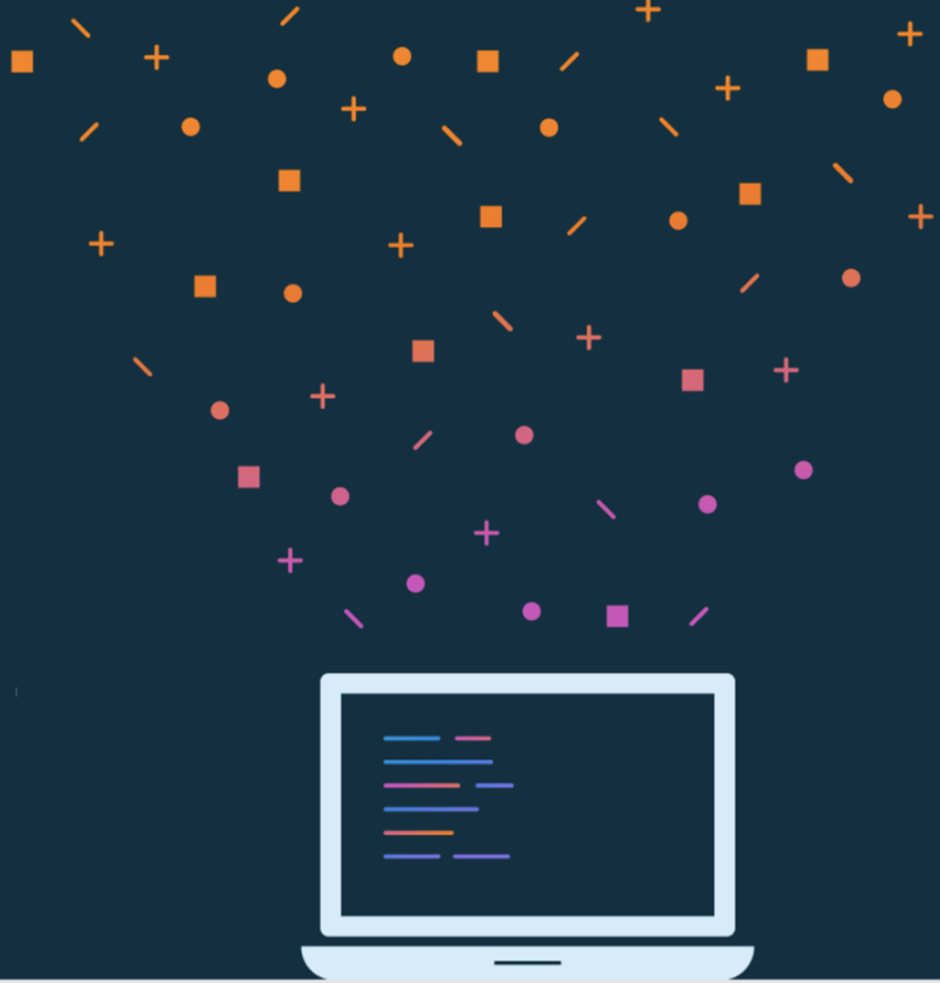
Lesson 2: Build your first Android app

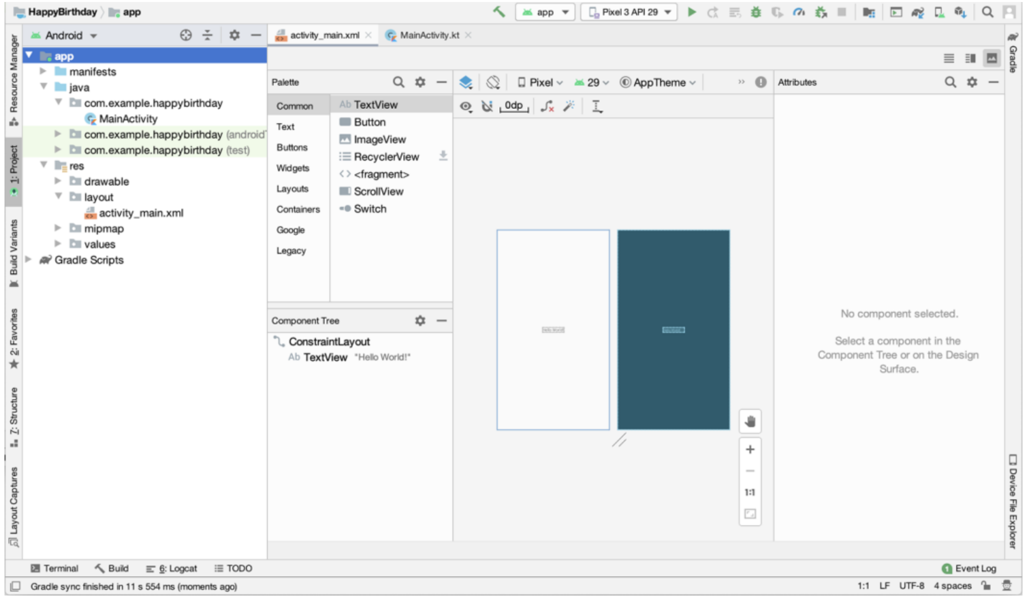


About this lesson

Lesson 2: Build your first Android app

Android Studio

Official IDE for building Android apps



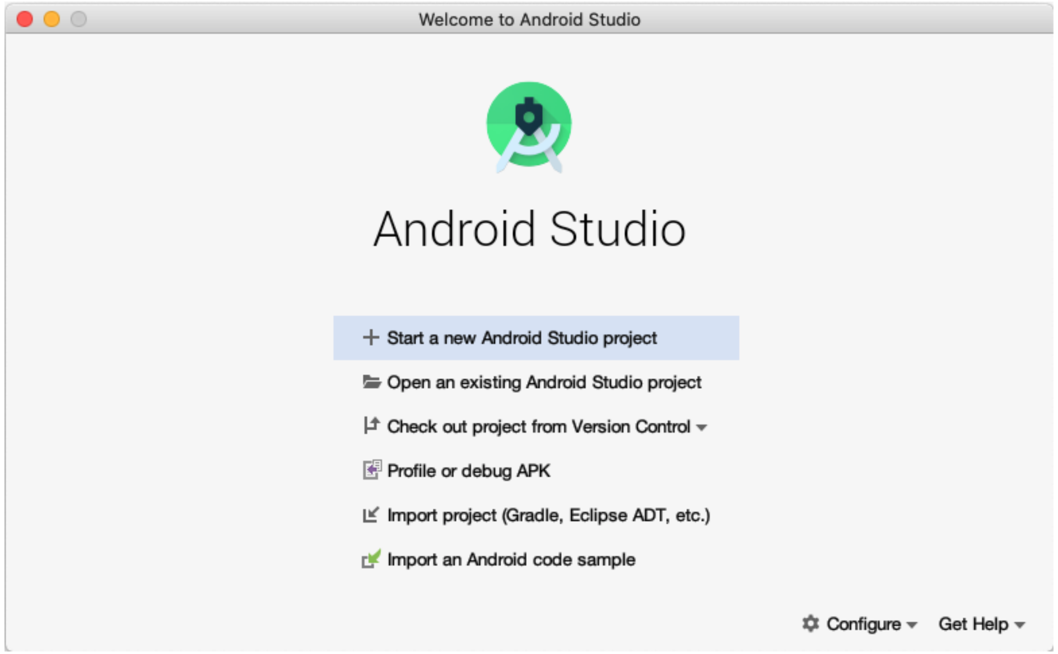
No component selected.

Select a component in the Component Tree or on the Design Surface.

