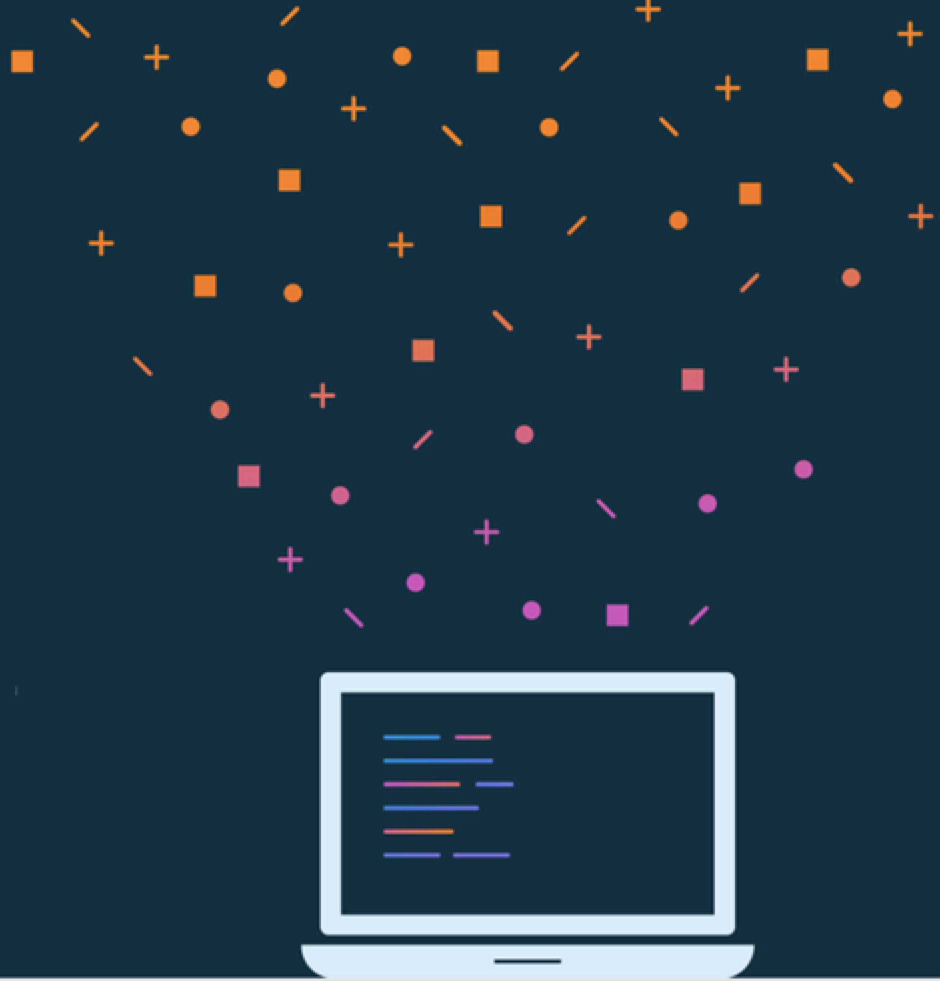
Android Development with Kotlin



Credits

The slides are adapted from:

- Android Development Fundamentals by Google
  - Under Common Criteria
- Android Development with Kotlin by Google
  - Under Apache 2.0

About this course

Prerequisites

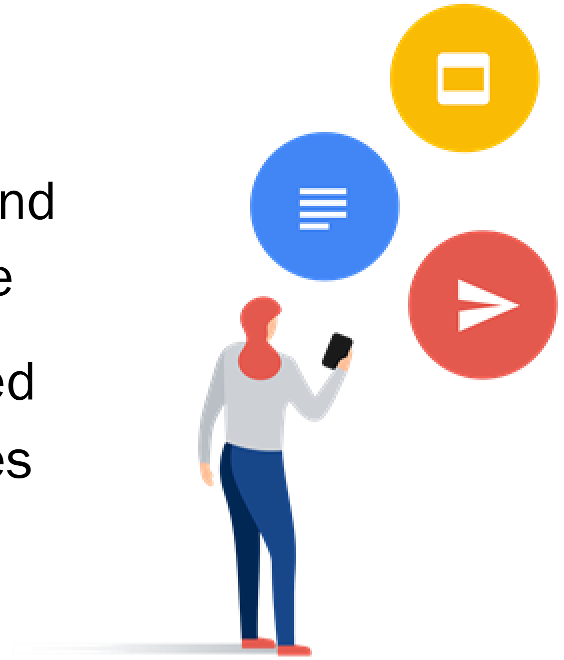
- Experience in an object-oriented programming language
- Comfortable using an IDE
- Familiar with using GitHub
- Access to a computer and internet connection
- (Optional) Android device and USB cable

What you'll learn

- How to build a variety of Android apps in Kotlin
- Kotlin language essentials
- Best practices for app development
- Resources to keep learning

The opportunity

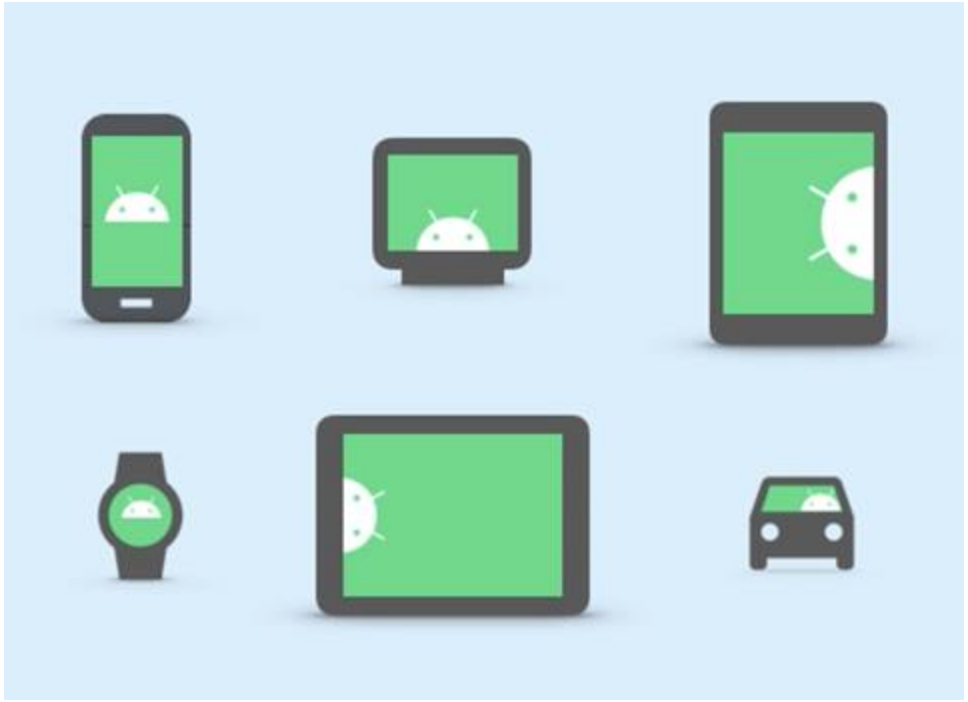
- Mobile devices are becoming increasingly commonplace
- Mobile apps connect users to information and services that can improve their quality of life
- Many industries have yet to be revolutionized through mobile, and offer great opportunities for new businesses and solutions



Android

- Open-source mobile platform
- 15 major platform releases so far
- 3+ billion monthly active Android devices
- 2.5+ billion monthly active Google Play users

Available across different form factors



Build Android apps in Kotlin

+

Kotlin

A modern programming language that helps developers be more productive.

Benefits of Kotlin

- Expressive and concise
- Safer code
- Interoperable
- Structured Concurrency

Idiomatic Kotlin

- Kotlin is at its best when used idiomatically
- Avoid just translating Java into Kotlin
- As you learn more Kotlin, you'll find easier, more concise ways to do things
- For a list of common Kotlin idioms, refer to the Kotlin Language Guide on Idioms

An Introduction to Android

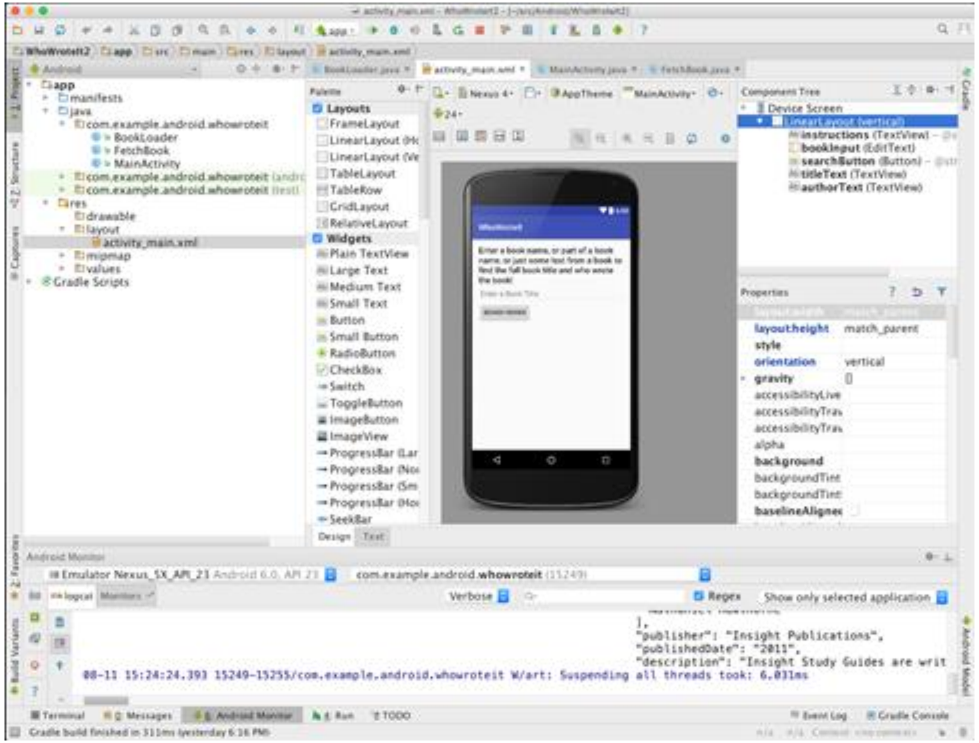
What is Android?

