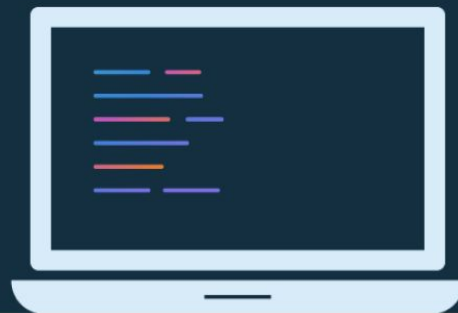




# Lesson 4: Build your first Android app



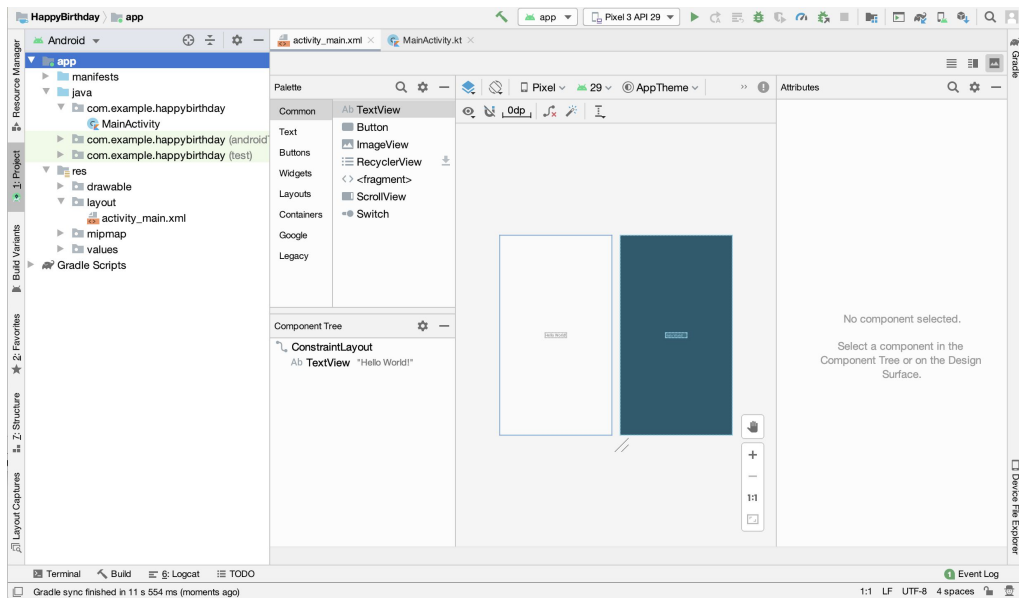
# About this lesson

## Lesson 4: Build your first Android app

- [Your first app](#)
- [Anatomy of an Android app](#)
- [Layouts and resources in Android](#)
- [Activities](#)
- [Make an app interactive](#)
- [Gradle: Building an Android app](#)
- [Accessibility](#)
- [Summary](#)