Gradle sync finished in 11 s 554 ms (moments ago)

Your first app

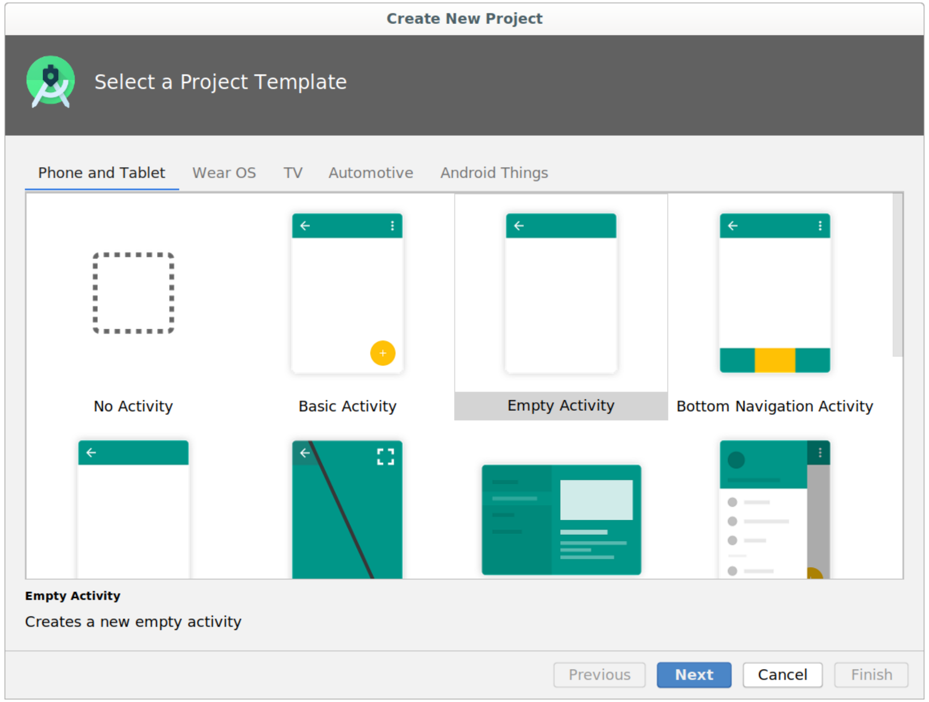
Open Android Studio


Welcome to Android Studio

Android Studio

+ Start a new Android Studio project
Open an existing Android Studio project
Check out project from Version Control
Profile or debug APK
Import project (Gradle, Eclipse ADT, etc.)
Import an Android code sample
Configure
Get Help