- Mobile operating system based on Linux kernel
- User Interface for touch screens
- Used on over 80% of all smartphones
- Powers devices such as watches, TVs, and cars
- Over 2 Million Android apps in Google Play store
- Highly customizable for devices / by vendors
- Open source

Android Software Developer Kit (SDK)

- Development tools (debugger, monitors, editors)
- Libraries (maps, wearables)
- Virtual devices (emulators)
- Documentation (developer.android.com)
- Sample code

Android Studio



```
],
"publisher": "Insight Publications",
"publishedDate": "2011",
"description": "Insight Study Guides are writ
08-11 15:24:24.393 15249-15255/com.example.android.whowroteit W/art: Suspending all threads took: 6.031ms
```

Gradle build finished in 311ms (yesterday 6:16 PM)

- Official Android IDE
- Develop, run, debug, test, and package apps
- Monitors and performance tools
- Virtual devices
- Project views
- Visual layout editor

Google Play store

Publish apps through Google Play store:

- Official app store for Android
- Digital distribution service operated by Google

Android versions

Codename	Version	Released	API Level
Honeycomb	3.0 - 3.2.6	Feb 2011	11 - 13
Ice Cream Sandwich	4.0 - 4.0.4	Oct 2011	14 - 15
Jelly Bean	4.1 - 4.3.1	July 2012	16 - 18
KitKat	4.4 - 4.4.4	Oct 2013	19 - 20
Lollipop	5.0 - 5.1.1	Nov 2014	21 - 22
Marshmallow	6.0 - 6.0.1	Oct 2015	23
Nougat	7.0 - 7.1	Sept 2016	24 - 25
Oreo	8.0 - 8.1	Aug 2017	26 - 27
Pie	9.0	Sept 2018	28
Android 10	10	Sept 2019	29
Android 11	11	Sept 2020	30
Android 12	12	Oktober 2021	31, 32
Android 13	13	August 2022	33

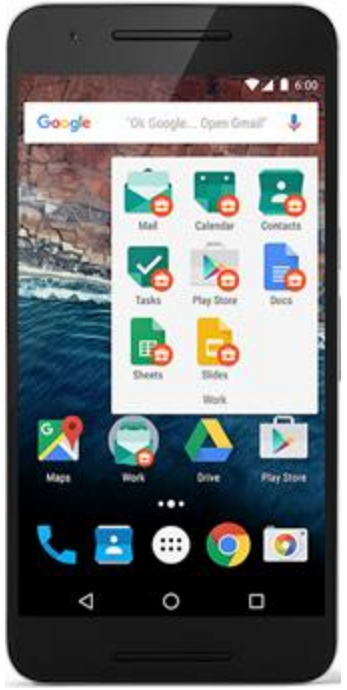
Android History and Platform Versions for more and earlier versions before 2011

What is an Android app?

- One or more interactive screens
- Written using Java Programming Language or Kotlin and XML
- Uses the Android Software Development Kit (SDK)
- Uses Android libraries and Android Application Framework
- Executed by Android Runtime Virtual machine (ART)

Android Special App: home screen

- Launcher icons for apps
- Self-updating widgets for live content
- Can be multiple pages
- Folders to organize apps
- "OK Google"



Challenges of Android development

- Multiple screen sizes and resolutions
- Performance: make your apps responsive and smooth
- Security: keep source code and user data safe
- Compatibility: run well on older platform versions
- Marketing: understand the market and your users

  (Hint: It doesn't have to be expensive, but it can be.)

App building blocks

- Resources: Resources define the visual and textual content of your app, which are stored separately from code to allow for easy updates, localization, and efficient management

    Folder: res/

    layouts, images, strings, colors as XML and media files

- Components:

Activities	Services
Broadcast Receivers	Content Providers
Helper Classes	

- Manifest: metadata information about app for the android runtime in XML.

    List of activities, services, Receivers, Content Providers

    App Permission

    Hardware/SW requirements (API level)

- Build configuration: Gradle scripts manage the app compilation and packaging

    build.gradle (Module: app): Defines dependencies, SDK versions, and flavors.

    build.gradle (Project level): Handles repositories and global configurations

    gradle.properties: Stores properties like API keys

    local.properties: Defines paths to the Android SDK.

Types of Application Component

- Activity: is a single screen with a user interface
- Service: performs long-running tasks in background
- Content provider: manages shared set of data
- Broadcast receiver: responds to system-wide announcements

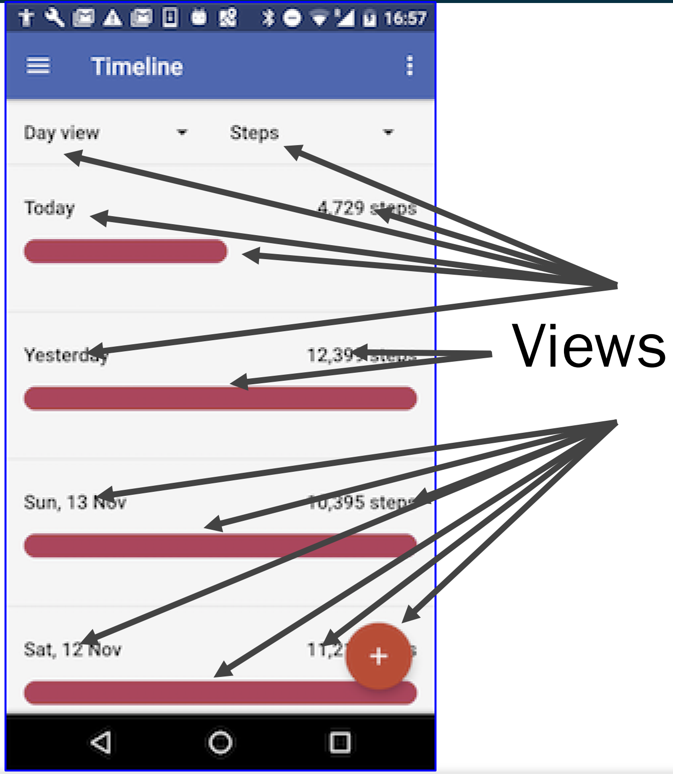
Think of Android as a hotel

- Your app is the guest
- The Android System is the hotel manager
- Services are available when you request them (intents)
    - In the foreground (activities) such as registration
    - In the background (services) such as laundry
- Calls you when a package has arrived (broadcast receiver)
- Access the city's tour companies (content provider)

Views

Everything you see is a view

If you look at your mobile device, every user interface element that you see is a View.



What is a view

Views are Android's basic user interface building blocks.

- display text (TextView class), edit text (EditText class)
- buttons (Button class), menus, other controls
- scrollable (ScrollView, RecyclerView)
- show images (ImageView)
- subclass of View class

Views have properties

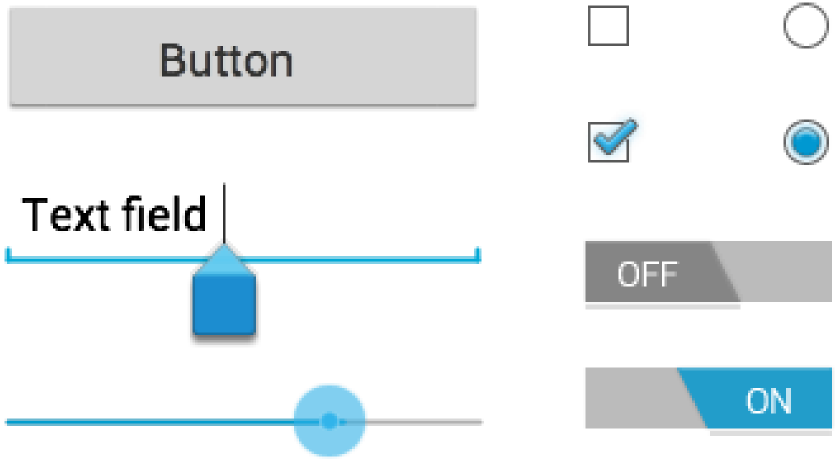
- Have properties (e.g., color, dimensions, positioning)
- May have focus (e.g., selected to receive user input)
- May be interactive (respond to user clicks)
- May be visible or not
- Have relationships to other views

