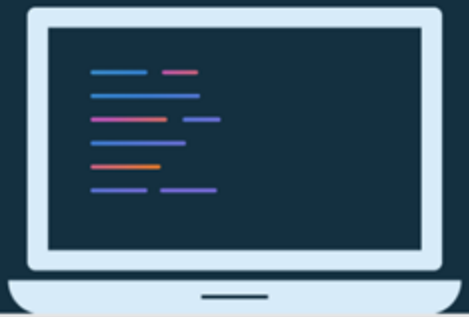
Lesson 4: Build your first Android app

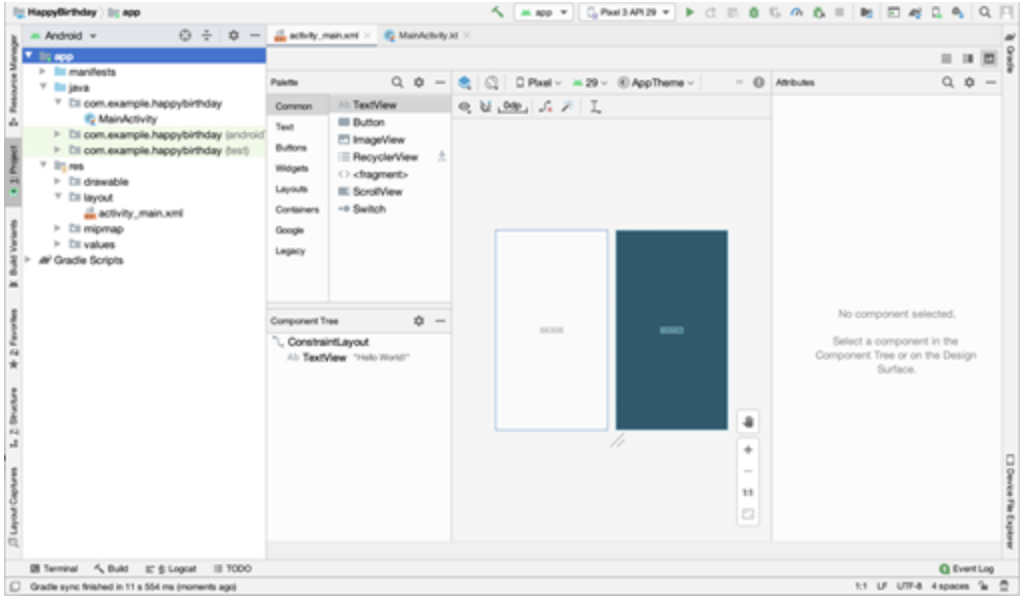


About this lesson

Lesson 4: Build your first Android app

Android Studio

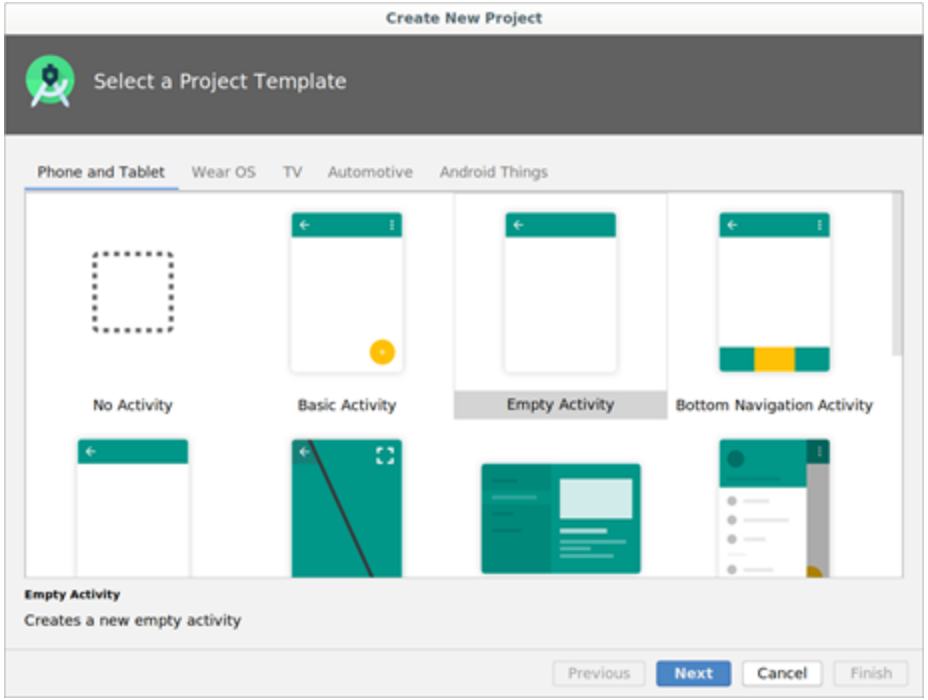
Official IDE for building Android apps



Your first app

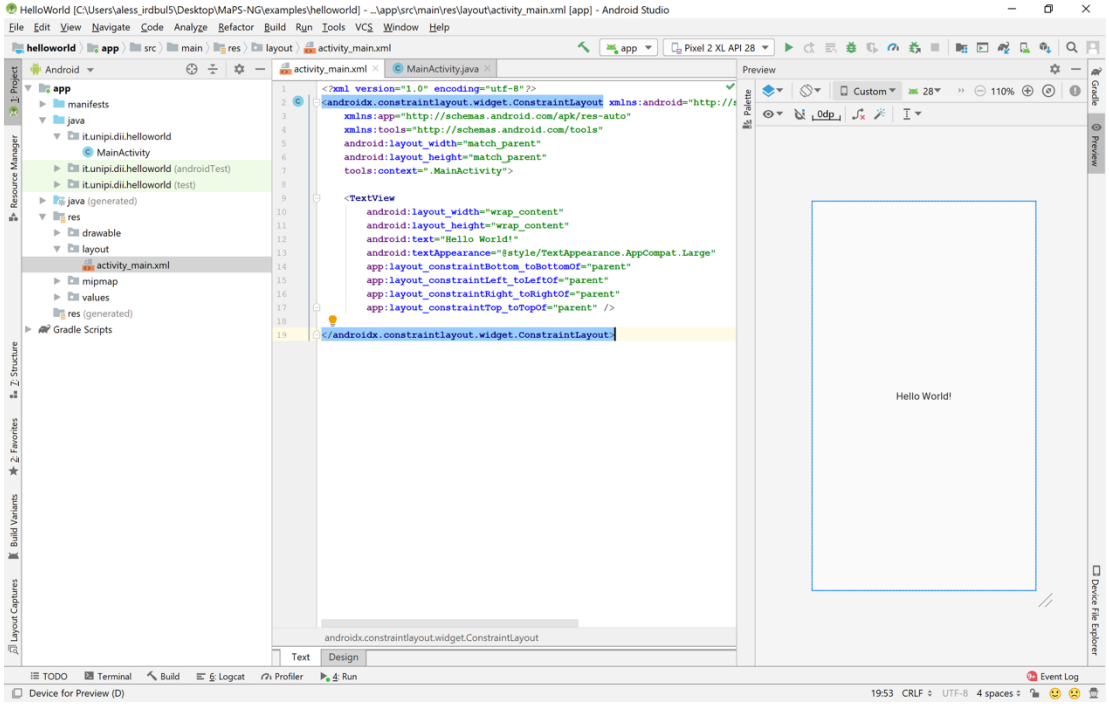
Open Android Studio

Create new project

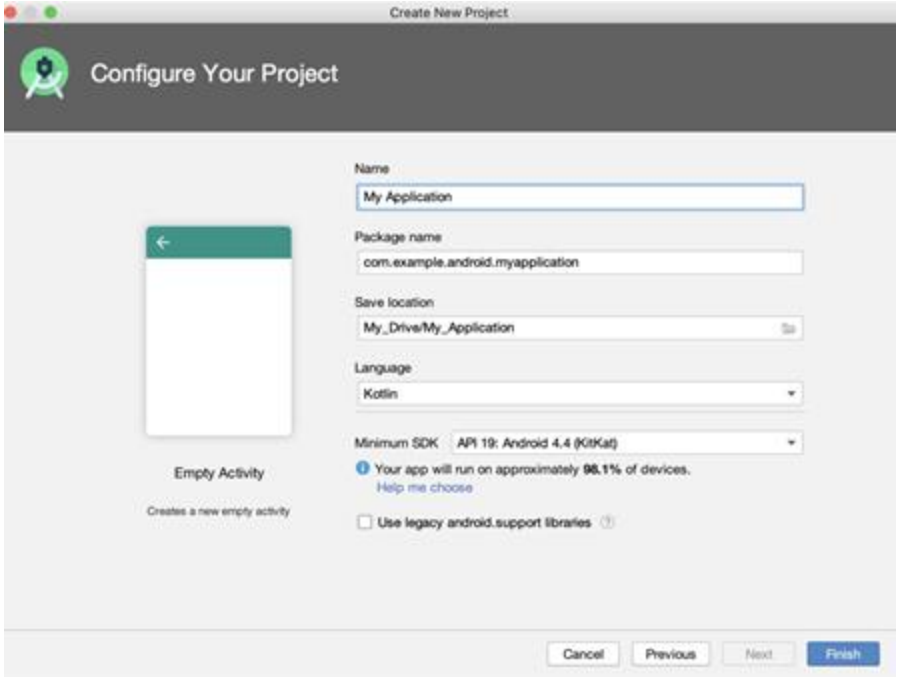


Development environment

- Android Studio, based on IntelliJ IDEA
- Compiles to executables (*.dex*), packages files (*.apk*), and deploys to phone



Enter your project details



Android releases and API levels

Platform Version	API Level	VERSION_CODE
Android 10.0	29	Q
Android 9	28	P
Android 8.1	27	O_MR1
Android 8.0	26	O
Android 7.1.1 Android 7.1	25	N_MR1
Android 7.0	24	N
Android 6.0	23	M
Android 5.1	22	LOLLIPOP_MR1
Android 5.0	21	LOLLIPOP

Android versions

- Version
- Codename
- API level

ANDROID PLATFORM VERSION		API LEVEL	CUMULATIVE DISTRIBUTION
4.4	KitKat	19	
5	Lollipop	21	99,6%
5.1	Lollipop	22	99,4%
6	Marshmallow	23	98,2%
7	Nougat	24	96,3%
7.1	Nougat	25	95,0%
8	Oreo	26	93,7%
8.1	Oreo	27	91,8%
9	Pie	28	86,4%
10	Q	29	75,9%