Create new project



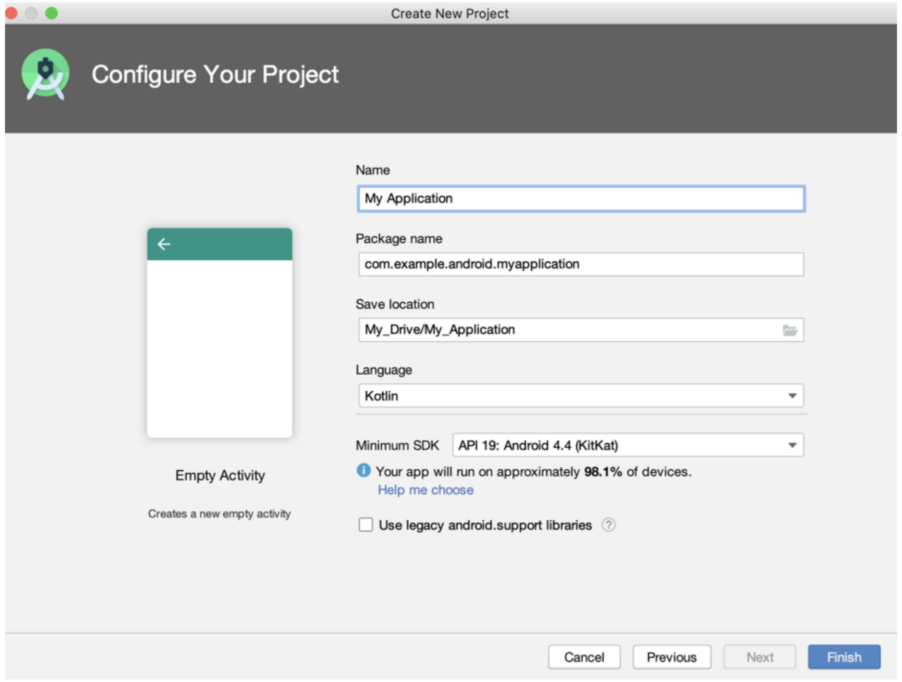
Create New Project

Select a Project Template

Empty Activity

Creates a new empty activity

Enter your project details



Create New Project

Configure Your Project

Name: My Application

Package name: com.example.android.myapplication

Save location: My_Drive/My_Application

Language: Kotlin

Minimum SDK: API 19: Android 4.4 (KitKat)

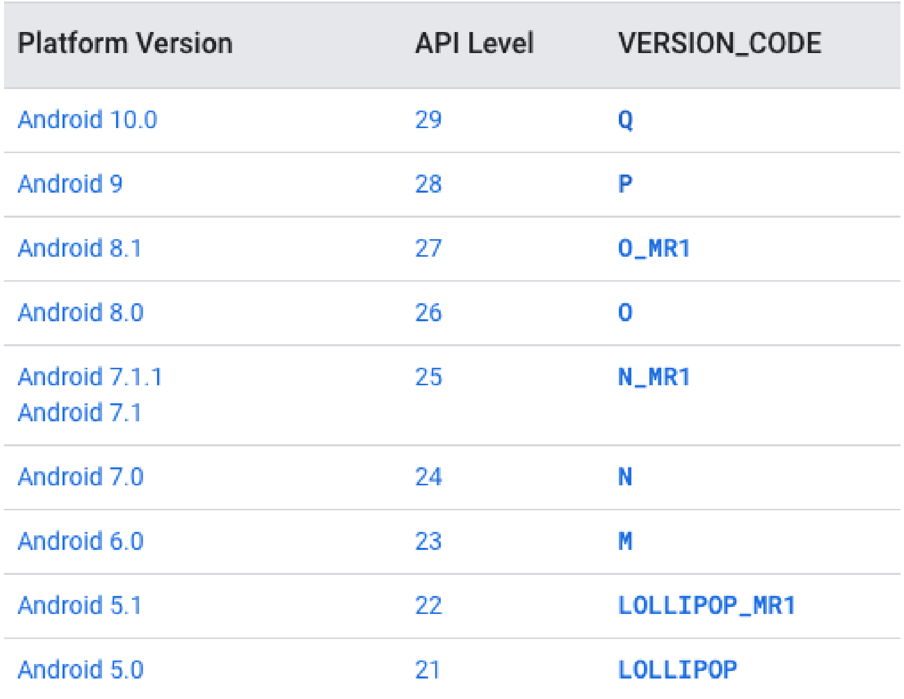
Your app will run on approximately **98.1%** of devices.
Help me choose