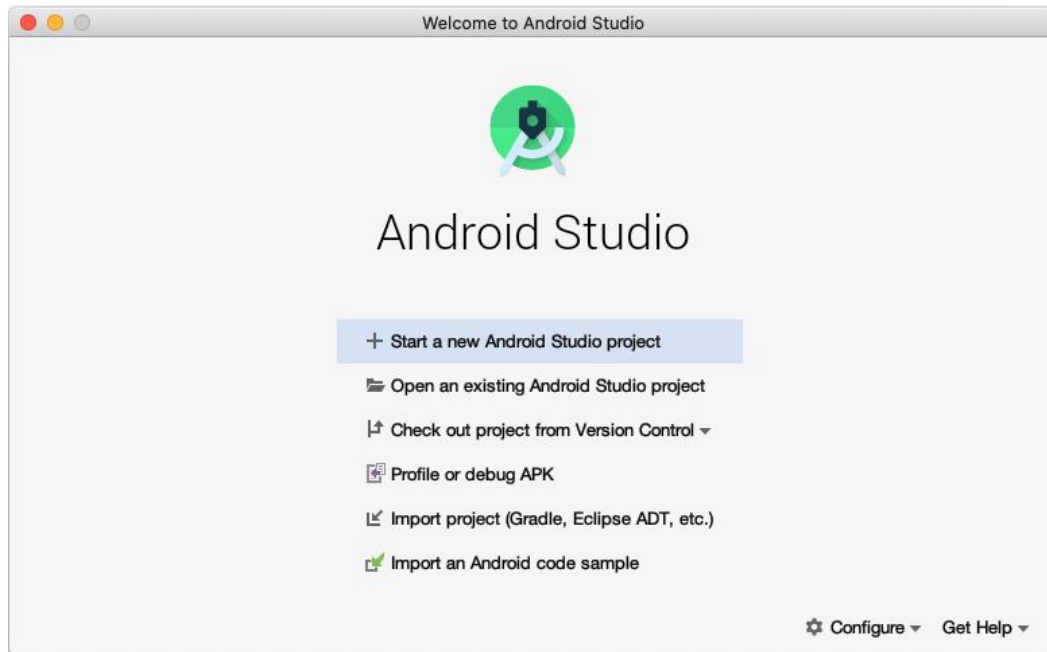
# Android Studio

Official IDE for building Android apps

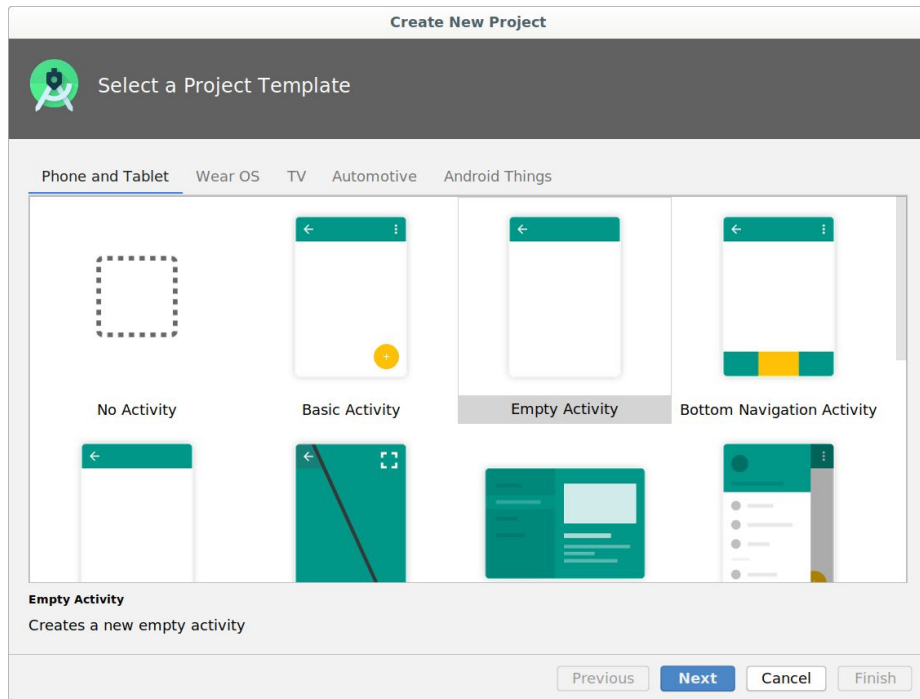


# Your first app

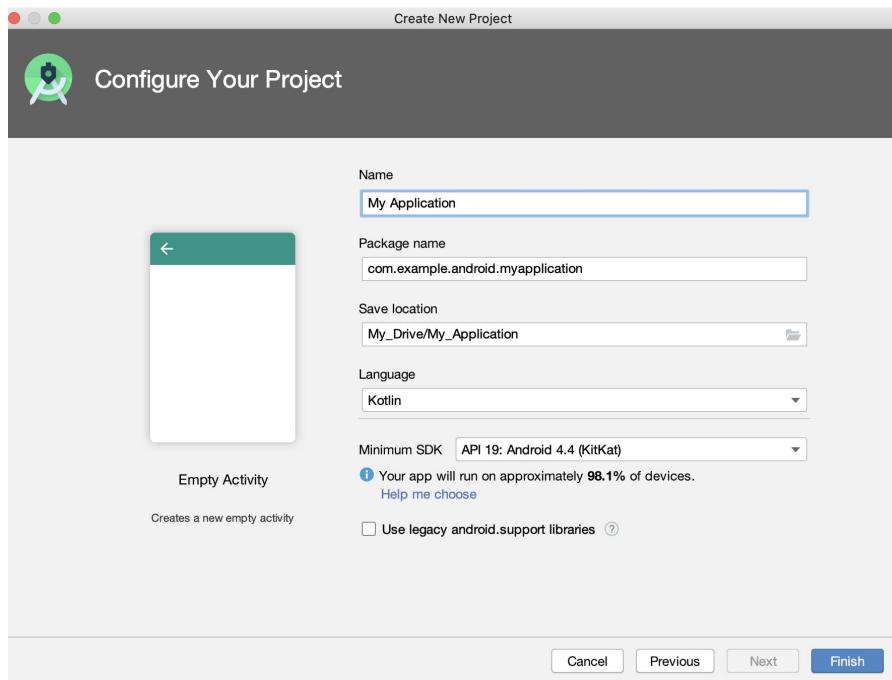
# Open Android Studio



# Create new project



# Enter your project details



The screenshot shows the 'Create New Project' dialog in Android Studio. The title bar says 'Create New Project'. The main header is 'Configure Your Project' with a green Android robot icon. On the left, there is a preview of an 'Empty Activity' with a green header bar and a white body. Below the preview, it says 'Empty Activity' and 'Creates a new empty activity'. On the right, there are several input fields and dropdowns:

- Name:** A text field containing 'My Application'.
- Package name:** A text field containing 'com.example.android.myapplication'.
- Save location:** A text field containing 'My\_Drive/My\_Application' with a folder icon on the right.
- Language:** A dropdown menu showing 'Kotlin'.
- Minimum SDK:** A dropdown menu showing 'API 19: Android 4.4 (KitKat)'.

Below these fields, there is a blue information icon followed by the text: 'Your app will run on approximately 98.1% of devices.' and a link 'Help me choose'. At the bottom, there is a checkbox labeled 'Use legacy android.support libraries' with a question mark icon. At the very bottom, there are four buttons: 'Cancel', 'Previous', 'Next', and 'Finish'.

