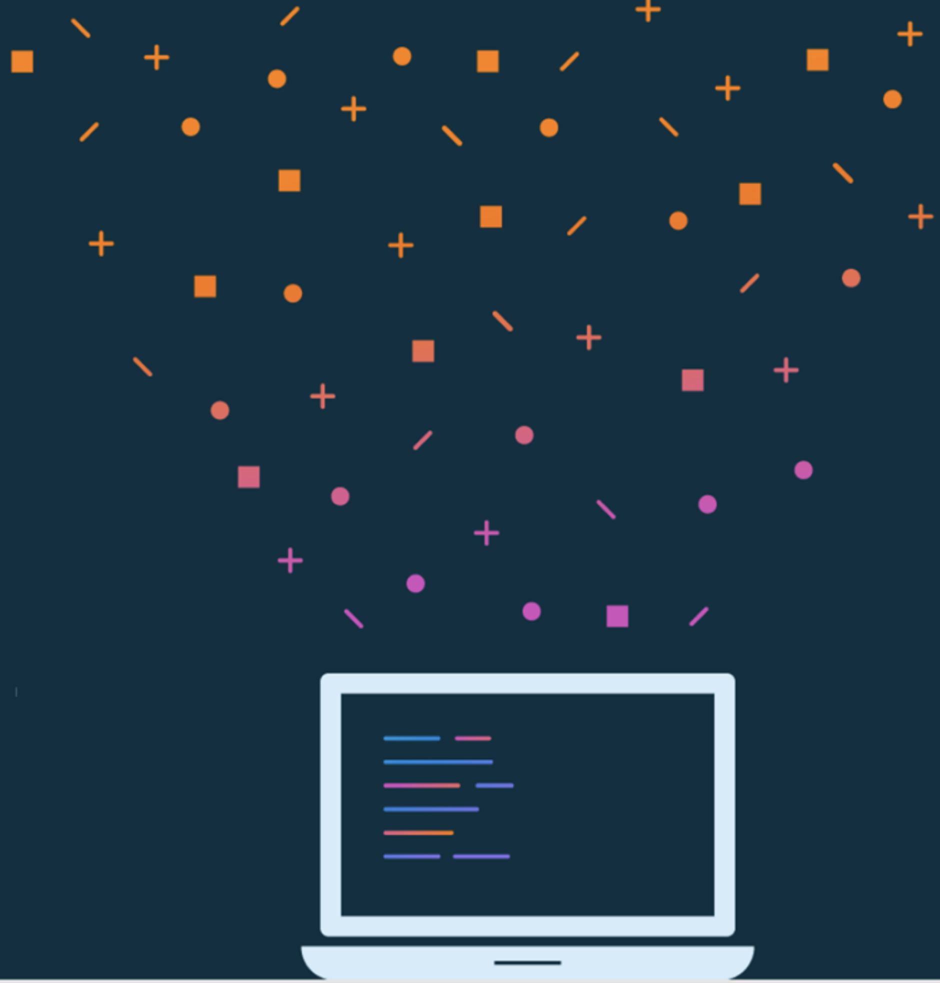
Android Development with Kotlin

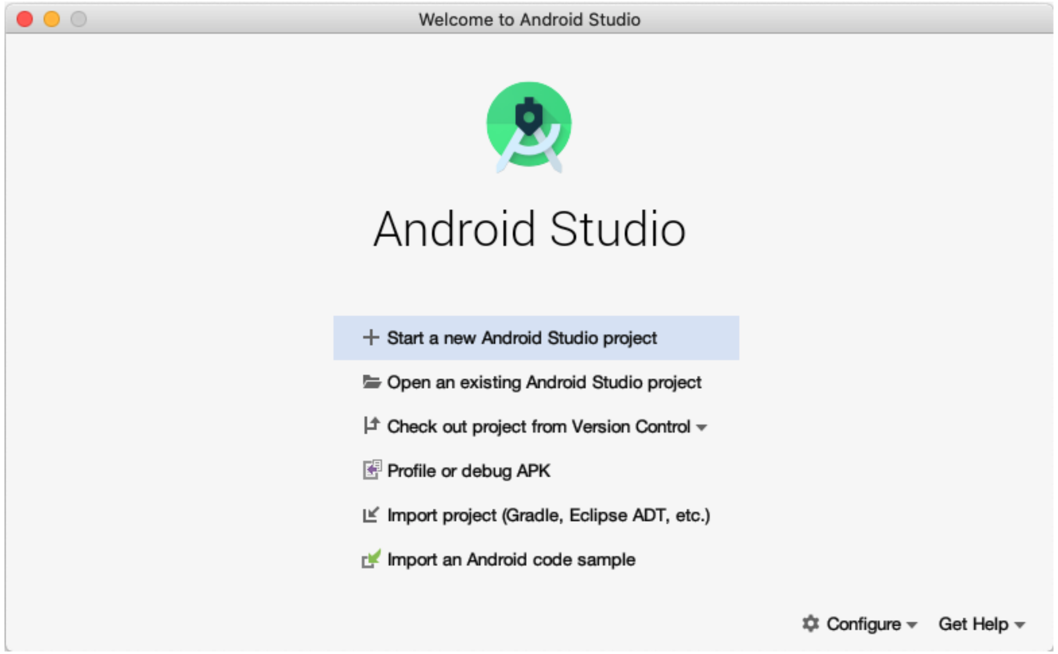


Your first app

