Introduction to Android

What is Android?

- Android is an operating system and programming platform developed by Google for mobile phones and other mobile devices, such as tablets
- It can run on many different devices from many different manufacturers
- Android includes a software development kit (SDK) that helps you write original code and assemble software modules to create apps for Android users
- Android also provides a marketplace to distribute apps
- All together, Android represents an *ecosystem* for mobile apps

Why develop apps for Android?

- Developers create apps for a variety of reasons. They may need to address business requirements or build new services or businesses, or they may want to offer games and other types of content for users
- Developers choose to develop for Android in order to reach the majority of mobile device users

Most popular platform for mobile apps

- world's most popular mobile platform. It has the largest installed base of any mobile platform and is still growing fast

Best experience for app users

- Android's UI is mainly based on direct manipulation.
- People use touch gestures such as swiping, tapping, and pinching to manipulate on-screen objects.
- In addition to the keyboard, there's a customizable on-screen keyboard for text input.
- Android can also support game controllers and full-size physical keyboards connected by Bluetooth or USB

- The Android home screen can contain several panes of *app icons*, which launch their associated apps.
- Home screen panes can also contain *app widgets*, which display live, auto-updating content such as the weather, the user's email inbox
- Android is designed to provide immediate response to user input
- Many apps take advantage of internal hardware such as accelerometers, gyroscopes, and proximity sensors to respond to additional user actions
- sensors can also detect screen rotation

It's easy to develop apps

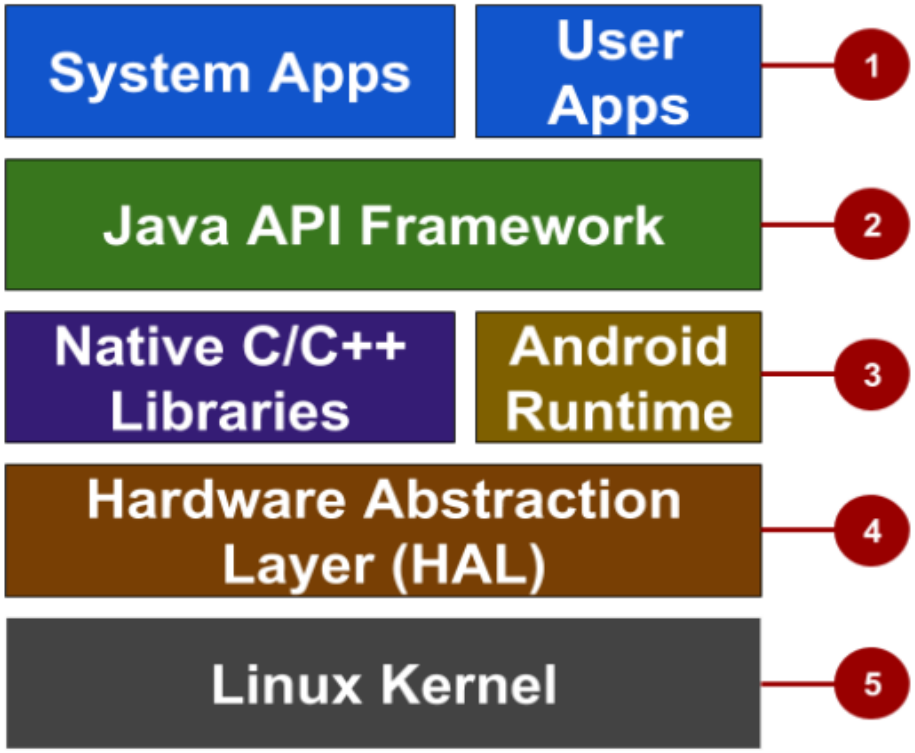
- To develop apps that take advantage of the Android operating system and UI, use the Android software development kit (SDK)
- The SDK includes software libraries of prewritten code, a debugger, a device emulator, documentation, sample code, and tutorials.
- Use the SDK to create apps that look great and take advantage of the hardware capabilities available on each Android-powered device.

- To develop apps using the SDK, you use the Java programming language to develop the app and Extensible Markup Language (XML) files to describe data resources
- By writing the code in Java and creating a single app binary, you create an app that can run on both phone and tablet form factors
- You can declare your UI in lightweight sets of XML resources. For example, create one set for parts of the UI that are common to all form factors, and other sets for features specific to phones or tablets
- At runtime, Android applies the correct resource sets based on the device's screen size, screen density, locale, and so on

- To help you develop your apps efficiently, Google offers an integrated development environment (IDE) called Android Studio
- It offers advanced features for developing, debugging, and packaging Android apps.
- Using Android Studio, you can develop for any Android-powered device, or create virtual devices that emulate any hardware configuration.

Architecture:

- Android provides a rich development architecture



Apps:

User's apps live at this level, along with core system apps for email, SMS messaging, calendars, internet browsing, and contacts

Java API framework:

All features for Android development, such as UI components, resource management, and lifecycle management, are available through application programming interfaces (APIs). You don't need to know the details of how the APIs work. You only need to learn how to use them.

Libraries and Android runtime:

- Each app runs in its own process, with its own instance of the Android runtime.
- Android includes a set of core runtime libraries that provide most of the functionality of the Java programming language.
- Many core Android system components and services are built from native code that require native libraries written in C and C++.
- These native libraries are available to apps through the Java API framework

Hardware abstraction layer (HAL):

- This layer provides standard interfaces that expose device hardware capabilities to the higher-level Java API framework
- The HAL consists of multiple library modules, each of which implements an interface for a specific type of hardware component, such as the camera or Bluetooth module

Linux kernel:

- The foundation of the Android platform is the Linux kernel.
- The layers above the Linux kernel rely on the Linux kernel for threading, low-level memory management, and other underlying functionality.
- Using a Linux kernel enables Android to take advantage of Linux-based security features and allows device manufacturers to develop hardware drivers for a well-known kernel.

Many distribution options

- You can distribute your Android app in many different ways: email, website, or an app marketplace Google Play
- Android users download billions of apps and games from the Google Play store each month.
- Google Play is a digital distribution service, operated and developed by Google, that serves as the official app store for Android.
- Google Play lets consumers to browse and download apps developed with the Android SDK.

The challenges of Android app development

Building for a multi-screen world

- Android runs on billions of handheld devices around the world and supports various form factors including wearable devices and televisions.
- Devices come in different sizes and shapes, which affects how you design the screens and UI elements in your apps.
- In addition, device manufacturers may add their own UI elements, styles, and colors to differentiate their products. Each manufacturer offers different features with respect to keyboard forms, screen size, or camera buttons.
- An app running on one device may look a bit different on another. Your challenge, as a developer, is to design UI elements that work on all devices.

Maximizing app performance

- An app's *performance* is determined by how fast it runs, how easily it connects to the network, and how well it manages battery and memory usage.
- Performance is affected by factors such as battery life, multimedia content, and internet access.
- Be aware that some features you design for your app may cause performance problems for users. For example, to save the user's battery power, enable background services only when they are necessary.

Keeping code and users more secure

- You need to take precautions to make your code, and the user's experience when they use your app, as secure as possible
- Use tools such as ProGuard, which is provided in Android Studio. ProGuard detects and removes unused classes, fields, methods, and attributes
- Encrypt all of your app's code and resources while packaging the app
- To protect critical user information such as logins and passwords, secure your communication channel to protect data in transit across the internet, as well as data at rest on the device

Remaining compatible with older versions of Android

- The Android platform continues to improve and provide new features you can add to your apps
- However, you should ensure that your app can still run on devices with older versions of Android
- It is impractical to focus only on the most recent Android version, as not all users may have upgraded or may be able to upgrade their devices
- Fortunately, Android Studio provides options for developers to more easily remain compatible with older versions

Two fundamental concepts about Android apps:

Apps provide multiple entry points

- Android apps are built as a combination of components that can be invoked individually. For example, an activity is a type of app component that provides a user interface (UI)
- The "main" activity starts when the user taps your app's icon. You can also direct the user to an activity from elsewhere, such as from a notification or even from a different app
- Other components, such as WorkManager, allow your app to perform background tasks without a UI

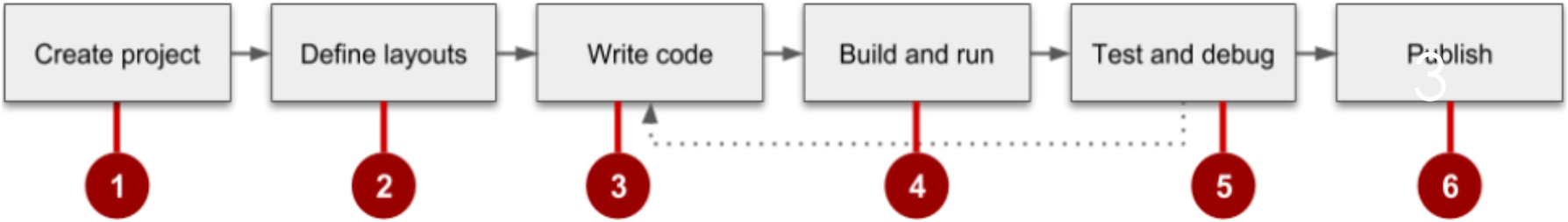
Apps adapt to different devices

- Android allows you to provide different resources for different devices. For example, you can create different layouts for different screen sizes. The system determines which layout to use based on the screen size of the current device.
- If any of your app's features need specific hardware, such as a camera, you can query at runtime whether the device has that hardware or not, and then disable the corresponding features if it doesn't. You can specify that your app requires certain hardware so that Google Play won't allow the app to be installed on devices without them.

First Android app

The development process

- An Android app project begins with an idea and a definition of the requirements necessary to realize that idea
- You may want to sketch user interfaces (UIs) for the various app functions
- To show what a UI would look like and how it would work, use drawings, mockups, and prototypes.



1. Create the project in Android Studio and choose an appropriate template
2. Define a layout for each screen that has UI elements. You can place UI elements on the screen using the layout editor, or you can write code directly in the Extensible Markup Language (XML)
3. Write code using the Java programming language. Create source code for all of the app's components
4. Build and run the app on real and virtual devices. Use the default build configuration or create custom builds for different versions of your app
5. Test and debug the app's logic and UI
6. Publish the app by assembling the final APK (package file) and distributing it through channels such as Google Play

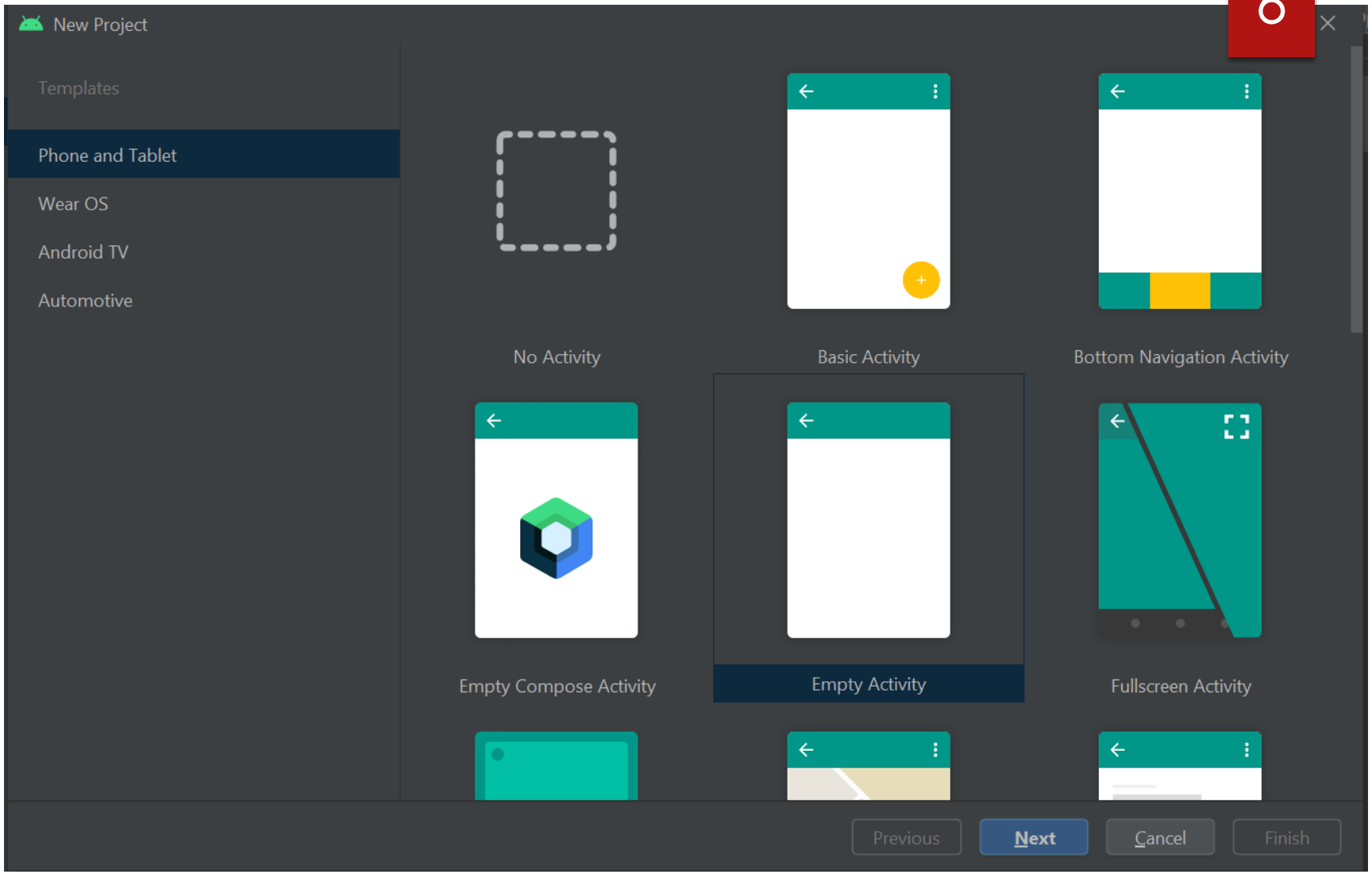
Using Android Studio

- Android Studio provides a unified development environment for creating apps for all Android-powered devices
- Android Studio includes code templates with sample code for common app features, extensive testing tools and frameworks, and a flexible build system.
- Latest version: Android Studio Arctic Fox (2020.3.1)