# 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





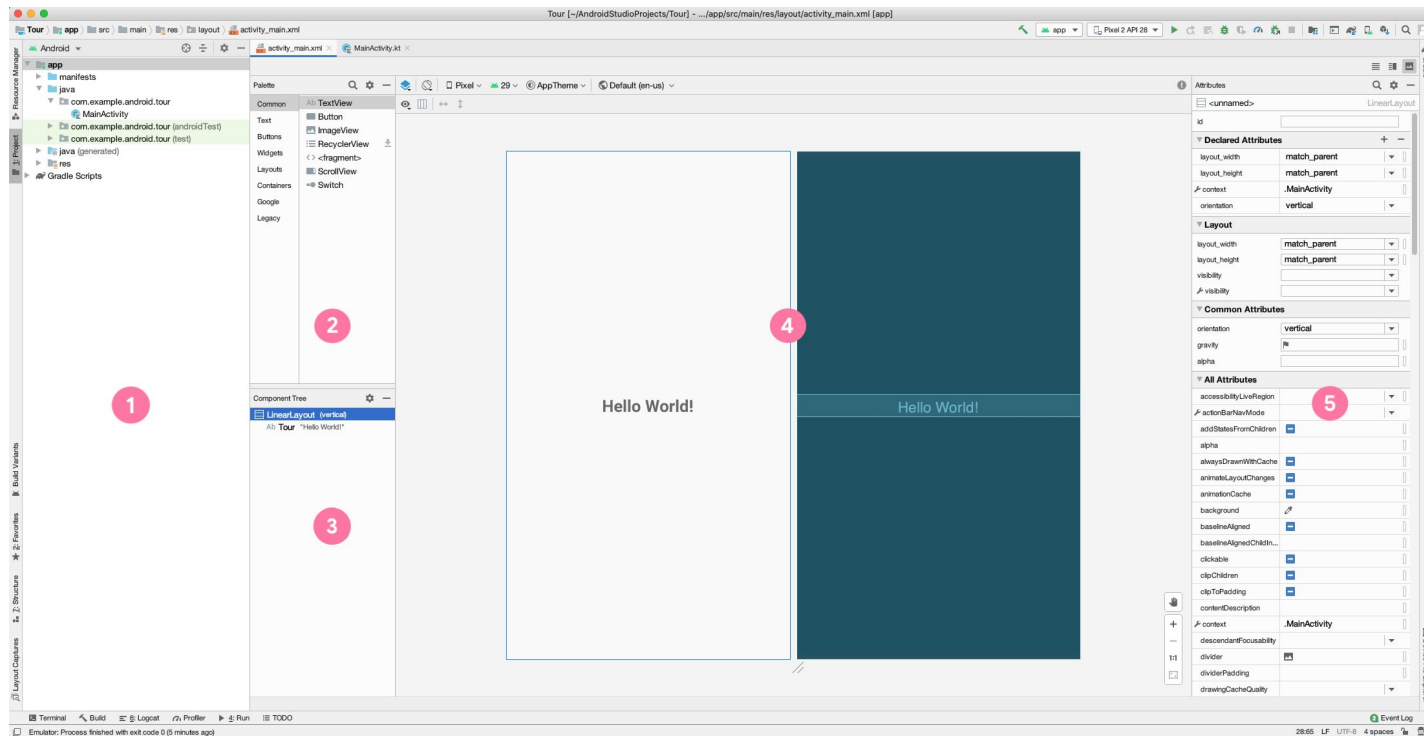
# Choose API levels for your app

- 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



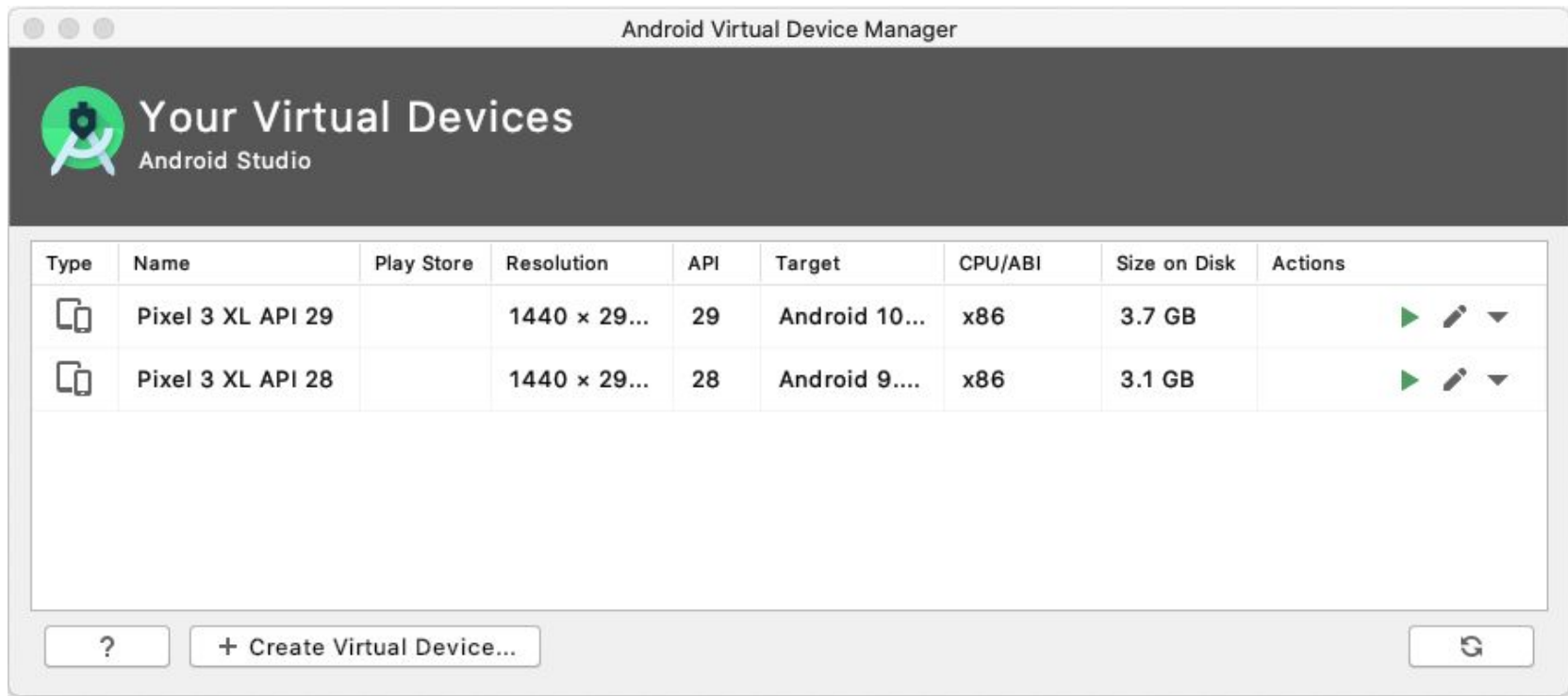
# Run your app



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



# Android Virtual Device (AVD) Manager

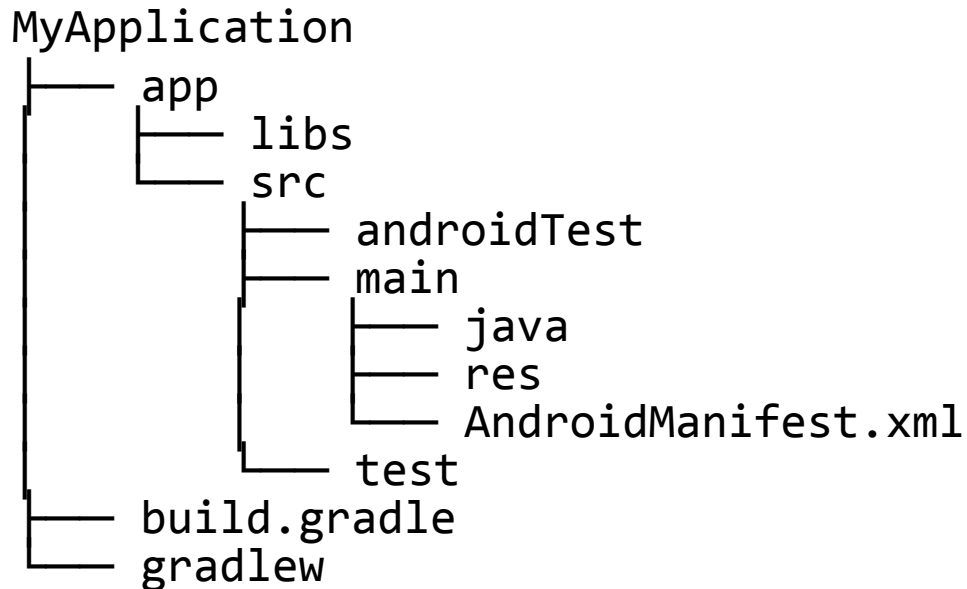