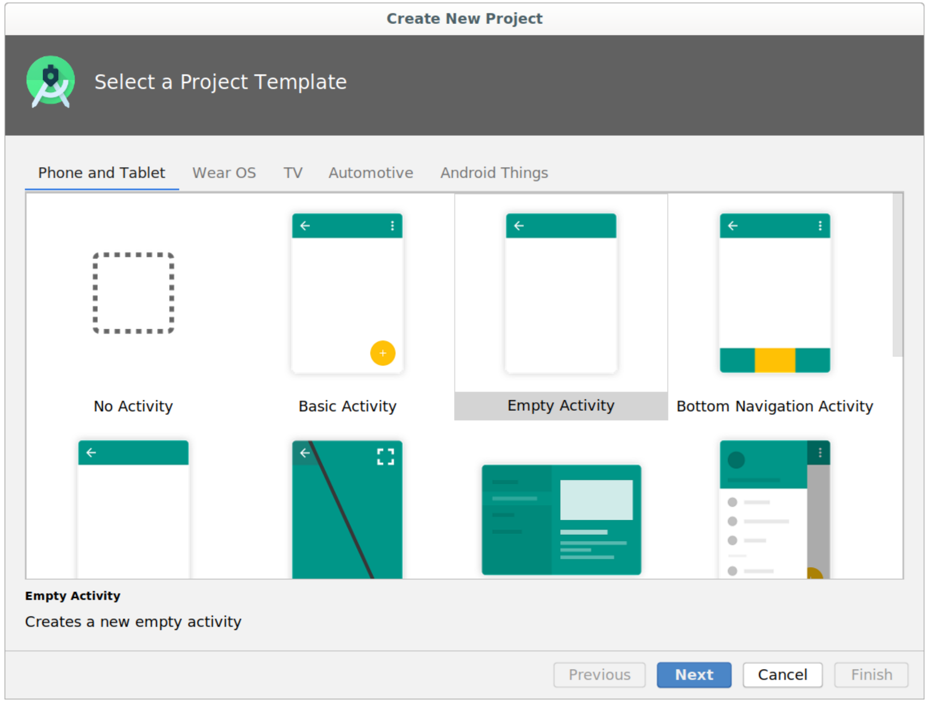
Development Tool

- [Download Android Studio]
- [Install Android Studio]
- [Meet Android Studio]