☐ Use legacy android.support libraries

Empty Activity

Creates a new empty activity

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

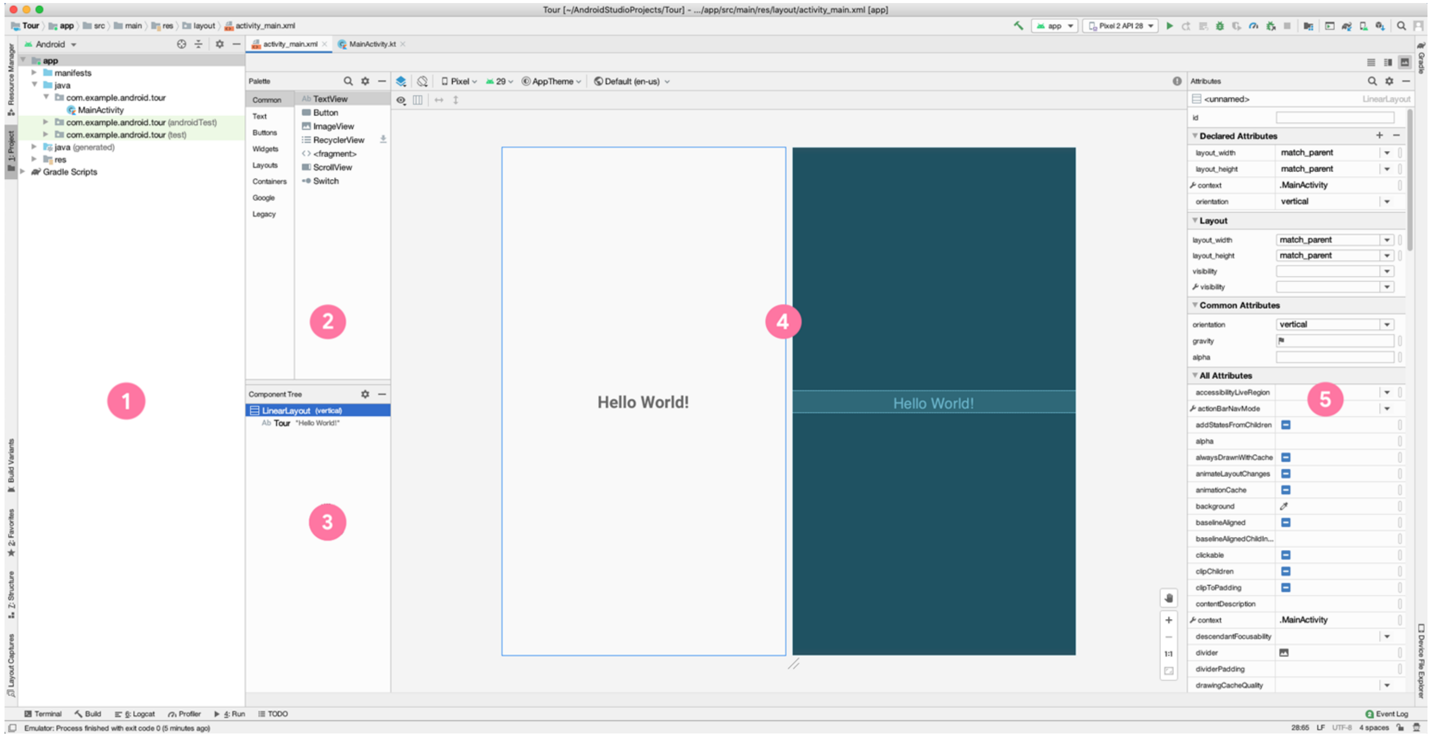
Choose API levels for your app

- 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



Declared Attributes

layout_width	match_parent
layout_height	match_parent
context	.MainActivity
orientation	vertical

Layout

layout_width	match_parent
layout_height	match_parent
visibility	
visibility	

Common Attributes

orientation	vertical
gravity	
alpha	

All Attributes

accessibilityLiveRegion	
actionBarNavMode	
addStatesFromChildren	
alpha	
alwaysDrawnWithCache	
animateLayoutChanges	
animationCache	
background	
baselineAligned	
baselineAlignedChildIn...	
clickable	
clipChildren	
clipToPadding	
contentDescription	
context	.MainActivity
descendantFocusability	
divider	
dividerPadding	
drawingCacheQuality	

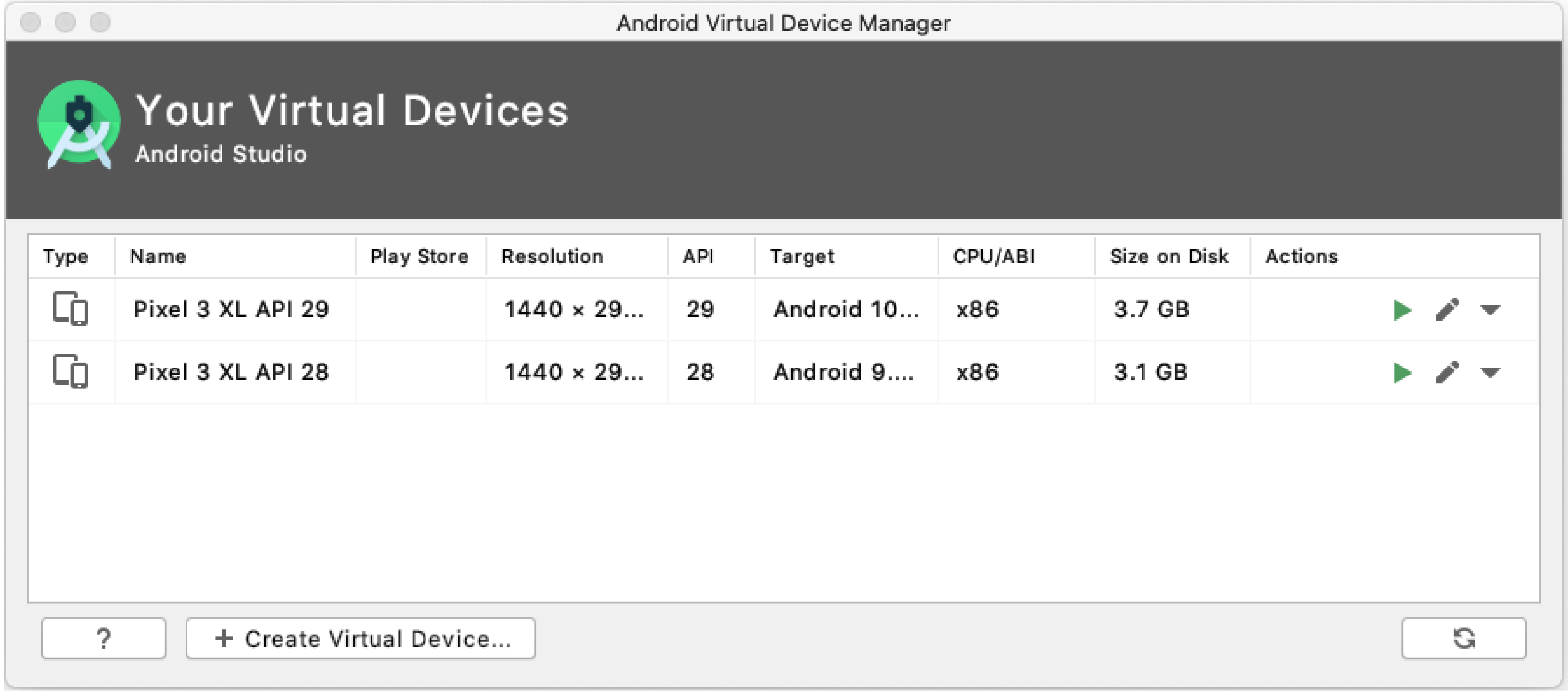
Emulator: Process finished with exit code 0 (5 minutes ago)