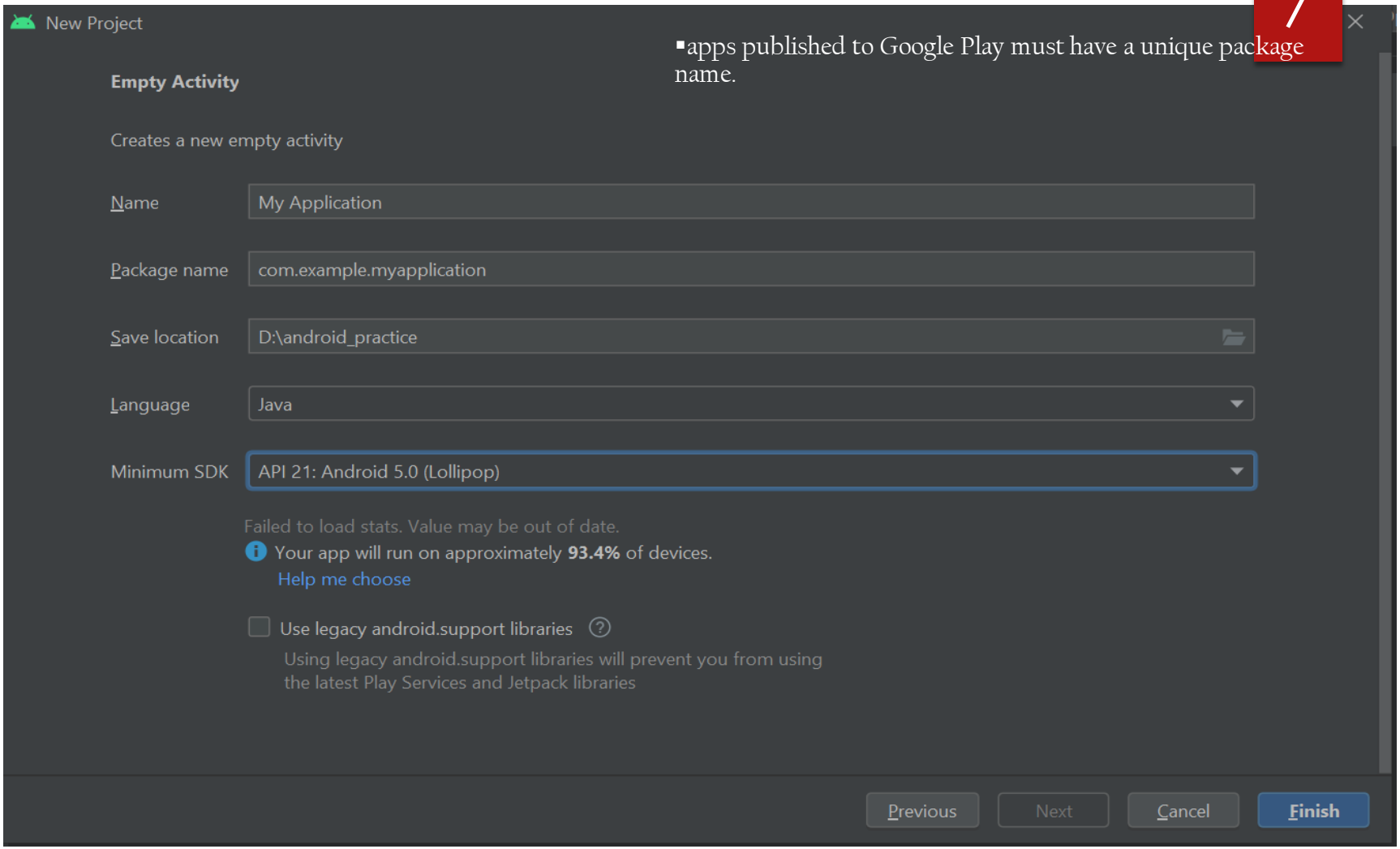
Create an Android project

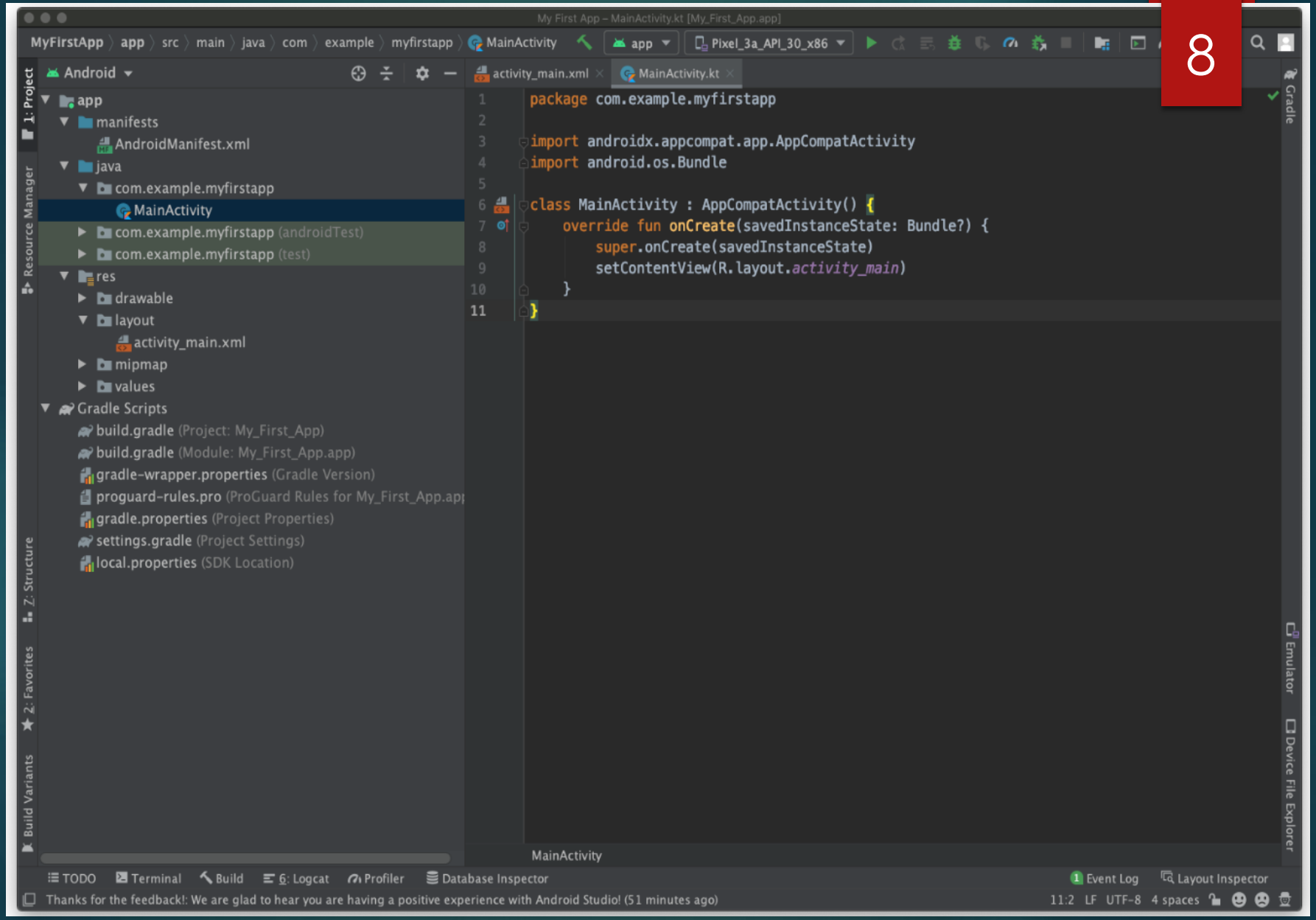
1. Install the latest version of Android Studio.
2. In the **Welcome to Android Studio** window, click **Create New Project**.

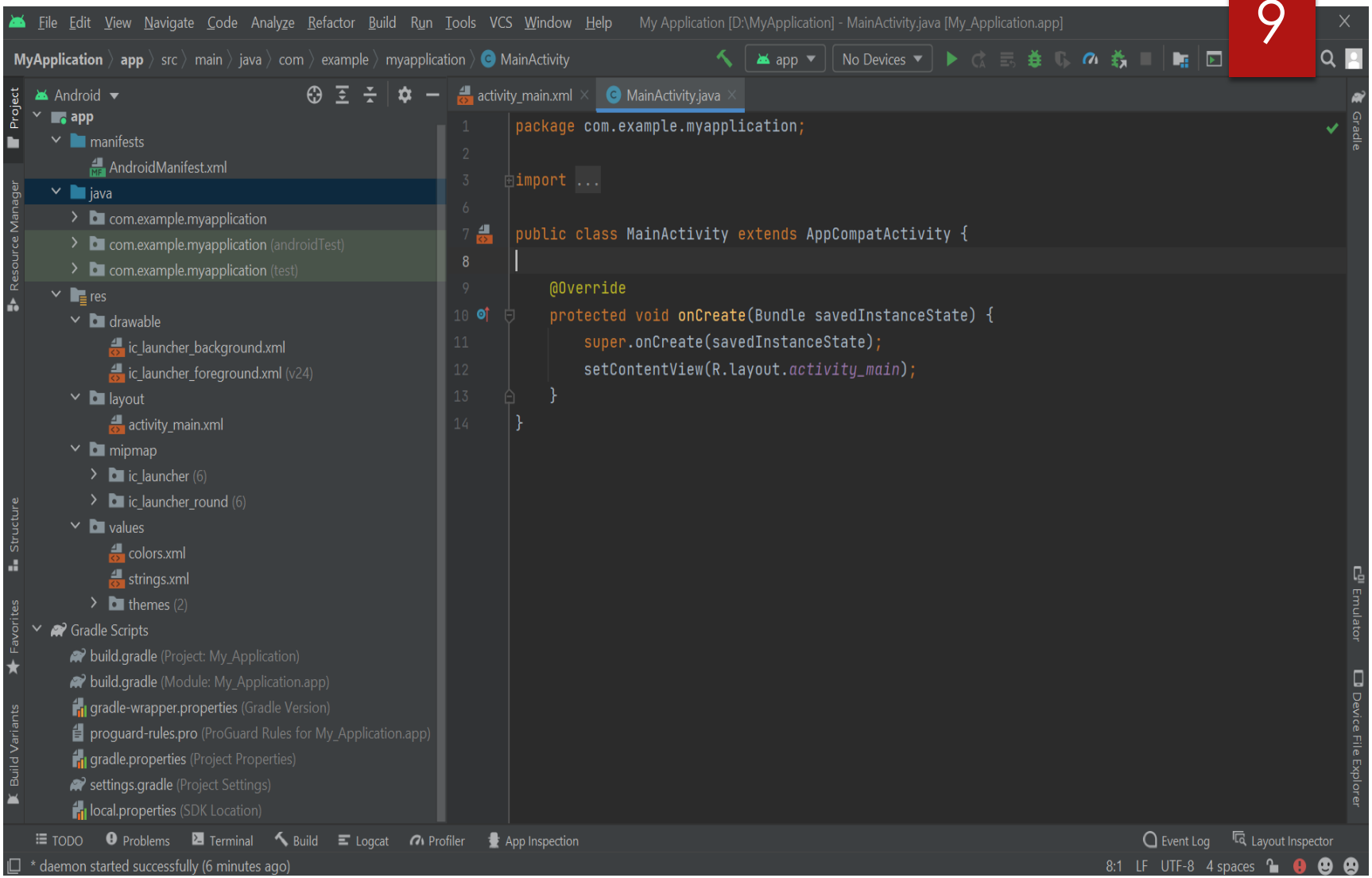
   If you have a project already opened, select **File** › **New** › **New Project**.
3. In the **Select a Project Template** window, select **Empty Activity** and click **Next**.
4. In the **Configure your project** window, complete the following:
   - Enter "My First App" in the **Name** field.
   - Enter "com.example.myfirstapp" in the **Package name** field.
   - If you'd like to place the project in a different folder, change its **Save** location.
   - Select either **Java** or **Kotlin** from the **Language** drop-down menu.
   - Select the lowest version of Android you want your app to support in the **Minimum SDK** field.
5. If your app will require legacy library support, mark the **Use legacy android.support libraries** checkbox.
   - Leave the other options as they are.
6. Click **Finish**.

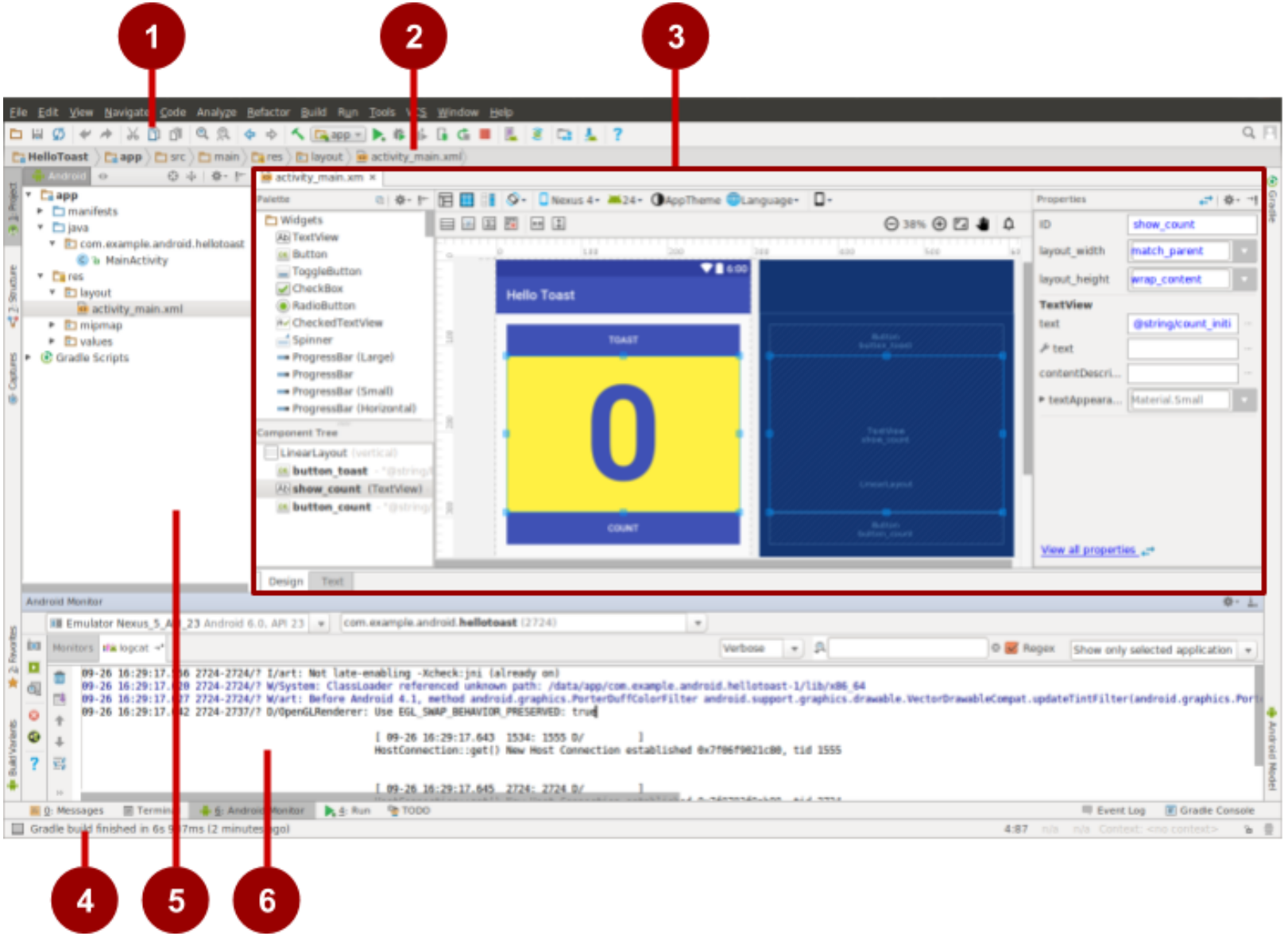

New Project
Templates
Phone and Tablet
Wear OS
Android TV
Automotive
No Activity
Basic Activity
Bottom Navigation Activity
Empty Compose Activity
Empty Activity
Fullscreen Activity
Previous
Next
Cancel
Finish

- apps published to Google Play must have a unique package name.




My First App – MainActivity.kt [My_First_App.app]
MyFirstApp 〉 app 〉 src 〉 main 〉 java 〉 com 〉 example 〉 myfirstapp 〉 MainActivity
app
Pixel_3a_API_30_x86
Android
app
manifests
AndroidManifest.xml
java
com.example.myfirstapp
MainActivity
com.example.myfirstapp (androidTest)
com.example.myfirstapp (test)
res
drawable
layout
activity_main.xml
mipmap
values
Gradle Scripts
build.gradle (Project: My_First_App)
build.gradle (Module: My_First_App.app)
gradle-wrapper.properties (Gradle Version)
proguard-rules.pro (ProGuard Rules for My_First_App.ap
gradle.properties (Project Properties)
settings.gradle (Project Settings)
local.properties (SDK Location)
activity_main.xml
MainActivity.kt
1 package com.example.myfirstapp
2
3 import androidx.appcompat.app.AppCompatActivity
4 import android.os.Bundle
5
6 class MainActivity : AppCompatActivity() {
7 override fun onCreate(savedInstanceState: Bundle?) {
8 super.onCreate(savedInstanceState)
9 setContentView(R.layout.activity_main)
10 }
11 }
MainActivity
TODO
Terminal
Build
6: Logcat
Profiler
Database Inspector
Event Log
Layout Inspector
Thanks for the feedback!: We are glad to hear you are having a positive experience with Android Studio! (51 minutes ago)
11:2 LF UTF-8 4 spaces
1: Project
Resource Manager
Z: Structure
2: Favorites
Build Variants
Gradle
Emulator
Device File Explorer

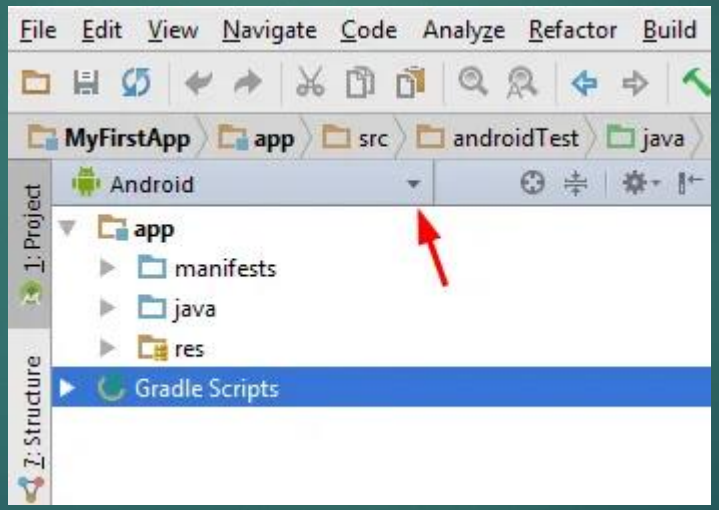

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
My Application [D:\MyApplication] - MainActivity.java [My_Application.app]
MyApplication app src main java com example myapplication MainActivity
app
No Devices
Android
app
manifests
AndroidManifest.xml
java
com.example.myapplication
com.example.myapplication (androidTest)
com.example.myapplication (test)
res
drawable
ic_launcher_background.xml
ic_launcher_foreground.xml (v24)
layout
activity_main.xml
mipmap
ic_launcher (6)
ic_launcher_round (6)
values
colors.xml
strings.xml
themes (2)
Gradle Scripts
build.gradle (Project: My_Application)
build.gradle (Module: My_Application.app)
gradle-wrapper.properties (Gradle Version)
proguard-rules.pro (ProGuard Rules for My_Application.app)
gradle.properties (Project Properties)
settings.gradle (Project Settings)
local.properties (SDK Location)
activity_main.xml
MainActivity.java
package com.example.myapplication;
import ...
public class MainActivity extends AppCompatActivity {
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
}
}
Project
Resource Manager
Structure
Favorites
Build Variants
Gradle
Emulator
Device File Explorer
TODO
Problems
Terminal
Build
Logcat
Profiler
App Inspection
Event Log
Layout Inspector
* daemon started successfully (6 minutes ago)
8:1 LF UTF-8 4 spaces

