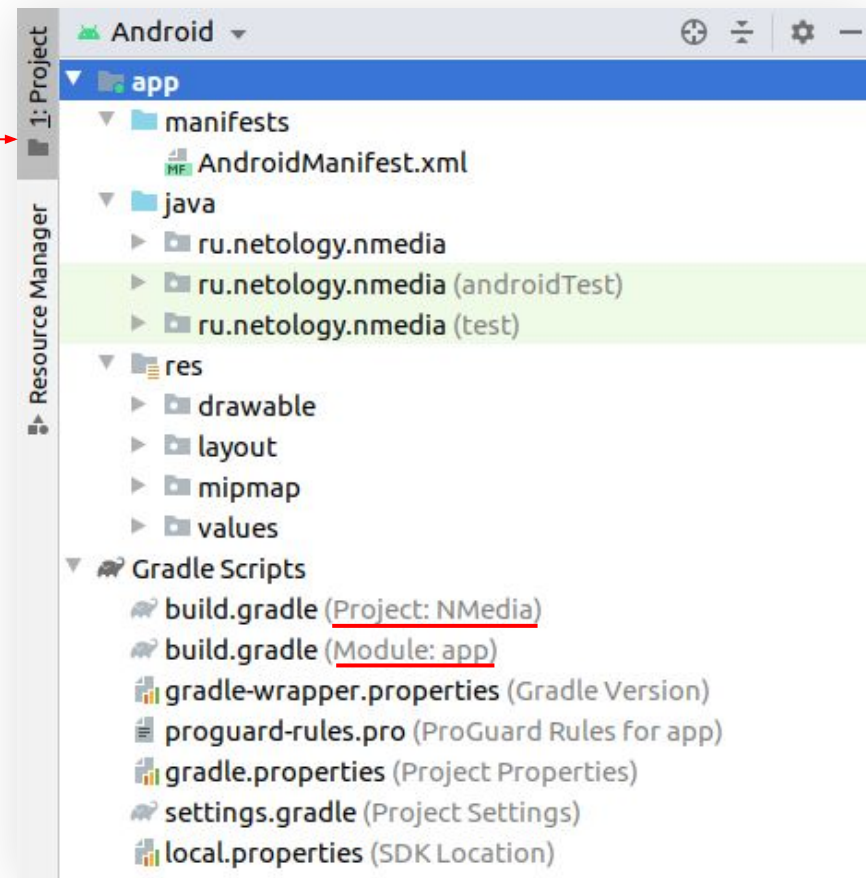
1. Имя пакета — оно же id для вашего приложения.
2. Язык — Kotlin.
3. Minimum SDK — совместимость с какой версией будем поддерживать.

будем обсуждать позже

СТРУКТУРА ПРОЕКТА

По умолчанию в боковой панели
открыт вид Project:

Он похож на то, с чем мы имели дело
раньше, за исключением следующего:
есть деление
на проект (Project) и модуль (Module).

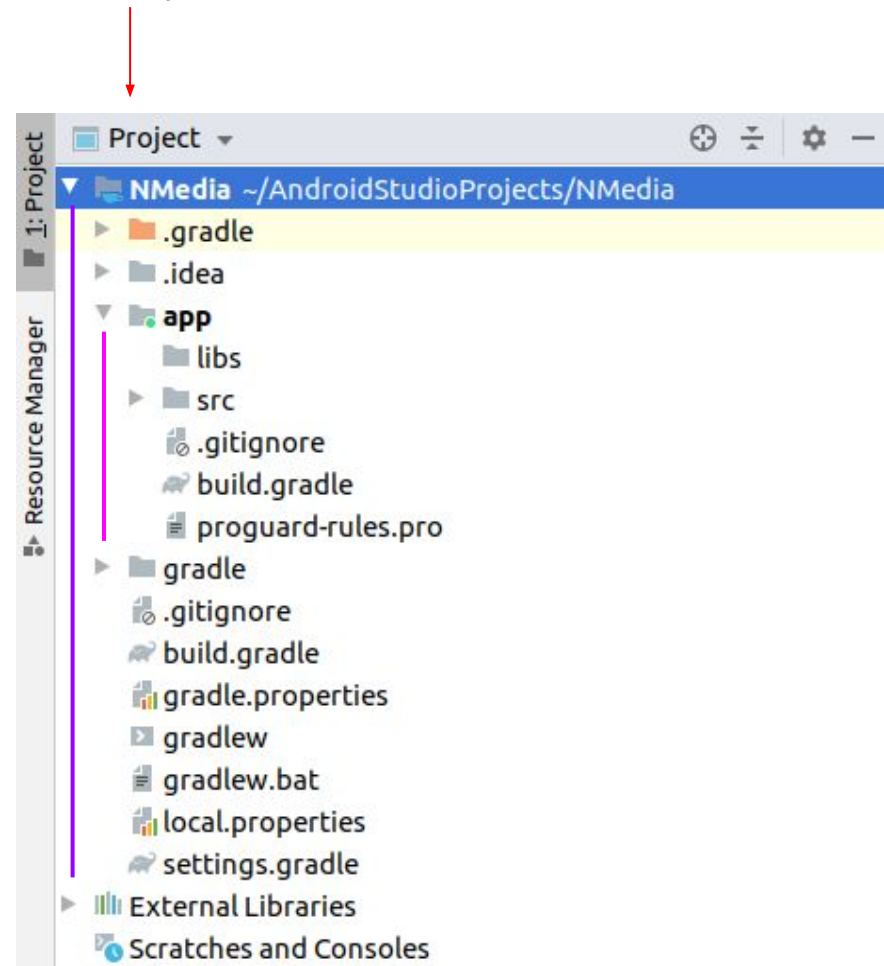


PROJECT vs MODULE

Gradle позволяет создавать мультимодульные проекты, разделяя конфигурационные файлы: **root** и **subprojects**.

Основная идея: настройки из build.gradle root проекта общие для всех подпроектов.

вид Project



PROJECT vs MODULE

root build.gradle

```
1 // Top-level build file where you can add configuration options common to all sub-projects/modules.
2 buildscript {
3     ext.kotlin_version = "1.4.31"
4     repositories {
5         google()
6         jcenter()
7     }
8     dependencies {
9         classpath "com.android.tools.build:gradle:4.1.3 "
10        classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_version"
11        ...
12    }
13 }
14
15
16
17 allprojects {
18     repositories {
19         google()
20         jcenter()
21     }
22 }
23
24 task clean(type: Delete) {
25     delete rootProject.buildDir
26 }
```



PROJECT vs MODULE

В данном базовом курсе мы не будем создавать другие подпроекты кроме app, поэтому нужно запомнить следующее правило: все изменения мы вносим только в [build.gradle](#) app (Module, а не Project).