Examples of views

Button

EditText

SeekBar



CheckBox

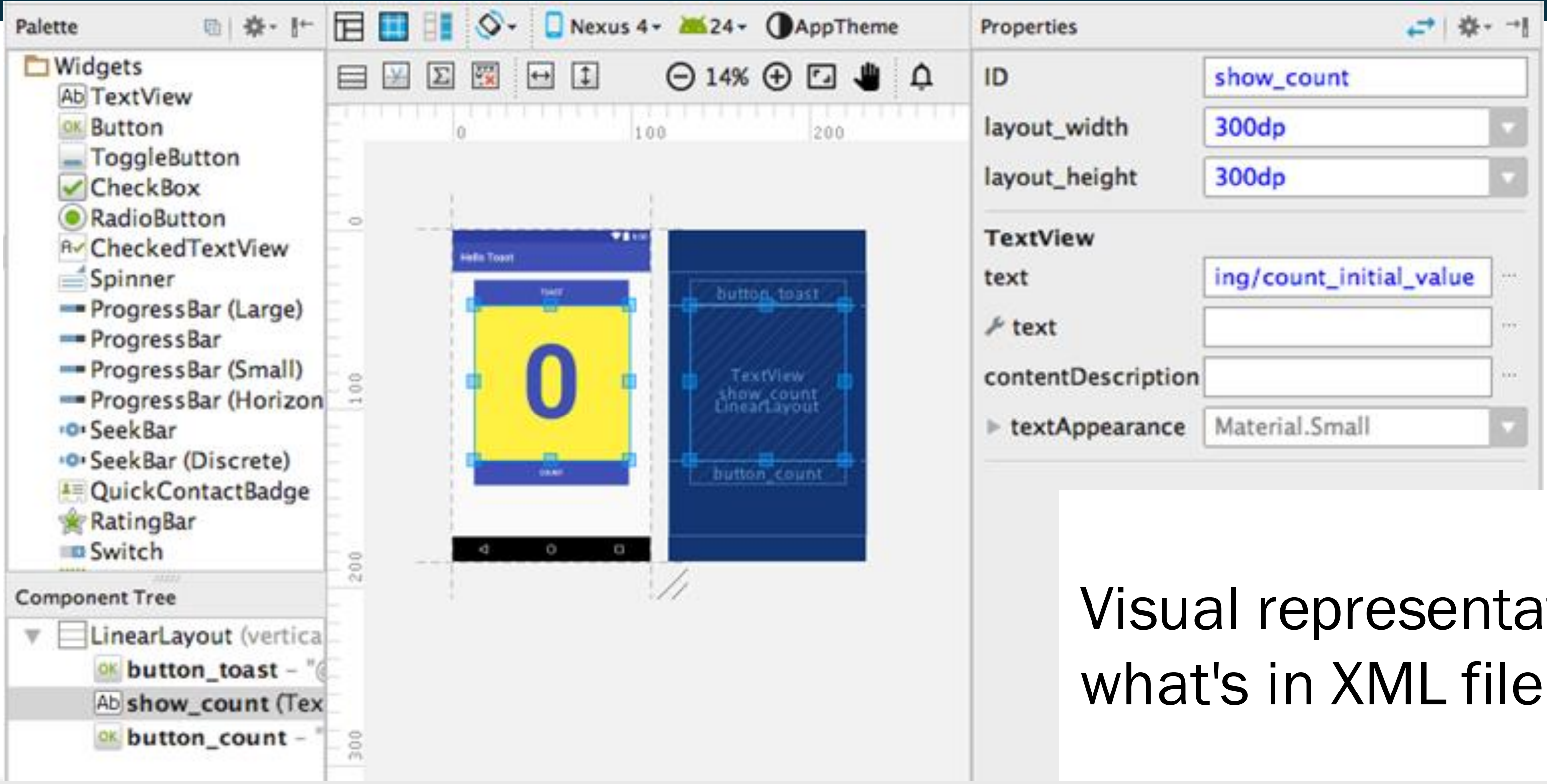
RadioButton

Switch

Creating and laying out views

- Graphically within Android Studio
- XML Files
- Programmatically

Views defined in Layout Editor

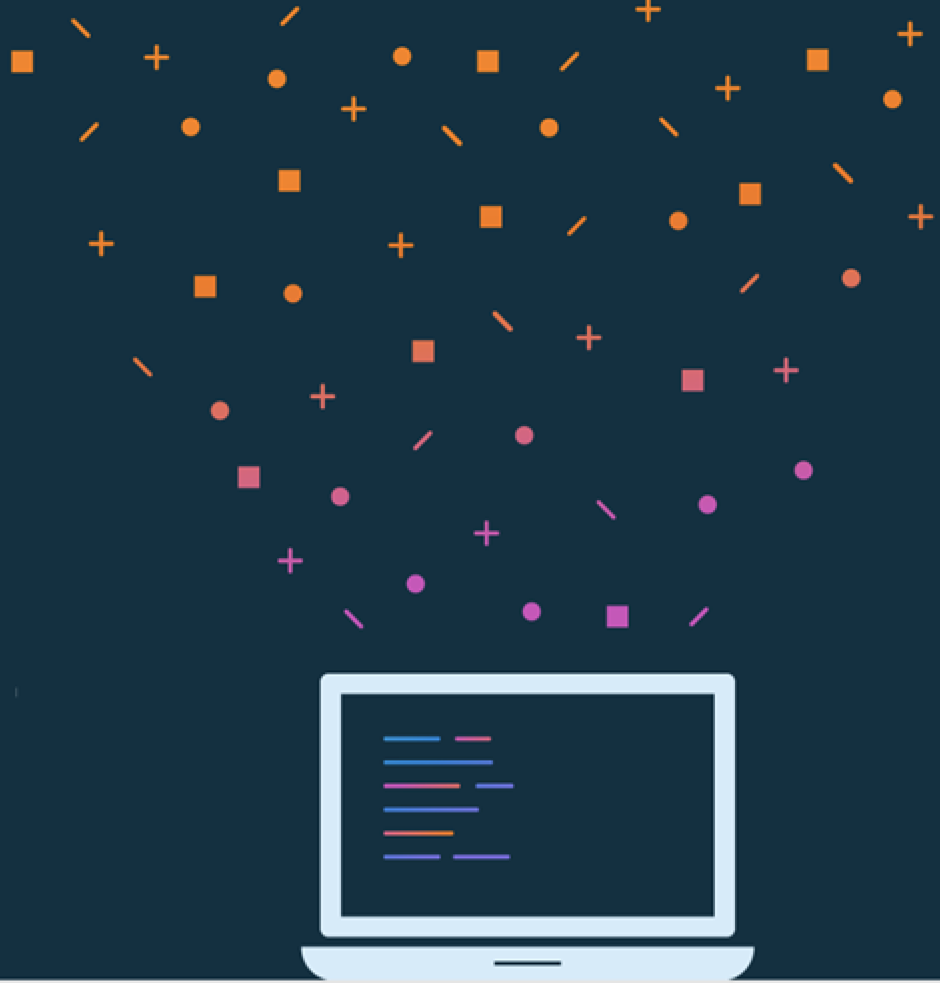


ID	show_count
layout_width	300dp
layout_height	300dp
TextView	
text	ing/count_initial_value
text	
contentDescription	
textAppearance	Material.Small

Visual representation of what's in XML file.

Views defined in XML

```
<TextView
        android:id="@+id/show_count"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@color/myBackgroundColor"
        android:text="@string/count_initial_value"
        android:textColor="@color/colorPrimary"
        android:textSize="@dimen/count_text_size"
        android:textStyle="bold"
/>
```