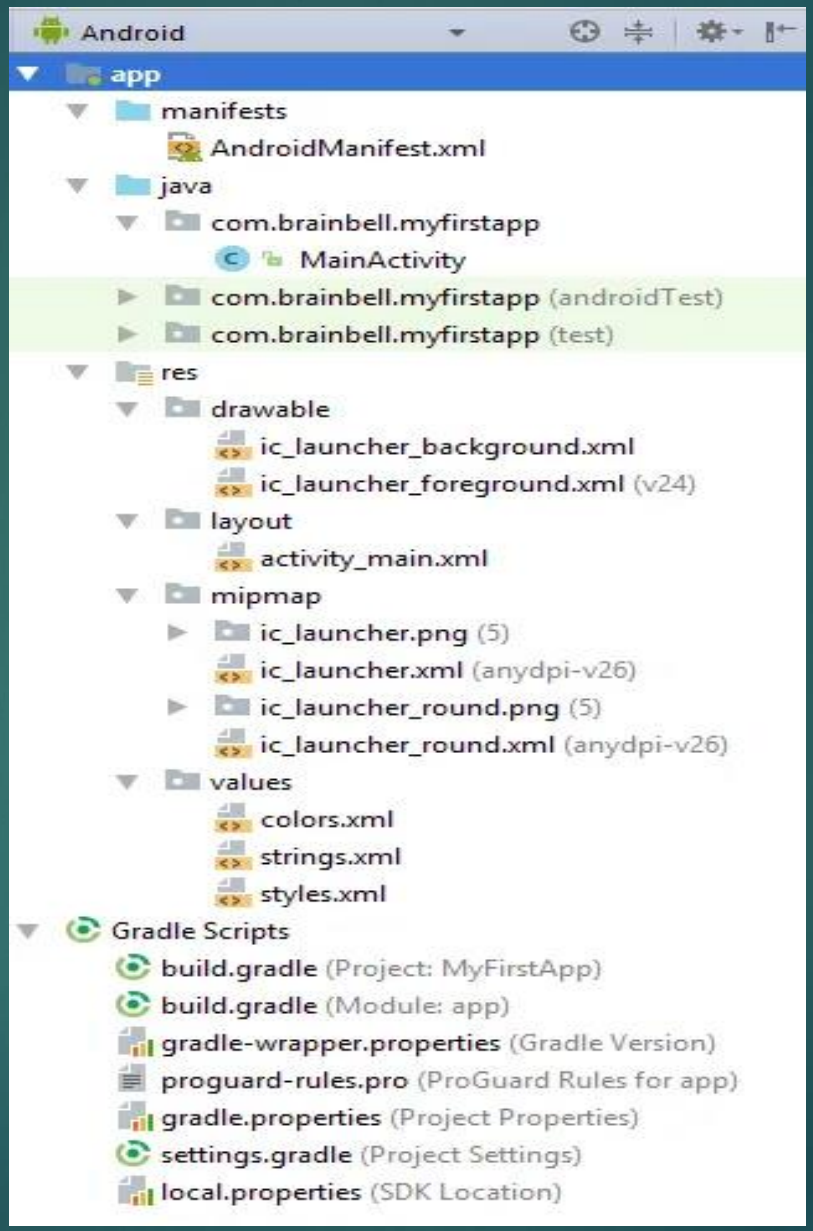

1
2
3
4
5
6
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
HelloToast app src main res layout activity_main.xml
Android
app
manifests
java
com.example.android.hellotoast
MainActivity
res
layout
activity_main.xml
mipmap
values
Gradle Scripts
activity_main.xml
Palette
Widgets
TextView
Button
ToggleButton
CheckBox
RadioButton
CheckedTextView
Spinner
ProgressBar (Large)
ProgressBar
ProgressBar (Small)
ProgressBar (Horizontal)
Component Tree
LinearLayout (vertical)
button_toast
show_count (TextView)
button_count
Nexus 4 24 AppTheme Language
Hello Toast
TOAST
0
COUNT
Properties
ID show_count
layout_width match_parent
layout_height wrap_content
TextView
text @string/count_initi
text
contentDescri...
textAppeara... Material.Small
View all properties
Design Text
Android Monitor
Emulator Nexus_5_API_23 Android 6.0, API 23
com.example.android.hellotoast (2724)
Monitors logcat
Verbose
Regex
Show only selected application
09-26 16:29:17.736 2724-2724/? I/art: Not late-enabling -Xcheck:jni (already on)
09-26 16:29:17.810 2724-2724/? W/System: ClassLoader referenced unknown path: /data/app/com.example.android.hellotoast-1/lib/x86_64
09-26 16:29:17.817 2724-2724/? W/art: Before Android 4.1, method android.graphics.PorterDuffColorFilter android.support.graphics.drawable.VectorDrawableCompat.updateTintFilter(android.graphics.Port
09-26 16:29:17.842 2724-2737/? D/OpenGLRenderer: Use EGL_SWAP_BEHAVIOR_PRESERVED: true
[09-26 16:29:17.643 1534: 1555 D/]
HostConnection::get() New Host Connection established 0x7f06f9021c80, tid 1555
[09-26 16:29:17.645 2724: 2724 D/]
0: Messages Terminal 6: Android Monitor 4: Run TODO
Event Log Gradle Console
Gradle build finished in 6s 507ms (2 minutes ago)
4:87 n/a n/a Context: <no context>

1. Toolbar
2. Navigation Bar
3. Editor Pane
4. Status Bar
5. Project Pane
6. Monitor Pane

Using the Project pane

- The Project tool window provides a simple tree interface with files and nested folders
- When you create an application in Android Studio, you find that the project is divided into an App folder and Gradle scripts




Android
app
manifests
AndroidManifest.xml
java
com.brainbell.myfirstapp
MainActivity
com.brainbell.myfirstapp (androidTest)
com.brainbell.myfirstapp (test)
res
drawable
ic_launcher_background.xml
ic_launcher_foreground.xml (v24)
layout
activity_main.xml
mipmap
ic_launcher.png (5)
ic_launcher.xml (anydpi-v26)
ic_launcher_round.png (5)
ic_launcher_round.xml (anydpi-v26)
values
colors.xml
strings.xml
styles.xml
Gradle Scripts
build.gradle (Project: MyFirstApp)
build.gradle (Module: app)
gradle-wrapper.properties (Gradle Version)
proguard-rules.pro (ProGuard Rules for app)
gradle.properties (Project Properties)
settings.gradle (Project Settings)
local.properties (SDK Location)

Manifests Folder

- AndroidManifest.xml file is generated inside manifest folder by Android Studio when you create a project
- it defines the structure and metadata of our application, its components, and its requirements
- This file contains the configuration parameters of the project such as permissions, services and additional libraries
- A manifest file also provides information to the OS and Google Play store about your app
- It acts as an intermediator between android OS and our application.

Java folder

- All of the source code files written in Java programming language are maintained here
- A MainActivity.java is automatically created in this folder by Android Studio

Resource (res) folder

- It contains all the non-code sources like images, XML layouts, UI strings for our android application.
- Each file type should be added to its own folder to comply with the Android development standards.
- **res/drawable folder**
- It contains the different type of images used for the development of the application.
- We need to add all the images in drawable folder for the application development

res/layout folder

- Layout folder contains all XML layout files which we used to define the user Interface of our application.
- These file are used to set up the layout for your Activity and is used for basic alignment of your layouts, components, widgets, and similar resources that are used for the UI of your application
- It contains the activity_main.xml file.
- Layout folder may have multiple layout folders to handle different devices. This can be helpful when working with layouts that need to be adjusted for devices with more or less screen space available.

res/midmap folder

- This folder contains launcher.xml files to define icons which are used to show on the home screen.
- It contains different density type of icons depends upon the size of the device such as hdpi, mdpi, xhdpi.

- Apk files are built using the gradle build system, which is integrated in the Android Studio
- When we start an app, the gradle build scripts are automatically created
- If you have special requirements for your project, you can specify these requirements here
- The gradle scripts folder contains the scripts used to build the app are: configuration files, properties files, and setting files

res/values folder

- The values folder contains XML files that contain simple values, such as strings, integers, and colors.
- The values folder is used to keep track of the values you will be using in your application.
- To create applications with an easier maintenance cycle, it is highly recommended to no longer hard-code values into your code.
- Instead, place values in XML files inside of the values folder.
- When you create a new project with Android Studio, the following XML files will be generated automatically:

  colors.xml

  strings.xml

  styles.xml

Important files:

app › java › com.example.myfirstapp › MainActivity

- This is the main activity.
- It's the entry point for your app.
- When you build and run your app, the system launches an instance of this Activity and loads its layout.

app › res › layout › activity_main.xml

- This XML file defines the layout for the activity's user interface (UI)
- It contains a TextView element with the text "Hello, World!"

Important files:

app › manifests › AndroidManifest.xml

- The manifest file describes the fundamental characteristics of the app and defines each of its components.

Gradle Scripts › build.gradle

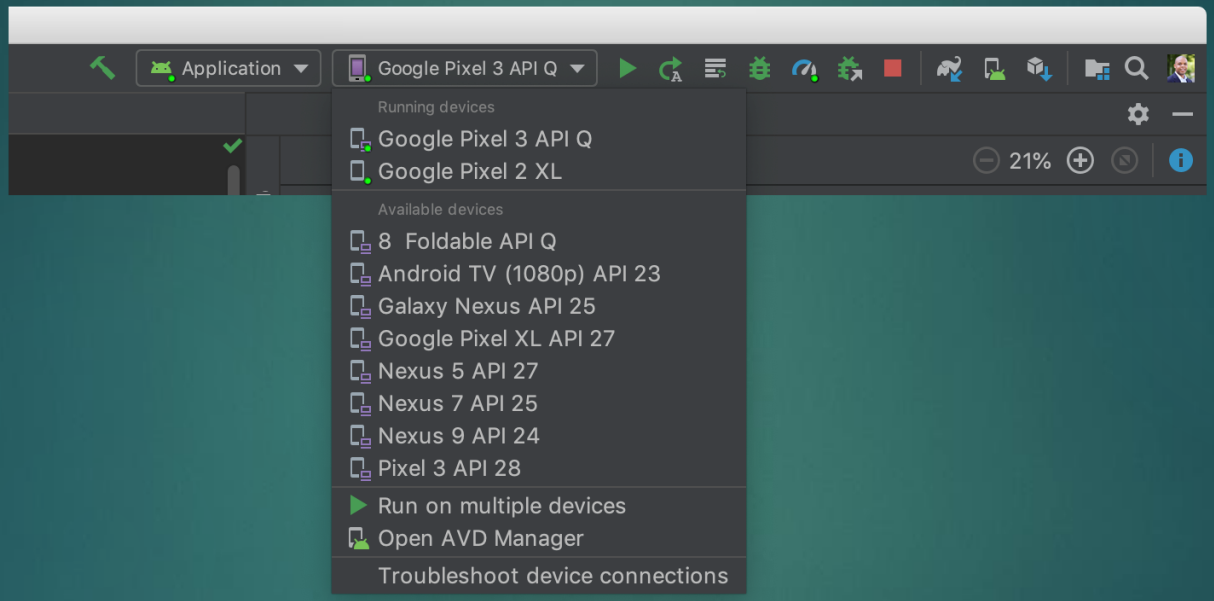
- There are two files with this name:
- one for the project, "Project: My_First_App," and
- one for the app module, "Module: My_First_App.app."
- Each module has its own build.gradle file
- Use each module's build.gradle file to control how the Gradle plugin builds your app.

Run your app

Run on an emulator

1. In Android Studio, create an Android Virtual Device (AVD) that the emulator can use to install and run your app.
2. In the toolbar, select your app from the run/debug configurations drop-down menu.
3. From the target device drop-down menu, select the AVD that you want to run your app on.
4. Click Run 

- Android Studio installs the app on the AVD and starts the emulator. You now see "Hello, World!" displayed in the app.


Application
Google Pixel 3 API Q
Running devices
Google Pixel 3 API Q
Google Pixel 2 XL
Available devices
8 Foldable API Q
Android TV (1080p) API 23
Galaxy Nexus API 25
Google Pixel XL API 27
Nexus 5 API 27
Nexus 7 API 25
Nexus 9 API 24
Pixel 3 API 28
Run on multiple devices
Open AVD Manager
Troubleshoot device connections
21%

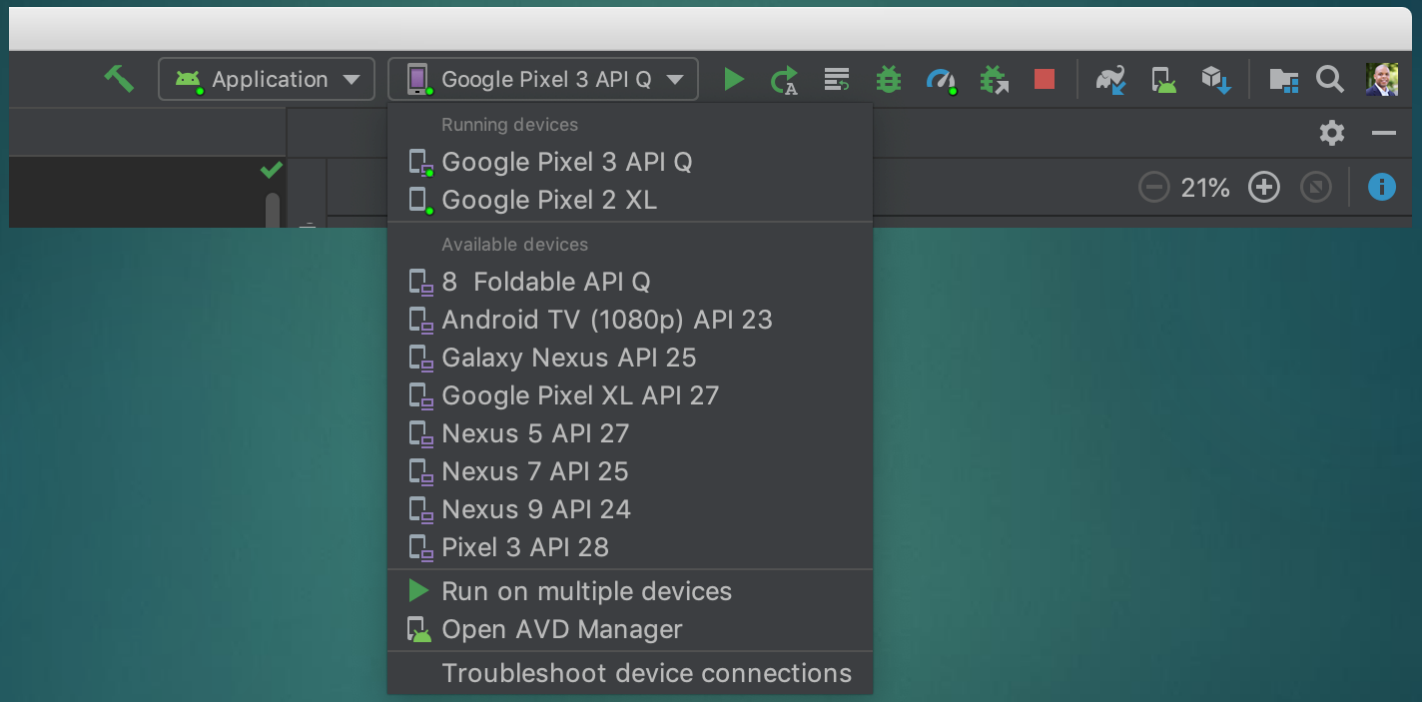
Run your app on device

Set up your device as follows:

1. Connect your device to your development machine with a USB cable. If you developed on Windows, you might need to install the appropriate USB driver for your device.
2. Perform the following steps to enable **USB debugging** in the **Developer options** window:
    a. Open the **Settings** app.
    b. Scroll to the bottom and select **About phone**.
    c. tap **Build number** seven times.
    d. Return to the previous screen, scroll to the bottom, and tap **Developer options**.
    e. In the **Developer options** window, scroll down to find and enable **USB debugging**.

Run the app on your device as follows:

1. In Android Studio, select your app from the run/debug configurations drop-down menu in the toolbar.
2. In the toolbar, select the device that you want to run your app on from the target device drop-down menu.
3. Click Run
4. Android Studio installs your app on your connected device and starts it. You now see "Hello, World!" displayed in the app on your device.