Open Android Studio


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

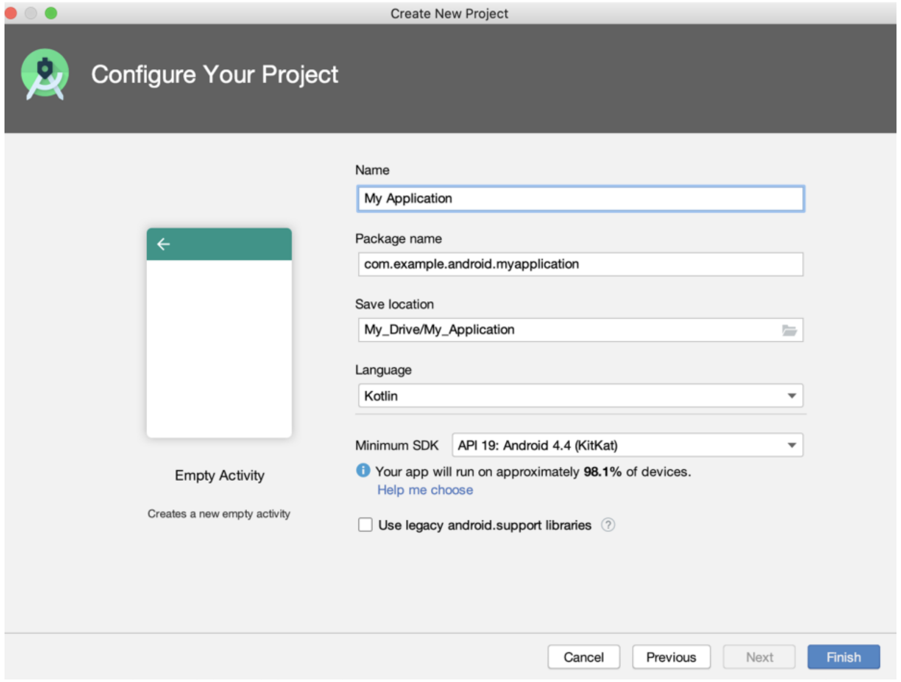
Create new project



Empty Activity

Creates a new empty activity

Enter your project details


Create New Project

Configure Your Project

Name: My Application

Package name: com.example.android.myapplication

Save location: My_Drive/My_Application

Language: Kotlin

Minimum SDK: API 19: Android 4.4 (KitKat)