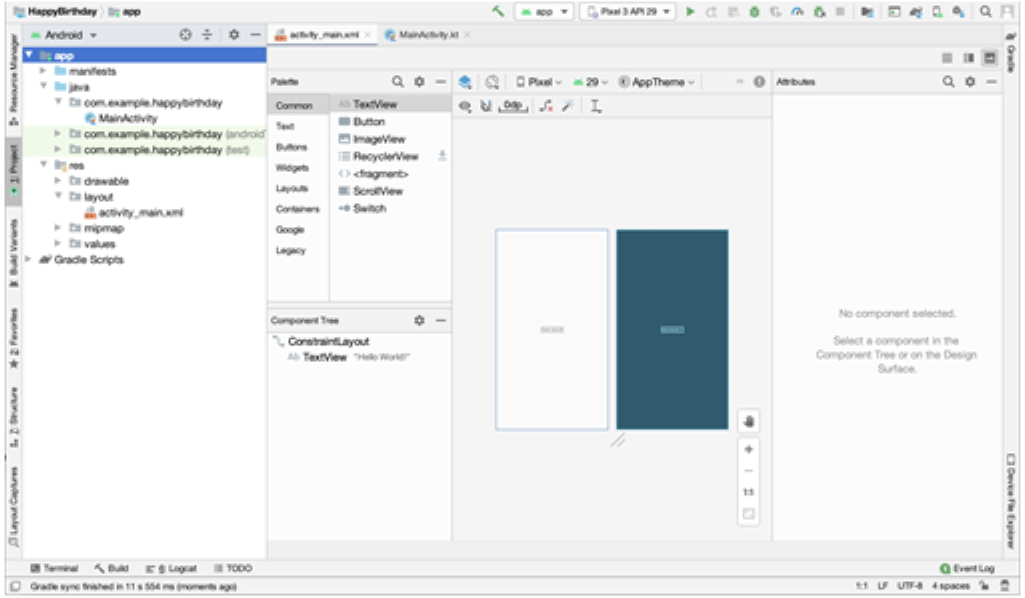
Build your first Android app

About this lesson

Lesson 4: Build your first Android app

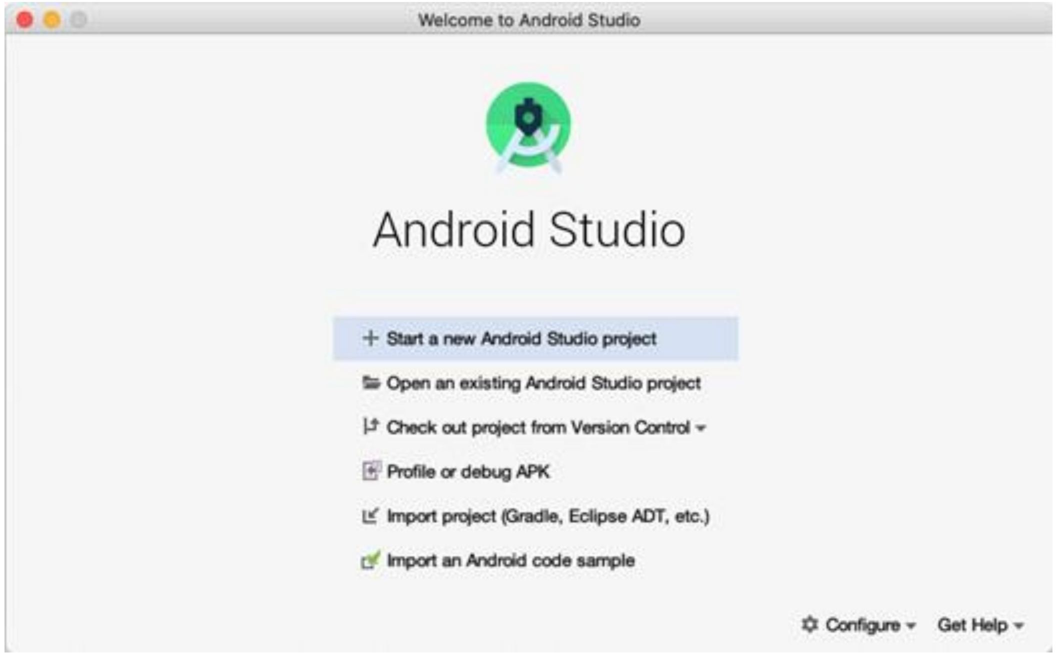
Android Studio

Official IDE for building Android apps



Your first app

Open Android Studio


Welcome to Android Study

Android Studio

+ Start a new Android Studio project

Open an existing Android Studio project

Check out project from Version Control

Profile or debug APK

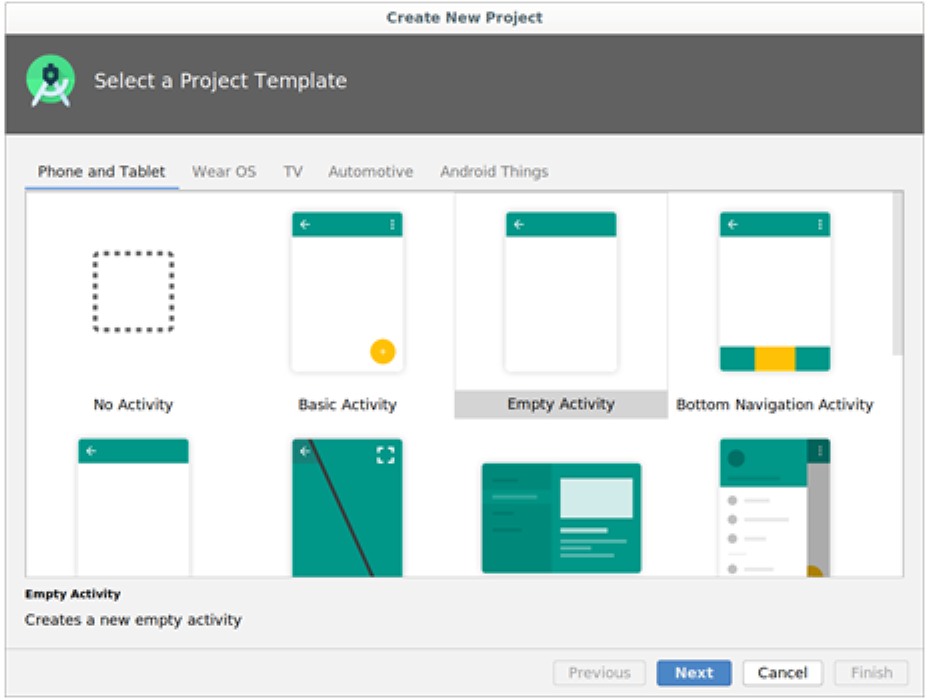
Import project (Gradle, Eclipse ADT, etc.)

Import an Android code sample

Configure

Get Help

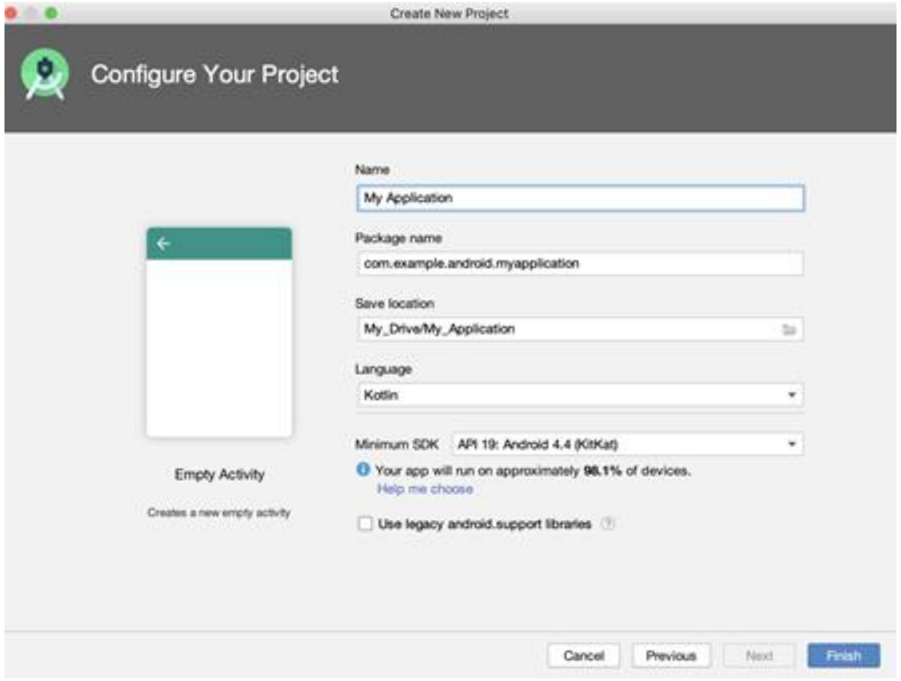
Create new project



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.

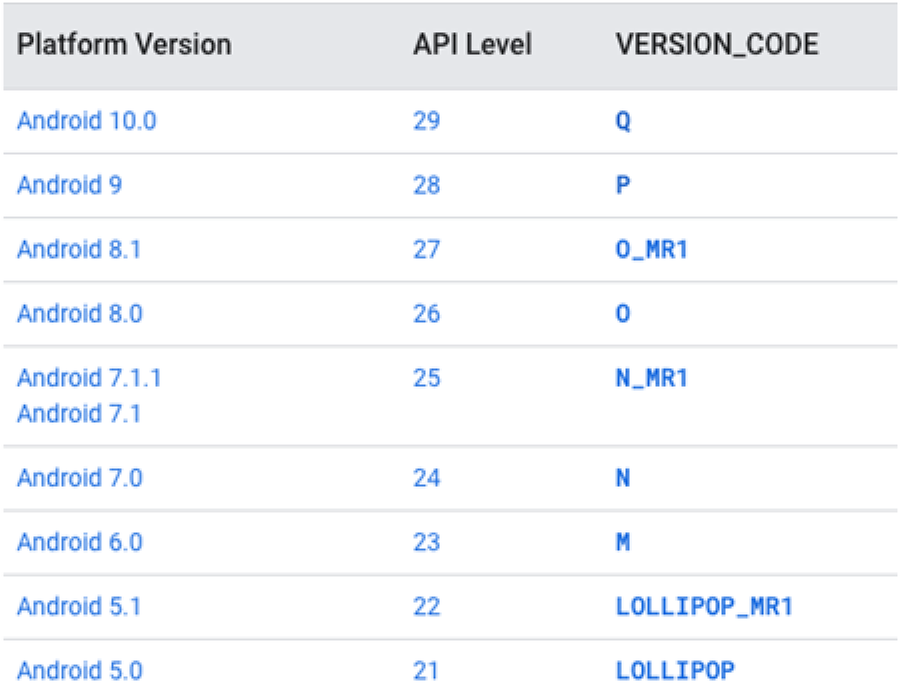
Help me choose

☐ Use legacy android.support libraries

Empty Activity

Creates a new empty activity

(Some) 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

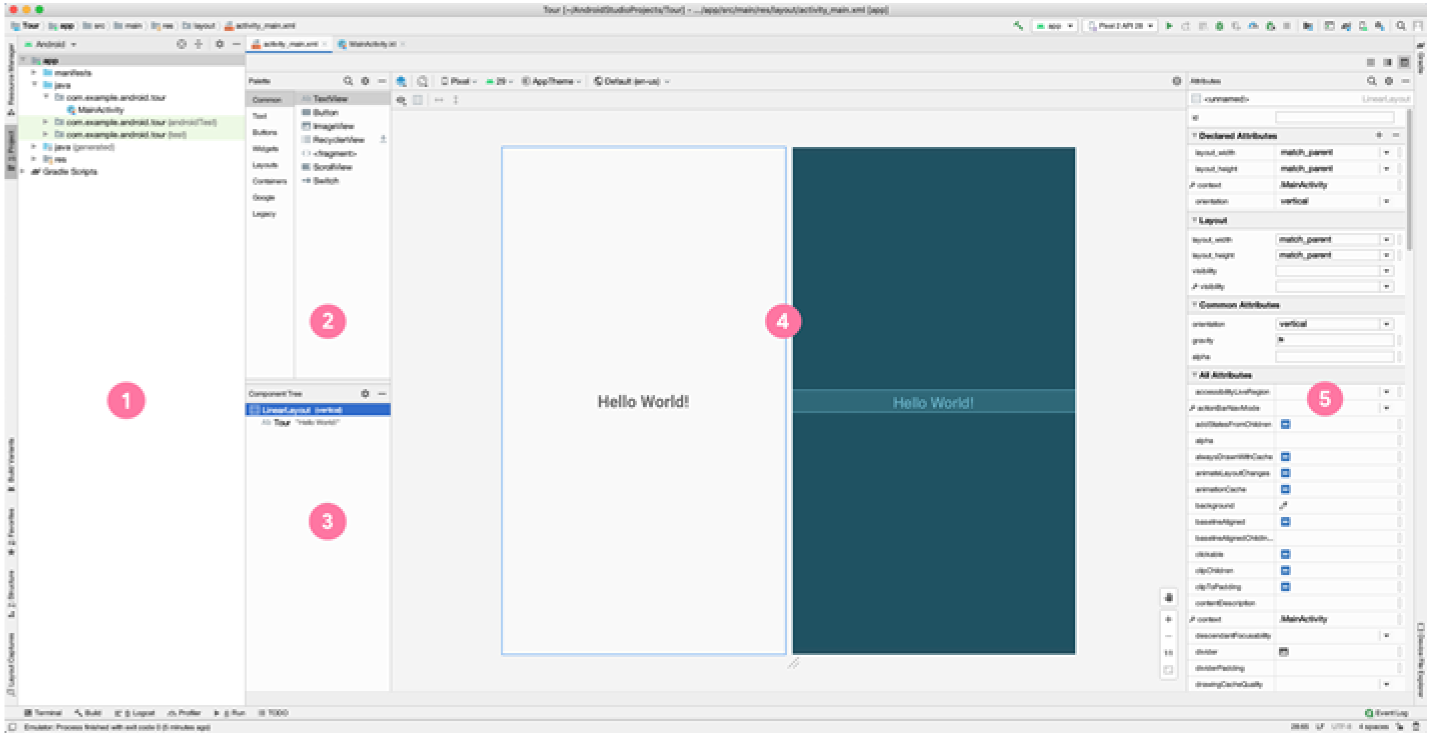
Choose API levels for your app

- 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

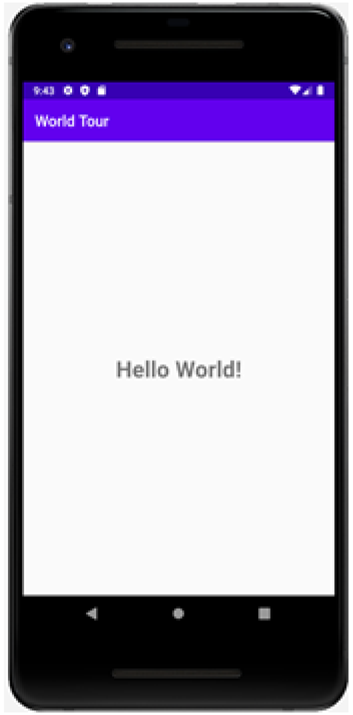
```
minSdkVersion ≤ targetSdkVersion ≤ compileSdkVersion
```