# Anatomy of an Android App project

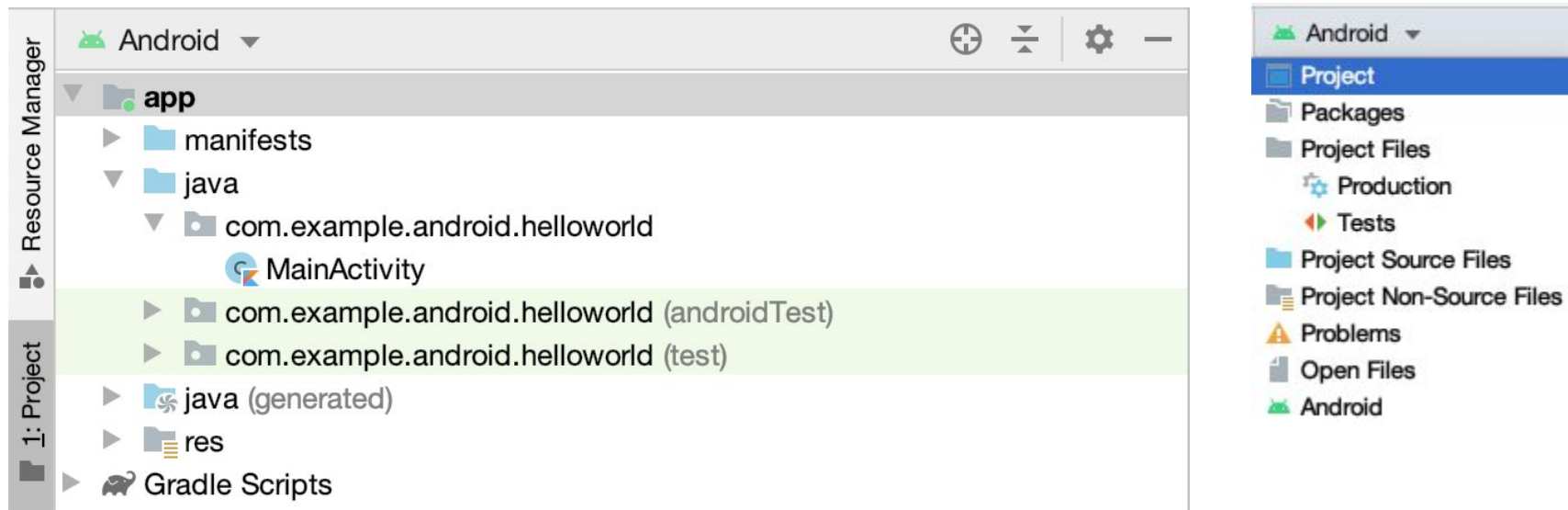
# Anatomy of a basic app project

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

# Android app project structure



# Browse files in Android Studio



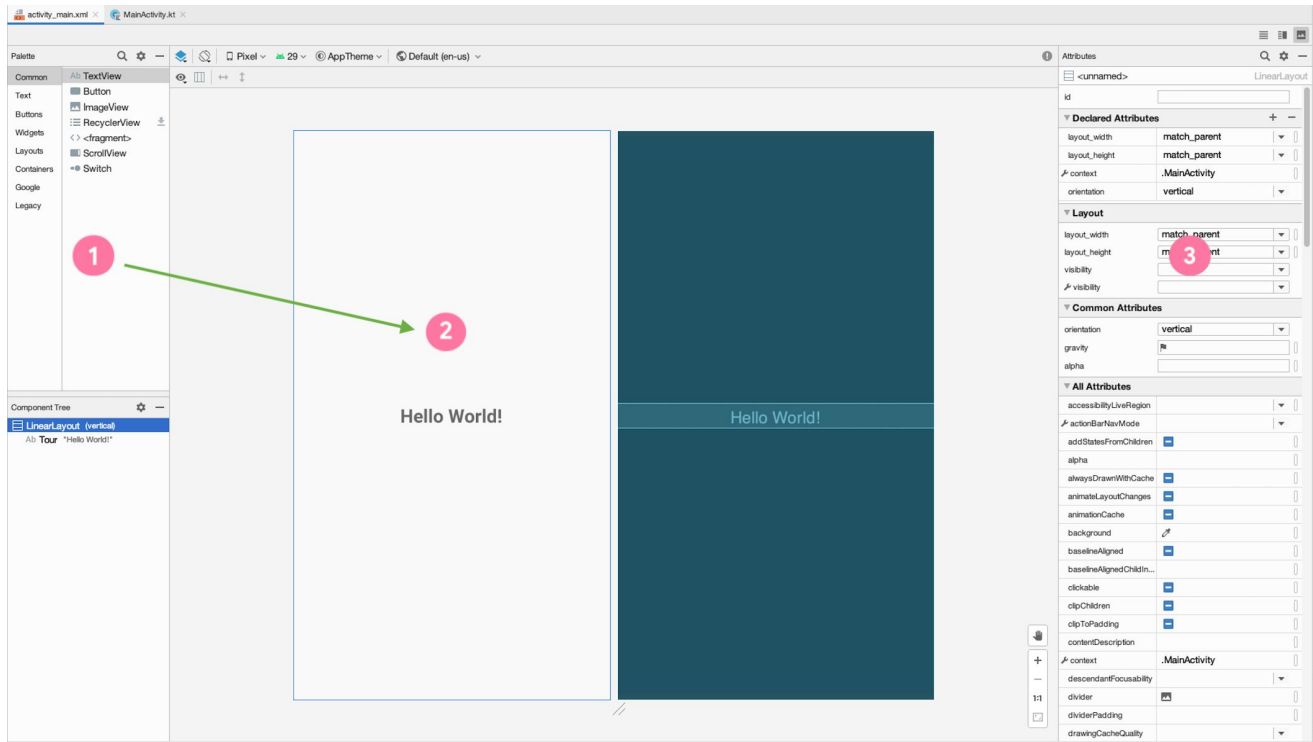


# Layouts and resources in Android

# 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"
```

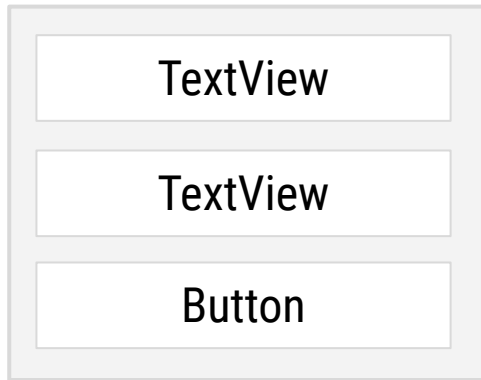
# ViewGroups

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