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

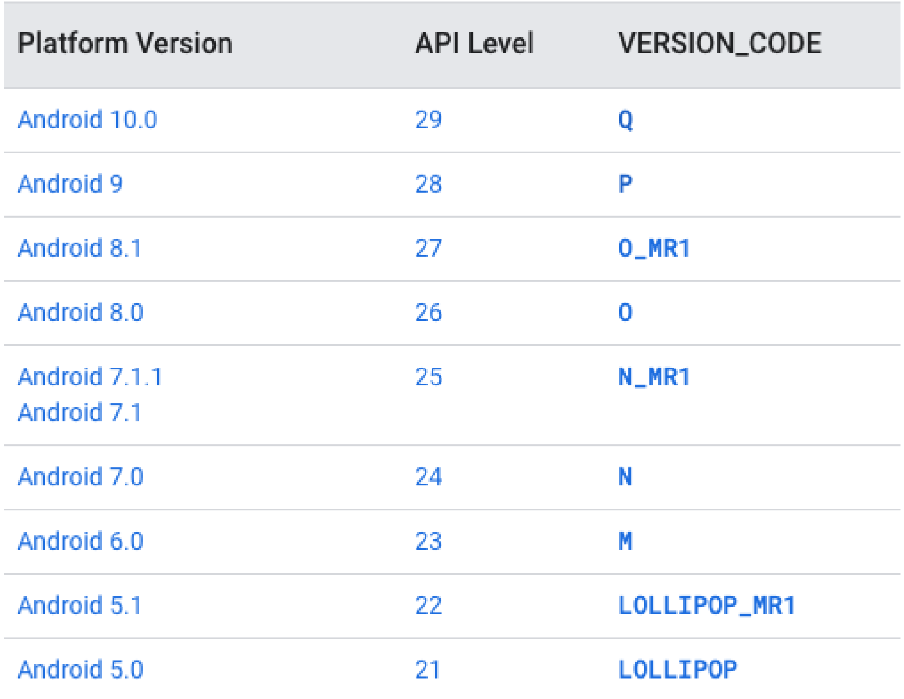
Help me choose

☐ Use legacy android.support libraries

Empty Activity

Creates a new empty activity

Android releases and API levels



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

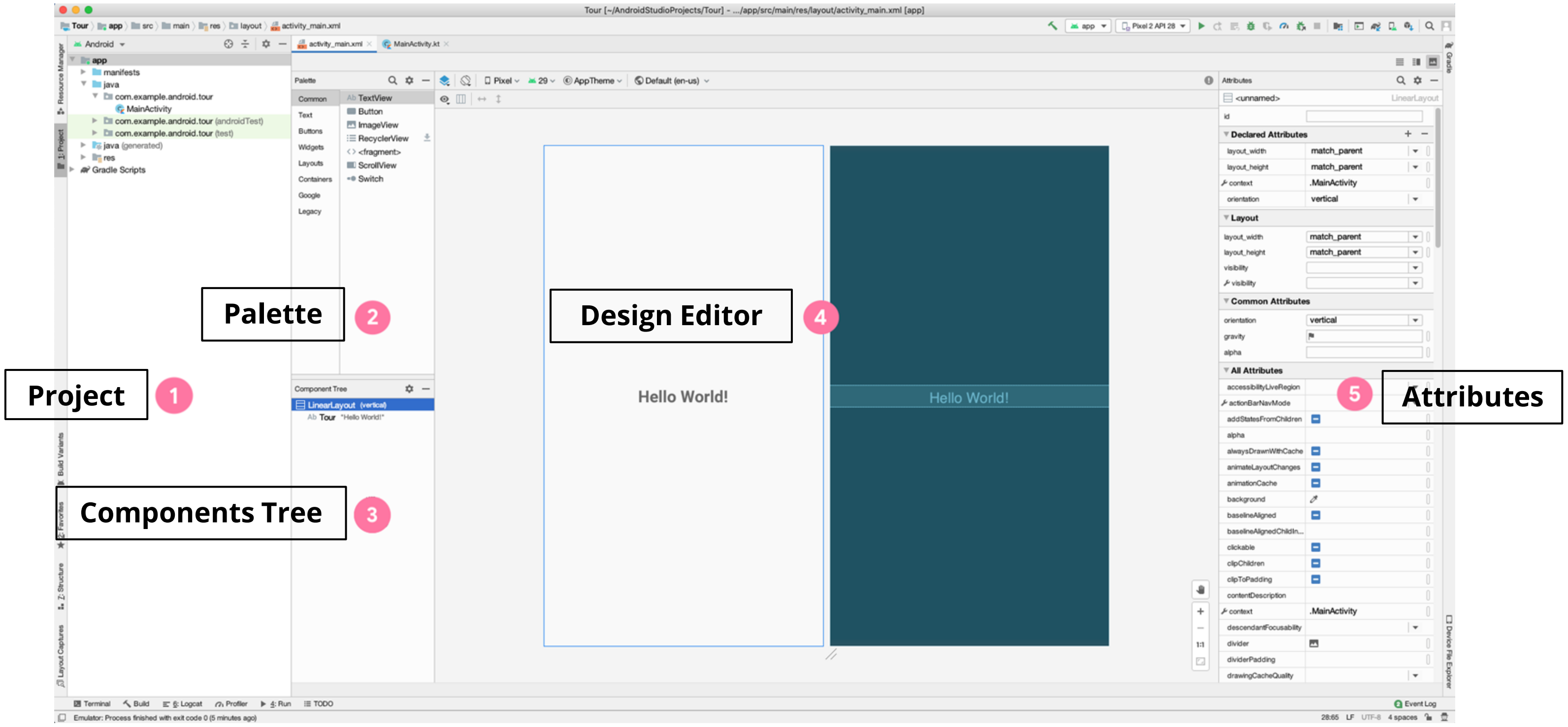
Choose API levels for your app

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

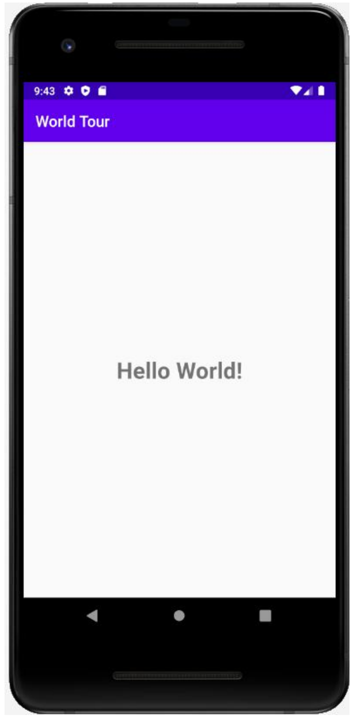
```
minSdkVersion <= targetSdkVersion <= compileSdkVersion
```

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