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

Tour of Android Studio

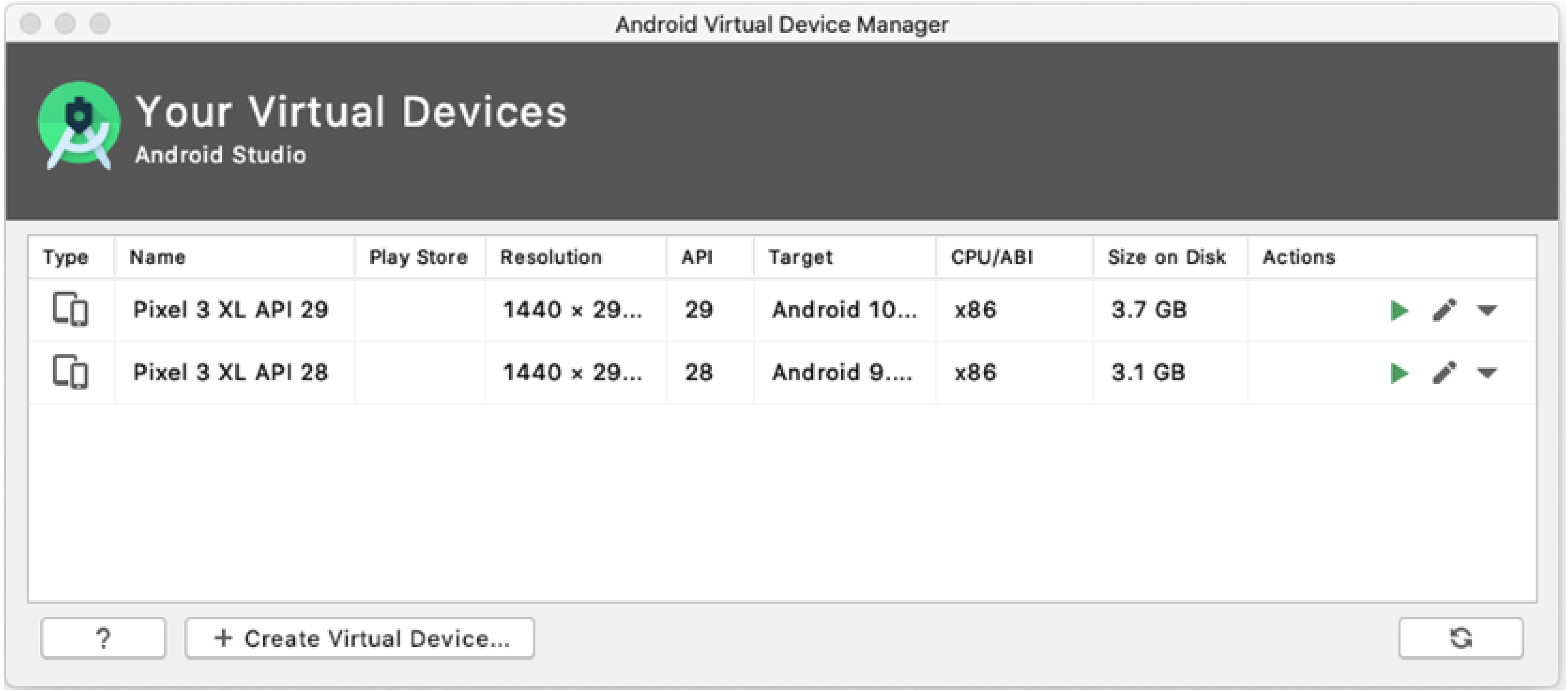


Run your app



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

Android Virtual Device (AVD) Manager



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

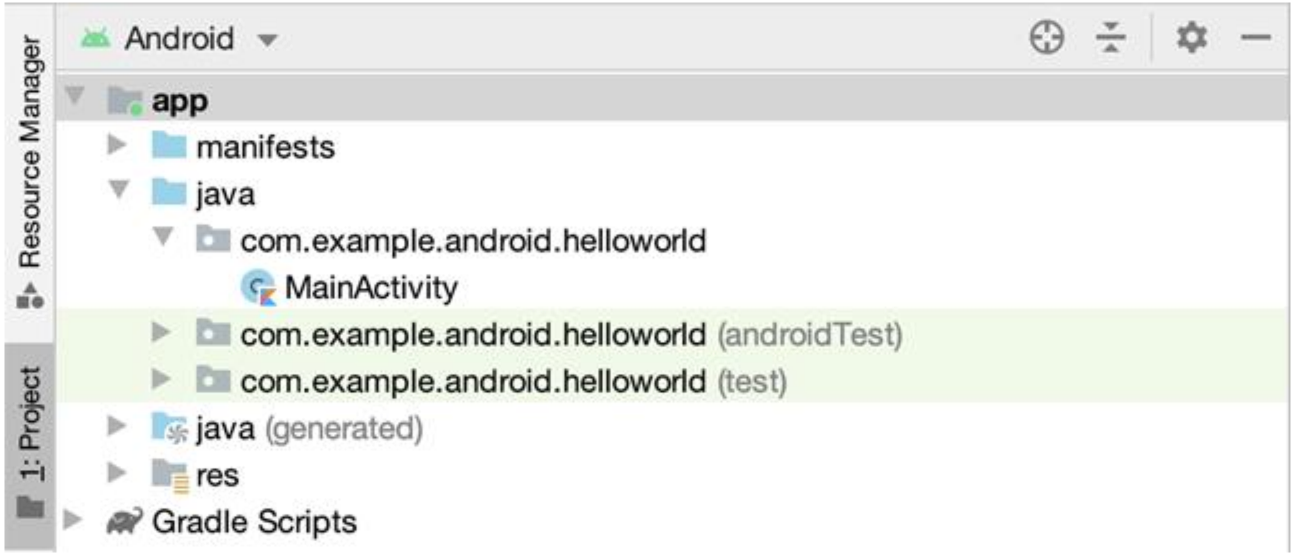
Anatomy of a basic app project

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

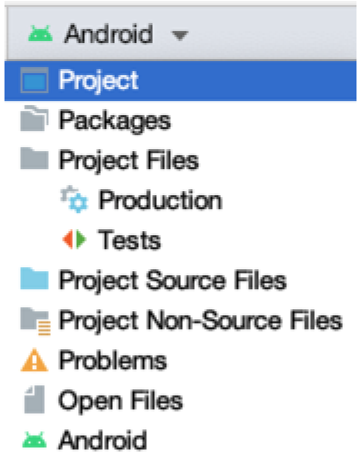
Android app project structure

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

Browse files in Android Studio



Resource Manager
1: Project
Android
app
manifests
java
com.example.android.helloworld
MainActivity
com.example.android.helloworld (androidTest)
com.example.android.helloworld (test)
java (generated)
res
Gradle Scripts



Exercise

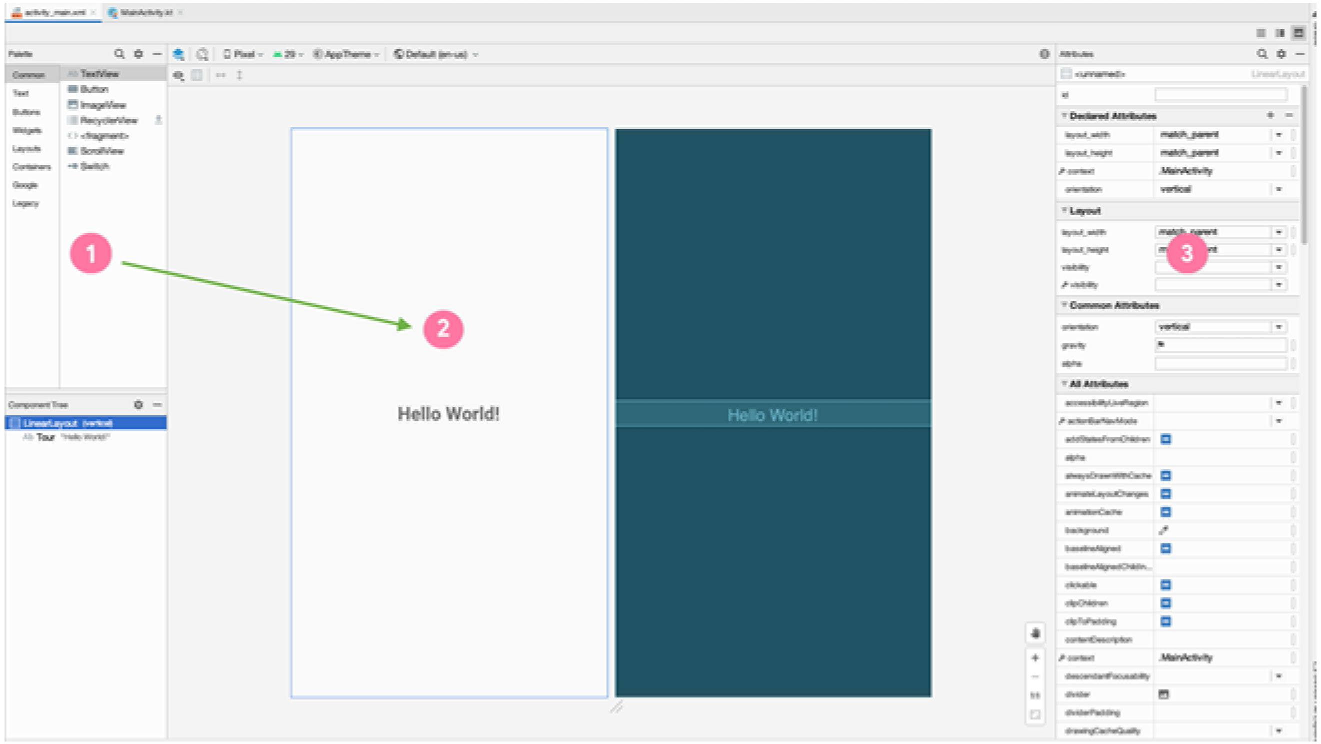
1. Perform the following exercise
    - https://developer.android.com/codelabs/basic-android-kotlin-compose-first-app
    - https://developer.android.com/codelabs/build-your-first-android-app-kotlin
2. Try to run on your Device and Android Virtual Device
3. Post your question/doubt on MS Team