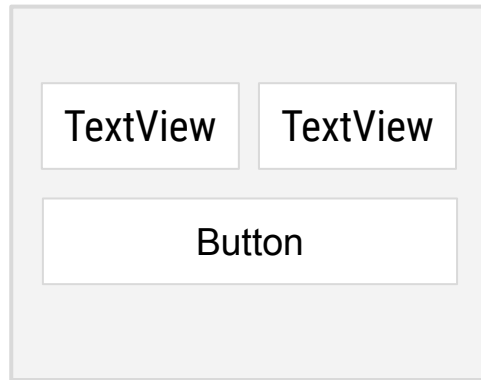
FrameLayout



LinearLayout



ConstraintLayout

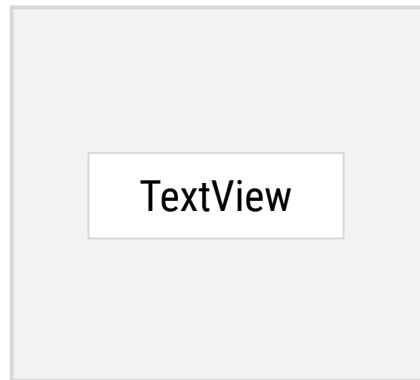


The `ViewGroup` is the parent and the views inside it are its children.

# 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**

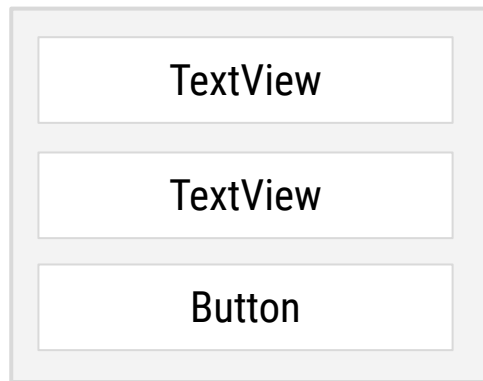
```
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:orientation="vertical">
```