28:65 LF UTF-8 4 spaces

Run your app

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

Android Virtual Device (AVD) Manager



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	

Anatomy of an Android App project

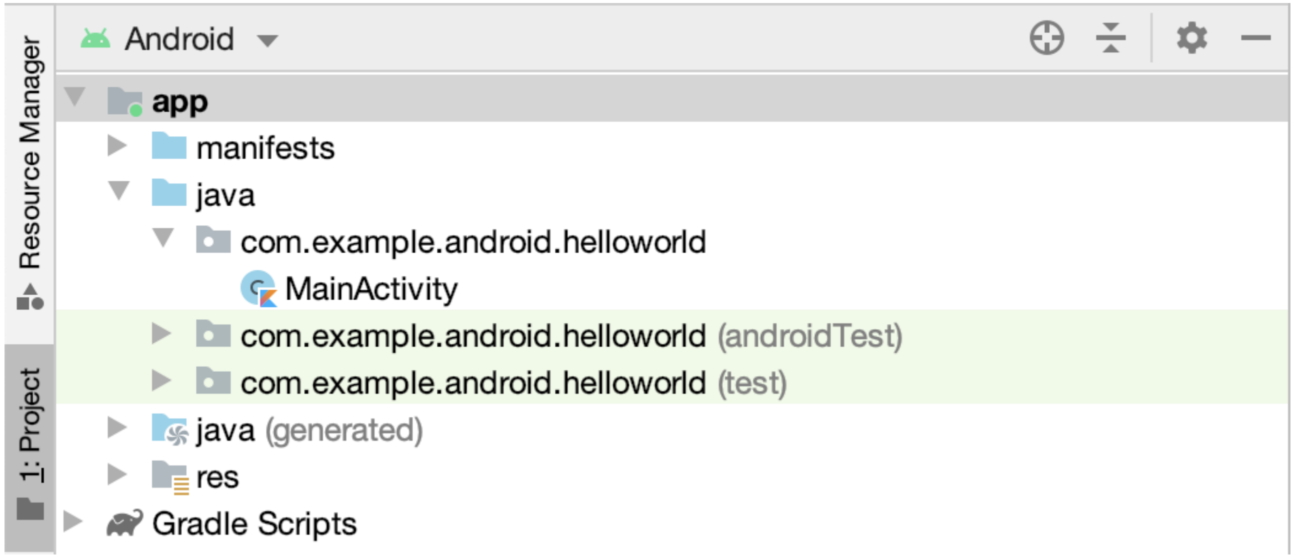
Anatomy of a basic app project

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

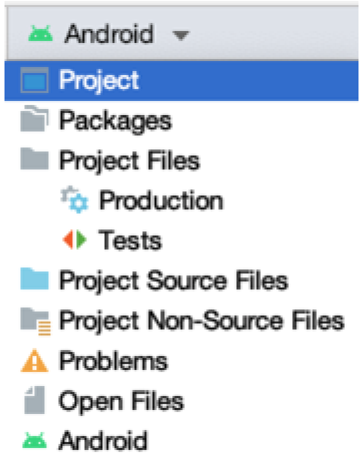
Android app project structure

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

Browse files in Android Studio



```
app
  manifests
  java
    com.example.android.helloworld
      MainActivity
    com.example.android.helloworld (androidTest)
    com.example.android.helloworld (test)
  java (generated)
  res
Gradle Scripts
```