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



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	
baselineAlignedChildIndex	
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!"/>
```

Size of a View

- wrap_content

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

- match_parent

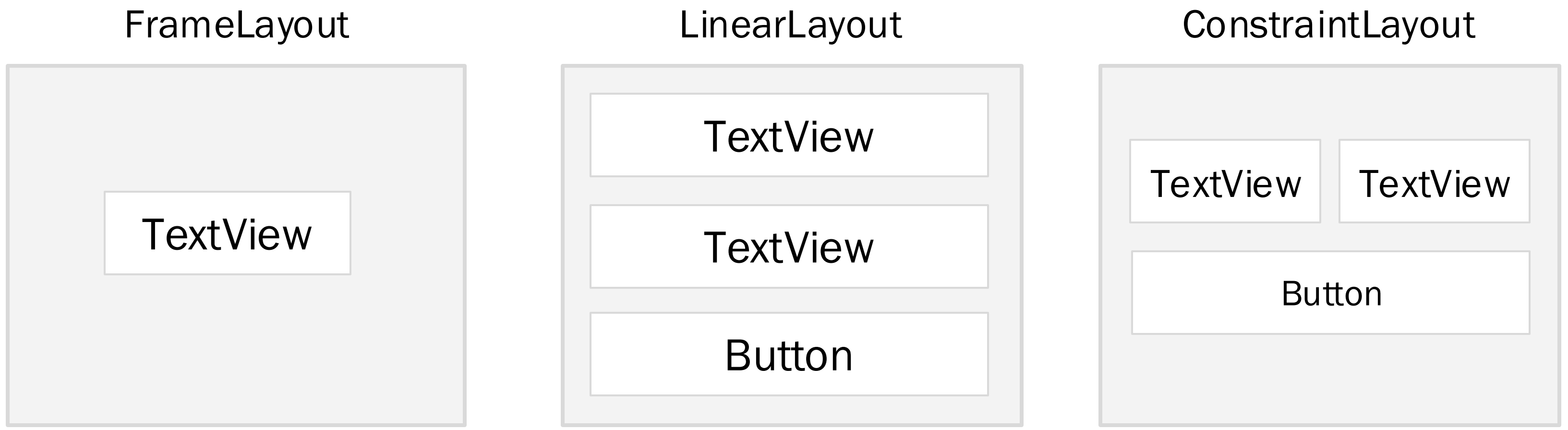
```
android:layout_width="match_parent"
```

- Fixed value (use dp units)

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

ViewGroups

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

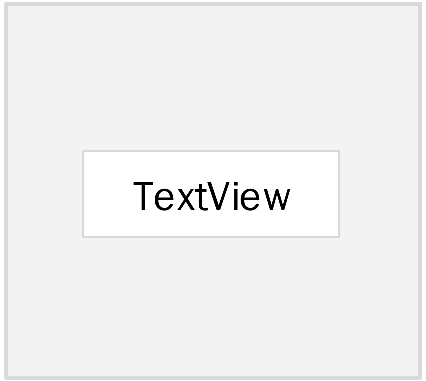


The ViewGroup is the parent and the views inside it are its children.

FrameLayout example

A FrameLayout generally holds a single child View.

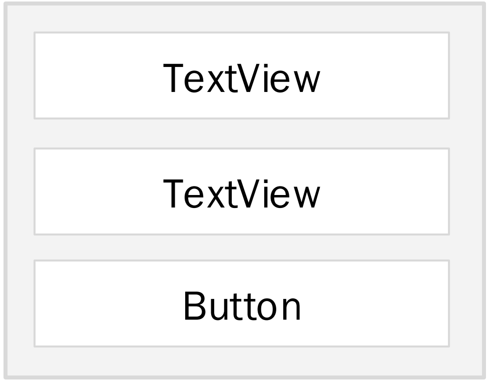
```
<FrameLayout
   android:layout_width="match_parent"
   android:layout_height="match_parent">
   <TextView
       android:layout_width="match_parent"
       android:layout_height="match_parent"
       android:text="Hello World!"/>
</FrameLayout>
```



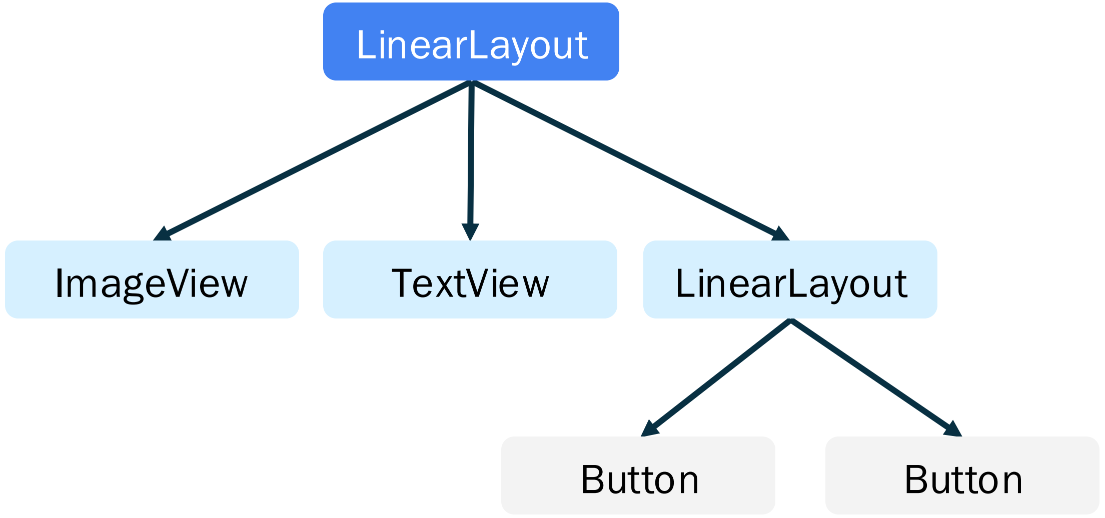
LinearLayout example

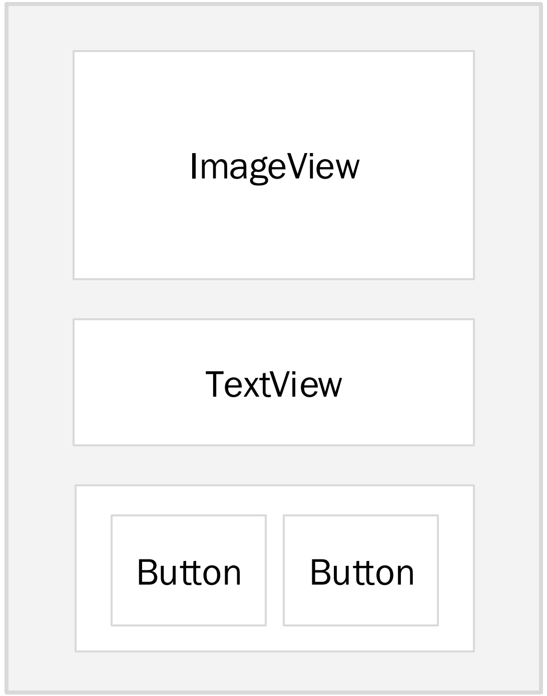
- Aligns child views in a row or column
- Set `android:orientation` to `horizontal` or `vertical`

```
<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.

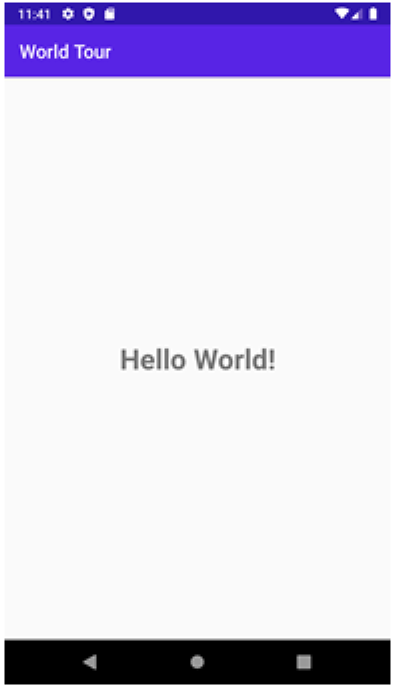
```
<TextView
     android:id="@+id/helloTextView"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:text="Hello World!"/>
```

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

`R.id.helloTextView`

Activities

What’s an Activity?