СТРУКТУРА ПРОЕКТА

Разберём ключевые файлы и каталоги:

- `manifests/AndroidManifest.xml` — описание нашего проекта;
- `java/ru.netology.nmedia*` — исходные файлы нашего приложения;
- `java/ru.netology.nmedia (androidTest/test)` — автотесты;
- `res` — ресурсы приложения (картинки, файлы переводов);
- Gradle Scripts — скрипты Gradle (пока нас не интересуют, т.к. там уже всё настроено).

Все эти каталоги мы рассмотрим в нашем курсе чуть позже, пока же наша задача — запустить приложение и увидеть результат работы.

Примечание*: исходники Kotlin вполне неплохо живут в каталоге `java`. На самом деле это просто настройки Gradle и плагина — где искать файлы Kotlin.



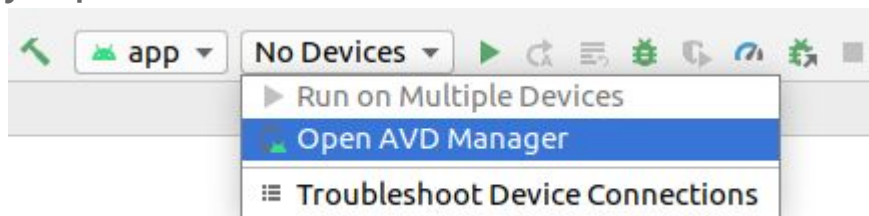
ЭМУЛЯТОР

ЭМУЛЯТОР



Для запуска приложения требуется либо физическое устройство, либо эмулятор — специальное ПО, которое эмулирует физическое устройство путём создания виртуальных.

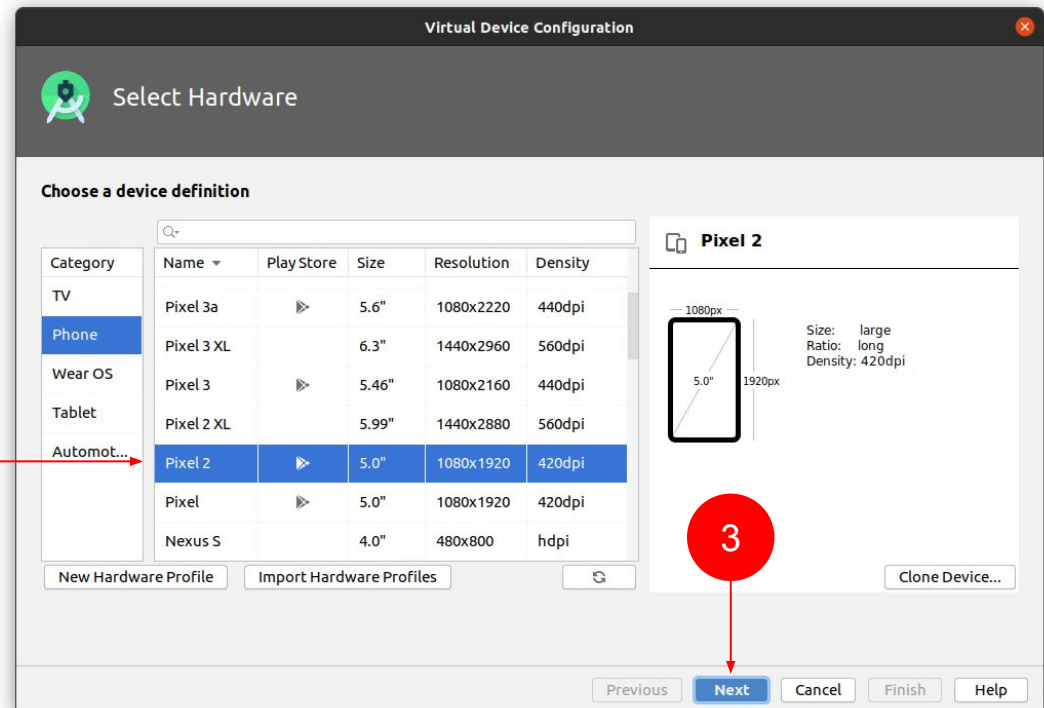
Установка Android Studio содержит в своём составе Android Virtual Device Manager (AVD), который позволяет создавать виртуальные устройства:



VIRTUAL DEVICE

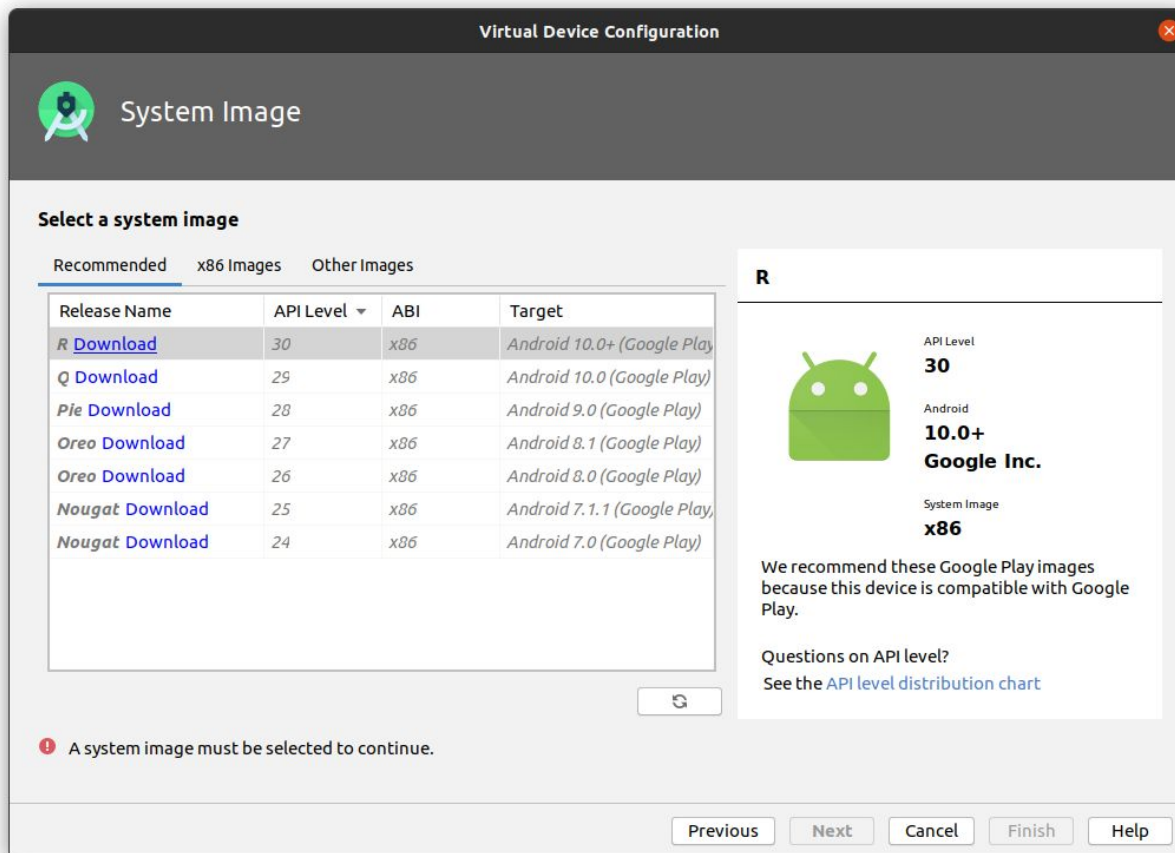


Выбранное устройство будет эмулировать характеристики реального (объём памяти, разрешение и т.д.)