11	R	30	59,8%
12	S	31	38,2%
13	T	33	22,4%

Last updated: October 1, 2023

T

New features

Tablet and large screen support
Programmable shaders
Color vector fonts
Predictive back gesture
Bluetooth LE Audio
Splash screen efficiency improvements
ART optimizations

Behavior changes

OpenJDK 11 updates
Battery Resource Utilization
Media controls derived from PlaybackState
Permission required for advertising ID
Updated non-SDK restrictions

Security and privacy

Safer exporting of context-registered receivers
Enhanced photo picker privacy
New runtime permission for nearby Wi-F devices
Exact alarms permission
Developer downgradable permissions
APK Signature Scheme v3.1
Better error reporting in Keystore and KeyMint

https://developer.android.com/about/versions/13

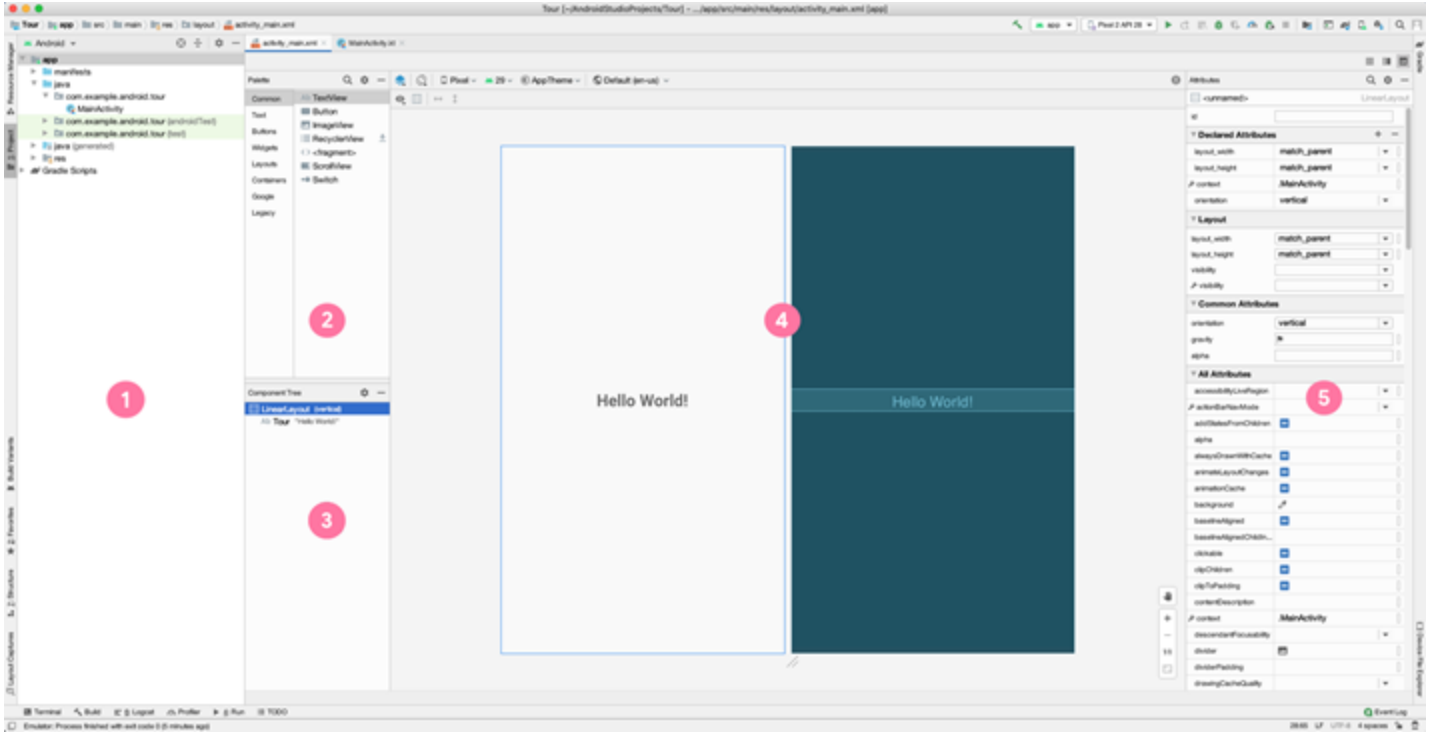
Choose API levels for your app

- Minimum SDK: Device needs at least this API level to install
- Target SDK: API version and highest Android version tested
- Compile SDK: Android OS library version compiled with

```
minSdkVersion <= targetSdkVersion <= compileSdkVersion
```

The API level identifies the framework API version of the Android SDK.