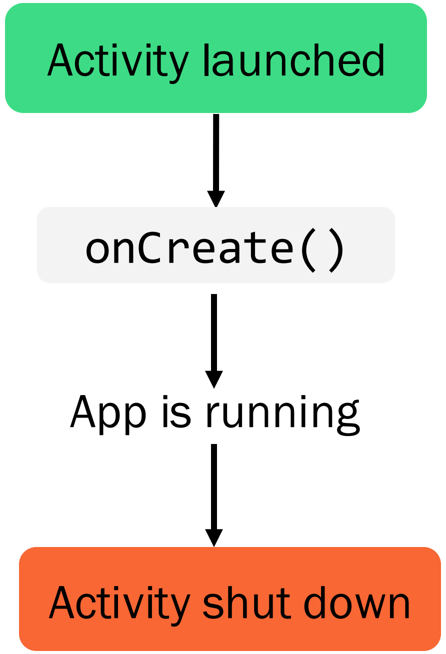
- An Activity is a means for the user to accomplish one main goal.
- An Android app is composed of one or more activities.

MainActivity.kt

```kotlin
class MainActivity : AppCompatActivity() {

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

How an Activity runs

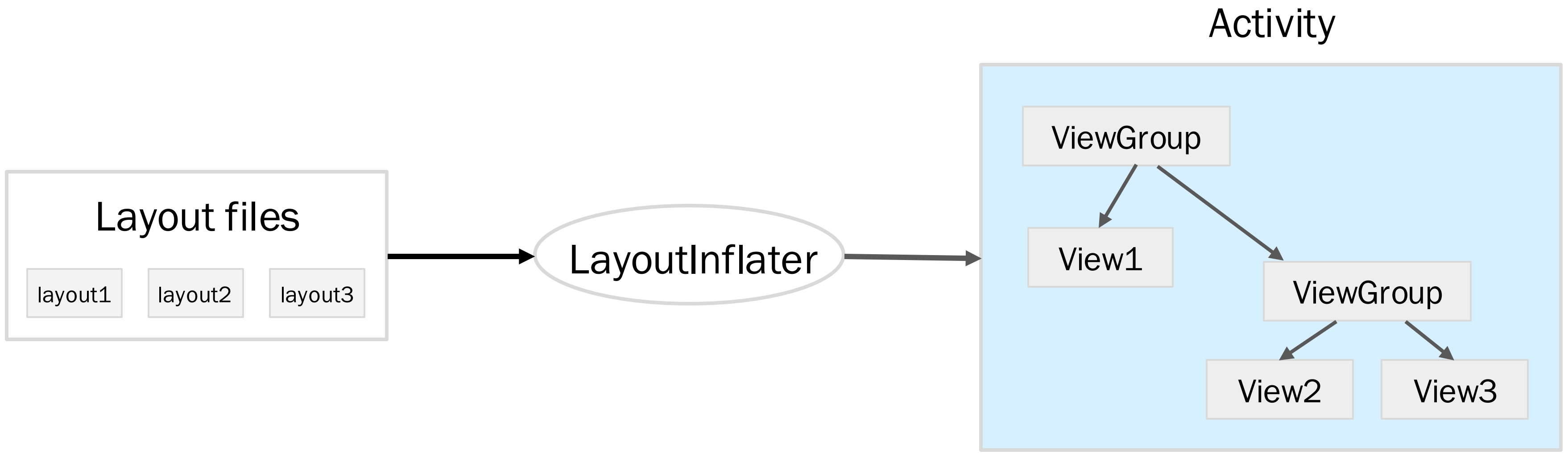


Implement the onCreate() callback

Called when the system creates your Activity

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

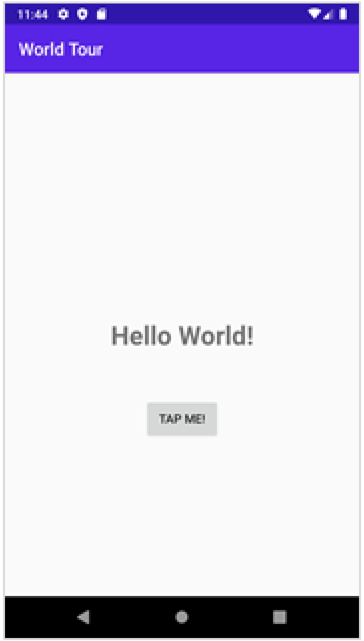
Layout inflation



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`:

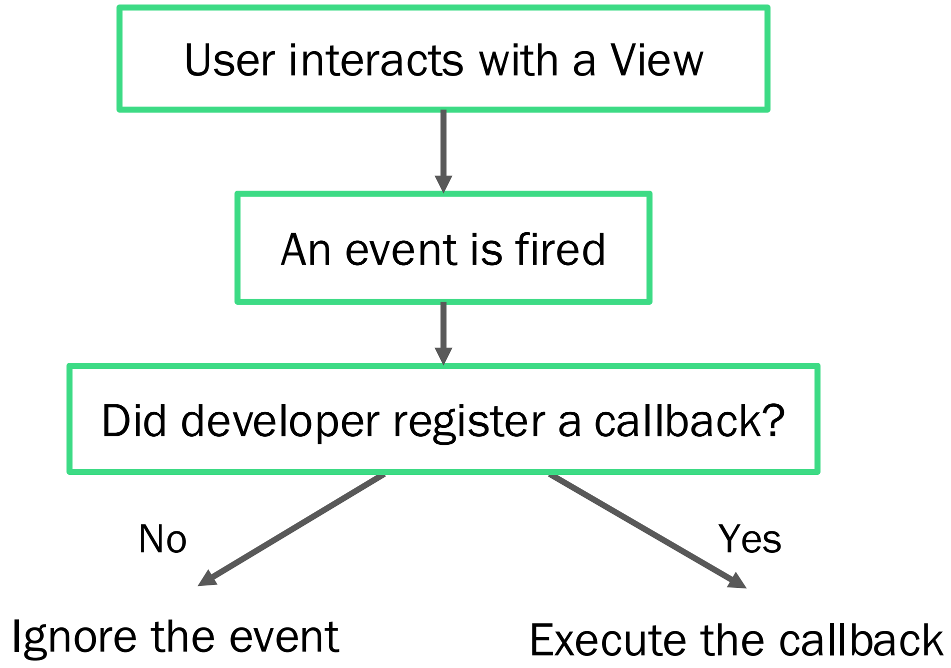
Get a reference to the View in the view hierarchy:

```
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

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

    lateinit var records: HashSet<Any>

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

Lateinit example in Activity

```kotlin
class MainActivity : AppCompatActivity() {

    lateinit var result: TextView

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

Gradle: Building an Android app

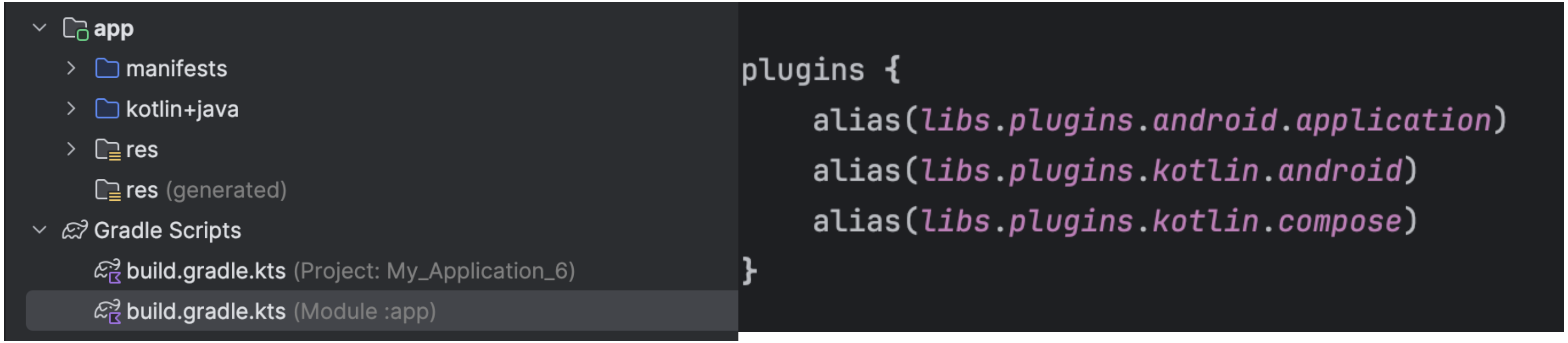
What is Gradle?

- 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



app
manifests
kotlin+java
res
res (generated)
Gradle Scripts
build.gradle.kts (Project: My_Application_6)
build.gradle.kts (Module :app)

```
plugins {
    alias(libs.plugins.android.application)
    alias(libs.plugins.kotlin.android)
    alias(libs.plugins.kotlin.compose)
}
```

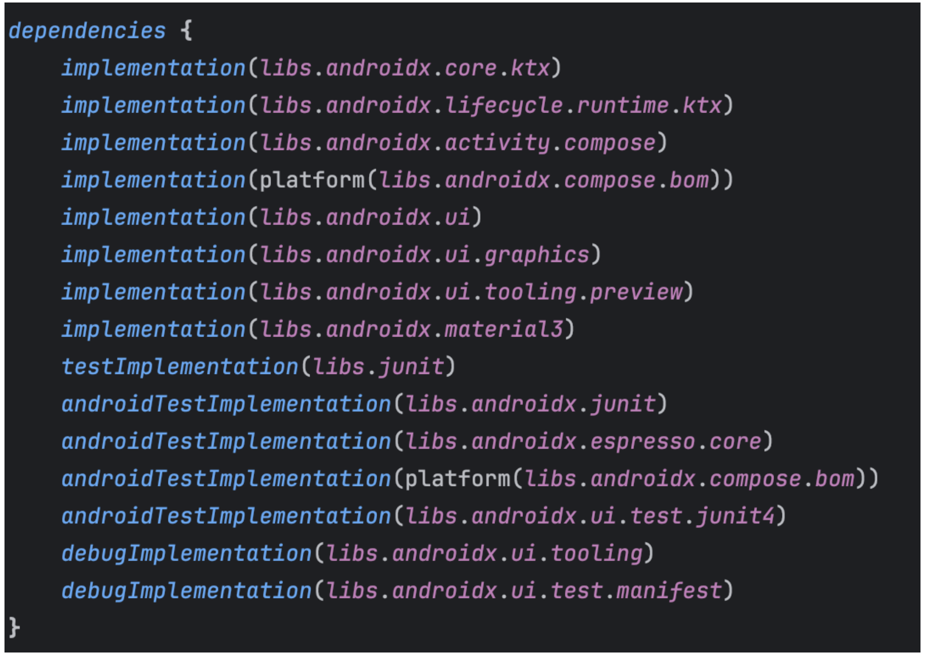
Plugins affect Gradle's behavior but do not add libraries to the app code.

Android configuration