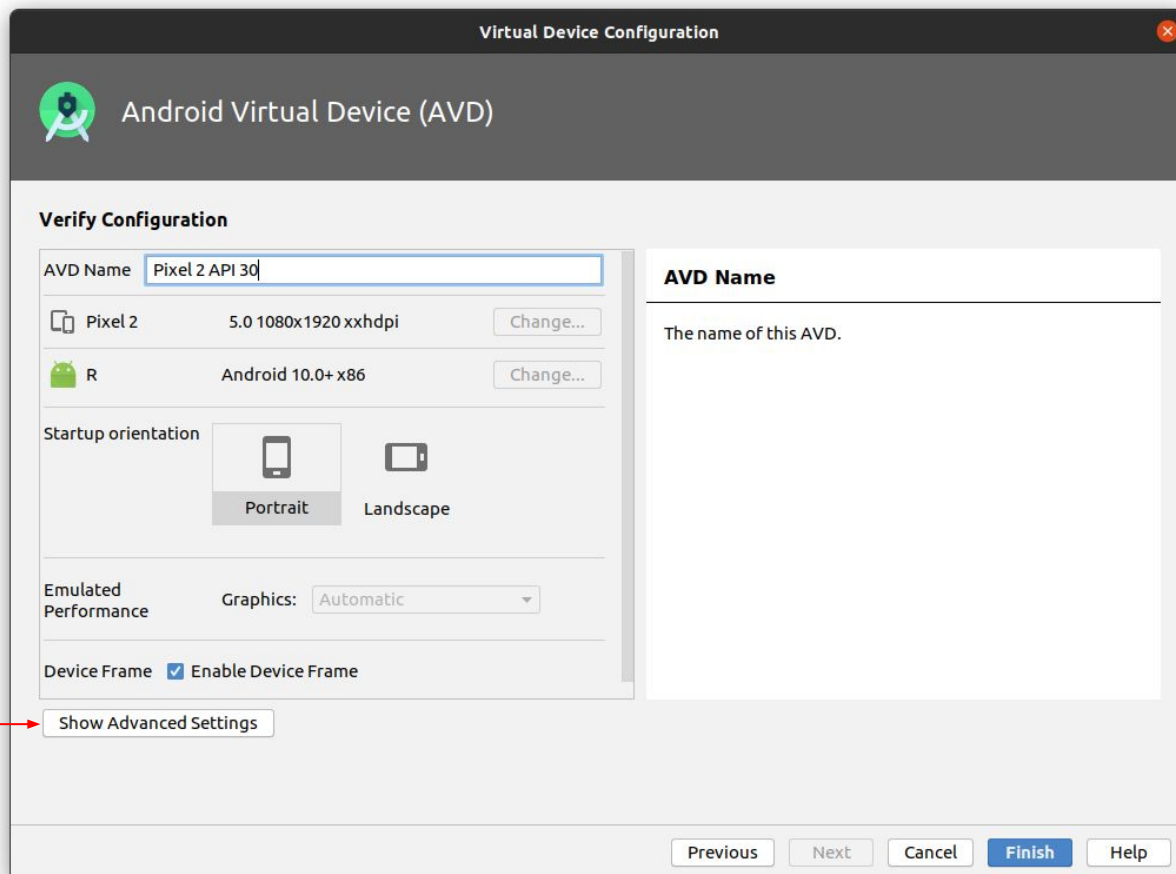
ЭМУЛЯТОР

На виртуальное устройство необходимо поставить ОС определённой версии (для этого образ ОС нужно скачать):



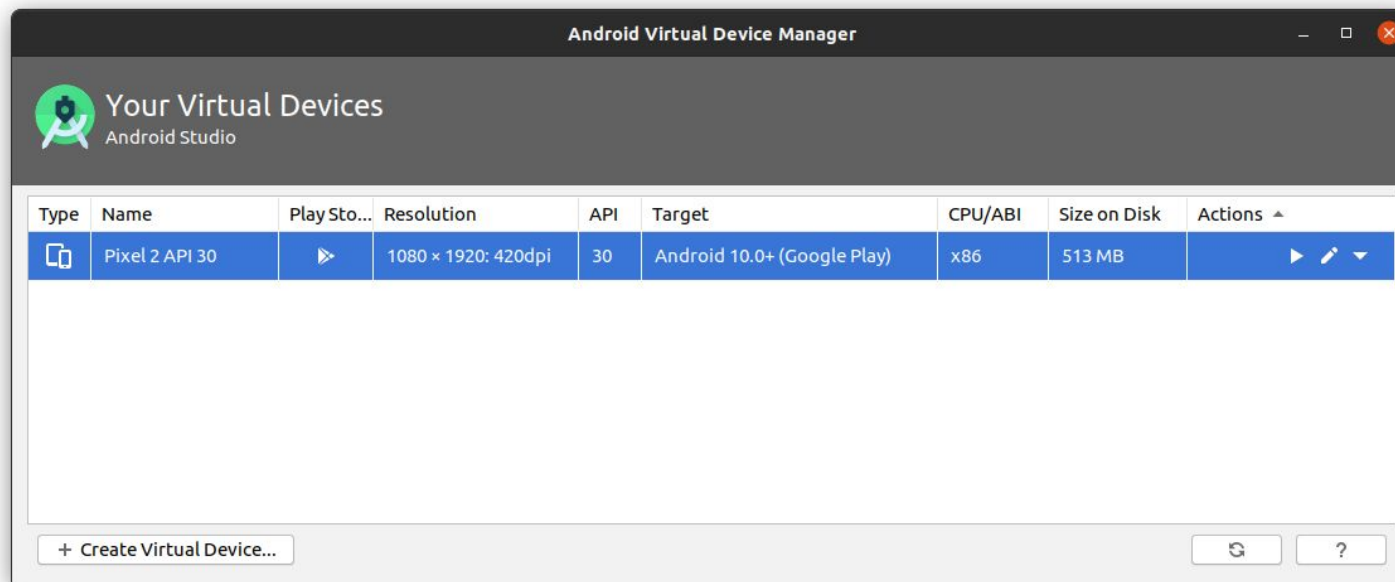
НАСТРОЙКА

После чего вы можете изменить настройки устройства, если это необходимо:



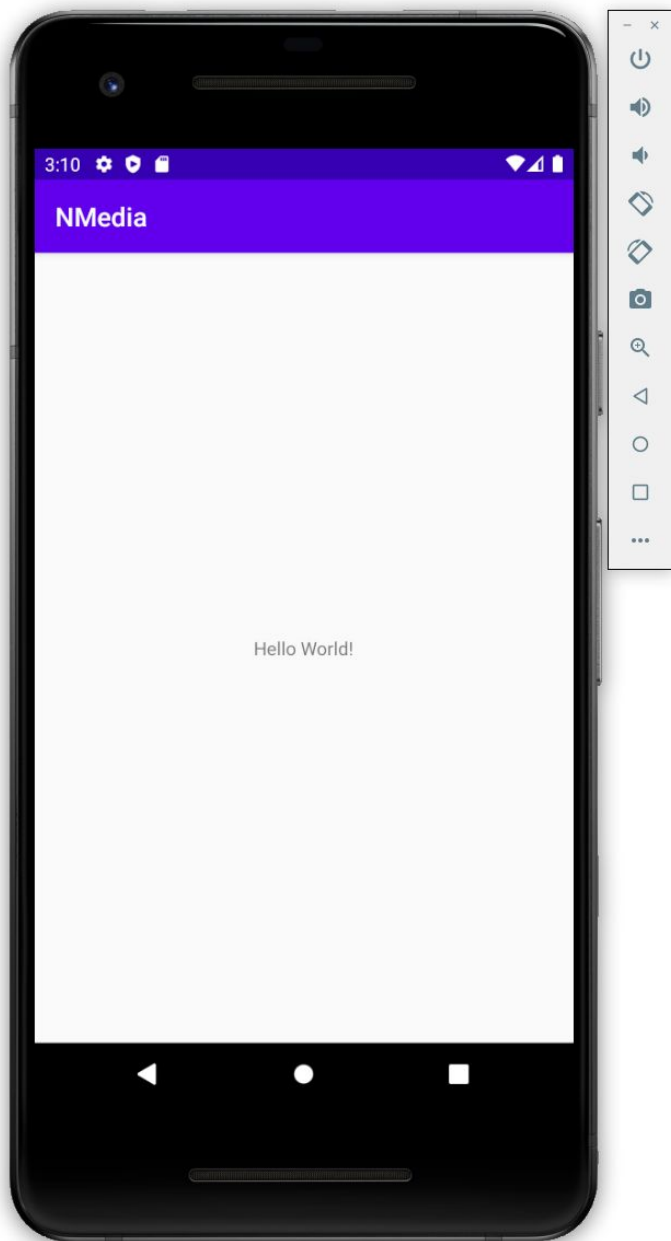
VIRTUAL DEVICES

Далее можно будет выбрать созданное устройство в списке виртуальных устройств:





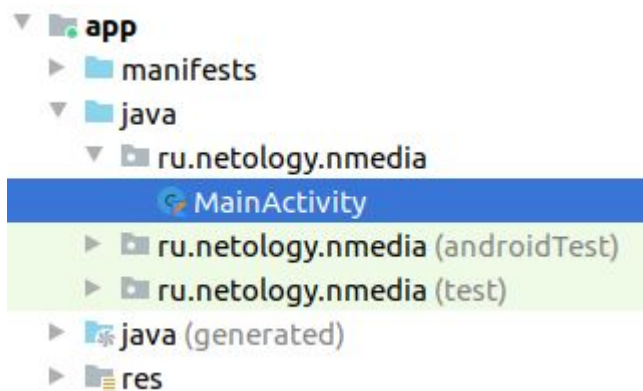
ACTIVITY



ЗАПУСК ПРИЛОЖЕНИЯ

ACTIVITY

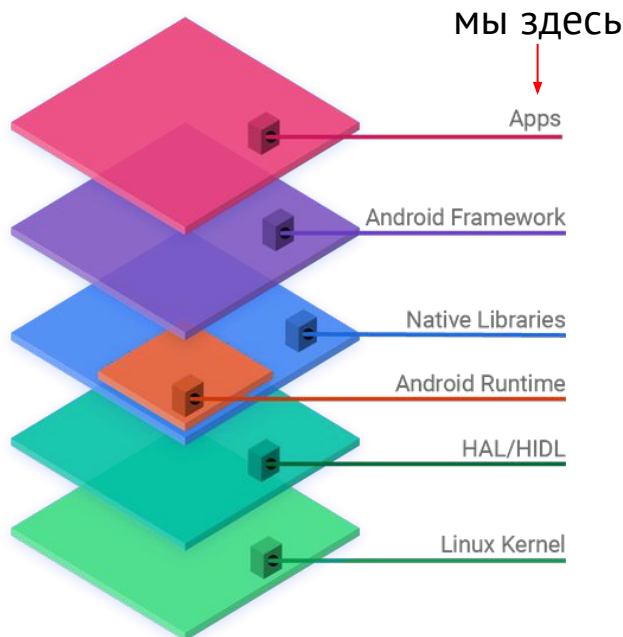
Начнём разбирать код нашего приложения. Никакого `main` у нас нет, зато есть `MainActivity`:



```
1 package ru.netology.nmedia
2
3 import ...
4
5
6 class MainActivity : AppCompatActivity() {
7     override fun onCreate(savedInstanceState: Bundle?) {
8         super.onCreate(savedInstanceState)
9         setContentView(R.layout.activity_main)
10    }
11 }
```

FRAMEWORK

Здесь важно запомнить следующую картинку:



source.android.com

Правила игры поменялись: если раньше мы писали функцию `main`, которую запускала JVM и сами управляли жизнью нашего приложения (решали, когда ему завершиться), то теперь это решает Framework.



FRAMEWORK

Q: что это значит?

A: это значит, что Framework диктует вам, что вы должны делать, чтобы ваше приложение запустилось. Например, вы должны отнаследоваться от [AppCompatActivity](#) (или [Activity](#)). Если вы этого не сделаете, то Framework не запустит ваше приложение.

И это не всё, Framework ещё определяет какие member functions он будет запускать и когда (например, [onCreate](#)). И вы можете только в этих member functions выстраивать логику своего приложения.

Но откуда он узнает, что запустить нужно именно эту Activity? Ведь, как мы говорили, их может быть много.

MANIFEST

Для настройки приложения существует файл [AndroidManifest.xml](#):



```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3     package="ru.netology.nmedia">
4
5     <application
6         android:allowBackup="true"
7         android:icon="@mipmap/ic_launcher"
8         android:label="NMedia"
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>
```

Пока нам достаточно знать, что он существует, настраивать мы его будем на следующих лекциях.



XML

[XML](#) (eXtensible Markup Language) — расширяемый язык разметки, специальный язык, созданный для обмена структурированной информацией.