Tour of Android Studio



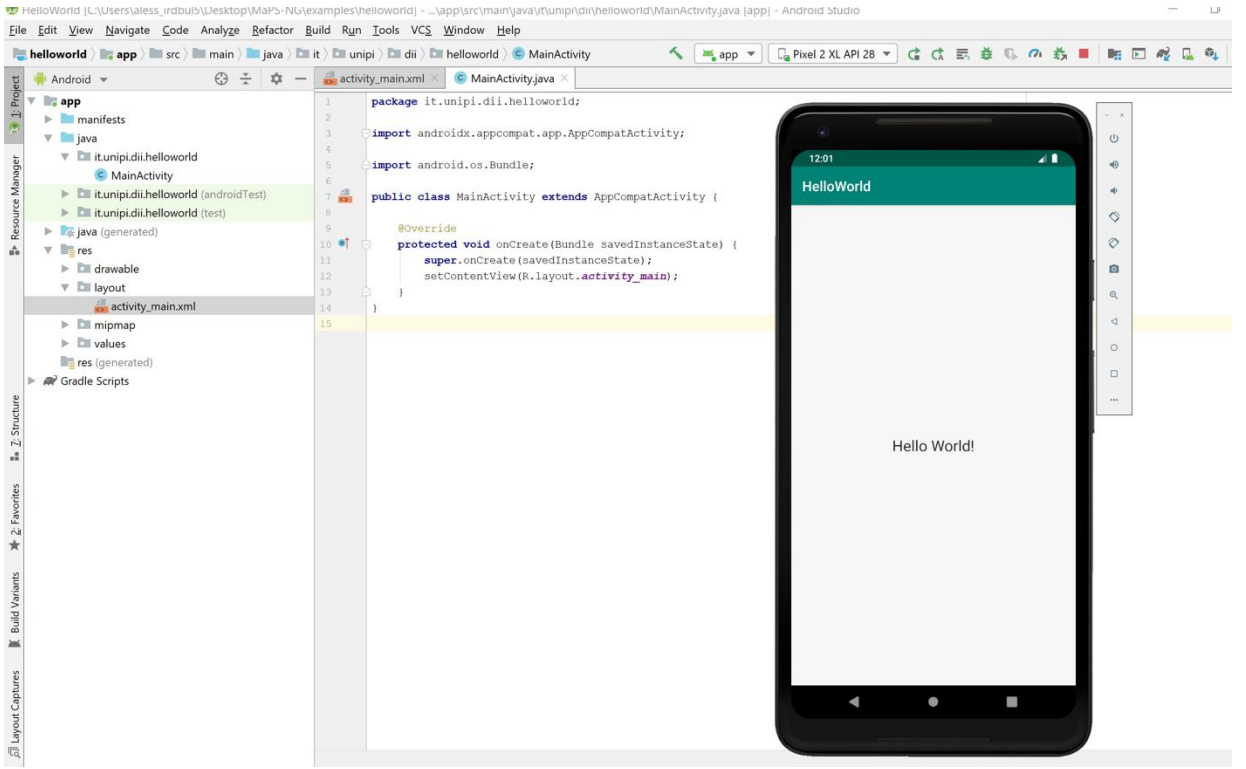
Run your app

- Android device (phone, tablet)
- Emulator on your computer

Running apps

- Apps can be executed
  - On a real device
  - On the emulator provided by Android Studio

- Real devices must be enabled:
  - *Settings > About phone* and tap *Build number* 7 times
  - *Settings > Developer options* enable *USB debugging*



Running

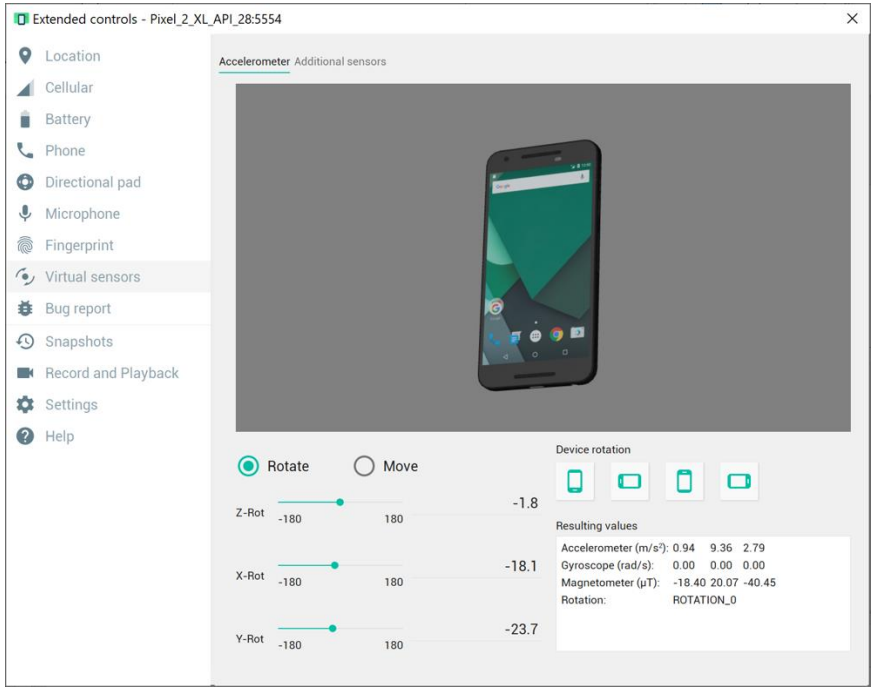
- The emulator boots like a real device, then your application is started
  - Keep the emulator running to save time
- If you store something on the persistent memory of the emulator, it will be persistent
- You can “navigate” within the emulator like a real device

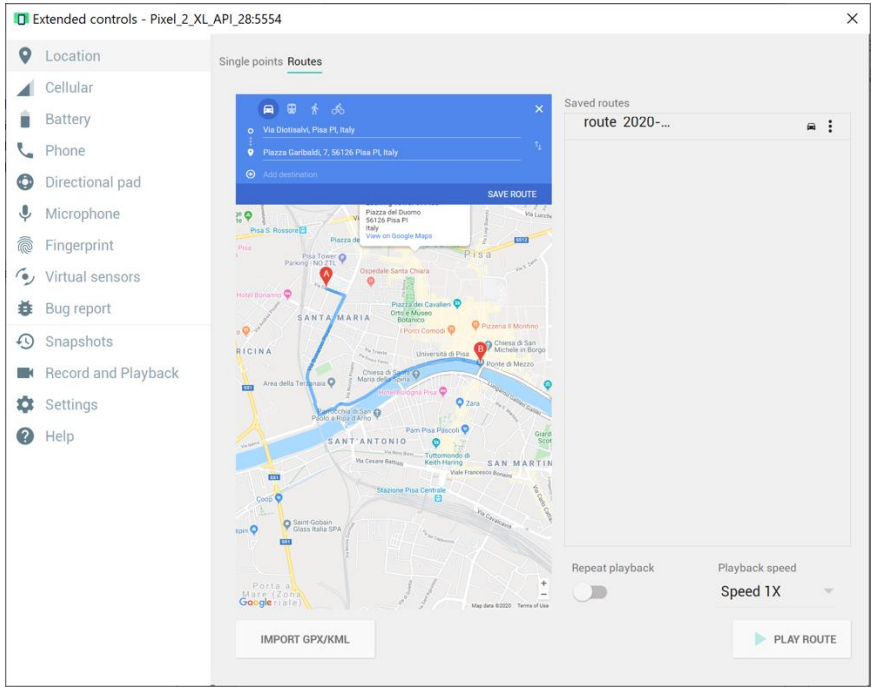
Emulator Pros and Cons (vs Real Phone)

- **Pros**:
  - Convenient execution of apps within the development environment
  - Easy to test app on various emulated devices (phones, tablets, TVs, etc), various screen sizes
- **Cons**:
  - Slower than real phone
  - Support for specific HW can be limited

- Emulator is OK for this class (in case you don’t have a real Android smartphone)