```
    <TextView ... />
```

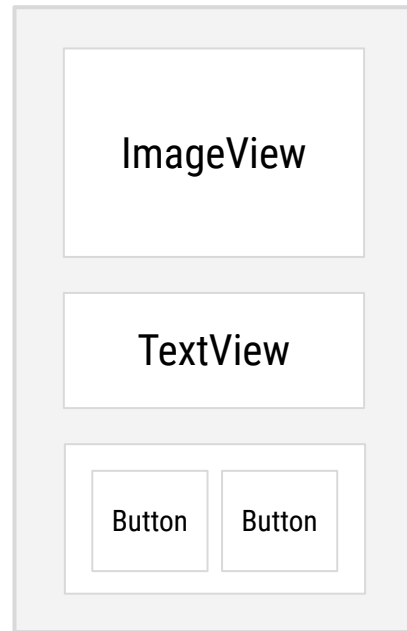
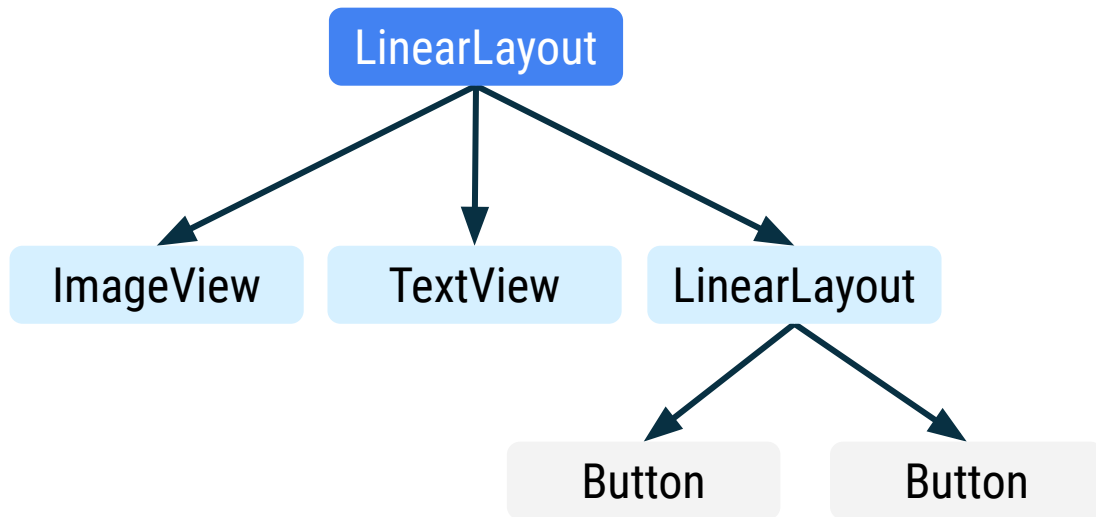
```
    <TextView ... />
```

```
    <Button ... />
```

**</LinearLayout>**



# View hierarchy



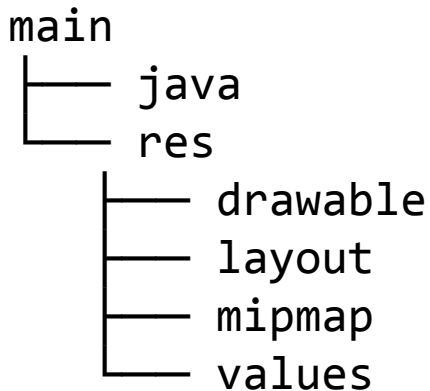
# App resources

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.



# 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.

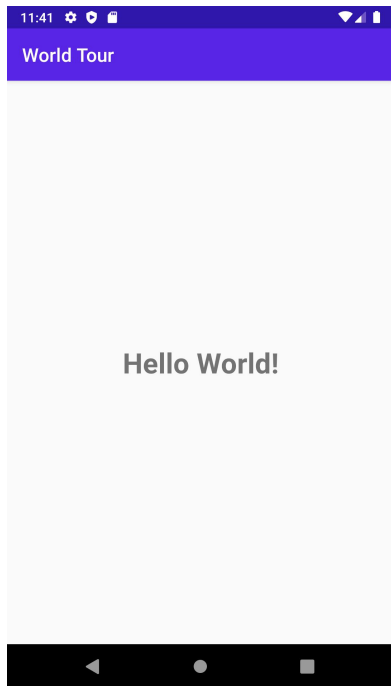
```
<TextView
    android:id="@+id/helloTextView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!"/>
```

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