Tour of Android Studio

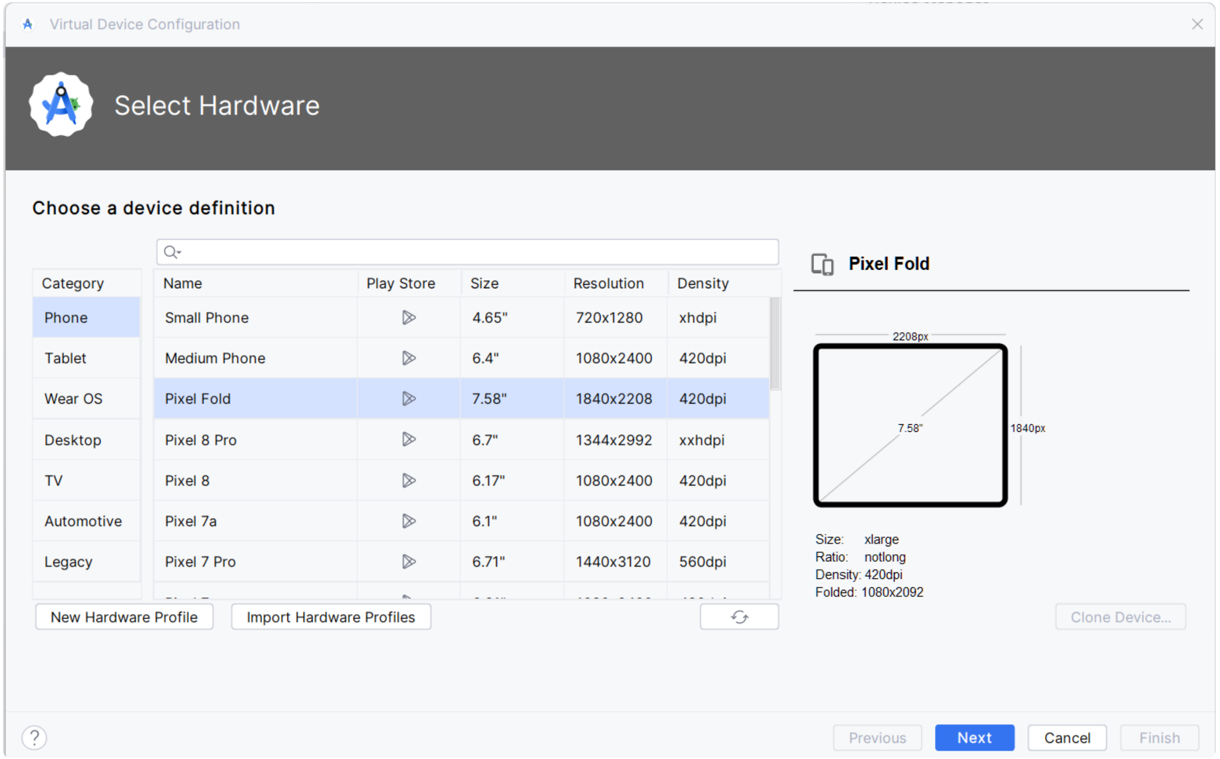


Run your app



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

Android Virtual Device (AVD) Manager



Virtual Device Configuration

Select Hardware

Choose a device definition

Category	Name	Play Store	Size	Resolution	Density
Phone	Small Phone	▷	4.65"	720x1280	xhdpi
Tablet	Medium Phone	▷	6.4"	1080x2400	420dpi
Wear OS	Pixel Fold	▷	7.58"	1840x2208	420dpi
Desktop	Pixel 8 Pro	▷	6.7"	1344x2992	xxhdpi
TV	Pixel 8	▷	6.17"	1080x2400	420dpi
Automotive	Pixel 7a	▷	6.1"	1080x2400	420dpi
Legacy	Pixel 7 Pro	▷	6.71"	1440x3120	560dpi