```
android {
    compileSdkVersion 30
    buildToolsVersion "30.0.2"

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

Dependencies



```
dependencies {
    implementation(libs.androidx.core.ktx)
    implementation(libs.androidx.lifecycle.runtime.ktx)
    implementation(libs.androidx.activity.compose)
    implementation(platform(libs.androidx.compose.bom))
    implementation(libs.androidx.ui)
    implementation(libs.androidx.ui.graphics)
    implementation(libs.androidx.ui.tooling.preview)
    implementation(libs.androidx.material3)
    testImplementation(libs.junit)
    androidTestImplementation(libs.androidx.junit)
    androidTestImplementation(libs.androidx.espresso.core)
    androidTestImplementation(platform(libs.androidx.compose.bom))
    androidTestImplementation(libs.androidx.ui.test.junit4)
    debugImplementation(libs.androidx.ui.tooling)
    debugImplementation(libs.androidx.ui.test.manifest)
}
```

Dependencies section specify libraries that is used by the app

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

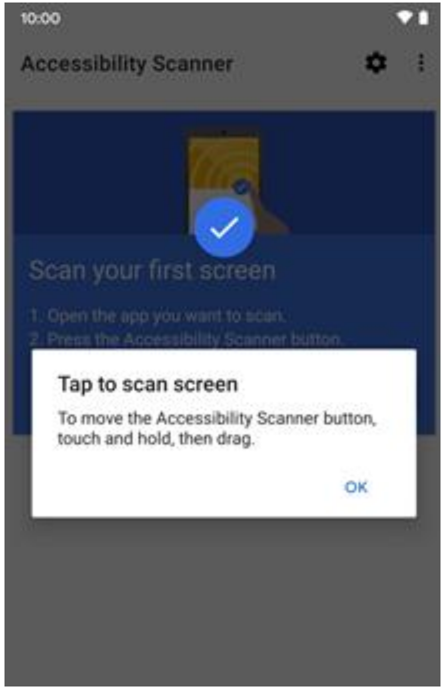
Make apps more accessible

- 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


10:00

Accessibility Scanner

Scan your first screen

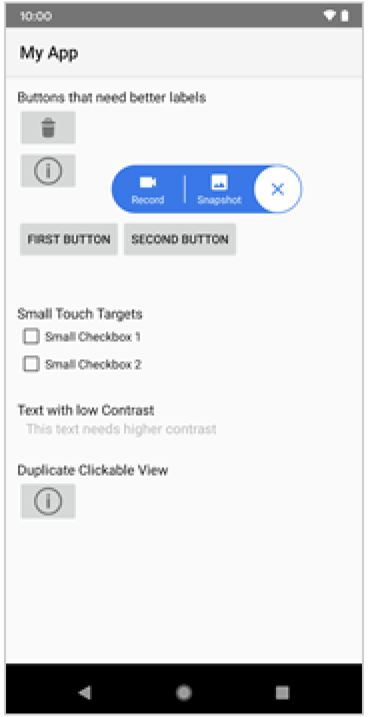
1. Open the app you want to scan.
2. Press the Accessibility Scanner button.

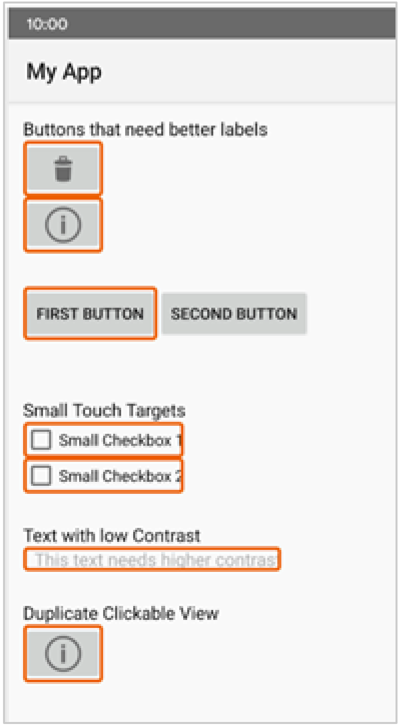
Tap to scan screen

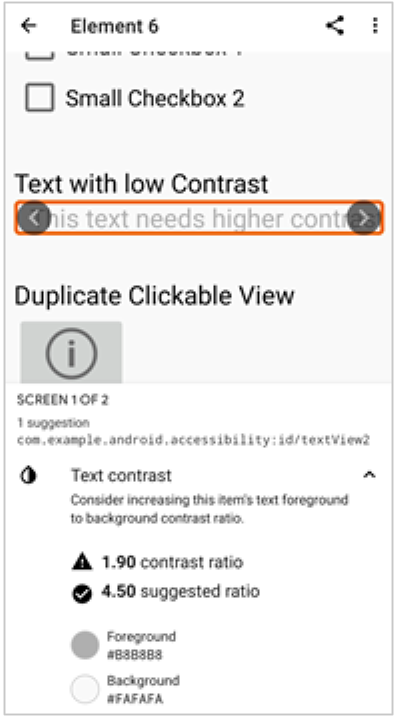
To move the Accessibility Scanner button, touch and hold, then drag.

OK

Accessibility Scanner example







SCREEN 1 OF 2

1 suggestion
com.example.android.accessibility:id/textView2

Text contrast
Consider increasing this item's text foreground to background contrast ratio.

1.90 contrast ratio
4.50 suggested ratio

Foreground
#B8B8B8

Background
#FAFAFA

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

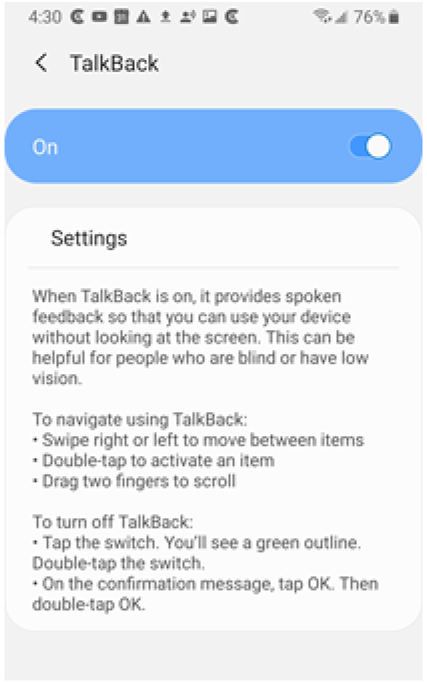
  `android:importantForAccessibility="no"`

- 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



4:30 76%

TalkBack

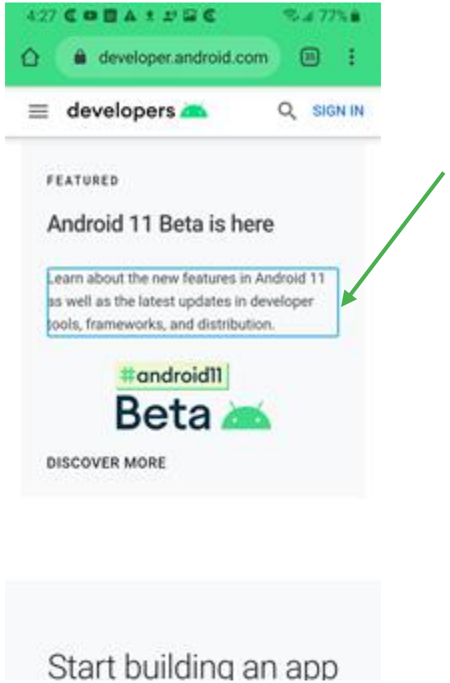
On

Settings

When TalkBack is on, it provides spoken feedback so that you can use your device without looking at the screen. This can be helpful for people who are blind or have low vision.

To navigate using TalkBack:
• Swipe right or left to move between items
• Double-tap to activate an item
• Drag two fingers to scroll

To turn off TalkBack:
• Tap the switch. You'll see a green outline. Double-tap the switch.
• On the confirmation message, tap OK. Then double-tap OK.



4:27 77%

developer.android.com

developers SIGN IN

FEATURED

Android 11 Beta is here

Learn about the new features in Android 11 as well as the latest updates in developer tools, frameworks, and distribution.

#android11 Beta

DISCOVER MORE

Start building an app

Reads text aloud as user navigates the screen

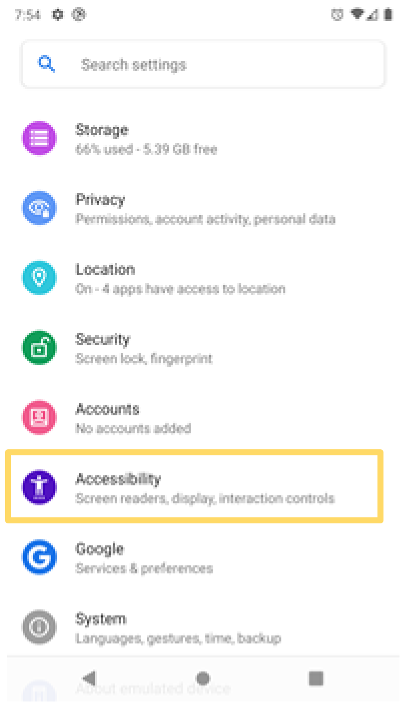
Switch access

- 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 this lesson, 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