Support for artificial sensor readings





Android Virtual Device (AVD) Manager

Android Virtual Device Manager

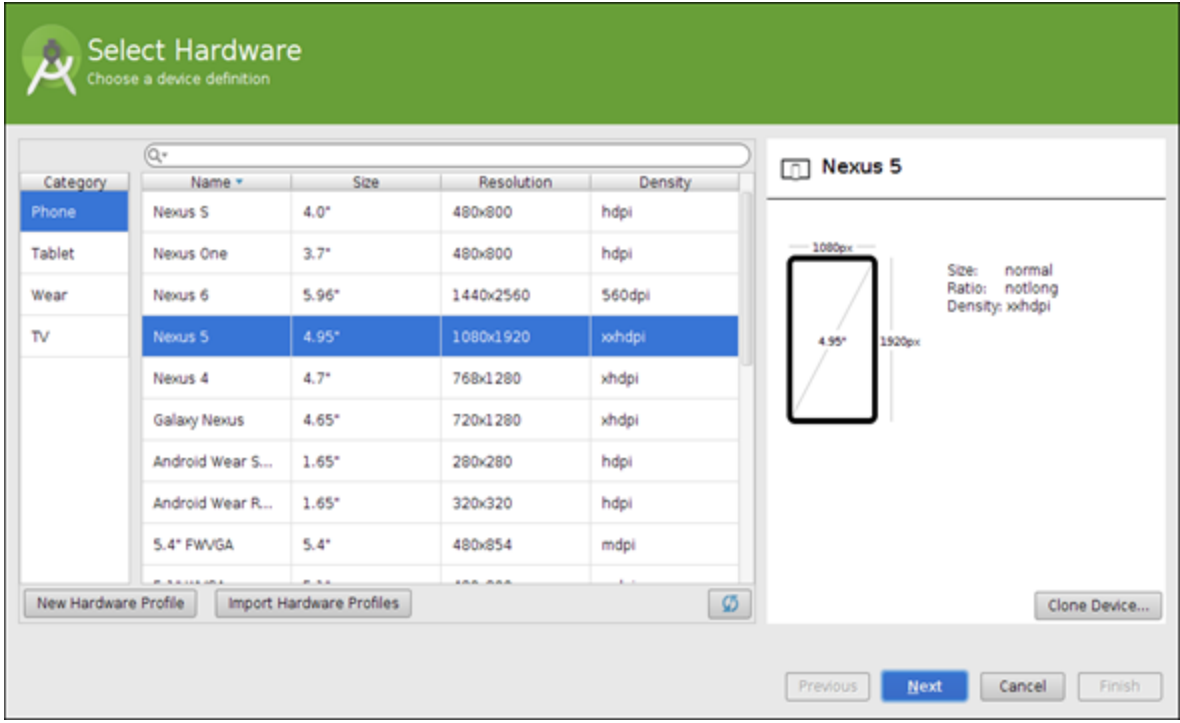
Your Virtual Devices
Android Studio

Type	Name	Play Store	Resolution	API	Target	CPU/ABI	Size on Disk	Actions
	Pixel 3 XL API 29		1440 × 29...	29	Android 10...	x86	3.7 GB	
	Pixel 3 XL API 28		1440 × 29...	28	Android 9....	x86	3.1 GB	

? + Create Virtual Device...

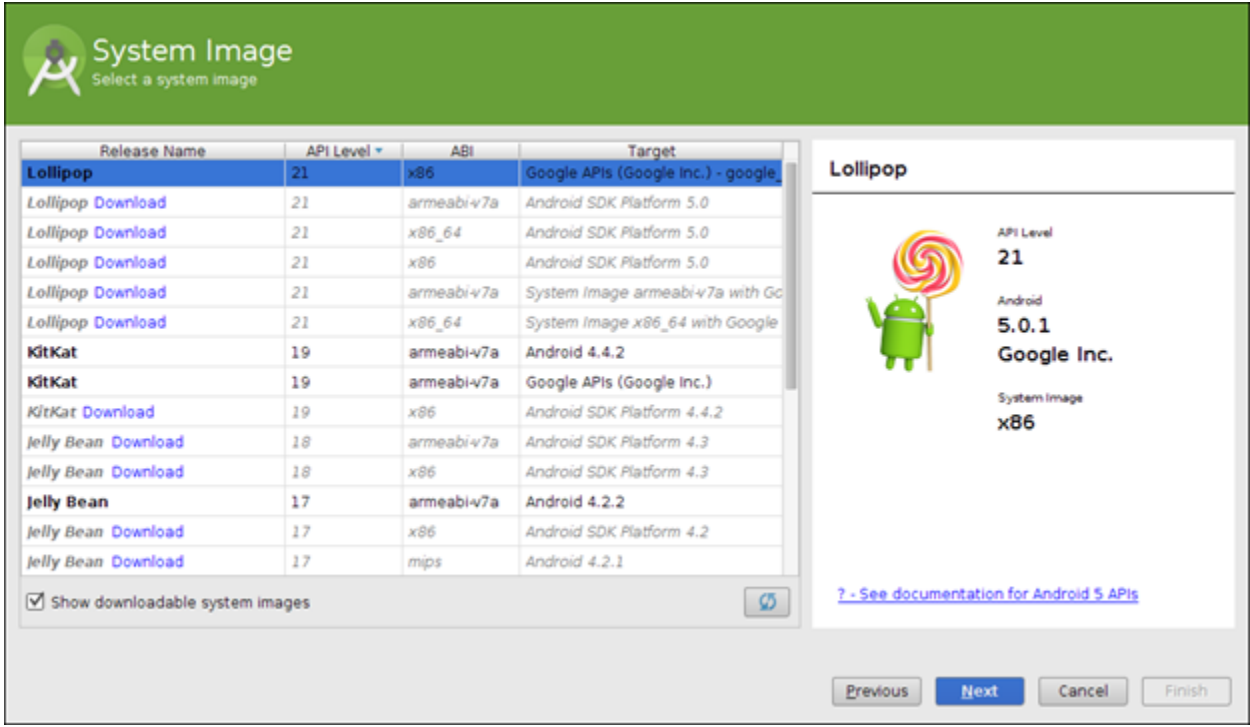
AVDs

- When creating an AVD, you can select category, device type, resolution



AVDs

- ... system image (API level, processor architecture)



Android SDK Manager

- SDK Manager allows you to install build-tools, system images, Android APIs, Google proprietary APIs, extras
- Extras include Google USB driver (Win), x86 emulator accelerator (Win), Google Play services

SDK Platforms | SDK Tools | SDK Update Sites

Each Android SDK Platform package includes the Android platform and sources pertaining to an API level by default. Once installed, Android Studio will automatically check for updates. Check "show package details" to display individual SDK components.

Name	API Level	Revision	Status
Android 6.0	23	2	Not installed
Android 5.1.1	22	2	Update available
Android 5.0.1	21	2	Update available
Android 4.4W.2	20	2	Not installed
Android 4.4.2	19	4	Update available
Android 4.3.1	18	3	Not installed
Android 4.2.2	17	3	Update available
Android 4.1.2	16	5	Update available
Android 4.0.3	15	5	Not installed
Android 4.0	14	4	Installed
Android 2.3.3	10	2	Update available
Android 2.2	8	3	Not installed

Show Package Details

SDK Platforms | SDK Tools | SDK Update Sites

Below are the available SDK developer tools. Once installed, Android Studio will automatically check for updates. Check "show package details" to display available versions of an SDK Tool.

Name	Version	Status
Android SDK Build Tools		Update Available: 23.0.2
Android SDK Tools 24.1.2	24.1.2	Update Available: 24.4.1
Android SDK Platform-Tools 22	22.0.0	Update Available: 23.1.0
Documentation for Android SDK	1	Update Available: 1
GPU Debugging tools	1.0.3	Not installed
Android Support Repository, rev 14	14.0.0	Update Available: 26
Android Support Library, rev 22.1.1	22.1.1	Update Available: 23.2.0
Android Auto Desktop Head Unit emulator	1.1.0	Not installed
Google Play services, rev 23	23.0.0	Update Available: 29
Google Repository, rev 16	16.0.0	Update Available: 24
Google Play APK Expansion Library	3.0.0	Not installed
Google Play Billing Library	5.0.0	Not installed
Google Play Licensing Library	2.0.0	Not installed
Android Auto API Simulators	1.0.0	Not installed
Google Web Driver	2.0.0	Not installed
Android NDK	1.0.0	Not installed
LLDB	2.1.2589848	Not installed