Application
Google Pixel 3 API Q
Running devices
Google Pixel 3 API Q
Google Pixel 2 XL
Available devices
8 Foldable API Q
Android TV (1080p) API 23
Galaxy Nexus API 25
Google Pixel XL API 27
Nexus 5 API 27
Nexus 7 API 25
Nexus 9 API 24
Pixel 3 API 28
Run on multiple devices
Open AVD Manager
Troubleshoot device connections
21%

Layouts

Views

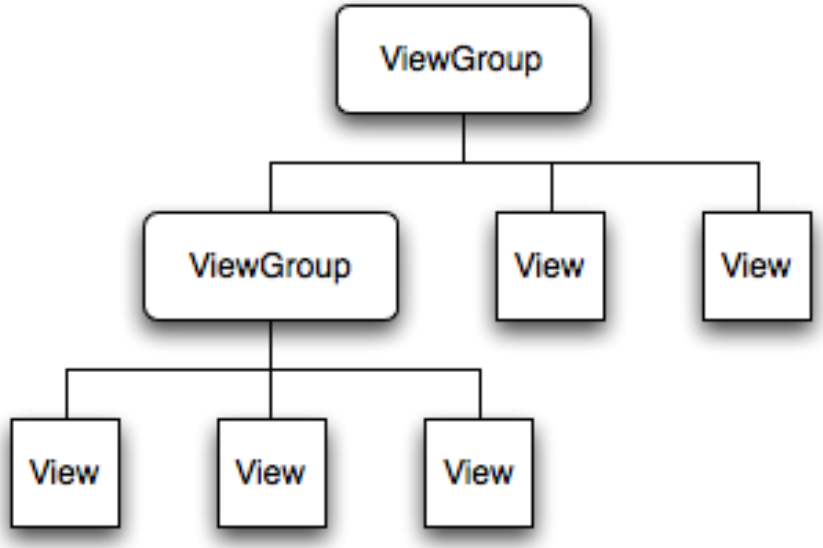
- UI consists of a hierarchy of objects called views
- view class represents the basic building block for all UI components
- It is the base class for classes that provide interactive UI components such as buttons, checkboxes, and text entry fields.
- Android system provides many predefined View subclasses- TextView, EditText, Button, ScrollView, RecyclerView, ImageView, ConstraintLayout and LinearLayout for containing other views
- View elements are specified in layout resource files. Layout resources are written in XML and listed within the layout folder in the res folder in the Project › Android pane

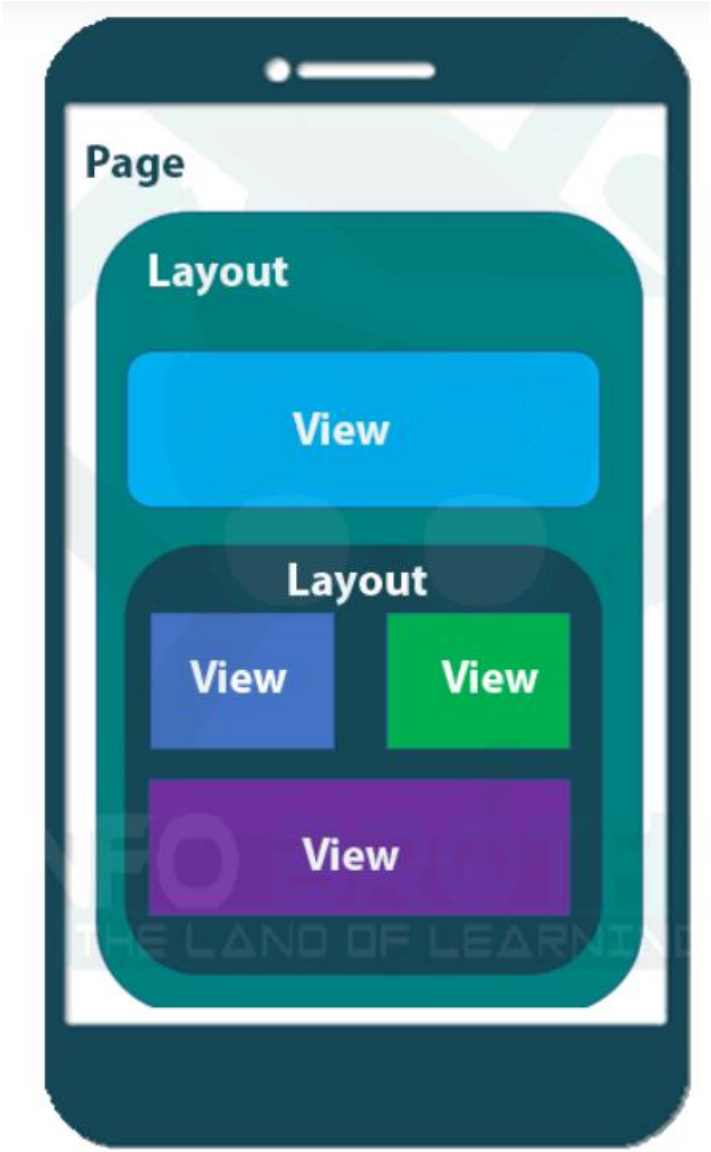
ViewGroup groups

- View elements can be grouped inside a ViewGroup, which acts as a container
- The relationship is parent-child, in which the parent is a ViewGroup, and the child is a View or another ViewGroup
- ViewGroup is a special kind of view which is extended from View as its base class. ViewGroup is the base class for layouts.
- Examples: ConstraintLayout, ScrollView, RecyclerView

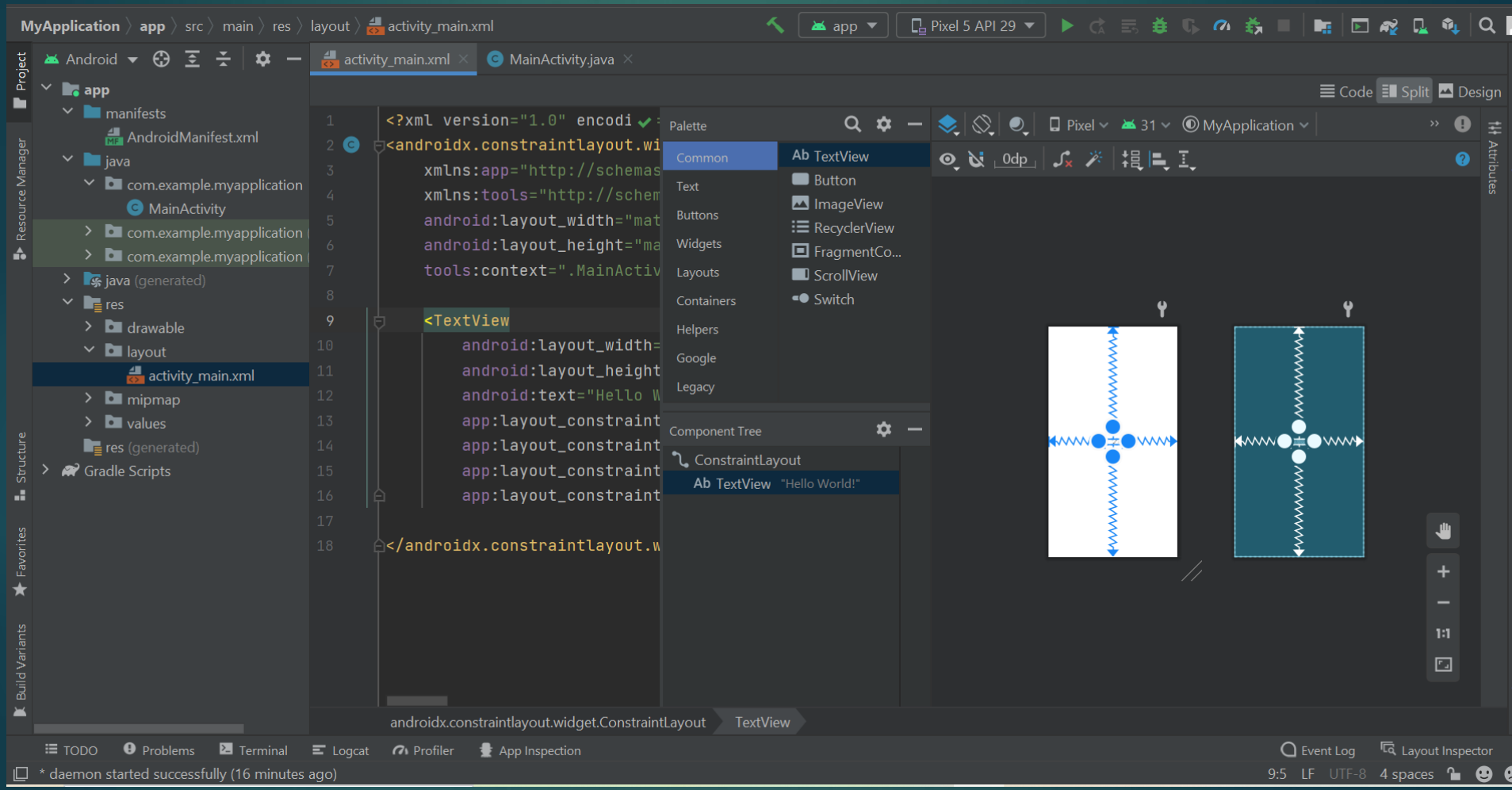
Layout ViewGroup groups

- View elements for a screen are organized in a hierarchy
- At the root of this hierarchy is a ViewGroup that contains the layout of the entire screen
- The ViewGroup can contain child View elements or other ViewGroup
- Some ViewGroup groups are designated as layouts because they organize child View elements in a specific way and are typically used as the root ViewGroup
- Examples: ConstraintLayout, GridLayout, TableLayout etc




Page
Layout
View
Layout
View
View
View

Layout editor



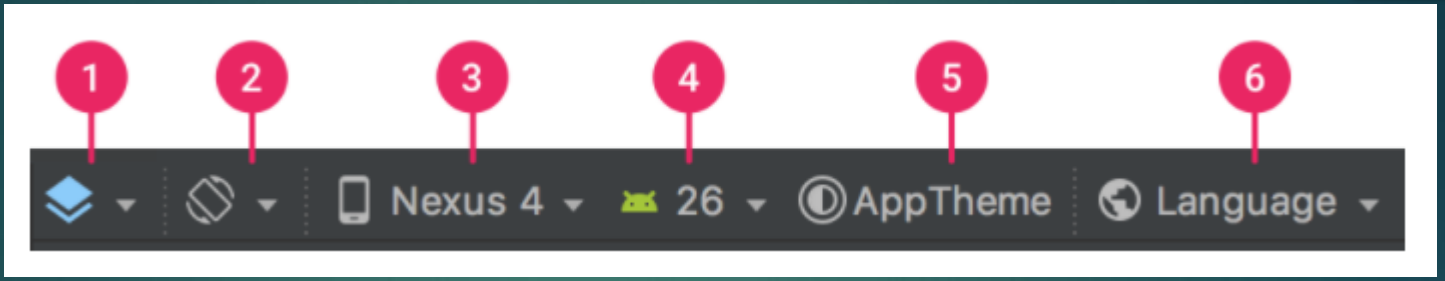
The layout editor

- layouts are defined using layout editor, or by entering XML code
- The layout editor shows a visual representation of XML code.
- One can drag View elements into the design or blueprint pane and arrange, resize, and specify attributes for them.

The layout editor

- XML layout file (activity_main.xml).
- Design and Code tabs: Click Design to see the layout editor, or Code to see XML code.
- Palette pane: The Palette pane provides a list of UI elements and layouts. Add an element or layout to the UI by dragging it into the design pane.
- Component Tree: The Component Tree pane shows the layout hierarchy. Click a View element or ViewGroup in this pane to select it. View elements are organized into a tree hierarchy of parents and children, in which a child inherits the attributes of its parent. In the figure above, the TextView is a child of the ConstraintLayout.
- Design and blueprint panes: Drag View elements from the Palette pane to the design or blueprint pane to position them in the layout.
- Toolbar: Click these buttons to configure your layout appearance in the editor and change layout attributes.
- Attributes tab: Click Attributes to display the Attributes pane for setting attributes for a View element.

Change the preview appearance



- Design and blueprint: Select how you'd like to view your layout in the editor. Choose Design to see a rendered preview of your layout. Choose Blueprint to see only outlines for each view. Choose Design + Blueprint to see both views side-by-side. You can also press B to cycle through these view types.
- Screen orientation and layout variants: Choose between landscape and portrait screen orientation, or choose other screen modes for which your app provides alternative layouts, such as night mode. This menu also contains commands for creating a new layout variant. You can also press O to change orientation.

Change the preview appearance

- Device type and size: Select the device type (phone/tablet, Android TV, or Wear OS) and screen configuration (size and density). You can select from several pre-configured device types and your own AVD definitions, or you can create a new AVD by selecting Add Device Definition from the list. You can resize the device size by dragging the bottom-right corner of the layout. You can also press D to cycle through the device list.
- API version: Select the version of Android on which to preview your layout.
- App theme: Select which UI theme to apply to the preview. Note that this works only for supported layout styles, so many themes in this list result in an error.
- Language: Select the language to show for your UI strings. This list displays only the languages available in your string resources.

Layout Properties:

- Each layout has a set of attributes which define the visual properties of that layout. There are few common attributes among all the layouts and there are other attributes which are specific to that layout

Common attributes that will be applied to all the layouts:

android:id	ID which uniquely identifies the view
android:layout_width	width of the layout
android:layout_height	height of the layout
android:layout_marginTop	extra space on the top side of the layout.
android:layout_gravity	specifies how child Views are positioned.
android:layout_weight	specifies how much of the extra space in the layout should be allocated to the View.
android:layout_x	specifies the x-coordinate of the layout.
android:paddingLeft	left padding filled for the layout.

width and height are the dimension of the layout/view can be specified in terms of:

- dp (Density-independent Pixels)
- sp (Scale-independent Pixels)
- pt (Points which is 1/72 of an inch)
- px(Pixels)
- mm (Millimeters)
- in(inches)

- We can specify width and height with exact measurements but more often, we use one of these:
  - android:layout_width=wrap_content tells view to size itself to the dimensions required by its content.
  - android:layout_width=fill_parent tells view to have its width as its parent view.

Gravity attribute values

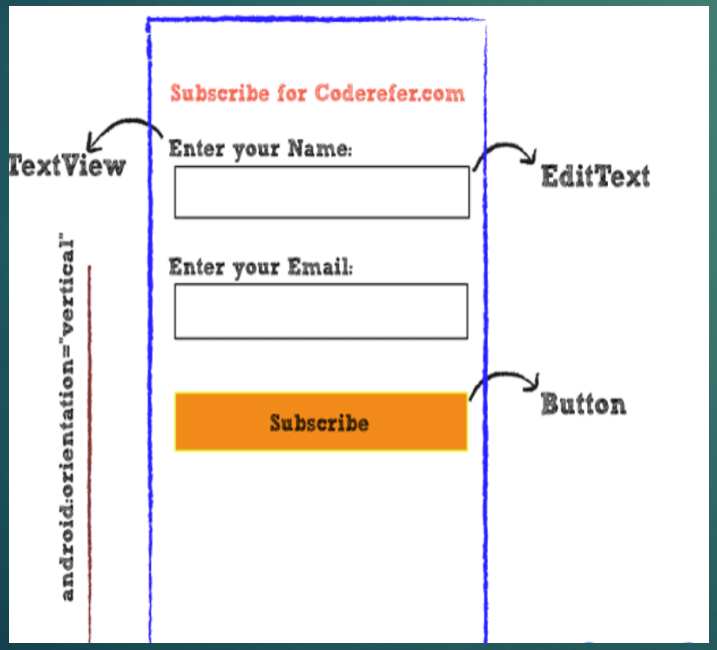
- top — Push object to the top of its container, not changing its size.
- bottom — Push object to the bottom of its container, not changing its size
- left — Push object to the left of its container, not changing its size.
- right — Push object to the right of its container, not changing its size.
- center_vertical — Place object in the vertical center of its container, not changing its size
- fill_vertical — Grow the vertical size of the object if needed so it completely fills its container
- center_horizontal — Place object in the horizontal center of its container, not changing its size
- fill_horizontal — Grow the horizontal size of the object if needed so it completely fills its container.

- center — Place the object in the center of its container in both the vertical and horizontal axis, not changing its size..
- fill — Grow the horizontal and vertical size of the object if needed so it completely fills its container.
- clip_vertical — Additional option that can be set to have the top and/or bottom edges of the child clipped to its container's bounds. The clip will be based on the vertical gravity: a top gravity will clip the bottom edge, a bottom gravity will clip the top edge, and neither will clip both edges
- clip_horizontal — Additional option that can be set to have the left and/or right edges of the child clipped to its container's bounds. The clip will be based on the horizontal gravity: a left gravity will clip the right edge, a right gravity will clip the left edge, and neither will clip both edges
- start — Push object to the beginning of its container, not changing its size.
- end — Push object to the end of its container, not changing its size

Linear Layout:

- Linear Layout is a layout which aligns the widgets or elements in a linear fashion.
- Linear Layout consists of two types of orientation:

1. Vertical Orientation,
2. Horizontal Orientation.



Linear Layout attributes

android:id

- This is the ID which uniquely identifies the layout.

android:orientation

- This specifies the direction of arrangement and you will use "horizontal" for a row, "vertical" for a column. The default is horizontal.

android:gravity

- This specifies how an object should position its content, on both the X and Y axes. Possible values are top, bottom, left, right, center, center_vertical, center_horizontal etc.

activity_main.xml


```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:orientation="vertical" >

   <Button android:id="@+id/btnStartService"
      android:layout_width="270dp"
      android:layout_height="wrap_content"
      android:text="start_service"/>

   <Button android:id="@+id/btnPauseService"
      android:layout_width="270dp"
      android:layout_height="wrap_content"
      android:text="pause_service"/>

   <Button android:id="@+id/btnStopService"
      android:layout_width="270dp"
      android:layout_height="wrap_content"
      android:text="stop_service"/>

</LinearLayout>
```