Size: xlarge
Ratio: notlong
Density: 420dpi
Folded: 1080x2092

Anatomy of an Android App project

Applications Building Blocks

- **Activity:** UI component, especially for one screen.
- **Broadcast Receiver (Intent Receiver):**
  Responds to notifications or status changes that can wake up your process.
- **Service:** A faceless task can run in the background.
- **Content Provider:** Enable applications to share data.

Anatomy of an Android Application

- **Resources:**
  layouts, images, strings, colors, and themes as XML and media files

- **Components:**
  activities, services, and helper classes as Java code

- **Manifest:** information about the app for the runtime

- **Build configuration:** APK versions in Gradle config files.

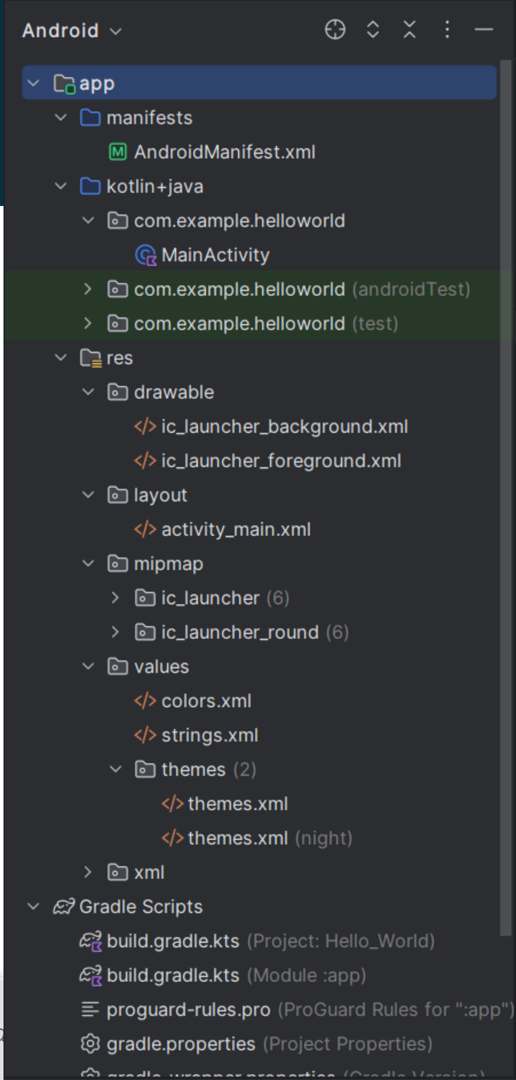
Android app project structure

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

Browse files in Android Studio

- **AndroidManifest.xml:**
  It is the manifest file
  for your Android application,
  where the application components
  must be registered,
  and the needed permissions listed.