Anatomy of an Android App project

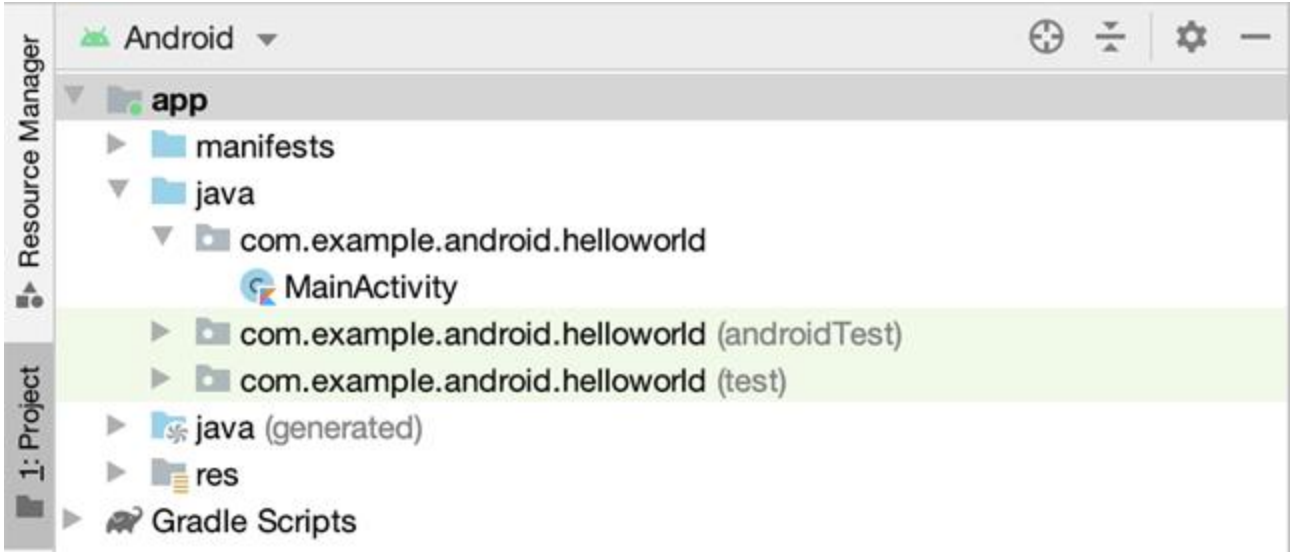
Anatomy of a basic app project

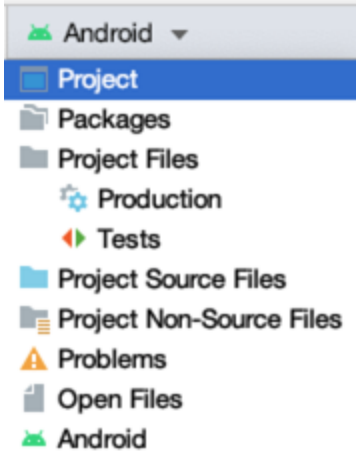
- Activity
- Resources (layout files, images, audio files, themes, and colors)
- Gradle files

Android app project structure

```
MyApplication
├── app
│   ├── libs
│   └── src
│       ├── androidTest
│       ├── main
│       │   ├── java
│       │   ├── res
│       │   └── AndroidManifest.xml
│       └── test
├── build.gradle
└── gradlew
```

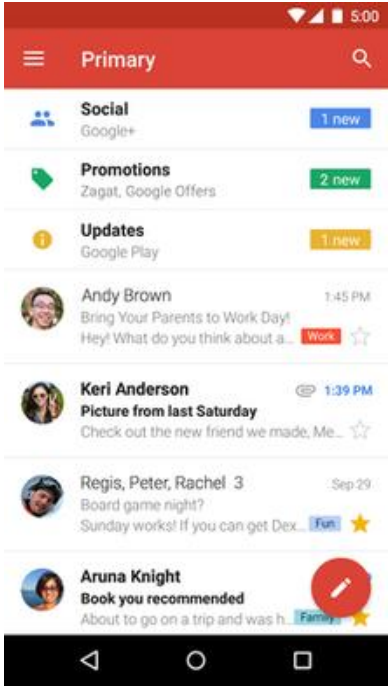
Browse files in Android Studio





Developing apps: UI + logic

- UI design code (XML) separate from the program (kotlin)
- Why? Can modify UI without changing kotlin code
- Example: Shapes, colors can be changed in XML file without changing kotlin program
- UI designed using either:
  - Drag-and drop graphical (WYSIWYG) tool or
  - Writing Extensible Markup Language (XML)
- XML: Markup language, both human-readable and machine-readable

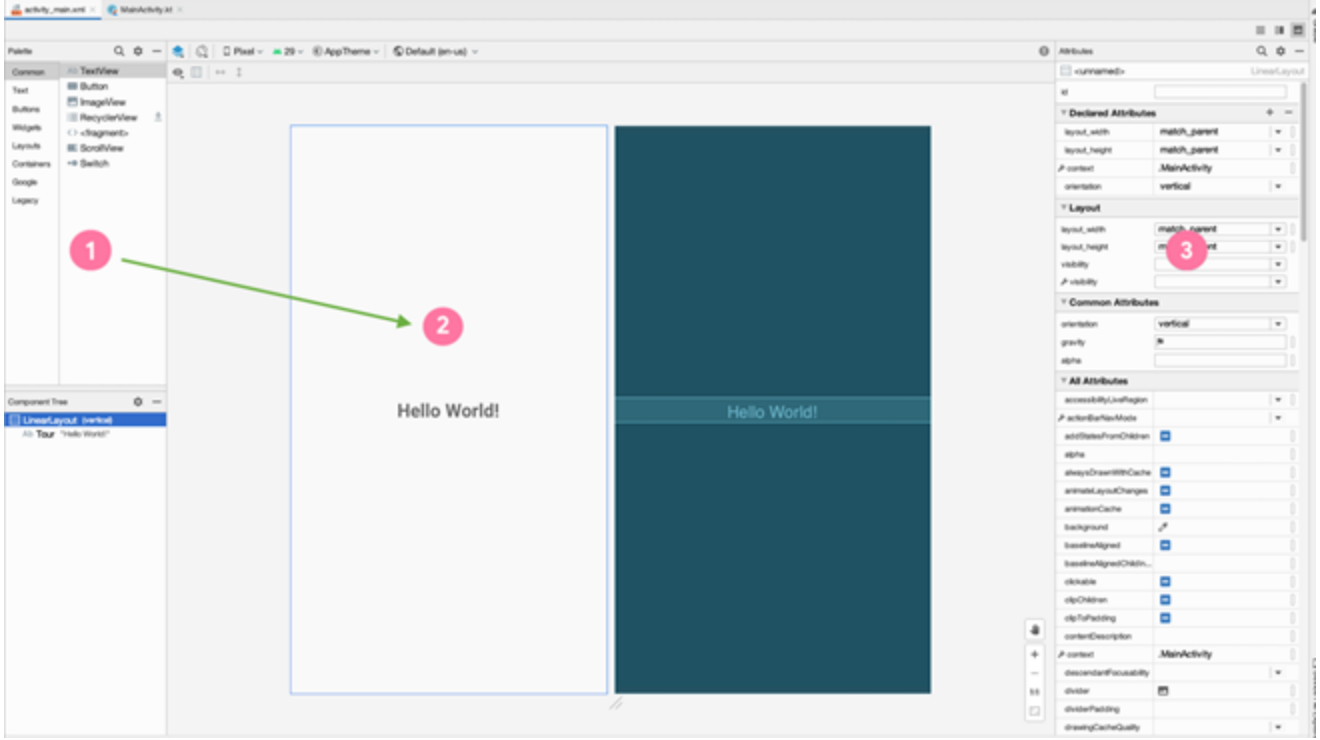


Layouts and resources in Android

Views

- Views are the user interface building blocks in Android
  - Bounded by a rectangular area on the screen
  - Responsible for drawing and event handling
  - Examples: TextView, ImageView, Button
- Can be grouped to form more complex user interfaces

Layout Editor



XML Layouts

You can also edit your layout in XML.