В нашем с вами случае, мы будем встречаться с ним в виде файла [AndroidManifest.xml](#), в котором описаны настройки нашего приложения и файлов в каталоге [res](#) (ресурсы).



XML

XML документ начинается с объявления, которое выглядит следующим образом: `<?xml version="1.0" encoding="UTF-8"?>`

Это означает, что наш документ использует стандарт 1.0 и написан в кодировке UTF-8.

XML

Сам документ состоит из тегов. Тег — это специальный структурный элемент, который состоит из 4 частей:

- открывающий тег `<manifest>`
- закрывающий тег `</manifest>`
- атрибуты (пишутся только в открывающем теге):
`<manifest attribute="value">` (значения пишутся в кавычках)
- содержимого (пишется между открывающим и закрывающим тегом — может быть либо текст, либо другой тег)

В случае, если у тега нет содержимого (например, `action`), то допускается использовать сокращённую форму:

`<action android:name="android.intent.action.MAIN"/>`

XML

Q: откуда берутся теги, атрибуты и т.д.? Где это написано?

A: к xml-документу обычно идёт [XSD](#) (Xml Schema Definition), она и определяет правила:

- какие теги можно использовать,
- какие атрибуты есть у тегов,
- какие теги какое содержимое могут иметь.

С Android всё немного сложнее, вы не найдёте XSD, поскольку правила либо зашиты в Android Studio, либо определяются путём «парсинга» обработки файлов вашего проекта. Об этом позже, когда будем говорить о создании собственных элементов.

XML

```
@DefinesXml
@Styleable("AndroidManifest")
public interface Manifest extends ManifestElement {
    Application getApplication(); // внутри может быть один application
    ...
}
```

```
@Styleable("AndroidManifestApplication")
public interface Application extends ManifestElement {
    List<Activity> getActivities(); // внутри может быть много activity
    ...
}
```

```
@Styleable("AndroidManifestActivity")
public interface Activity extends ApplicationComponent {
    @Attribute("name")
    @Required // атрибут name обязателен
    AndroidAttribute<PsiClass> getActivityClass();
    ...
}
```

XML

Пробуем убрать атрибут name:

```
<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>
        <intent
            'name' attribute should be defined
            Define name attribute Alt+Shift+Enter More actions... Alt+Enter
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>
```

XML

Конечно же, подобный вариант с поиском по исходникам IDE не подходит для работы, поэтому на сайте есть исчерпывающая документация:

<https://developer.android.com/guide/topics/manifest/manifest-element>

<https://developer.android.com/guide/topics/manifest/activity-element>

Важно: смотрите, пожалуйста, сначала в документацию и только потом на StackOverflow.

MANIFEST

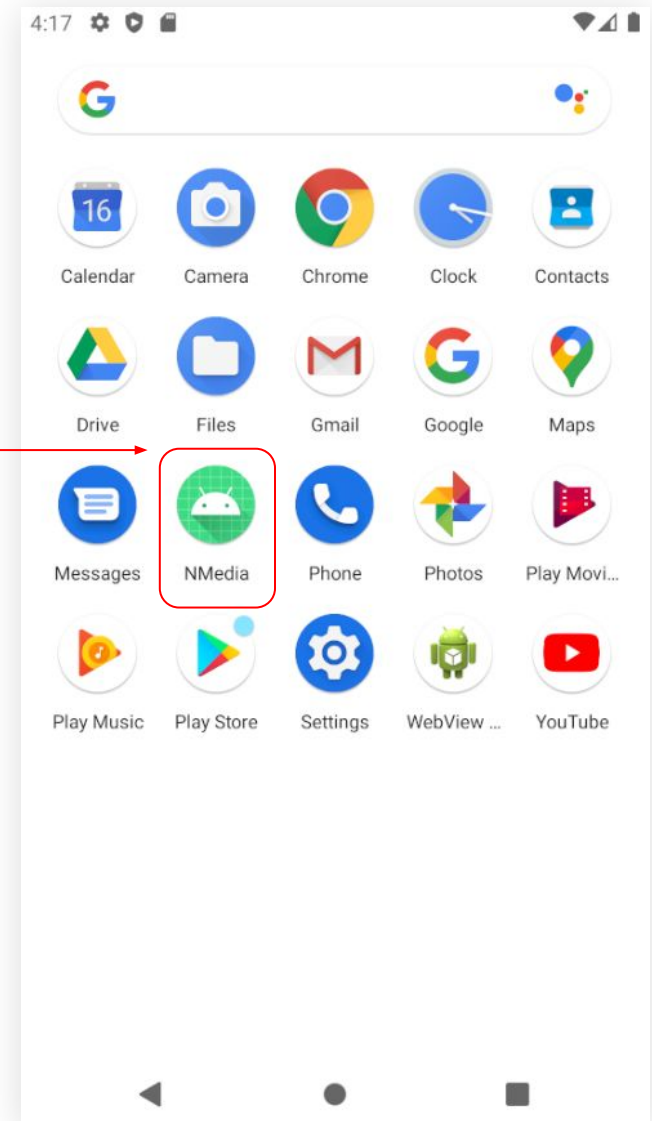
В нашем манифесте указано, что Activity `MainActivity` будет выступать:

- в качестве основной точки входа в приложение `android.intent.action.MAIN`;
- значок Activity (либо всего приложения) `android:icon` следует разместить в лаунчере (средстве запуска приложений).

Детально с Intent мы будем разбираться на одной из следующих лекций.

LAUNCHER

Эта пара (MAIN + LAUNCHER) необходима, чтобы мы увидели наше приложение в лаунчере: ●



MANIFEST

Если мы там что-то поменяем (поломаем):