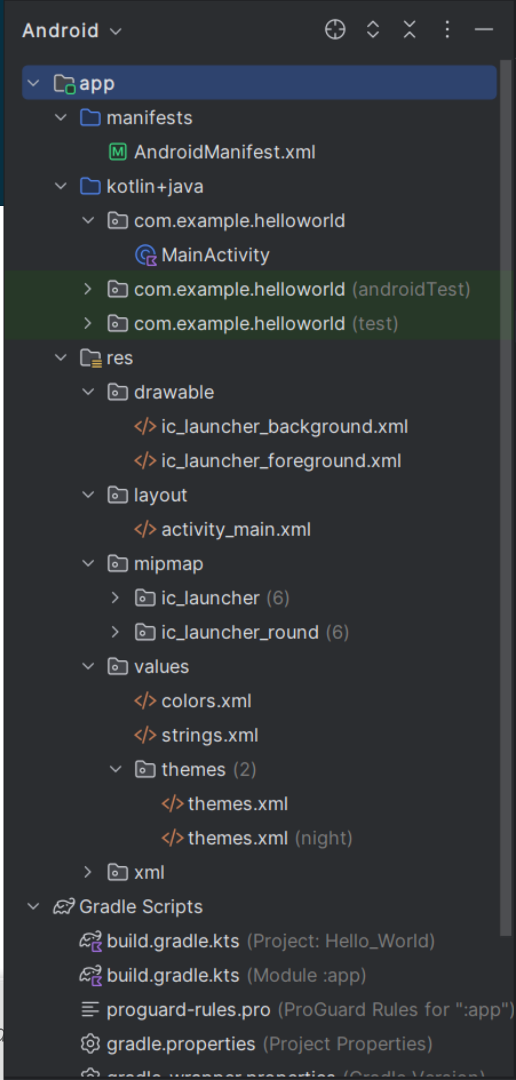
- **kotlin+Java:**
  Contains the .kt source files for the project.



```
Android
app
  manifests
    AndroidManifest.xml
  kotlin+java
    com.example.helloworld
      MainActivity
    com.example.helloworld (androidTest)
    com.example.helloworld (test)
  res
    drawable
      ic_launcher_background.xml
      ic_launcher_foreground.xml
    layout
      activity_main.xml
    mipmap
      ic_launcher (6)
      ic_launcher_round (6)
    values
      colors.xml
      strings.xml
      themes (2)
        themes.xml
        themes.xml (night)
    xml
Gradle Scripts
  build.gradle.kts (Project: Hello_World)
  build.gradle.kts (Module :app)
  proguard-rules.pro (ProGuard Rules for ":app")
  gradle.properties (Project Properties)
```

Browse files in Android Studio

- **res:**
  This folder contains all the resources used in your application.
  - **drawable:**
    A drawable resource is a general concept for a graphic that can be drawn to the screen.
  - **layout:** The XML layout files.
  - **mipmap:**
    The Launcher icon with different densities.
  - **values:**
    colors, strings, and styles used in the application.

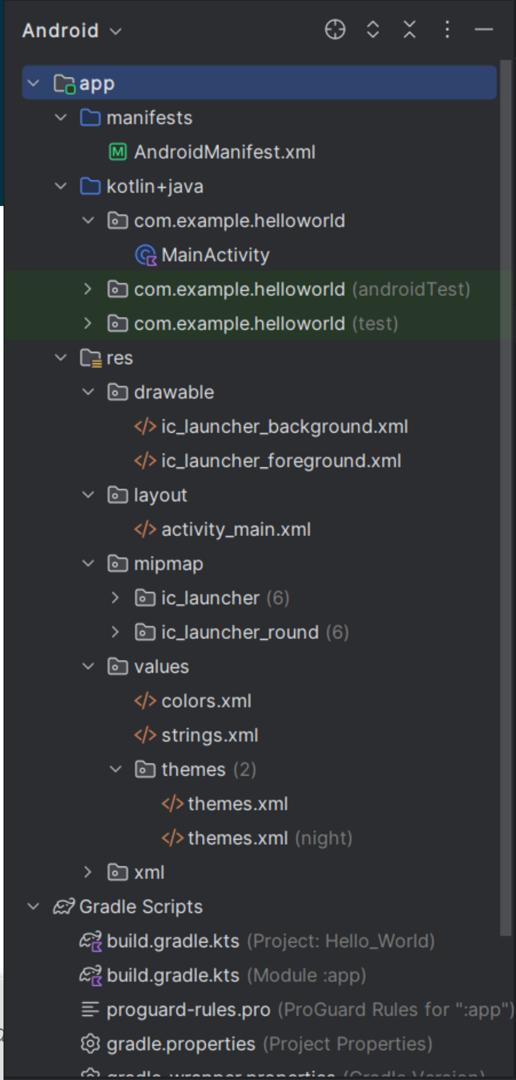


```
app
  manifests
    AndroidManifest.xml
  kotlin+java
    com.example.helloworld
      MainActivity
    com.example.helloworld (androidTest)
    com.example.helloworld (test)
  res
    drawable
      ic_launcher_background.xml
      ic_launcher_foreground.xml
    layout
      activity_main.xml
    mipmap
      ic_launcher (6)
      ic_launcher_round (6)
    values
      colors.xml
      strings.xml
      themes (2)
        themes.xml
        themes.xml (night)
    xml
Gradle Scripts
  build.gradle.kts (Project: Hello_World)
  build.gradle.kts (Module :app)
  proguard-rules.pro (ProGuard Rules for ":app")
  gradle.properties (Project Properties)
```