- Android uses XML to specify the layout of user interfaces (including View attributes)

- Each View in XML corresponds to a class in Kotlin that controls how that View functions

XML for a TextView

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!"/>
```

Hello World!

Size of a View

- wrap_content

```
android:layout_width="wrap_content"
```

- match_parent

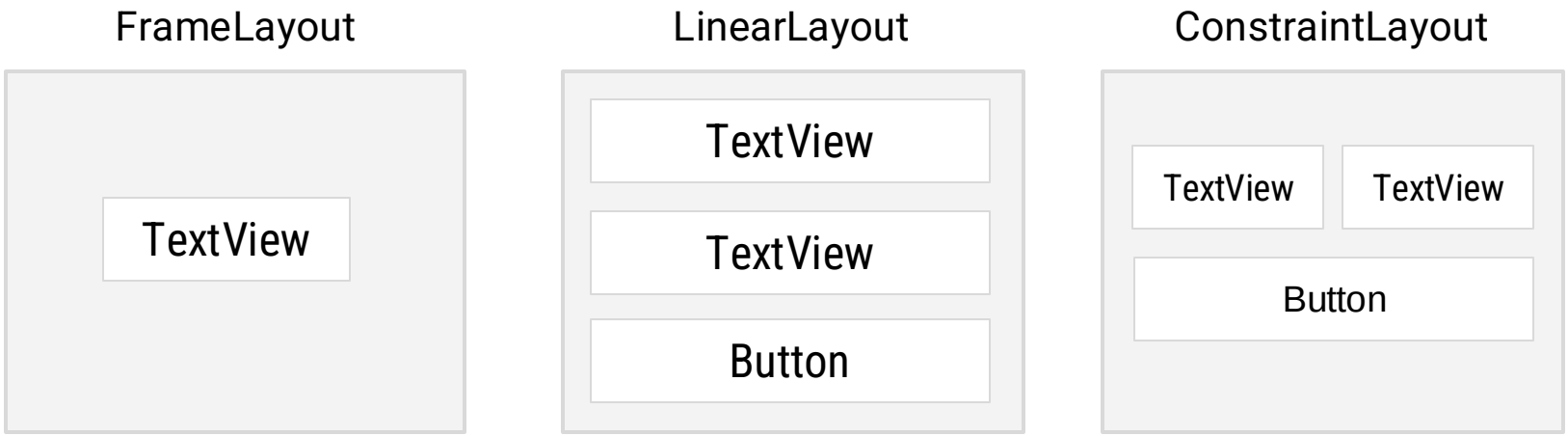
```
android:layout_width="match_parent"
```

- Fixed value (use dp units)

```
android:layout_width="48dp"
```

ViewGroups

A `ViewGroup` is a container that determines how views are displayed.



The ViewGroup is the parent and the views inside it are its children.

FrameLayout example

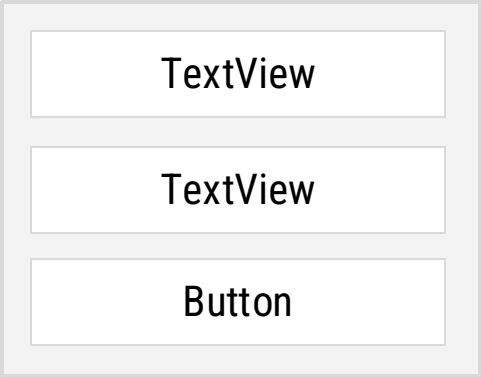
A `FrameLayout` generally holds a single child `View`.

```
<FrameLayout
   android:layout_width="match_parent"
   android:layout_height="match_parent">
   <TextView
       android:layout_width="match_parent"
       android:layout_height="match_parent"
       android:text="Hello World!"/>
</FrameLayout>
```

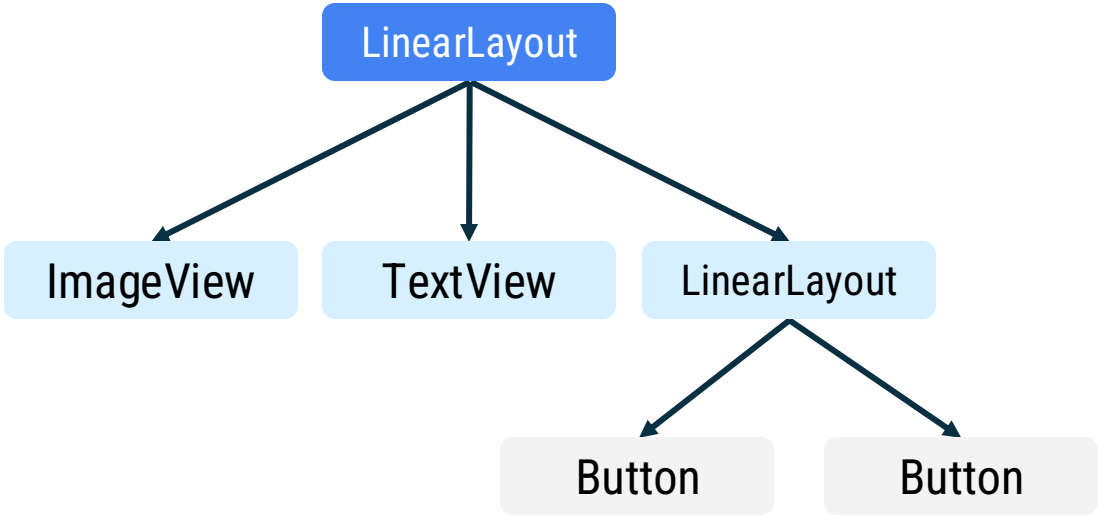
LinearLayout example

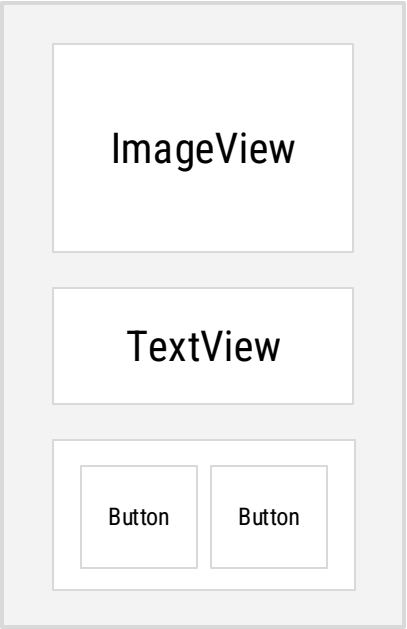
- Aligns child views in a row or column
- Set `android:orientation` to `horizontal` or `vertical`

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <TextView ... />
    <TextView ... />
    <Button ... />
</LinearLayout>
```



View hierarchy





App resources

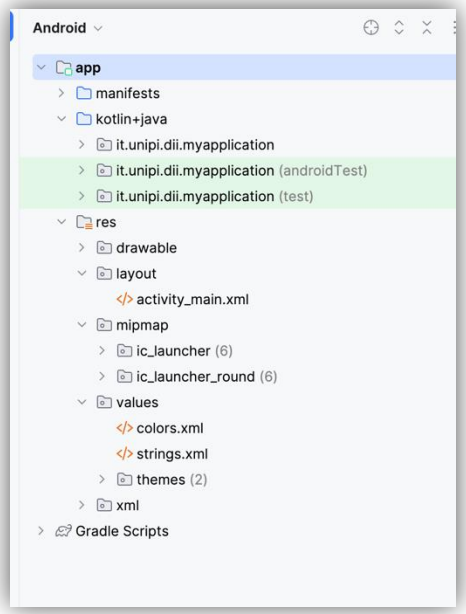
Static content or additional files that your code uses

- Layout files
- Images
- Audio files
- User interface strings
- App icon

Common resource directories

Add resources to your app by including them in the appropriate resource directory under the parent `res` folder.

```
main
├── java
└── res
    ├── drawable
    ├── layout
    ├── mipmap
    └── values
```



Resource IDs

- Each resource has a resource ID to access it.
- When naming resources, the convention is to use all lowercase with underscores (for example, `activity_main.xml`).
- Android autogenerates a class file named `R.java` with references to all resources in the app.
- Individual items are referenced with:

  `R.<resource_type>.<resource_name>`

Examples:

```
R.drawable.ic_launcher (res/drawable/ic_launcher.xml)
R.layout.activity_main (res/layout/activity_main.xml)
```

Resource IDs for views

Individual views can also have resource IDs.

Add the `android:id` attribute to the View in XML. Use `@+id/name` syntax.