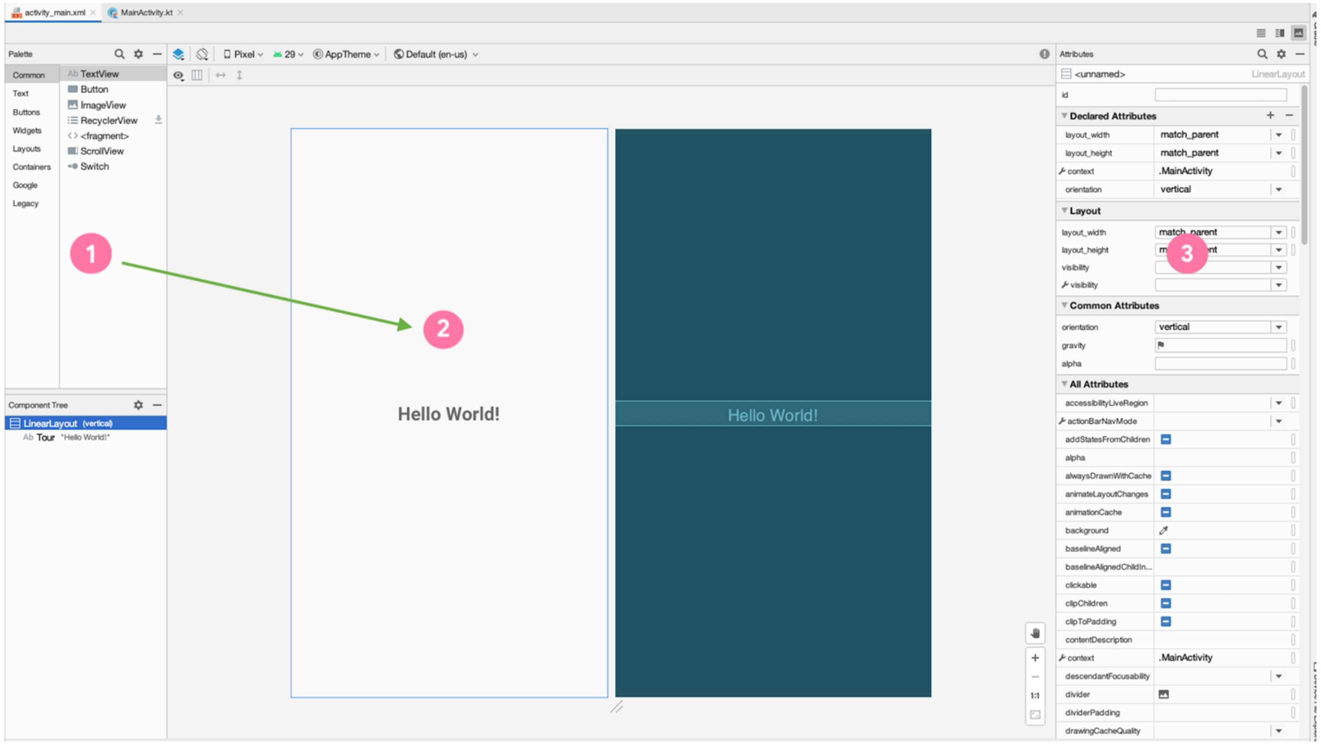


Layouts and resources in Android

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



<unnamed> LinearLayout

id	
Declared Attributes	
layout_width	match_parent
layout_height	match_parent
context	.MainActivity
orientation	vertical
Layout	
layout_width	match_parent
layout_height	m[illegible]nt
visibility	
visibility	
Common Attributes	
orientation	vertical
gravity	
alpha	
All Attributes	
accessibilityLiveRegion	
actionBarNavMode	
addStatesFromChildren	
alpha	
alwaysDrawnWithCache	
animateLayoutChanges	
animationCache	
background	
baselineAligned	
baselineAlignedChildIn...	
clickable	
clipChildren	
clipToPadding	
contentDescription	
context	.MainActivity
descendantFocusability	
divider	
dividerPadding	
drawingCacheQuality	

XML Layouts

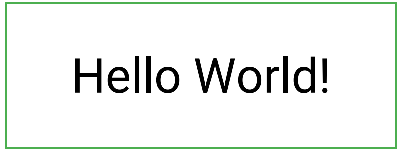
You can also edit your layout in XML.

- 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

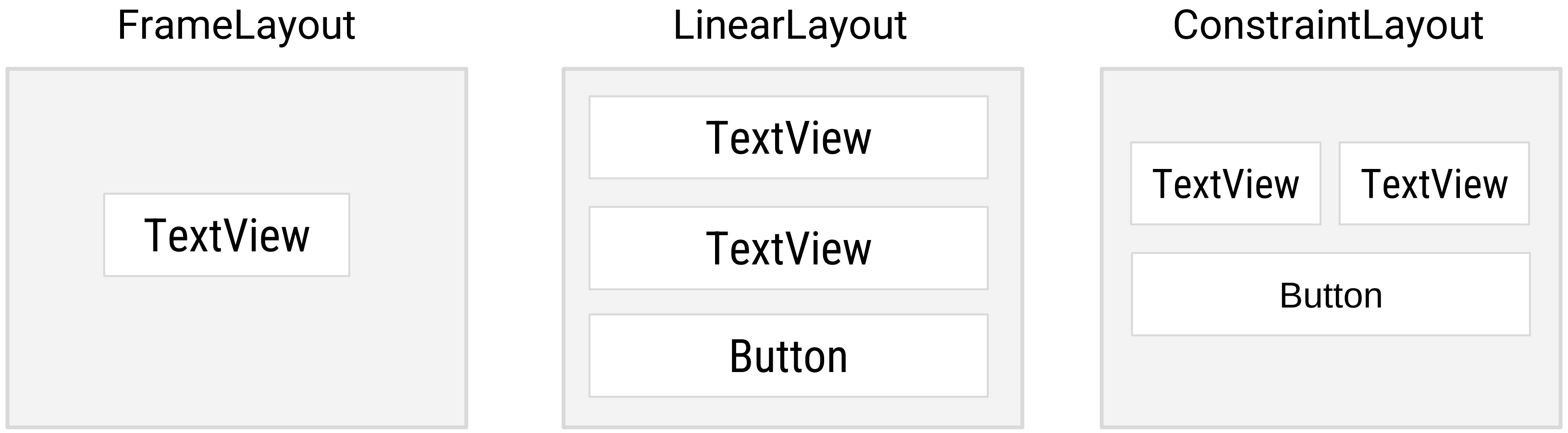
```
android:layout_width="match_parent"
```

- Fixed value (use dp units)

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

ViewGroups

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

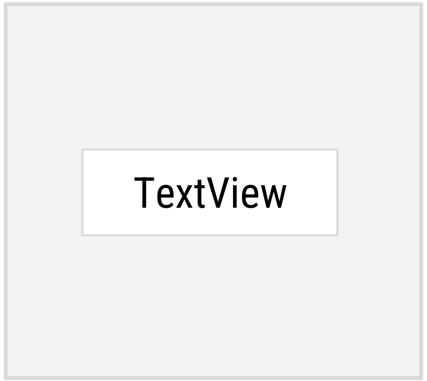


The ViewGroup is the parent and the views inside it are its children.