```
R.id.helloTextView
```

# Activities

# What's an Activity?



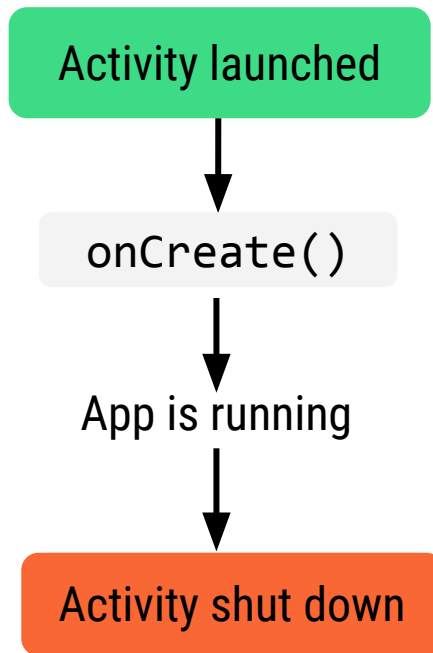
- An Activity is a means for the user to accomplish one main goal.
- An Android app is composed of one or more activities.



# MainActivity.kt

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

# How an Activity runs

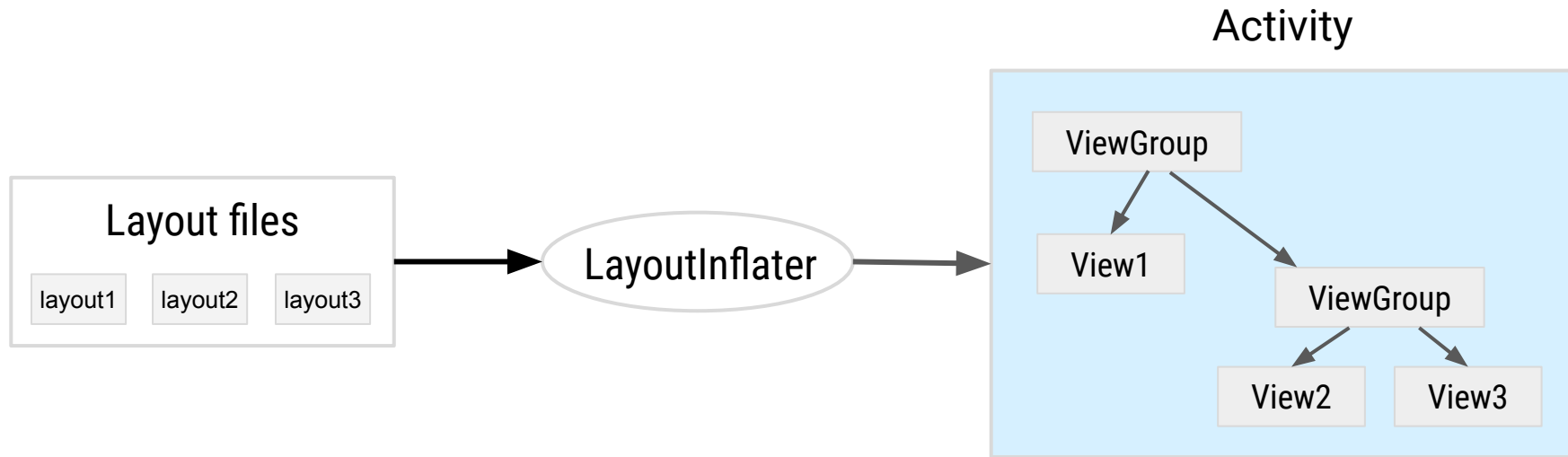


# Implement the onCreate() callback

Called when the system creates your Activity

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

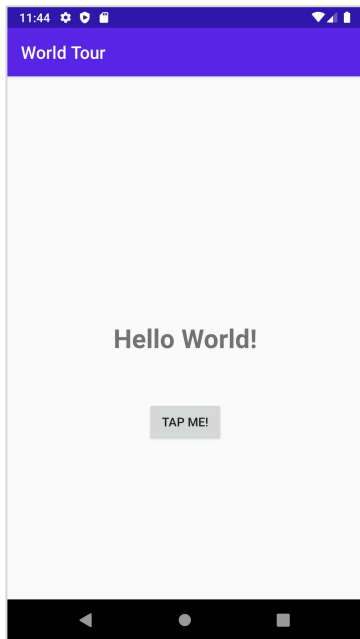
# Layout inflation



# 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`:

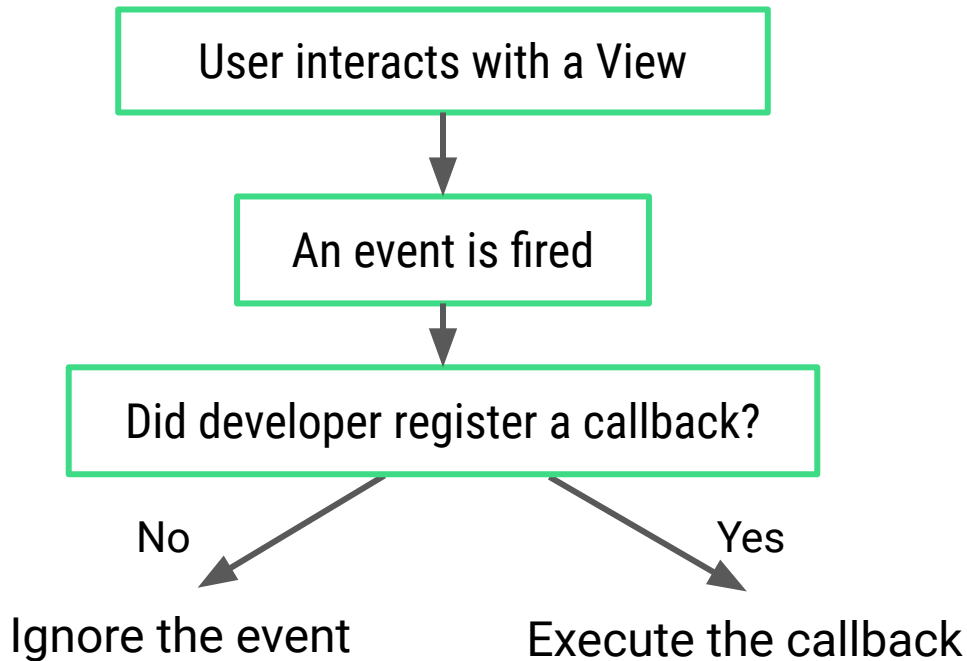
Get a reference to the View in the view hierarchy:

```
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()  
    mavenCentral()  
}
```

# 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

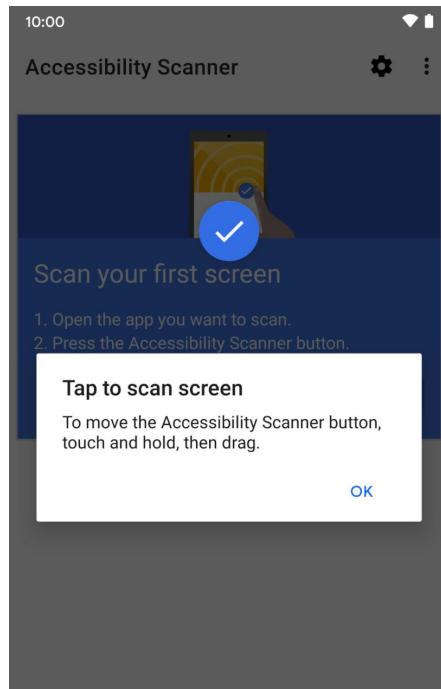




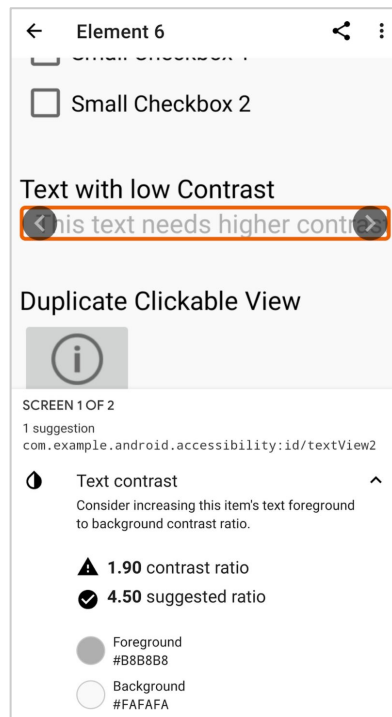
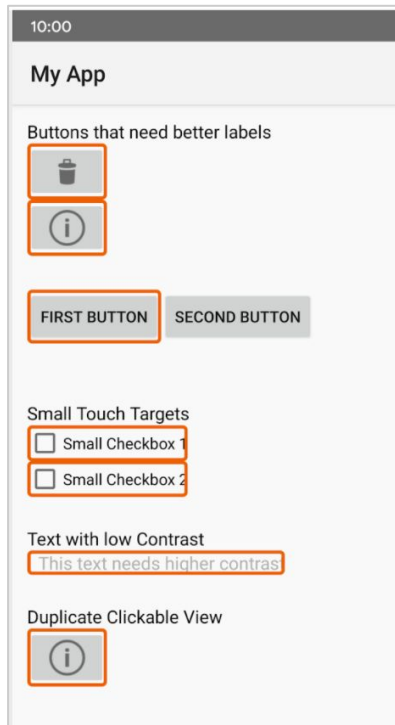
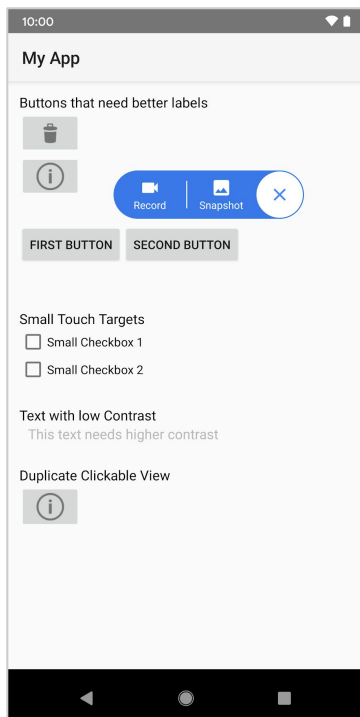
# Accessibility Scanner

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

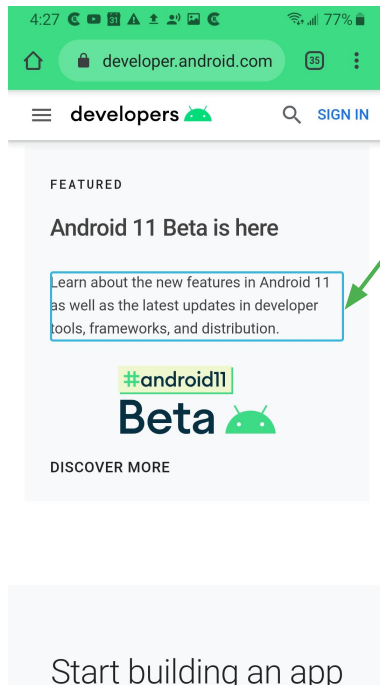
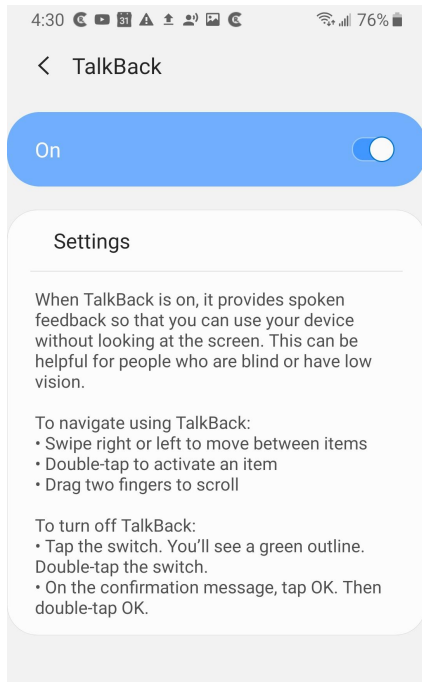
```
android:importantForAccessibility="no"
```

- 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

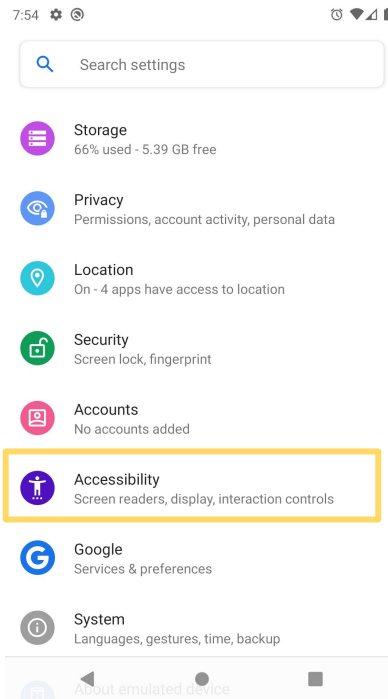
# Switch access

- 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





# Accessibility Resources

- [Build more accessible apps](#)
- [Principles for improving app accessibility](#)
- [Basic Android Accessibility codelab](#)
- [Material Design best practices on accessibility](#)



# Summary

# Summary

In Lesson 4, 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
- Follow best practices to make your apps more accessible

# 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](#)