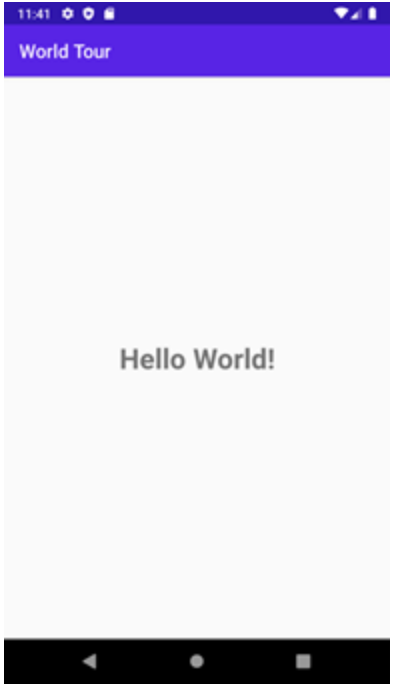
```
<TextView
    android:id="@+id/helloTextView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!"/>
```

Within your app, you can now refer to this specific TextView using:

`R.id.helloTextView`

Activities

What’s an Activity?



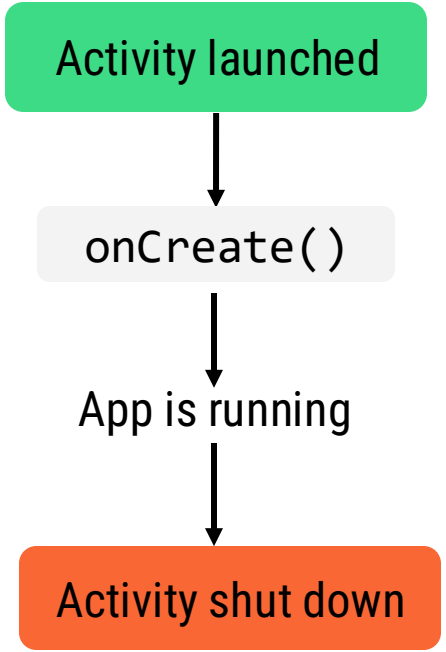
- An Activity is a means for the user to accomplish one main goal.
- An Android app is composed of one or more activities.

MainActivity.kt

```
class MainActivity : AppCompatActivity() {

   override fun onCreate(savedInstanceState: Bundle?) {
       super.onCreate(savedInstanceState)
       setContentView(R.layout.activity_main)
   }
}
```

How an Activity runs

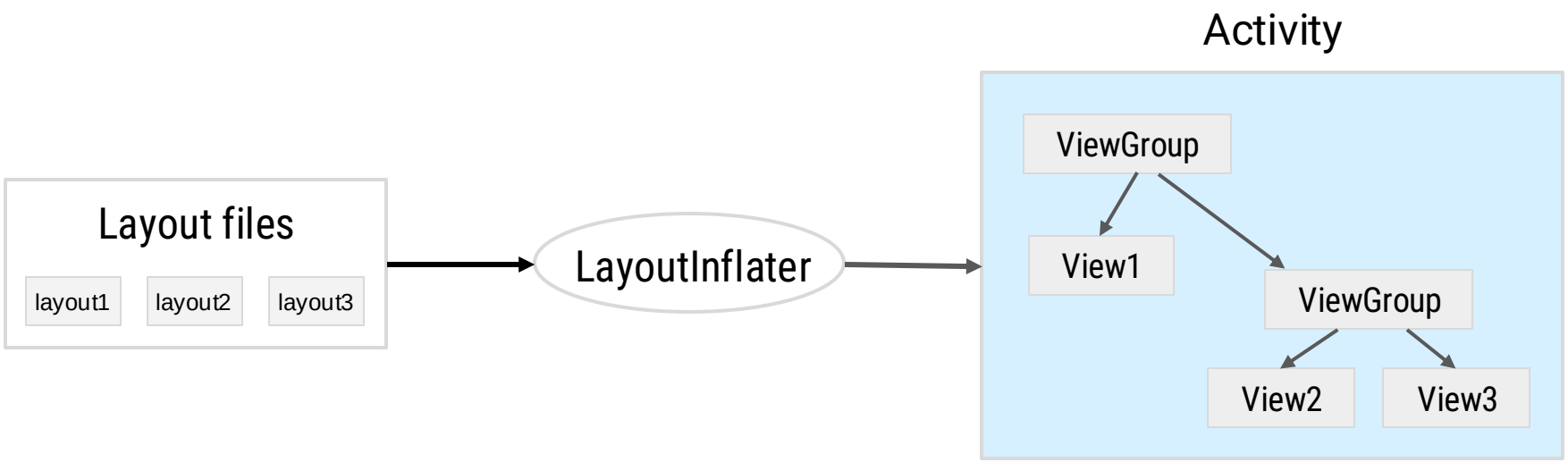


Implement the onCreate() callback

Called when the system creates your Activity

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
}
```

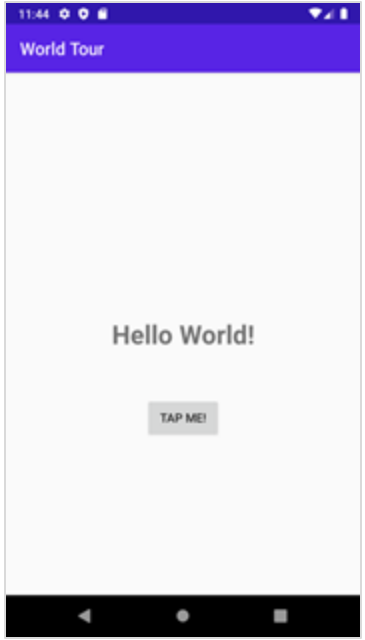
Layout inflation



Make an app interactive

Define app behavior in Activity

Modify the Activity so the app responds to user input, such as a button tap.



Modify a View dynamically

Within `MainActivity.kt`:

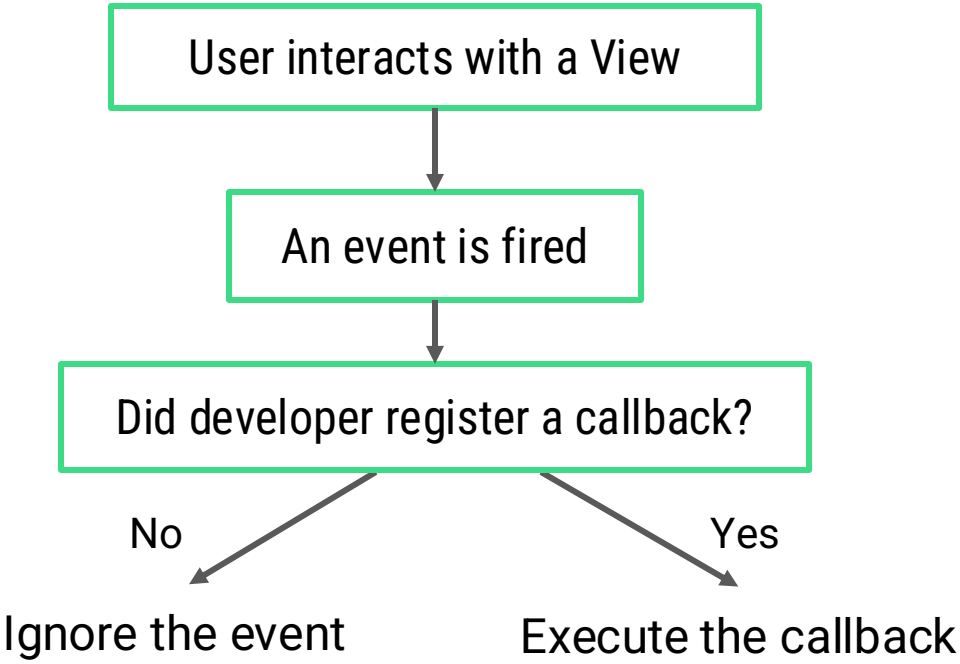
Get a reference to the View in the view hierarchy:

```
val resultTextView: TextView = findViewById(R.id.textView)
```

Change properties or call methods on the View instance:

```
resultTextView.text = "Goodbye!"
```

Set up listeners for specific events



View.OnClickListener