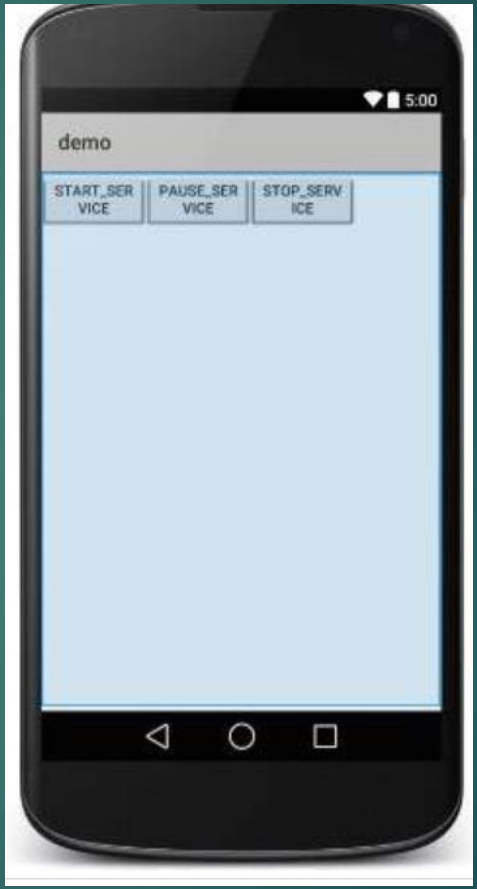
MainActivity.java

.....

```
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

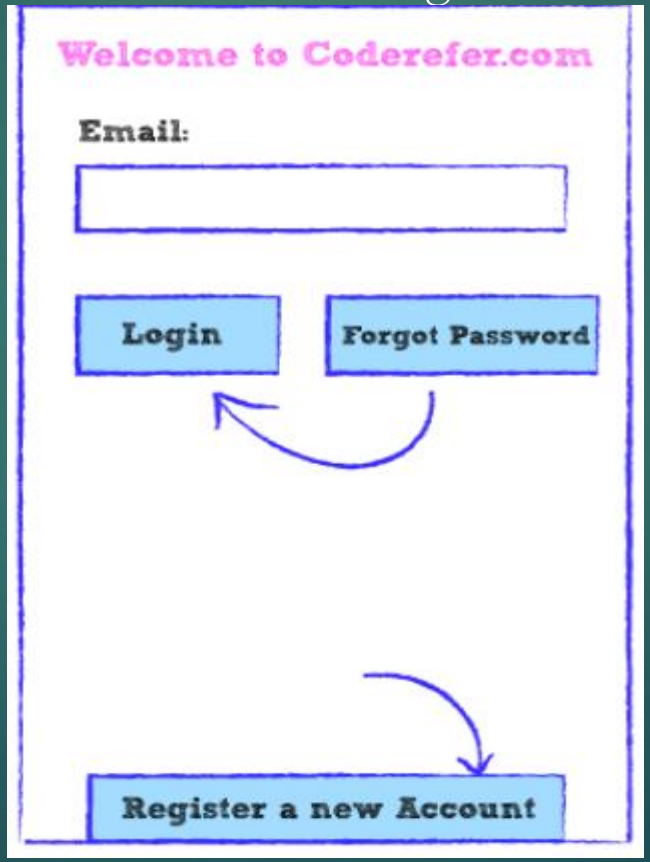
Orientation=vertical

Orientation=horizontal



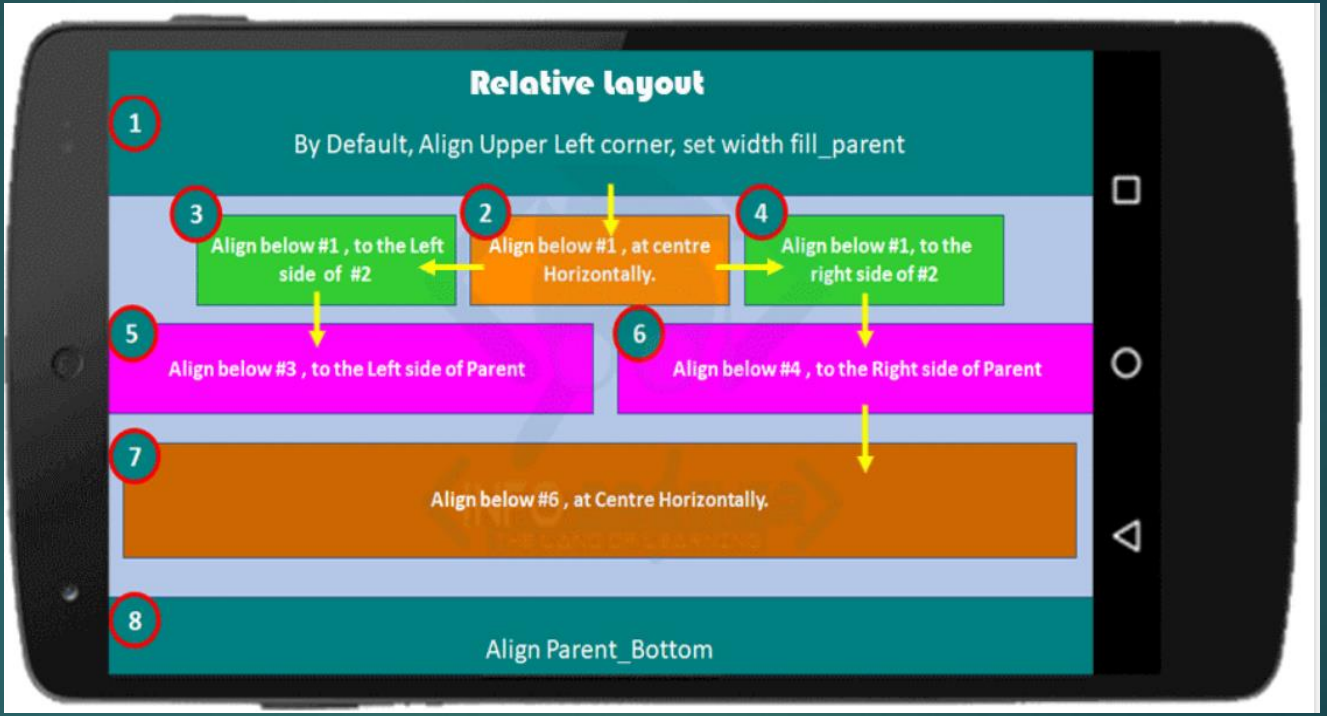
Relative Layout:

- Relative Layout is a layout where the widgets (such as Text Views, Buttons, etc.) are represented with respect to sibling elements or relative to the parent.
- In the figure, 'Forgot Password' button is positioned relative to 'Login' Button, whereas 'Register a new Account' is aligned with respect to Parent Layout.



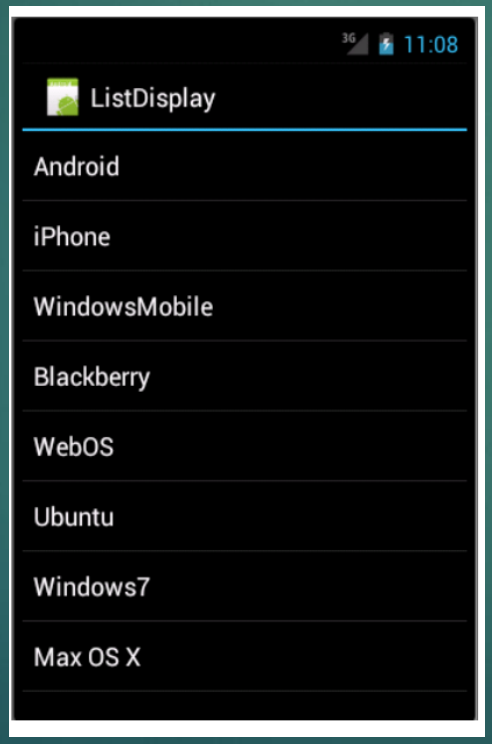
```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:paddingLeft="16dp"
   android:paddingRight="16dp" >

   <EditText
      android:id="@+id/name"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:hint="@string/reminder" />

   <LinearLayout
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:layout_alignParentStart="true"
      android:layout_below="@+id/name">

      <Button
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="New Button"
         android:id="@+id/button" />

      <Button
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="New Button"
         android:id="@+id/button2" />

   </LinearLayout>

</RelativeLayout>
```


5:00
demo
Enter your name
NEW BUTTON
NEW BUTTON


Relative layout
1
By Default, Align Upper Left corner, set width fill_parent
3
Align below #1 , to the Left side of #2
2
Align below #1 , at centre Horizontally.
4
Align below #1, to the right side of #2
5
Align below #3 , to the Left side of Parent
6
Align below #4 , to the Right side of Parent
7
Align below #6 , at Centre Horizontally.
8
Align Parent_Bottom

List View:

- A List View is a View Layout where all the items are specified in the form of vertical scrollable list
- The list items are automatically inserted to the list using an Adapter that pulls content from a source such as an array or database.



- You can use this adapter when your data source is an array. By default, ArrayAdapter creates a view for each array item by calling toString() on each item and placing the contents in a TextView. Consider you have an array of strings you want to display in a ListView, initialize a new ArrayAdapter using a constructor to specify the layout for each string and the string array

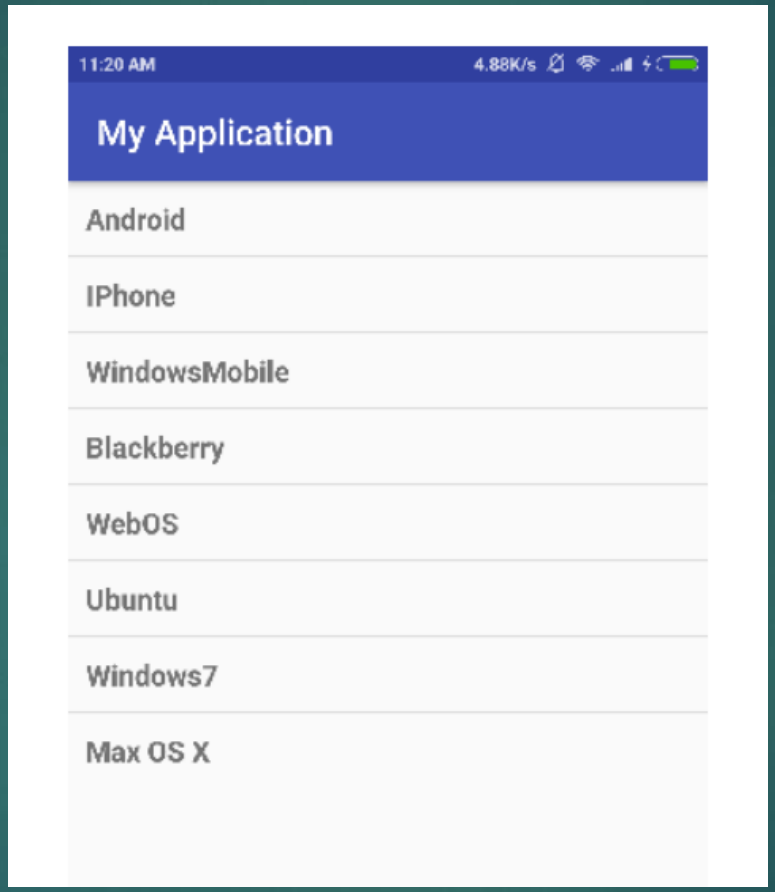
```
ArrayAdapter adapter = new ArrayAdapter<String>(this,R.layout.ListView,StringArray);
```

- Once you have array adapter created, then simply call setAdapter() on your ListView object as follows -
- ListView listView = (ListView) findViewById(R.id.listview);
- listView.setAdapter(adapter);

```
public class ListDisplay extends Activity {
   // Array of strings...
   String[] mobileArray = {"Android","IPhone","WindowsMobile","Blackberry",
      "WebOS","Ubuntu","Windows7","Max OS X"};

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      ArrayAdapter adapter = new ArrayAdapter<String>(this,
         R.layout.activity_listview, mobileArray);

      ListView listView = (ListView) findViewById(R.id.mobile_list);
      listView.setAdapter(adapter);
   }
}
```

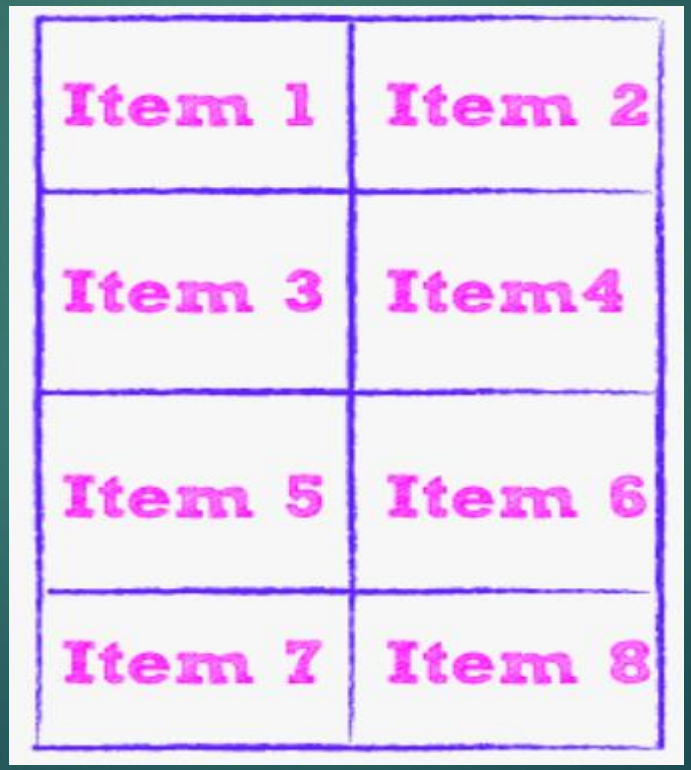
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical"
   tools:context=".ListActivity" >

   <ListView
      android:id="@+id/mobile_list"
      android:layout_width="match_parent"
      android:layout_height="wrap_content" >
   </ListView>

</LinearLayout>
```

```
<?xml version="1.0" encoding="utf-8"?>
<!--  Single List Item Design -->

<TextView xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/label"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:padding="10dip"
   android:textSize="16dip"
   android:textStyle="bold" >
</TextView>
```


11:20 AM
4.88K/s
My Application
Android
IPhone
WindowsMobile
Blackberry
WebOS
Ubuntu
Windows7
Max OS X

Grid View:

- A Grid View is a View Layout where the items (such as pictures, files etc.) are placed in a Grid manner
- the grid items are not necessarily predetermined but they automatically inserted to the layout using a ListAdapter



- The ListView and GridView are subclasses of AdapterView and they can be populated by binding them to an Adapter, which retrieves data from an external source and creates a View that represents each data entry.

```
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        GridView gridview = (GridView) findViewById(R.id.gridview);
        gridview.setAdapter(new ImageAdapter(this));
    }
}
```

Following will be the content of **res/layout/activity_main.xml** file −

```
<?xml version="1.0" encoding="utf-8"?>
<GridView xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/gridview"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:columnWidth="90dp"
   android:numColumns="auto_fit"
   android:verticalSpacing="10dp"
   android:horizontalSpacing="10dp"
   android:stretchMode="columnWidth"
   android:gravity="center"
/>
```