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3     package="ru.netology.nmedia">
4
5     <application
6         android:allowBackup="true"
7         android:icon="@mipmap/ic_launcher"
8         android:label="NMedia"
9         android:roundIcon="@mipmap/ic_launcher_round"
10        android:supportsRtl="true"
11        android:theme="@style/AppTheme">
12        <activity android:name=".MyActivity">
13            <intent-filter>
14                <action android:name="android.intent.action.MAIN">
15                <category android:name="android.intent.category.LAUNCHER">
16            </intent-filter>
17        </activity>
18    </application>
19
20 </manifest>
```

Class referenced in the manifest, ru.netology.nmedia.MyActivity, was not found in the project or the libraries

Unresolved class 'MyActivity'

Create class 'MyActivity' Alt+Shift+Enter More actions... Alt+Enter

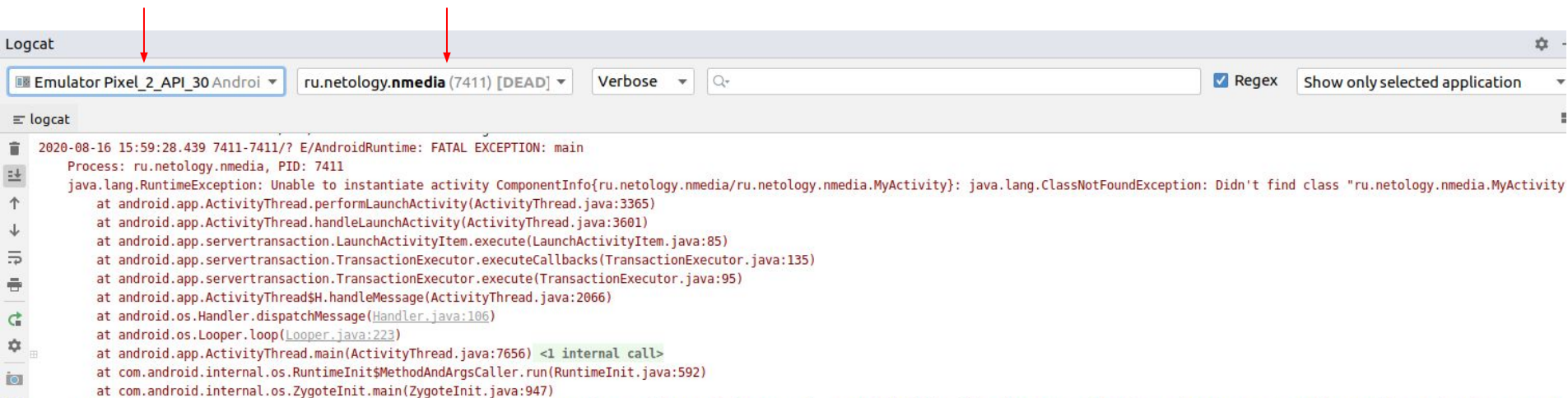
То приложение просто не запустится. К этому нужно привыкнуть: логические ошибки в xml-файлах можно обнаружить только в логах.

LOGCAT

Для просмотра логов есть специальная вкладка Logcat:

устройство

приложение



LOGCAT

java.lang.RuntimeException: Unable to instantiate activity

ComponentInfo{ru.netology.nmedia/**ru.netology.nmedia.MyActivity**}:

java.lang.ClassNotFoundException: **Didn't find class "ru.netology.nmedia.MyActivity"**

at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:3365)

at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:3601)

at android.app.servertransaction.LaunchActivityItem.execute(LaunchActivityItem.java:85)

... // часть вывода опущено

Caused by: java.lang.ClassNotFoundException: Didn't find class "ru.netology.nmedia.MyActivity"

at dalvik.system.BaseDexClassLoader.findClass(BaseDexClassLoader.java:207)

at java.lang.ClassLoader.loadClass(ClassLoader.java:379)

at java.lang.ClassLoader.loadClass(ClassLoader.java:312)

... // часть вывода опущено



APPCOMPACTIVITY

APPCOMPATACTIVITY

Q: почему мы говорим об [Activity](#), а сами наследуемся от [AppCompatActivity](#)?

A: разработчики Android стремились обеспечить совместимость с предыдущими версиями платформы, поэтому, добавляя новые классы в систему, стали придерживаться следующего подхода:

- в новых версиях предоставлять новые возможности «нативно», т.е. реализованными на уровне системы;
- а в старых версиях предоставляя реализацию в виде библиотеки [appcompat](#).

APPCOMPACTACTIVITY

Поэтому для поддержки старых устройств мы используем библиотеку `appcompat`, прописывая её в виде зависимости в Gradle*:

```
37 dependencies {  
38     implementation 'androidx.core:core-ktx:1.3.2'  
39     implementation 'androidx.appcompat:appcompat:1.2.0'  
40     implementation 'com.google.android.material:material:1.3.0'  
41     implementation 'androidx.constraintlayout:constraintlayout:2.0.4'  
42     testImplementation 'junit:junit:4.13.2'  
43     androidTestImplementation 'androidx.test.ext:junit:1.1.2'  
44     androidTestImplementation 'androidx.test.espresso:espresso-core:3.3.0'  
45 }
```

Примечание*: на самом деле она там уже прописана —конструктор постарался.



JETPACK

Библиотека `appcompat`, как и ряд других библиотек и инструментов, собраны в набор, который называется [Android Jetpack](#).

Библиотеки из Jetpack находятся в пакете `androidx.*` и вынесены из платформы для того, чтобы мы с вами могли подключать и использовать в нашем проекте наиболее свежие версии, не полагаясь на версию платформы на устройстве пользователя.



JETPACK

Jetpack Libraries [Explore all libraries](#)

Filter by keyword or use case

* Popular and often-used libraries are listed first

activity *	Access composable APIs built on top of Activity.
appcompat *	Allows access to new APIs on older API versions of the platform (many using Material Design).
camera *	Build mobile camera apps.
compose *	Define your UI programmatically with composable functions that describe its shape and data dependencies.
databinding *	Bind UI components in your layouts to data sources in your app using a declarative format.
fragment *	Segment your app into multiple, independent screens that are hosted within an Activity.
hilt *	Extend the functionality of Dagger Hilt to enable dependency injection of certain classes from the androidx libraries.

VIEW MORE

APPCOMPACTIVITY

?

Как вы думаете, какие возможности нам предоставляет наследование?

APPCompatActivity

Наследование позволяет нам переиспользовать тот код, который написан в базовом классе (хоть он и написан на Java):

```
public class AppCompatActivity extends FragmentActivity implements AppCompatActivity,
    TaskStackBuilder.SupportParentable, ActionBarDrawerToggle.DelegateProvider {

    private AppCompatActivity mDelegate;
    private Resources mResources;

    /** Default constructor for AppCompatActivity. All Activities must have a default constructor ...*/
    public AppCompatActivity() {
        super();
    }

    /** Alternate constructor that can be used to provide a default layout ...*/
    @ContentView
    public AppCompatActivity(@LayoutRes int contentLayoutId) { super(contentLayoutId); }

    @Override
    protected void attachBaseContext(Context newBase) {...}

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        final AppCompatActivity delegate = getDelegate();
        delegate.installViewFactory();
        delegate.onCreate(savedInstanceState);
        super.onCreate(savedInstanceState);
    }
}
```


APPSCOMPACTACTIVITY

Q: остался последний вопрос — мы видим на экране текст, но нигде внутри нашей Activity его не задаём.

A: в Android отделять код от внешнего представления (или макета UI). Поэтому код будет размещён в нашей Activity, а UI хранится отдельно, в каталоге [res/layout](#):

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:app="http://schemas.android.com/apk/res-auto"
4     xmlns:tools="http://schemas.android.com/tools"
5     android:layout_width="match_parent"
6     android:layout_height="match_parent"
7     tools:context=".MainActivity">
8
9     <TextView
10         android:layout_width="wrap_content"
11         android:layout_height="wrap_content"
12         android:text="Hello World!"
13         app:layout_constraintBottom_toBottomOf="parent"
14         app:layout_constraintLeft_toLeftOf="parent"
15         app:layout_constraintRight_toRightOf="parent"
16         app:layout_constraintTop_toTopOf="parent" />
17
18 </androidx.constraintlayout.widget.ConstraintLayout>
```

LAYOUT

Установка UI (а именно создание реальных объектов «из xml») происходит благодаря вот этой строке в нашей Activity:

```
6 class MainActivity : AppCompatActivity() {  
7     override fun onCreate(savedInstanceState: Bundle?) {  
8         super.onCreate(savedInstanceState)  
9         setContentView(R.layout.activity_main)  
10    }  
11 }
```

где `R.layout.activity_main` — это специальная «константа», которая позволяет из Kotlin-кода ссылаться на определённый ресурс типа layout (в данном случае). Чаще всего, следуют правилу: к каждому Activity прикрепляется свой layout.