FrameLayout example

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

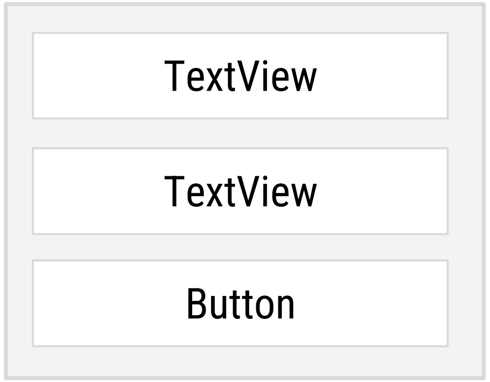
```
<FrameLayout
   android:layout_width="match_parent"
   android:layout_height="match_parent">
   <TextView
       android:layout_width="match_parent"
       android:layout_height="match_parent"
       android:text="Hello World!"/>
</FrameLayout>
```



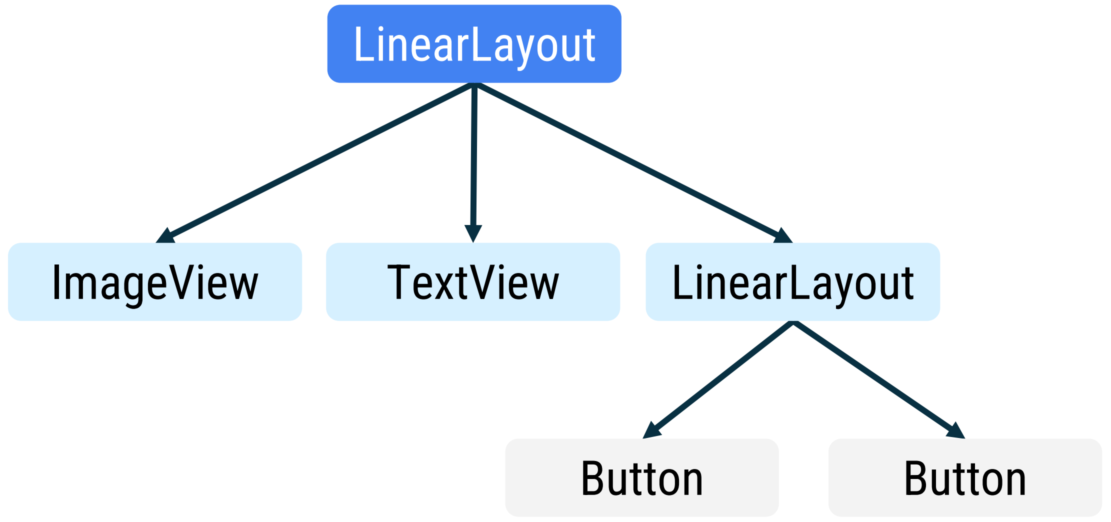
LinearLayout example

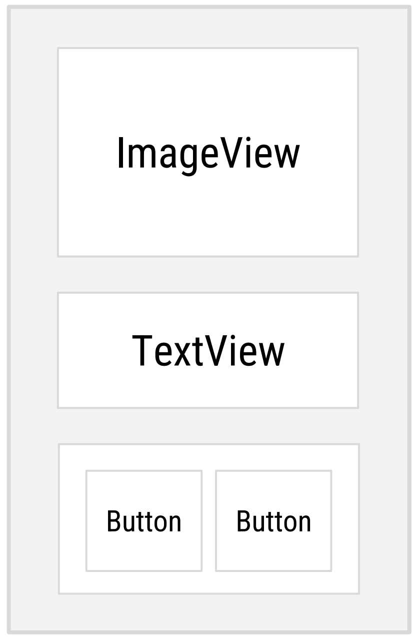
- Aligns child views in a row or column
- Set `android:orientation` to `horizontal` or `vertical`

```
<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.

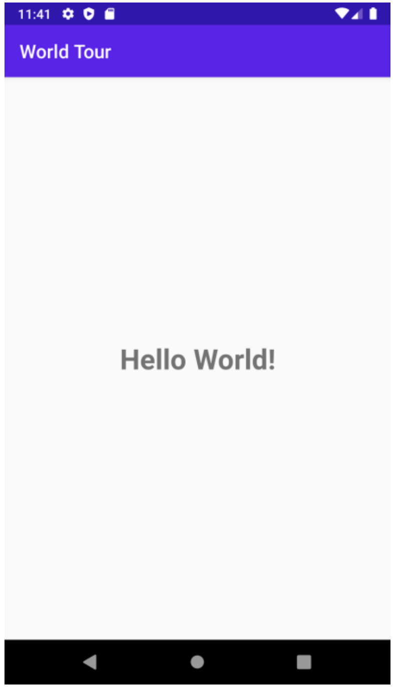
```
<TextView
     android:id="@+id/helloTextView"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:text="Hello World!"/>
```

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

`R.id.helloTextView`

Activities

What's an Activity?



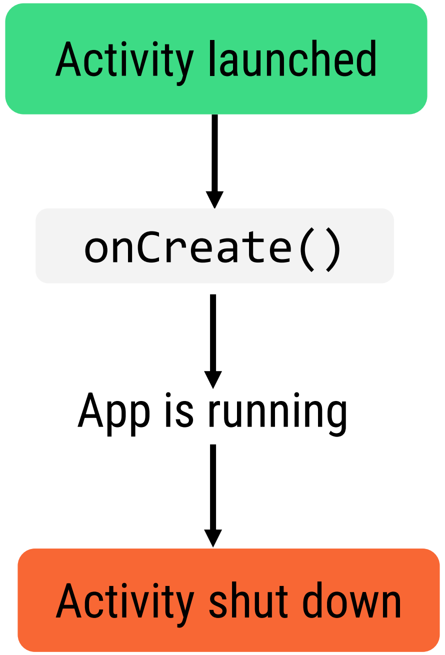
- An Activity is a means for the user to accomplish one main goal.
- An Android app is composed of one or more activities.

MainActivity.kt