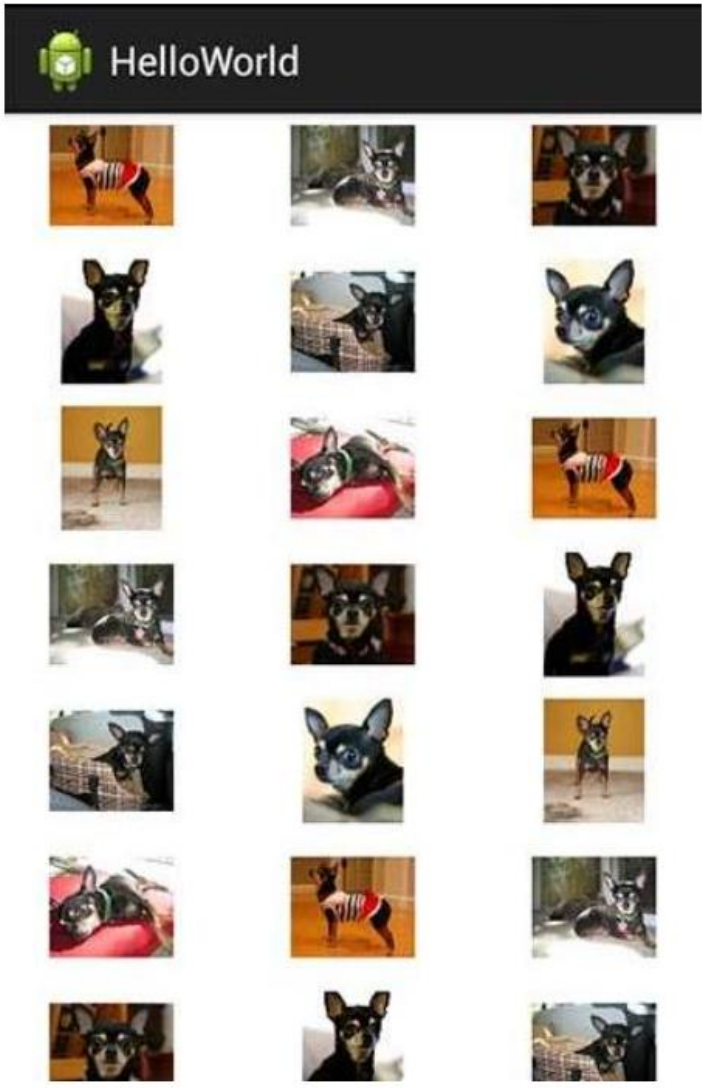
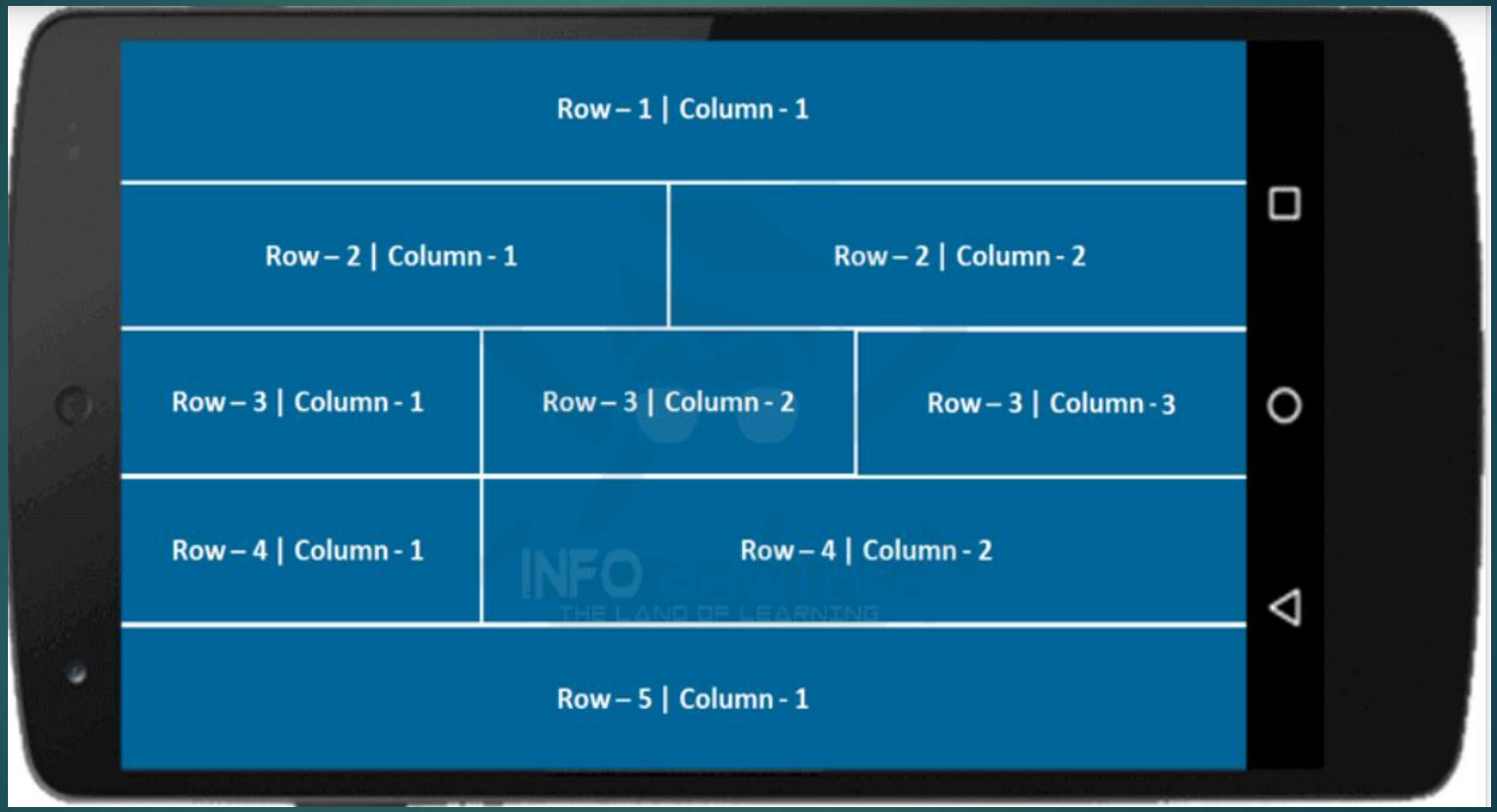
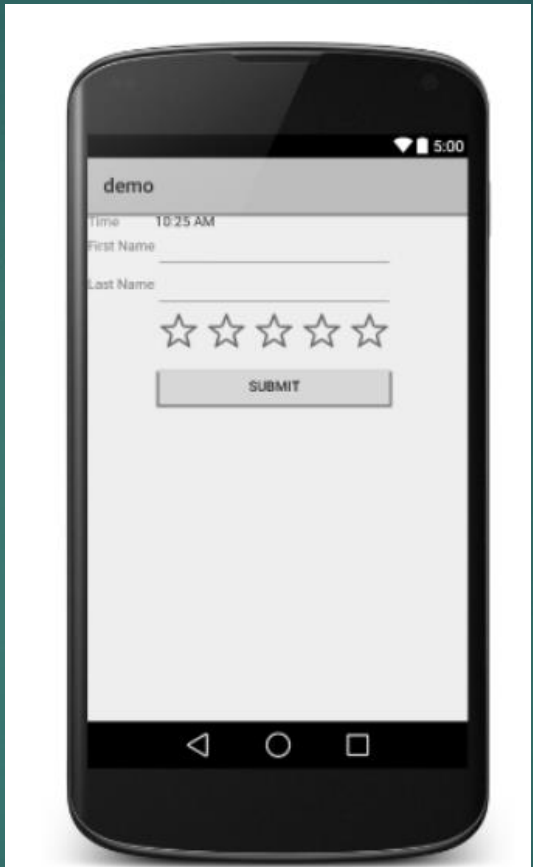

HelloWorld

Table layout

- It let us arrange elements in a rows and columns. We will use the ‹TableRow› element to build a row in the table. Each row has zero or more cells; each cell can hold one View object.

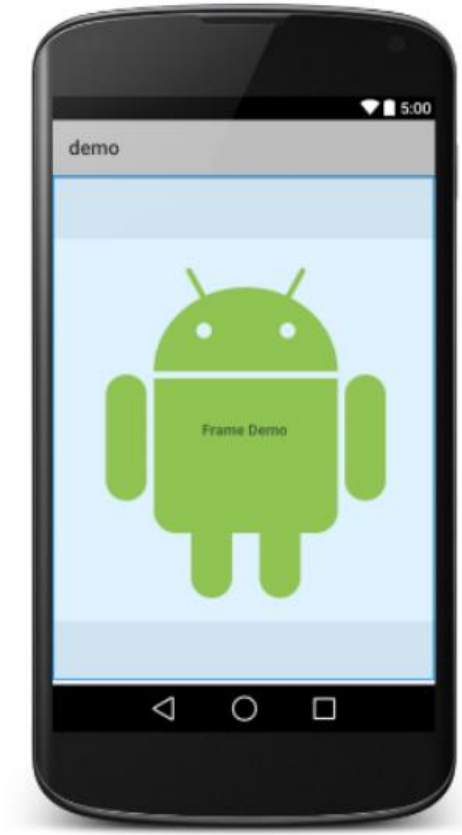



5:00
demo
Time
10:25 AM
First Name
Last Name
SUBMIT

Frame layout

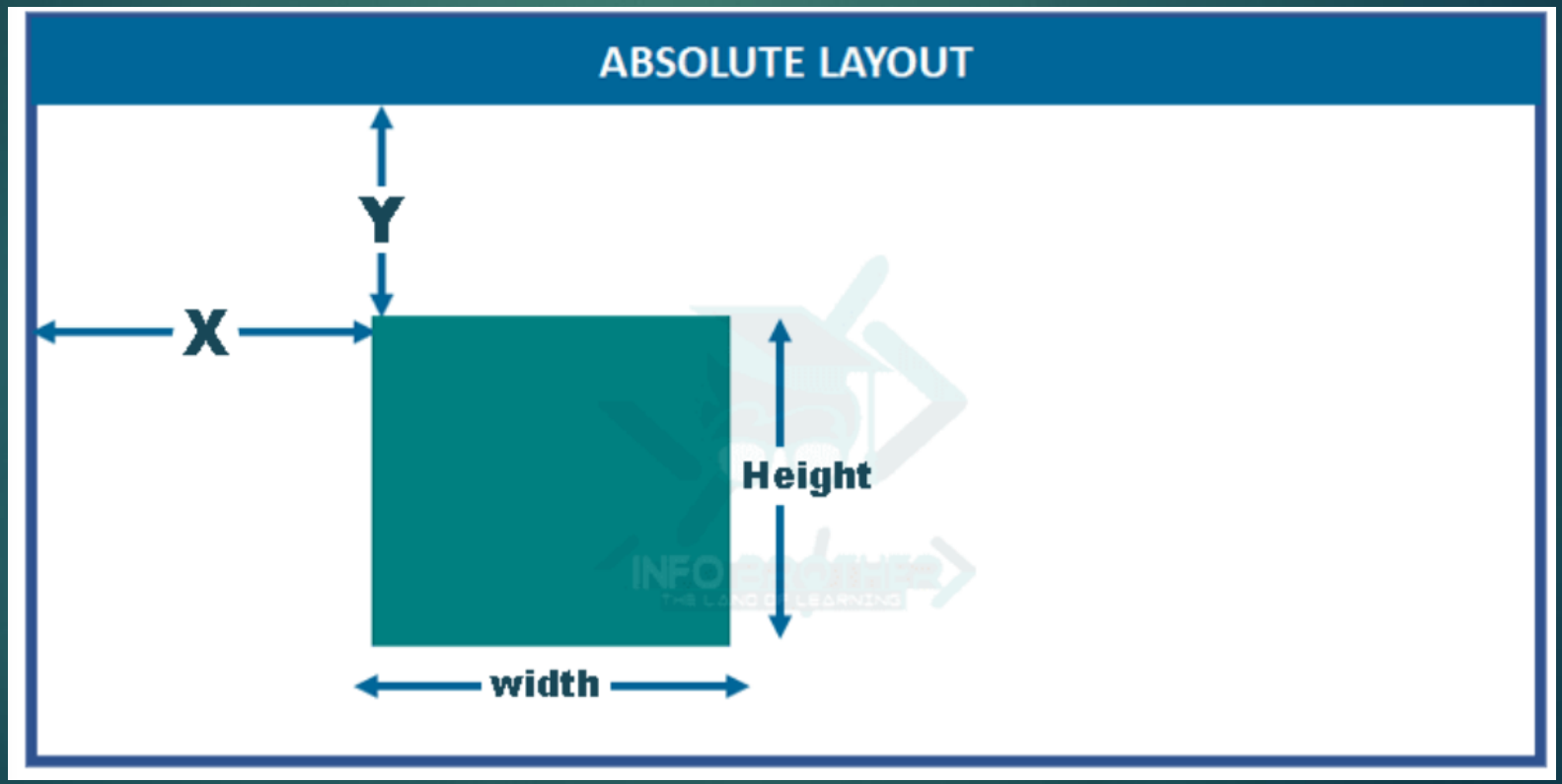
- Frame Layout is designed to block out an area on the screen to display a single item.
- Generally, FrameLayout should be used to hold a single child view, because it can be difficult to organize child views in a way that's scalable to different screen sizes without the children overlapping each other.
- We can, however, add multiple children to a FrameLayout and control their position within the FrameLayout by assigning gravity to each child, using the android:layout_gravity attribute.
-

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/androi
   android:layout_width="fill_parent"
   android:layout_height="fill_parent">

   <ImageView
      android:src="@drawable/ic_launcher"
      android:scaleType="fitCenter"
      android:layout_height="250px"
      android:layout_width="250px"/>

   <TextView
      android:text="Frame Demo"
      android:textSize="30px"
      android:textStyle="bold"
      android:layout_height="fill_parent"
      android:layout_width="fill_parent"
      android:gravity="center"/>
</FrameLayout>
```

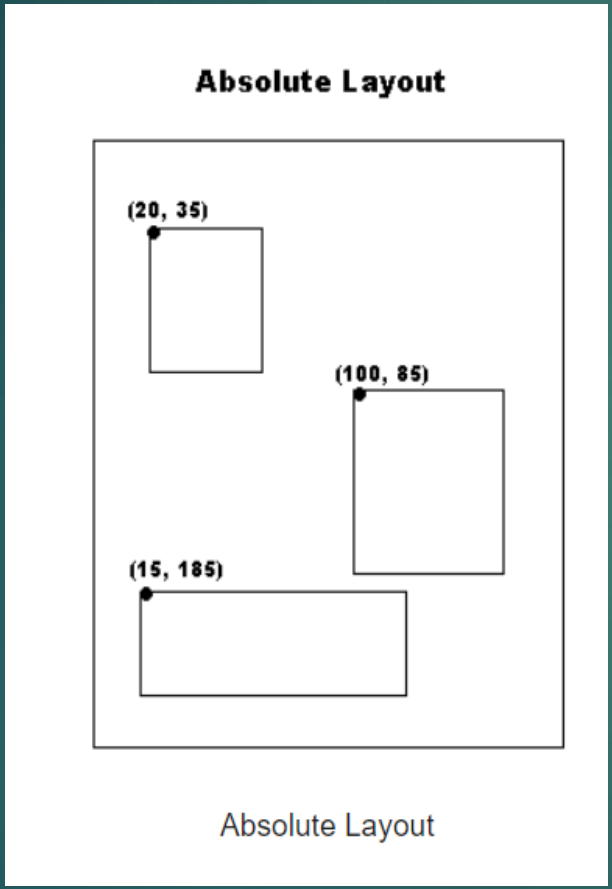

5:00
demo
Frame Demo

Absolute layout

- It let us specify the exact location of its children using and x and Y coordinates
- Absolute layouts are less flexible and harder to maintain than other types of layouts
-



```
<AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/andr
   android:layout_width="fill_parent"
   android:layout_height="fill_parent">

   <Button
      android:layout_width="100dp"
      android:layout_height="wrap_content"
      android:text="OK"
      android:layout_x="50px"
      android:layout_y="361px" />
   <Button
      android:layout_width="100dp"
      android:layout_height="wrap_content"
      android:text="Cancel"
      android:layout_x="225px"
      android:layout_y="361px" />

</AbsoluteLayout>
```


Absolute Layout
(20, 35)
(100, 85)
(15, 185)
Absolute Layout


5:00
demo
OK
CANCEL

Text Views

- TextView in Android is one of the basic and important UI elements
- It is used to display text on the screen
- A TextView can be used to show a message, a response from a database, or even entire magazine-style articles that users can scroll
- We can use TextView to view a text of any size, from a single character or word to a full screen of text
- We can add a resource id to the TextView and can control how the text appears using attributes in the layout file
- We can refer to a TextView in Java code by using its resource id in order to update the text or its attributes from code

TextView Attributes

We can use XML attributes for a TextView to control:

- Where the TextView is positioned in a layout (like any other view)
- How the TextView itself appears, such as with a background color
- What the text looks like within the TextView, such as the initial text and its style, size, and color

TextView Attributes

id: id is an attribute used to uniquely identify a text view

gravity: The gravity attribute is an optional attribute which is used to control the alignment of the text like left, right, center, top, bottom, center_vertical, center_horizontal etc

Example:

```
<TextView
android:id="@+id/simpleTextView"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="NMAMIT Nitte"
android:textSize="20sp"
android:gravity="center_horizontal"/>
```

TextView Attributes

text: text attribute is used to set the text in a text view. We can set the text in xml as well as in the java class.

Setting the text in a textview programmatically:

```
TextView textView = (TextView)findViewById(R.id.textView);
textView.setText("NMAMIT Nitte");
```

TextView Attributes

textColor: textColor attribute is used to set the text color of a text view. Color value is in the form of “#argb”, “#rgb”, “#rrggbb”, or “#aarrggbb”

Setting the text color of a text view programmatically(main_activity.java)

```
public class MainActivity extends AppCompatActivity {
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        TextView textView = (TextView)findViewById(R.id.textView);
        textView.setTextColor(Color.RED);

    }
}
```

TextView Attributes

textSize: textSize attribute is used to set the size of text of a text view. We can set the text size in sp(scale independent pixel) or dp(density pixel)

Setting the text size of a text view programmatically:

```
TextView textView = (TextView)findViewById(R.id.textView);
textView.setTextSize(20);
```

TextView Attributes

textStyle: textStyle attribute is used to set the text style of a text view. The possible text styles are bold, italic and normal. If we need to use two or more styles for a text view then "|" operator is used

background: background attribute is used to set the background of a text view. We can set a color or a drawable in the background of a text view

setting the background color of a text view programmatically

```
TextView textView = (TextView)findViewById(R.id.textView);
textView.setBackgroundColor(Color.BLACK);
```

padding: padding attribute is used to set the padding from left, right, top or bottom.

TextView Attributes

Text Shadow

android:shadowDx=”integer_value” -> which decides the distance of text from its shadow with respect to x axis, if the integer_value is positive the shadow is on positive of the x axis

android:shadowDy=”integer_value” -> which decides the distance of text from its shadow with respect to y axis, if the integer_value is positive the shadow is on negative of the y axis

android:shadowRadius=”integer_value” -> which decides the amount of the shadow to be given for the text view

TextView Attributes

Letter Spacing

android:letterSpacing="floatingTypeValue" -> This attribute is used to give the space between each of the letters

All Caps

android:textAllCaps="trueOrfalse" -> This attribute decides, all the letters should be in uppercase or not.

Adding Icons for TextView

Android also allows adding drawable with the text views

positions to add the icons for the TextView: start, end, top, and bottom

TextView Attributes

android:autoLink: Controls whether links such as URLs and email addresses are automatically found and converted to clickable (touchable) links.

Use one of the following with android:autoLink:

none: Match no patterns (default).

web: Match web URLs.

email: Match email addresses.

phone: Match phone numbers.

map: Match map addresses.

all: Match all patterns (equivalent to web|email|phone|map).

TextView Attributes

- We can extract the text string into a string resource
- It is easier to maintain for multiple-language versions of the app, or if we need to change the string in the future.
- After extracting the string, use the string resource name with @string/ to specify the text
- Example:

```
<TextView
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="@string/greet"
 <!-- more attributes -->
/>
```

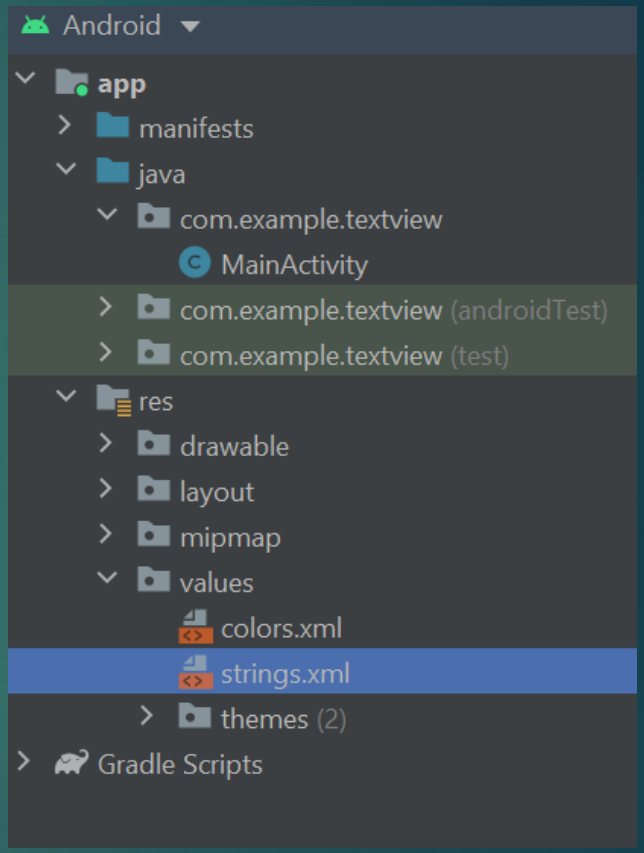
TextView Attributes

In strings.xml file:

‹resources›

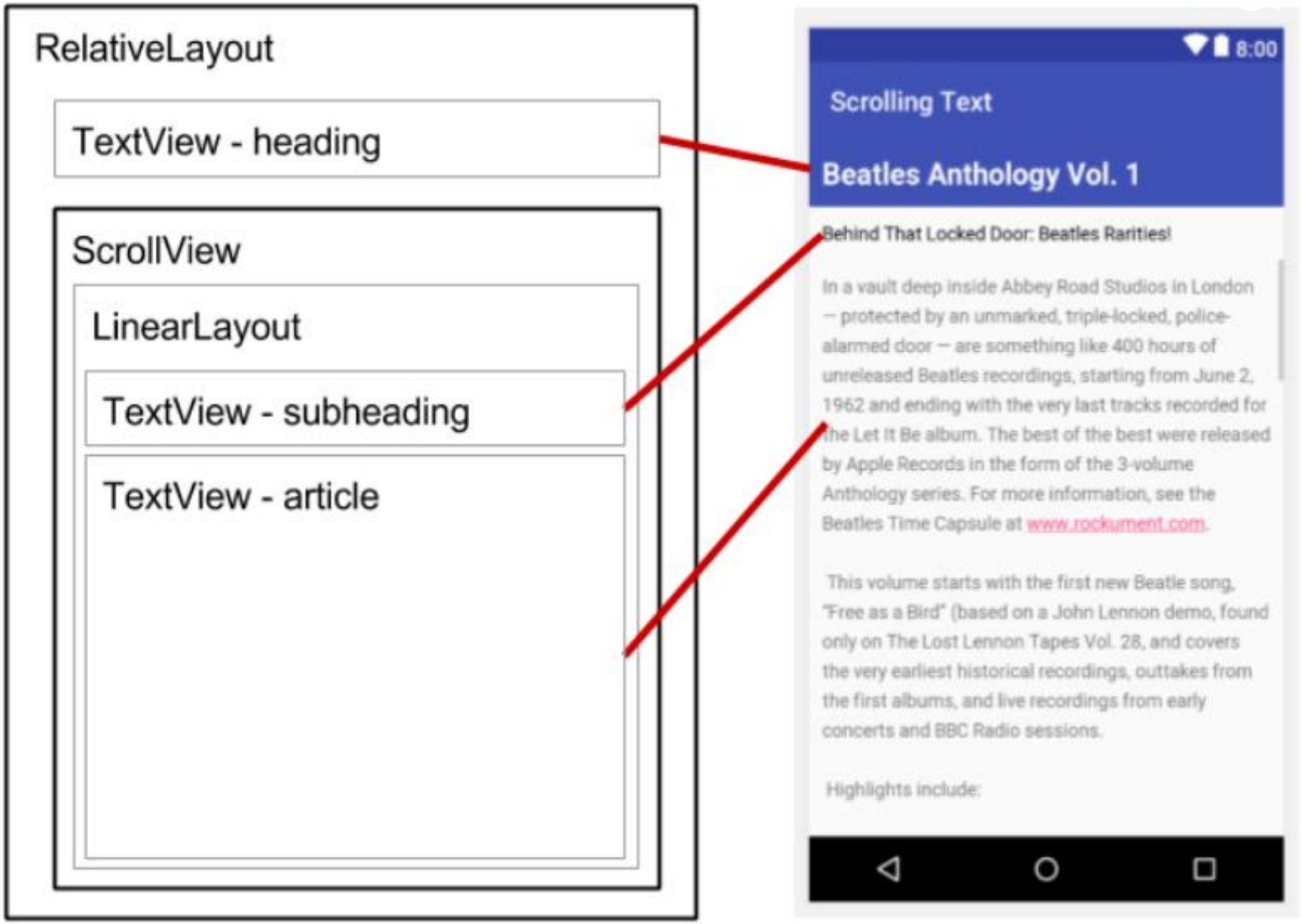
‹string name=“greet"›Good Morning‹/string›

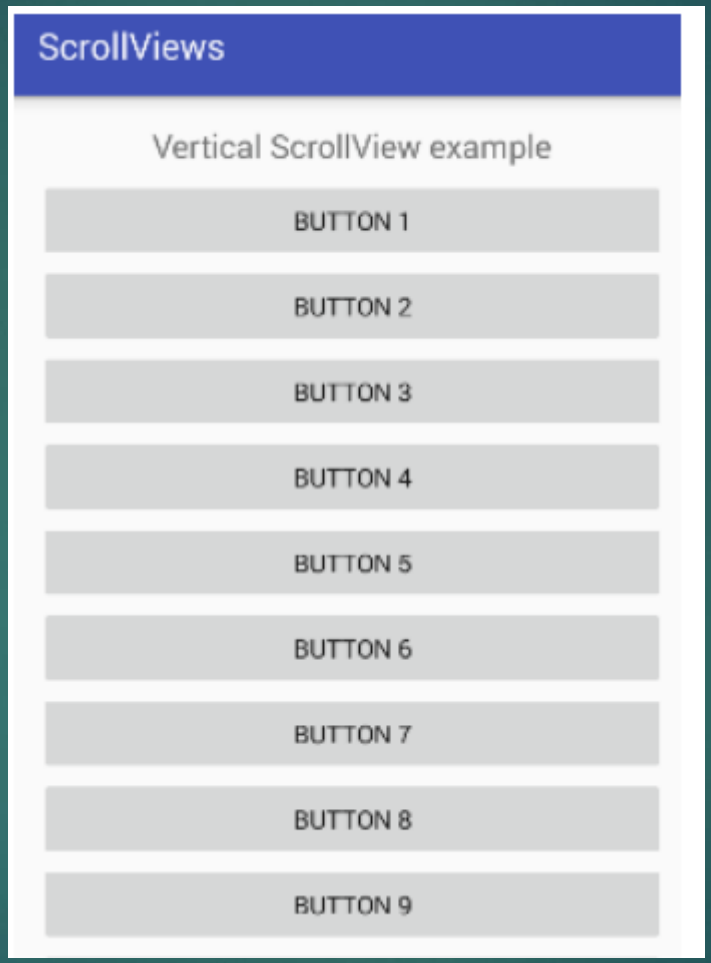
‹/resources›



ScrollView

- ScrollView is a view group that is used to make vertically scrollable views
- A scroll view contains a single direct child only
- In order to place multiple views in the scroll view, one needs to make a view group(like LinearLayout) as a direct child and then we can define many views inside it
- Scroll view supports vertical scrolling only. For horizontal scrolling, use HorizontalScrollView instead.


RelativeLayout
TextView - heading
ScrollView
LinearLayout
TextView - subheading
TextView - article
8:00
Scrolling Text
Beatles Anthology Vol. 1
Behind That Locked Door: Beatles Rarities!
In a vault deep inside Abbey Road Studios in London — protected by an unmarked, triple-locked, police-alarmed door — are something like 400 hours of unreleased Beatles recordings, starting from June 2, 1962 and ending with the very last tracks recorded for the Let It Be album. The best of the best were released by Apple Records in the form of the 3-volume Anthology series. For more information, see the Beatles Time Capsule at www.rockument.com.
This volume starts with the first new Beatle song, "Free as a Bird" (based on a John Lennon demo, found only on The Lost Lennon Tapes Vol. 28, and covers the very earliest historical recordings, outtakes from the first albums, and live recordings from early concerts and BBC Radio sessions.
Highlights include:


ScrollViews
Vertical ScrollView example
BUTTON 1
BUTTON 2
BUTTON 3
BUTTON 4
BUTTON 5
BUTTON 6
BUTTON 7
BUTTON 8
BUTTON 9

Activities

- An activity represents a single screen in an app with an interface the user can interact with
- Usually, any app includes collection of activities
- Although the activities in your app work with each other to form a cohesive user experience, each activity is independent of the others.
- This enables your app to start an activity in another app, and it enables other apps to start activities in your app (if your app allows this)
- Example: a messaging app could start an activity in a camera app to take a picture, then start an activity in an email app to let the user share the picture in email.

- Typically, one Activity in an app is specified as the "main" activity, (MainActivity)
- The user sees the main activity when they launch the app for the first time.
- Each activity can start other activities to perform different actions
- Each time a new activity starts, the previous activity is stopped, but the system preserves the activity in a stack (the "back stack")
- When the user is done with the current activity and presses the Back button, the activity is popped from the stack and destroyed, and the previous activity resumes
- An Activity has a life cycle state—created, started, runs, is paused, resumed, stopped, and destroyed

- When a new project is created or when a new Activity is added, automatically following steps are performed:
- Creation of an Activity Java class
- Implementing a basic UI for the Activity in an XML layout file
- Declaring the new Activity in the AndroidManifest.xml file.

Create the Activity

- When you create a new project in Android Studio and choose the Backwards Compatibility (AppCompat) option, the MainActivity is, by default, a subclass of the AppCompatActivity class.
- The AppCompatActivity class lets you use up-to-date Android app features such as the app bar and Material Design, while still enabling your app to be compatible with devices running older versions of Android.

skeleton subclass of AppCompatActivity:

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

- The first task for you in your Activity subclass is to implement the standard Activity lifecycle callback methods to handle the state changes for your Activity

- The one required callback your app must implement is the onCreate() method
- The system calls this method when it creates your Activity, and all the essential components of your Activity should be initialized here.
- Most importantly, the onCreate() method calls setContentView() to create the primary layout for the Activity
- When the setContentView() method is called with the path to a layout file, the system creates all the initial views from the specified layout and adds them to your Activity - inflating the layout.
- In addition to lifecycle callbacks, you may also implement methods in your Activity to handle other behavior such as user input or button clicks.

Implement the activity's UI

- The most common way to define a UI using View elements is with an XML layout file stored as part of your app's resources.
- Defining layout in XML enables you to maintain the design of your UI separately from the source code that defines the activity behavior.
- One can also create new View elements directly in your activity code

Declare the Activity in AndroidManifest.xml

- Each Activity in your app must be declared in the AndroidManifest.xml file with the ‹activity› element
- When you create a new project or add a new Activity to your project in Android Studio, the AndroidManifest.xml file is created or updated to include skeleton declarations for each Activity

```
<activity android:name=".MainActivity" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Declare the Activity in AndroidManifest.xml

- The ‹activity› element includes a number of attributes to define properties of the Activity such as its label, icon, or theme.
- The only required attribute is android:name, which specifies the class name for the Activity

- The ‹activity› element can also include declarations for Intent filters. The Intent filters specify the kind of Intent your Activity will accept.
- Intent filters must include at least one ‹action› element, and can also include a ‹category›
- The MainActivity for your app needs an Intent filter that defines the "main" action and the "launcher" category so that the system can launch your app.
- Android Studio creates this Intent filter for the MainActivity in your project.

Declare the Activity in AndroidManifest.xml

- The ‹action› element specifies that this is the "main" entry point to the app.
- The ‹category› element specifies that this Activity should be listed in the system's app launcher (to allow users to launch this Activity).
- Each Activity in your app can also declare Intent filters, but only your MainActivity should include the "main" action.

Add another Activity to your project

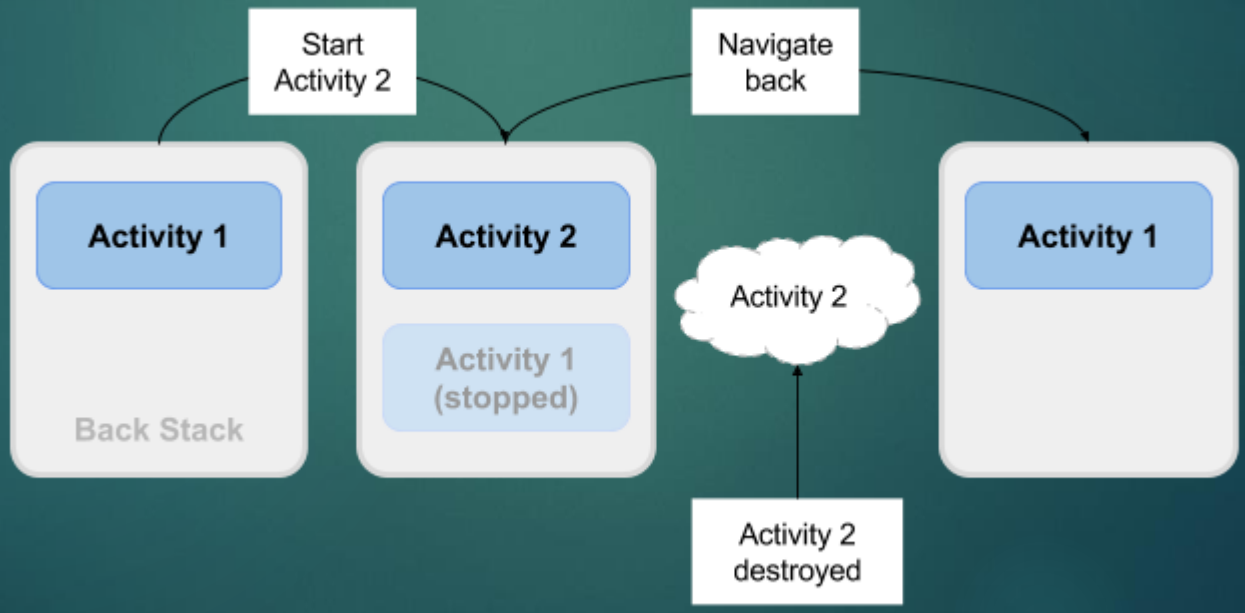
- You can add a new Activity to your project by choosing File › New › Activity
- Android Studio provides three things for each new activity in your app:
  - A Java file for the new Activity with a skeleton class definition and onCreate() method. The new Activity, like MainActivity, is a subclass of AppCompatActivity.
  - An XML file containing the layout for the new activity. Note that the setContentView() method in the Activity class inflates this new layout.
  - An additional ‹activity› element in the AndroidManifest.xml file that specifies the new activity. The second Activity definition does not include any Intent filters. If you plan to use this activity only within your app (and not enable that activity to be started by any other app), you do not need to add filters.

Activity lifecycle

- The activity lifecycle is the set of states an activity can be in during its entire lifetime, from the time it's created to when it's destroyed, and the system reclaims its resources
- As the user interacts with your app and other apps on the device, activities move into different state