```
class MainActivity : AppCompatActivity(), View.OnClickListener {

    override fun onCreate(savedInstanceState: Bundle?) {
        ...
        val button: Button = findViewById(R.id.button)
        button.setOnClickListener(this)
    }

    override fun onClick(v: View?) {
        TODO("not implemented")
    }
}
```

SAM (single abstract method)

Converts a function into an implementation of an interface

Format: `InterfaceName { lambda body }`

```
val runnable = Runnable { println("Hi there") }
```

is equivalent to

```
val runnable = (object: Runnable {
    override fun run() {
        println("Hi there")
    }
})
```

View.OnClickListener as a SAM

A more concise way to declare a click listener

```
class MainActivity : AppCompatActivity() {

   override fun onCreate(savedInstanceState: Bundle?) {
       ...

       val button: Button = findViewById(R.id.button)
       button.setOnClickListener({ view -> /* do something*/ })
   }
}
```

Late initialization

```
class Student(val id: String) {

    lateinit var records: HashSet<Any>

    init {
        // retrieve records given an id
    }
}
```

Lateinit example in Activity

```
class MainActivity : AppCompatActivity() {

    lateinit var result: TextView

    override fun onCreate(savedInstanceState: Bundle?) {
        ...
        result = findViewById(R.id.result_text_view)
    }
}
```

Gradle: Building an Android app

What is Gradle?

- Builds automation system
- Manages the build cycle via a series of tasks (for example, compiles Kotlin sources, runs tests, installs app to device)
- Determines the proper order of tasks to run
- Manages dependencies between projects and third-party libraries

Gradle build file

- Declare plugins
- Define Android properties
- Handle dependencies
- Connect to repositories

Plugins

Provide libraries and infrastructure needed by your app

```
apply plugin: 'com.android.application'
apply plugin: 'kotlin-android'
apply plugin: 'kotlin-android-extensions'
```

Android configuration

```
android {
   compileSdkVersion 30
   buildToolsVersion "30.0.2"

   defaultConfig {
       applicationId "com.example.sample"
       minSdkVersion 19
       targetSdkVersion 30
   }
}
```

Dependencies

```
dependencies {
    implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk7:$kotlin_version"
    implementation 'androidx.core:core-ktx:1.3.2'
    implementation 'androidx.appcompat:appcompat:1.2.0'
    implementation 'com.google.android.material:material:1.2.1'
    ...
}
```

Repositories

```
repositories {
    google()
    jcenter()
    maven {
        url "https://maven.example.com"
    }
}
```

Common Gradle tasks

- Clean
- Tasks
- InstallDebug

Accessibility

Accessibility

- Refers to improving the design and functionality of your app to make it easier for more people, including those with disabilities, to use
- Making your app more accessible leads to an overall better user experience and benefits all your users

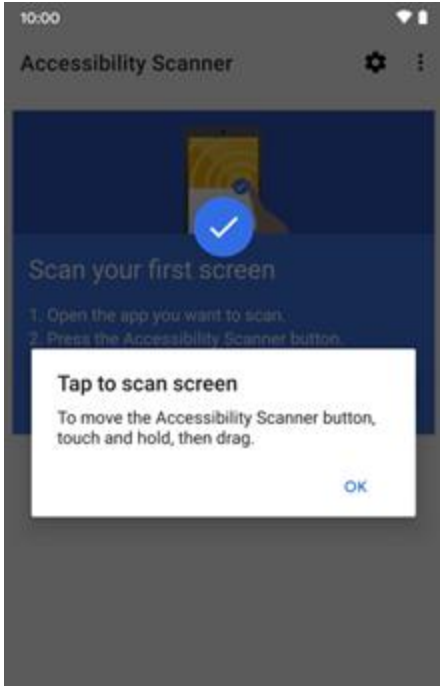
Make apps more accessible

- Increase text visibility with foreground and background color contrast ratio:
  - At least 4.5:1 for small text against the background
  - At least 3.0:1 for large text against the background
- Use large, simple controls
  - Touch target size should be at least 48dp x 48dp
- Describe each UI element
  - Set content description on images and controls

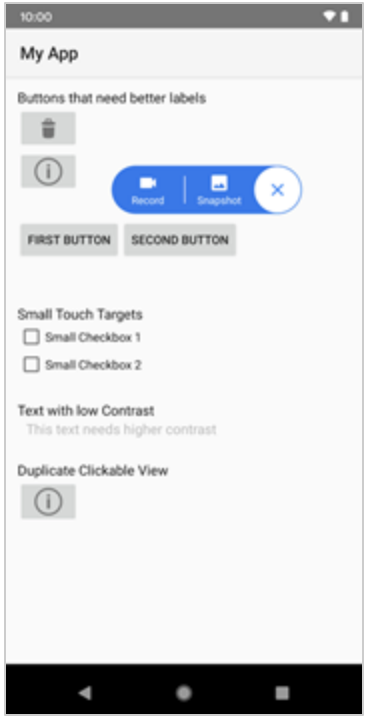
Accessibility Scanner

Tool that scans your screen and suggests improvements to make your app more accessible, based on:

- Content labels
- Touch target sizes
- Clickable views
- Text and image contrast



Accessibility Scanner example



Add content labels

- Set `contentDescription` attribute → read aloud by screen reader

```
<ImageView
    ...
    android:contentDescription="@string/stop_sign" />
```

- Text in TextView already provided to accessibility services, no additional label needed

No content label needed

- For graphical elements that are purely for decorative purposes, you can set

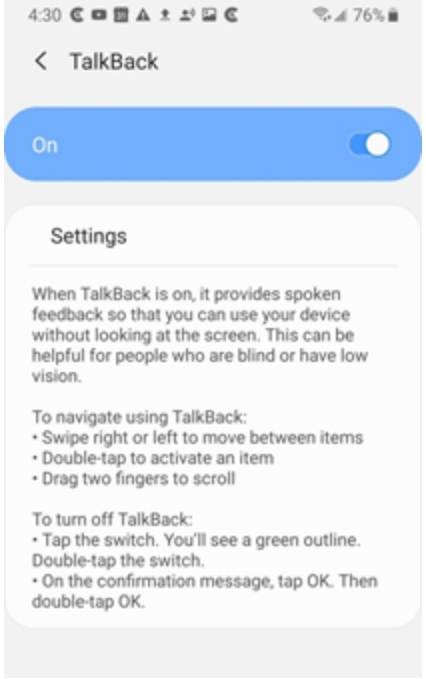
  ```
  android:importantForAccessibility="no"
  ```

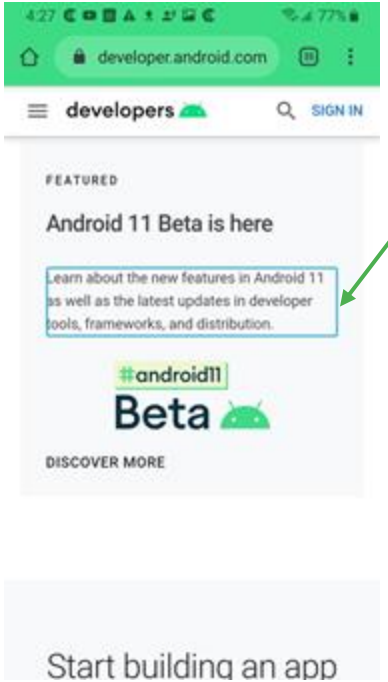
- Removing unnecessary announcements is better for the user

TalkBack

- Google screen reader included on Android devices
- Provides spoken feedback so you don’t have to look at the screen to use your device
- Lets you navigate the device using gestures
- Includes braille keyboard for Unified English Braille

TalkBack example





Reads text aloud as user navigates the screen

Switch access

- Allows for controlling the device using one or more switches instead of the touchscreen
- Scans your app UI and highlights each item until you make a selection
- Use with external switch, external keyboard, or buttons on the Android device (e.g., volume buttons)

Android Accessibility Suite

Collection of accessibility apps that help you use your Android device eyes-free, or with a switch device. It includes:

- Talkback screen reader
- Switch Access
- Accessibility Menu
- Select to Speak