Осталось только понять, что такое `R` и откуда оно берётся.

R.java

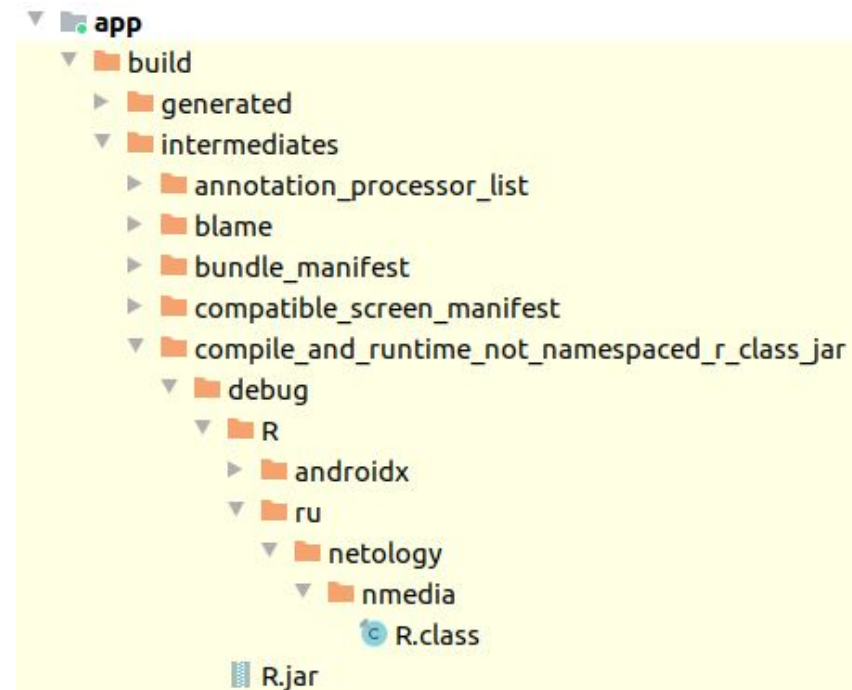
В состав Build Tools входит инструмент [appt2](#) (Android Asset Packaging Tool), задачей которого является компиляция и упаковка ресурсов приложения.

В части XML — он преобразовывает все файлы в формат flat (вид Project, каталог `app/build/intermediates/res/merged/debug`):



R.java

Кроме того, генерируется файл R.java, который содержит идентификаторы тех самых ресурсов (с предыдущего слайда).



```
public final class R {  
    private R() {  
    }  
  
    public static final class layout {  
        // ...  
        public static final int activity_main = 2131361820;  
        // ...  
  
        private layout() {  
        }  
    }  
  
    public static final class mipmap {  
        public static final int ic_launcher = 2131427328;  
        public static final int ic_launcher_round = 2131427329;  
  
        private mipmap() {  
        }  
    }  
}
```

РЕСУРСЫ

Таким образом, мы всегда используем готовые константы из [R](#), чтобы ссылаться на ресурсы.



UI



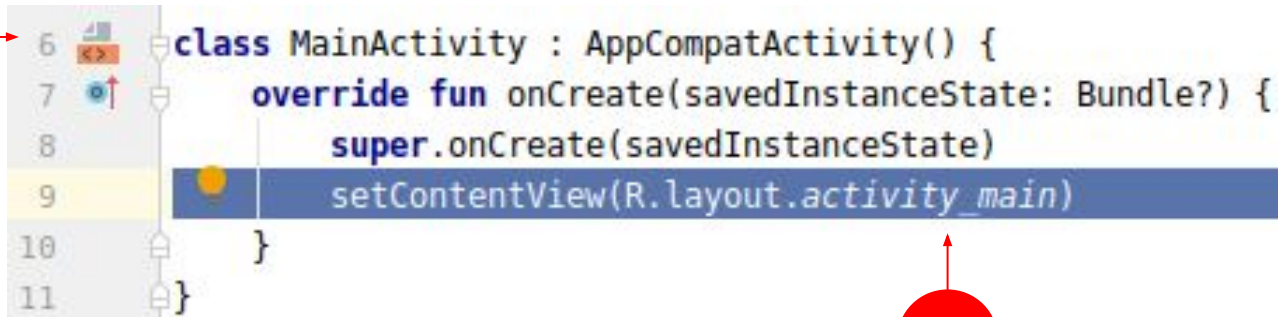
UI

Мы разобрались с ресурсами, теперь давайте поговорим о том, как строить интерфейсы.

Android Framework предоставляет готовые «компоненты» для построения интерфейсов. Осталось лишь разобраться, как с ними работать из кода, например, программно менять текст.

LAYOUT

Откроем файл [layout/activity_main.xml](#). Сделать это можно либо в панели проекта, либо кликнув на иконку (1) или resource id (2) в файле Activity:



The screenshot shows the `MainActivity` class in an IDE. A red circle with the number '1' points to the Android Studio icon in the left margin of line 6. A red circle with the number '2' points to the resource identifier `R.layout.activity_main` in the `setContentView` call on line 9.

```
6 class MainActivity : AppCompatActivity() {  
7     override fun onCreate(savedInstanceState: Bundle?) {  
8         super.onCreate(savedInstanceState)  
9         setContentView(R.layout.activity_main)  
10    }  
11 }
```




CODE vs DESIGN MODE

При работе с файлами layout доступно два режима работы:

1. Code Mode — мы редактируем xml в виде текста;
2. Design Mode — мы редактируем xml в визуальном режиме.

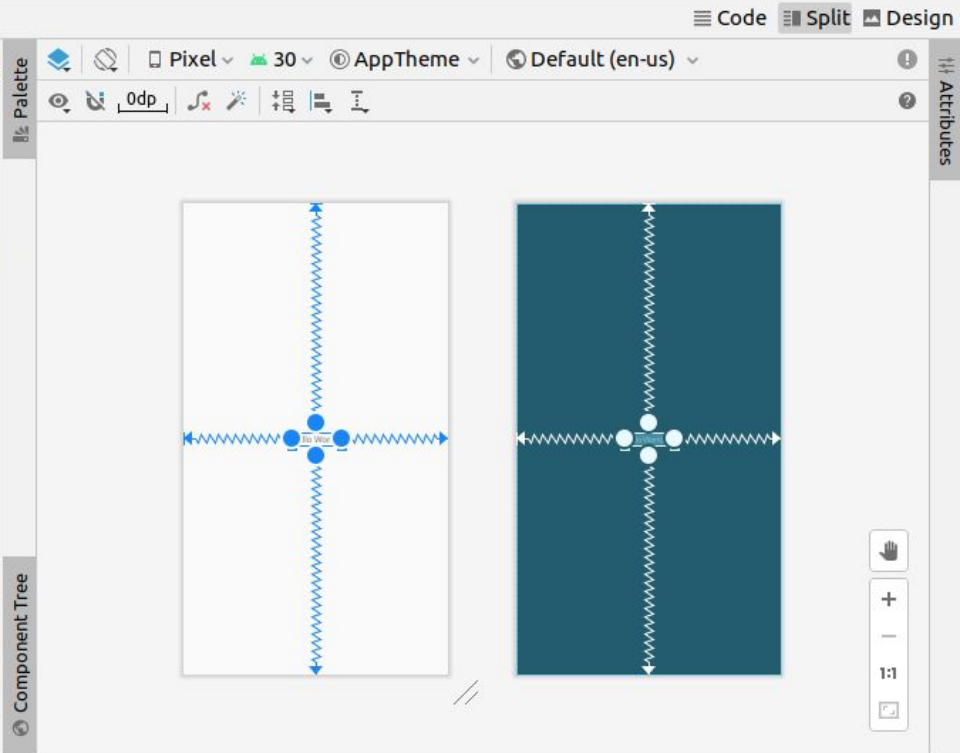
И в том, и в другом режиме доступно окошко Preview, которое показывает, как примерно будет выглядеть наш интерфейс.

CODE MODE

activity_main.xml

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res-auto"
3   xmlns:app="http://schemas.android.com/apk/res-auto"
4   xmlns:tools="http://schemas.android.com/tools"
5   android:layout_width="match_parent"
6   android:layout_height="match_parent"
7   tools:context=".MainActivity">
8
9   <TextView
10     android:layout_width="wrap_content"
11     android:layout_height="wrap_content"
12     android:text="Hello World!"
13     app:layout_constraintBottom_toBottomOf="parent"
14     app:layout_constraintLeft_toLeftOf="parent"
15     app:layout_constraintRight_toRightOf="parent"
16     app:layout_constraintTop_toTopOf="parent" />
17
18 </androidx.constraintlayout.widget.ConstraintLayout>
```

androidx.constraintlayout.widget.ConstraintLayout > TextView



DESIGN MODE

activity_main.xml

Code Split Design

Palette

Common

- TextView
- Button
- ImageView
- RecyclerView
- <> <fragment>
- ScrollView
- Switch

Text

Buttons

Widgets

Layouts

Containers

Google

Legacy

Component Tree

- ConstraintLayout
- TextView "Hello World!"

androidx.constraintlayout.widget.ConstraintLayout > TextView

Attributes

Ab <unnamed> TextView

id

Declared Attributes

Layout

Constraint Widget

Constraints (4)


layout_width wrap_content

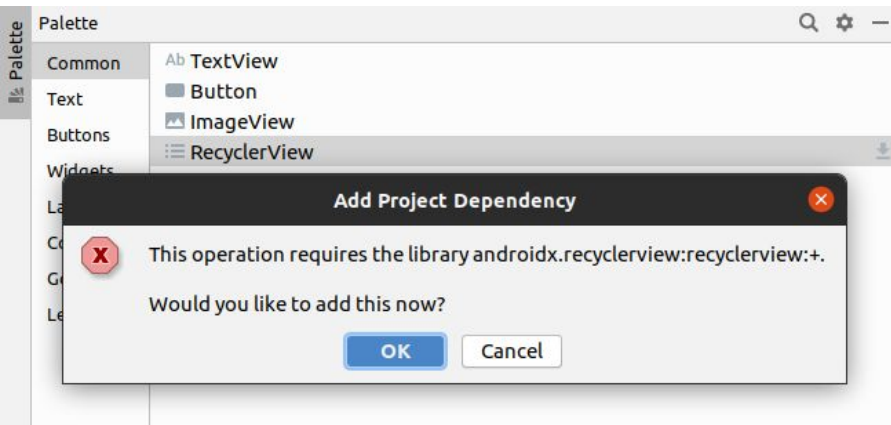
layout_height wrap_content

visibility

visibility

LIBRARIES

Рядом с некоторыми компонентами отображён значок скачивания  (например, [RecyclerView](#)). Это значит, что для их использования нужно дополнительно скачать и подключить в Gradle соответствующую зависимость, что эта кнопка за нас и сделает:



```
dependencies {  
    implementation fileTree(dir: "libs", include: ["*.jar"])  
    implementation "org.jetbrains.kotlin:kotlin-stdlib:$kotlin_version"  
    implementation 'androidx.core:core-ktx:1.1.0'  
    implementation 'androidx.appcompat:appcompat:1.1.0'  
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'  
    implementation 'androidx.recyclerview:recyclerview:1.1.0'  
    testImplementation 'junit:junit:4.12'  
    androidTestImplementation 'androidx.test.ext:junit:1.1.1'  
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'  
}
```

ATTRIBUTES

При выборе конкретного компонента вы увидите:

- все заданные атрибуты (Declared Attributes);
- настройки компоновки (Layout);
- самые часто используемые атрибуты для данного компонента (Common Attributes);
- все доступные атрибуты для этого компонента (All Attributes).

Attributes			Q	⚙	—
Ab <unnamed>		TextView			
id		<input type="text"/>			
▼ Declared Attributes		+ —			
layout_width	wrap_content		▼		
layout_height	wrap_content		▼		
layout_constraintBott...	parent		▼		
layout_constraintLeft_...	parent		▼		
layout_constraintRight...	parent		▼		
layout_constraintTop_...	parent		▼		
text	Hello World!				
► Layout					
► Common Attributes					
► All Attributes					

ATTRIBUTES

Т.е. мы можем поменять `text` на `Hello Kotlin`, и теперь в нашем приложении вместо `Hello World` будет `Hello Kotlin`.

Кроме того, благодаря панели All Attributes вам не нужно запоминать все существующие атрибуты, вы их можете просто подглядеть.

Следующим шагом для нас станет изучение ключевых компонентов и возможностей их компоновки. Об этом мы и поговорим на следующей лекции.



ИТОГИ

ИТОГИ

Сегодня мы обсудили с вами основополагающие вопросы:

- SDK,
- Android Framework,
- Общую идею разделения на ресурсы и код.

На базе этих знаний мы и будем строить наше дальнейшее обучение.