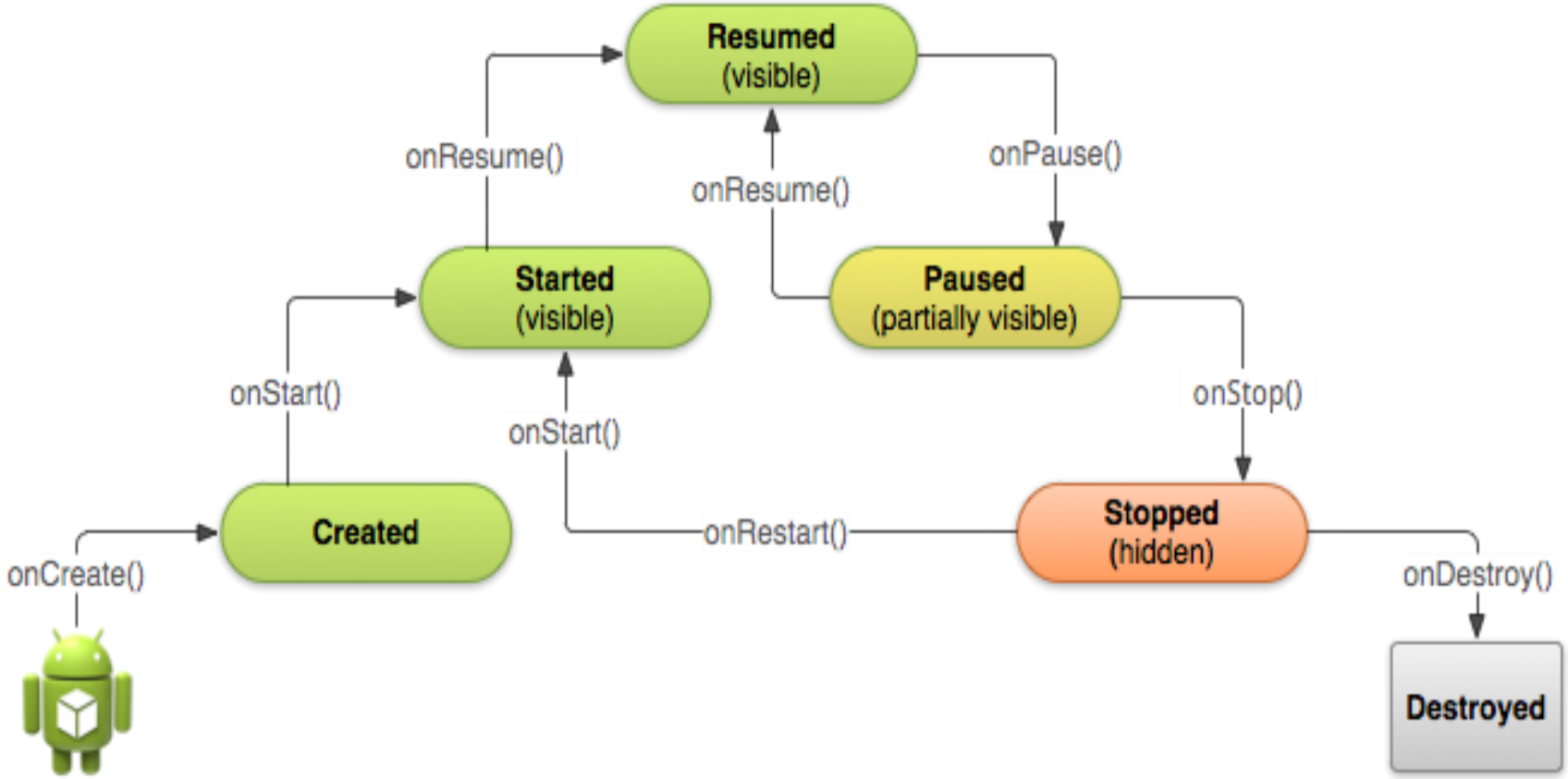
Activity lifecycle

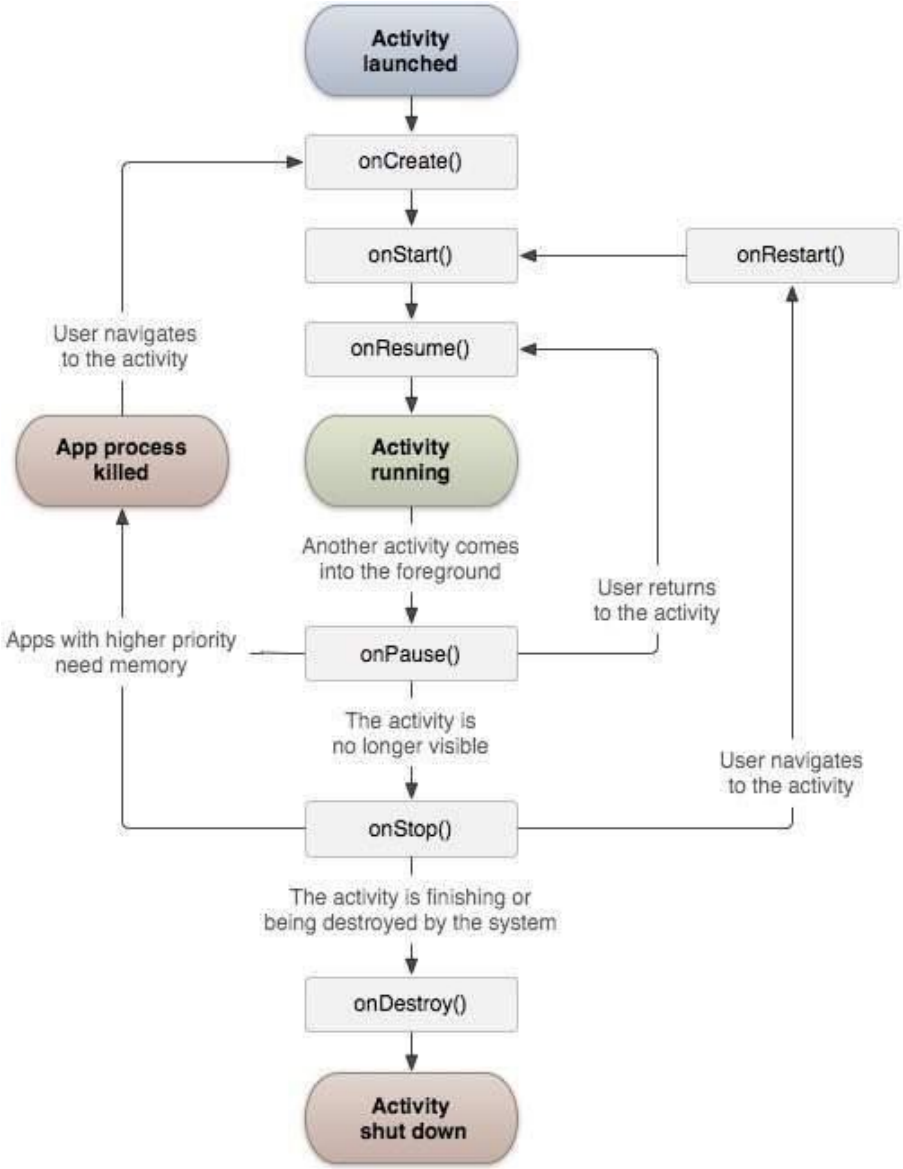
- When you start an app, the app's main activity ("Activity 1" in the figure below) is started, comes to the foreground, and receives the user focus.
- When you start a second activity ("Activity 2" in the figure below), a new activity is created and started, and the main activity is stopped.
- When you're done with the Activity 2 and navigate back, Activity 1 resumes. Activity 2 stops and is no longer needed.
- If the user doesn't resume Activity 2, the system eventually destroys it



Activity states and lifecycle callback methods

- When an Activity transitions into and out of the different lifecycle states as it runs, the Android system calls several lifecycle callback methods at each stage.
- All of the callback methods are hooks that you can override in each of your Activity classes to define how that Activity behaves when the user leaves and re-enters the Activity
- Lifecycle states (and callbacks) are per Activity, not per app, and you may implement different behavior at different points in the lifecycle of each Activity
- Depending on the complexity of your Activity, you probably don't need to implement all the lifecycle callback methods in an Activity
- But implement those that ensure your app behaves the way users expect.
- Managing the lifecycle of an Activity by implementing callback methods is crucial to developing a strong and flexible app.


Resumed
(visible)
onResume()
onResume()
onPause()
Started
(visible)
Paused
(partially visible)
onStart()
onStart()
onStop()
Created
onRestart()
Stopped
(hidden)
onCreate()
onDestroy()
Destroyed


Activity launched
onCreate()
onStart()
onRestart()
onResume()
Activity running
App process killed
User navigates to the activity
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Activity created: the onCreate() method

- Activity enters into the created state when it is started for the first time
- When an Activity is first created, the system calls the onCreate() method to initialize that Activity.
- Example, when the user taps your app icon from the Home screen to start that app, the system calls the onCreate() method for the Activity in your app that you've declared to be the "launcher" or "main" Activity.
- In this case the main Activity onCreate() method is analogous to the main() method in other programs.
- The onCreate() method is the only required callback you must implement in your Activity class.

Activity created: the onCreate() method

- Inside onCreate() method, perform basic app startup logic that should happen only once, such as setting up the user interface, assigning class-scope variables, or setting up background tasks
- Created is a transient state; the Activity remains in the created state only as long as it takes to run onCreate(), and then the Activity moves to the started state

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // The activity is being created.
}
```

Activity started: the onStart() method

- After an Activity is initialized with onCreate(), the system calls the onStart() method, and the Activity is in the started state
- The onStart() method is also called if a stopped Activity returns to the foreground, such as when the user clicks the Back button or the Up button to navigate to the previous screen
- While onCreate() is called only once when the Activity is created, the onStart() method may be called many times during the lifecycle of the Activity as the user navigates around your app
- When an Activity is in the started state and visible on the screen, the user cannot interact with it until onResume() is called, the Activity is running, and the Activity is in the foreground

Activity started: the onStart() method

- Typically you implement onStart() in your Activity as a counterpart to the onStop() method
- For example, if you release hardware resources (such as GPS or sensors) when the Activity is stopped, you can re-register those resources in the onStart() method
- Started, like created, is a transient state. After starting, the Activity moves into the resumed (running) state

```
@Override
protected void onStart() {
    super.onStart();
    // The activity is about to become visible.
}
```

Activity resumed/running: the onResume() method

- Activity is in the resumed state when it is initialized, visible on screen, and ready to use.
- The resumed state is often called the running state, because it is in this state that the user is actually interacting with your app
- The first time the Activity is started the system calls the onResume() method just after onStart()
- The onResume() method may also be called multiple times, each time the app comes back from the paused state

Activity resumed/running: the onResume() method

- As with the onStart() and onStop() methods, which are implemented in pairs, you typically only implement onResume() as a counterpart to onPause()
- If in the onPause() method you halt any animations, you would start those animations again in onResume()
- The Activity remains in the resumed state as long as the Activity is in the foreground and the user is interacting with it. From the resumed state the Activity can move into the paused state

```
@Override
protected void onResume() {
    super.onResume();
    // The activity has become visible (it is now "resumed").
}
```

Activity paused: the onPause() method

- The paused state can occur in several situations:

- The Activity is going into the background, but has not yet been fully stopped. This is the first indication that the user is leaving your Activity.
- The Activity is only partially visible on the screen, because a dialog or other transparent Activity is overlaid on top of it.
- In multi-window or split screen mode (API 24), the Activity is displayed on the screen, but some other Activity has the user focus.

Activity paused: the onPause() method

- The system calls the onPause() method when the Activity moves into the paused state.
- Because the onPause() method is the first indication you get that the user may be leaving the Activity, you can use onPause() to stop animation or video playback, release any hardware-intensive resources, or commit unsaved Activity changes (such as a draft email).
- The onPause() method should execute quickly. Don't use onPause() for CPU-intensive operations such as writing persistent data to a database.
- The app may still be visible on screen as it passes through the paused state, and any delays in executing onPause() can slow the user's transition to the next Activity.
- Implement any heavy-load operations when the app is in the stopped state instead.
- Activity can move from the paused state into the resumed state (if the user returns to the Activity) or to the stopped state (if the user leaves the Activity altogether).

Activity paused: the onPause() method

- The onPause() method should execute quickly. Don't use onPause() for CPU-intensive operations such as writing persistent data to a database.
- The app may still be visible on screen as it passes through the paused state, and any delays in executing onPause() can slow the user's transition to the next Activity.
- Implement any heavy-load operations when the app is in the stopped state instead.
- Activity can move from the paused state into the resumed state (if the user returns to the Activity) or to the stopped state (if the user leaves the Activity altogether)

```
@Override
protected void onPause() {
    super.onPause();
    // Another activity is taking focus
    // (this activity is about to be "paused").
}
```

Activity stopped: the onStop() method

- An Activity is in the stopped state when it's no longer visible on the screen
- This is usually because the user started another activity or returned to the home screen
- The Android system retains the activity instance in the back stack, and if the user returns to the activity, the system restarts it
- If resources are low, the system might kill a stopped activity altogether
- The system calls the onStop() method when the activity stops

Activity stopped: the onStop() method

- Implement the onStop() method to save persistent data and release resources that you didn't already release in onPause(), including operations that may have been too heavyweight for onPause()

```
@Override
protected void onStop() {
    super.onStop();
    // The activity is no longer visible (it is now "stopped")
}
```

Activity destroyed: the onDestroy() method

- When your Activity is destroyed it is shut down completely, and the Activity instance is reclaimed by the system.
- This can happen in several cases:
  - You call finish() in your Activity to manually shut it down.
  - The user navigates back to the previous Activity.
  - The device is in a low memory situation where the system reclaims any stopped Activity to free more resources.

Activity destroyed: the onDestroy() method

- Use onDestroy() to fully clean up after your Activity so that no component (such as a thread) is running after the Activity is destroyed.
- There are situations where the system will simply kill the hosting process for the Activity without calling this method (or any others), so you should not rely on onDestroy() to save any required data or Activity state. Use onPause() or onStop() instead

```
@Override
protected void onDestroy() {
    super.onDestroy();
    // The activity is about to be destroyed.
}
```

Activity restarted: the onRestart() method

- The restarted state is a transient state that only occurs if a stopped Activity is started again.
- The onRestart() method is called between onStop() and onStart().
- If you have resources that need to be stopped or started you typically implement that behavior in onStop() or onStart() rather than onRestart()

```
@Override
protected void onRestart() {
    super.onRestart();
    // The activity is about to be restarted.
}
```

Intent

Intent

- Intent is a messaging object used to request an action from another app component
- It facilitates communication between different components in several ways
- Starting an activity:
- We can start a new instance of an Activity by passing an Intent to startActivity()
- In addition to starting an activity, an intent can also be used to pass data between one activity and another.
- When you create an intent to start a new activity, you can include information about the data you want that new activity to operate on

- Starting a service:
- A Service is a component that performs operations in the background without a user interface
- We can start a service to perform a one-time operation such as downloading a file by passing an Intent to startService()
- The Intent describes the service to start and carries any necessary data

- Delivering a broadcast
- A broadcast is a message that any app can receive
- The system delivers various broadcasts for system events, such as when the system boots up or the device starts charging.
- We can deliver a broadcast to other apps by passing an Intent to sendBroadcast() or sendOrderedBroadcast()

Intent types

Explicit intents

- Explicit intents specify which application will satisfy the intent, by supplying either the target app's package name or a fully-qualified component class name.
- We use an explicit intent to start a component in our own app, because we know the class name of the activity or service we want to start
- For example, we may start a new activity within our app in response to a user action, or start a service to download a file in the background

Intent types

Implicit intents

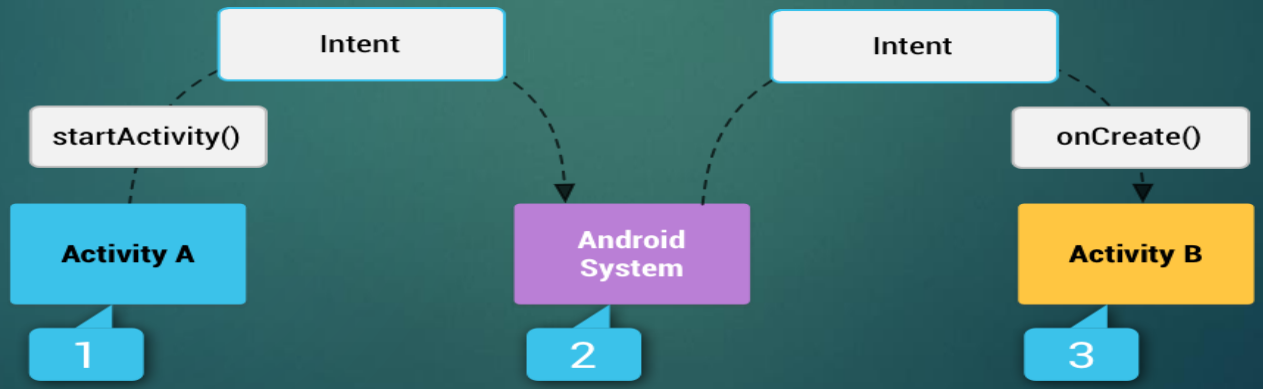
- They do not name a specific component, but instead declare a general action to perform, which allows a component from another app to handle it.
- For example, if you want to show the user a location on a map, you can use an implicit intent to request that another capable app show a specified location on a map.

Intent type

- Intent is used when starting an activity
- Use of Intent to start an activity:
- When the Intent object names a specific activity component explicitly, the system immediately starts that component.
- When we use implicit intent, the Android system finds the appropriate component to start by comparing the contents of the intent to the intent filters declared in the manifest file of other apps on the device.
- If the intent matches an intent filter, the system starts that component and delivers it the Intent object.
- If multiple intent filters are compatible, the system displays a dialog so the user can pick which app to use.

Intent type

- Implicit intent is delivered through the system to start another activity as follows:
- Activity A creates an Intent with an action description and passes it to startActivity().
- The Android System searches all apps for an intent filter that matches the intent. When a match is found,
- The system starts the matching activity (Activity B) by invoking its onCreate() method and passing it the Intent



Intent filter

- An intent filter is an expression in an app's manifest file that specifies the type of intents that the component would like to receive.
- For instance, by declaring an intent filter for an activity, you make it possible for other apps to directly start your activity with a certain kind of intent.
- Likewise, if you do not declare any intent filters for an activity, then it can be started only with an explicit intent..

Starting an Activity with an explicit Intent

- To start a specific Activity from another Activity, use an explicit Intent and the startActivity() method
- An explicit Intent includes the fully qualified class name for the Activity or other component in the Intent object
- Example: to start the ShowMessageActivity:

```
Intent messageIntent = new Intent(this, ShowMessageActivity.class);
startActivity(messageIntent);
```

Starting an Activity with an explicit Intent

- The intent constructor takes two arguments for an explicit Intent:

- An application context. In this example, the Activity class provides the context (this)
- The specific component to start (ShowMessageActivity.class)

- Use the startActivity() method with the new Intent object as the only argument
- The startActivity() method sends the Intent to the Android system, which launches the ShowMessageActivity class on behalf of your app
- The started Activity remains on the screen until the user taps the Back button on the device, at which time that Activity closes and is reclaimed by the system, and the originating Activity is resumed

Implicit Intent

- An implicit intent specifies an action that can invoke any app on the device able to perform the action.
- Using an implicit intent is useful when your app cannot perform the action, but other apps probably can and you'd like the user to pick which app to use
- For example, if you have content that you want the user to share with other people, create an intent with the ACTION_SEND action and add extras that specify the content to share.
- When you call startActivity() with that intent, the user can pick an app through which to share the content.

Implicit Intent

```
// Create the text message with a string.
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
sendIntent.setType("text/plain");

// Try to invoke the intent.
try {
    startActivity(sendIntent);
} catch (ActivityNotFoundException e) {
    // Define what your app should do if no activity can handle the intent.
}
```

Implicit Intent

- When startActivity() is called, the system examines all of the installed apps to determine which ones can handle this kind of intent (an intent with the ACTION_SEND action and that carries "text/plain" data).
- If there's only one app that can handle it, that app opens immediately and is given the intent.
- If no other apps can handle it, your app can catch the ActivityNotFoundException that occurs.
- If multiple activities accept the intent, the system displays a dialog so the user can pick which app to use.