```
class MainActivity : AppCompatActivity() {

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

How an Activity runs

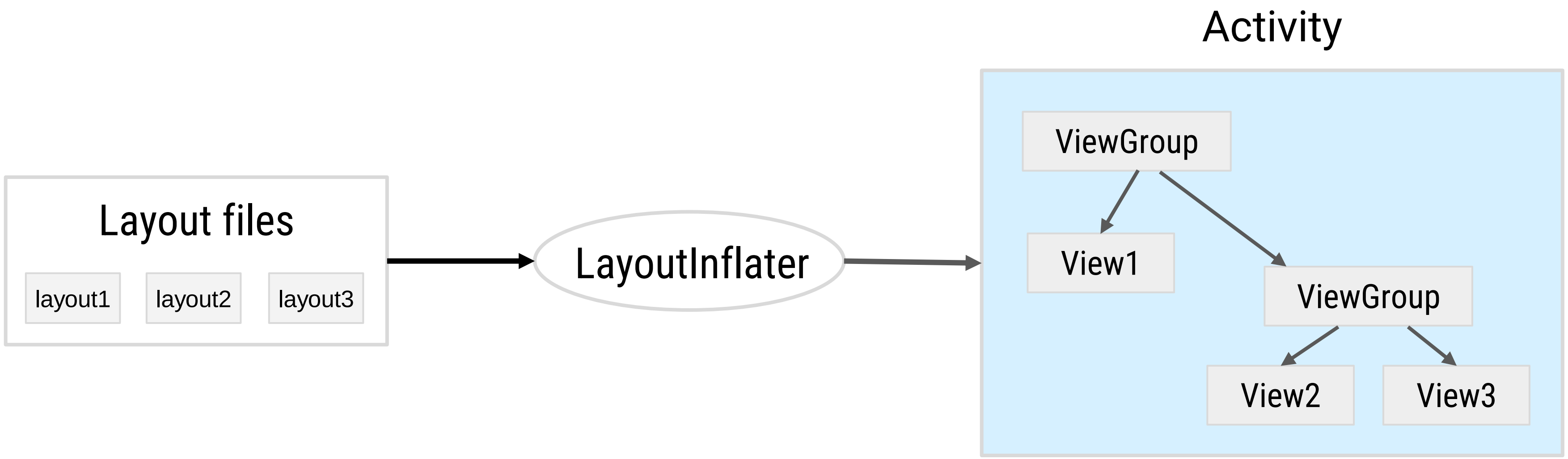


Implement the onCreate() callback

Called when the system creates your Activity

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

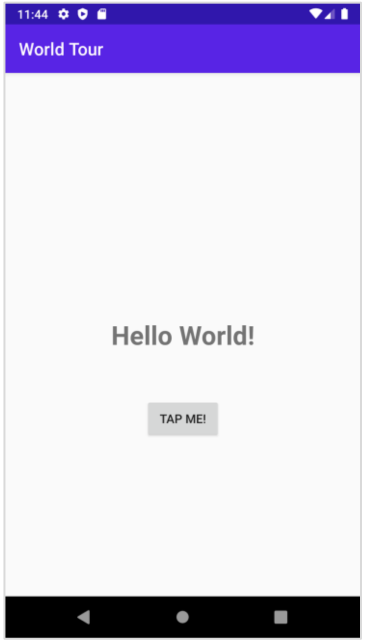
Layout inflation



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`:

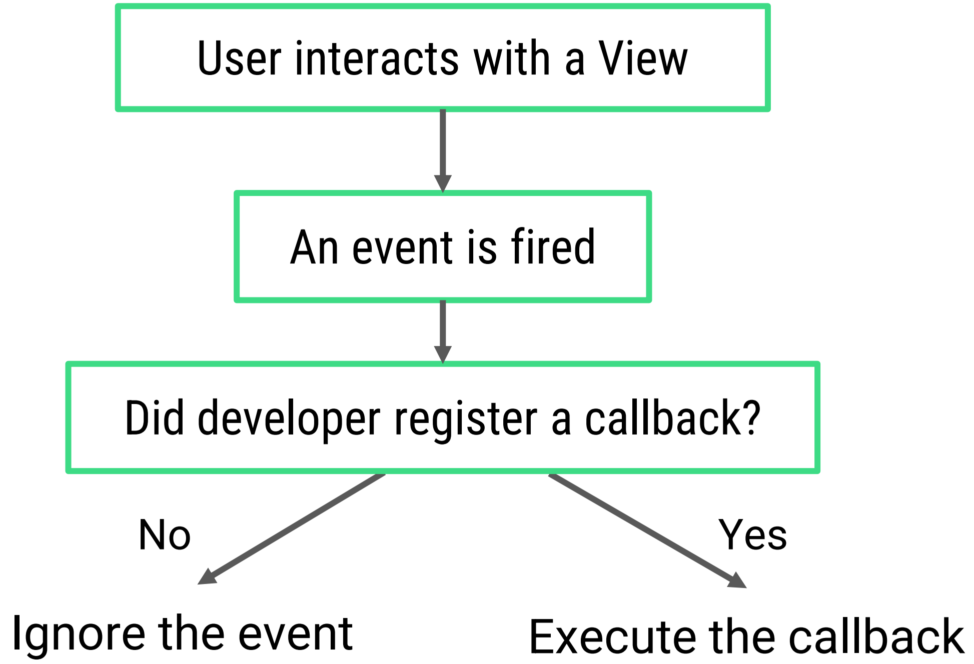
Get a reference to the View in the view hierarchy:

```
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


User interacts with a View
An event is fired
Did developer register a callback?
No
Yes
Ignore the event
Execute the callback

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

```kotlin
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

Summary

Summary

In Lesson 2, you learned how to:

- Use Views and `ViewGroups` to build the user interface of your app
- Access resources in your app from `R.<resource_type>.<resource_name>`
- Define app behavior in the Activity (for example, register `OnClickListener`)
- Use Gradle as the build system to build your app

Learn more

- Layouts
- LinearLayout
- Input events overview
- View
- ViewGroup

Pathway

Practice what you've learned by completing the pathway:

Lesson 4: Build your first Android app