Browse files in Android Studio

- **Gradle Scripts:**
  Gradle is a custom build tool
  used to build **Android** packages (APK files)
  by managing dependencies
  and providing custom build logic.

- **APK** file (**A**ndroid **A**pplication **P**ackage)
  is a specially formatted zip file that contains
  Byte code, Resources(images, UI, XML, etc.)



```
Android
app
  manifests
    AndroidManifest.xml
  kotlin+java
    com.example.helloworld
      MainActivity
    com.example.helloworld (androidTest)
    com.example.helloworld (test)
  res
    drawable
      ic_launcher_background.xml
      ic_launcher_foreground.xml
    layout
      activity_main.xml
    mipmap
      ic_launcher (6)
      ic_launcher_round (6)
    values
      colors.xml
      strings.xml
      themes (2)
        themes.xml
        themes.xml (night)
    xml
Gradle Scripts
  build.gradle.kts (Project: Hello_World)
  build.gradle.kts (Module :app)
  proguard-rules.pro (ProGuard Rules for ":app")
  gradle.properties (Project Properties)
```

Manifest File

- Where the application components must be declared:
  - ❑ <activity> elements for activities
  - ❑ <service> elements for services
  - ❑ <receiver> elements for broadcast receivers
  - ❑ <provider> elements for content providers

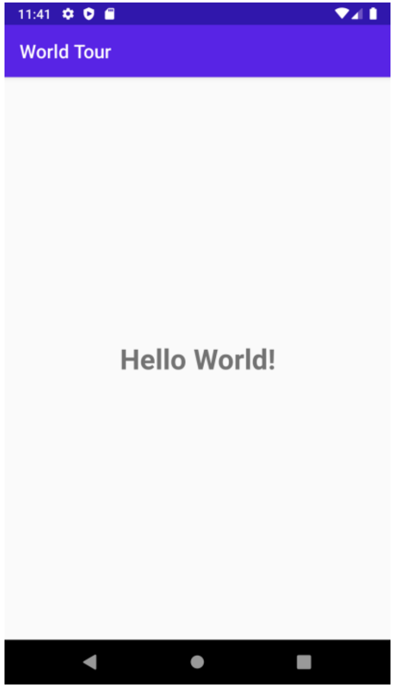
Manifest File Cont'd

- You can declare an intent filter for your component by adding an `<intent-filter>` element as a child of the component's declaration element.

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

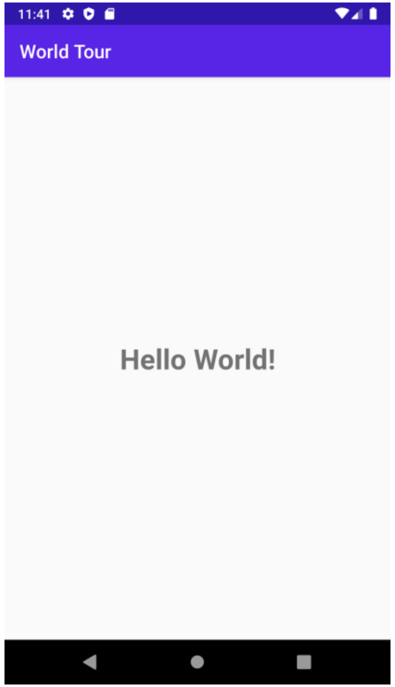
Activities

What's an Activity?



- An Activity is a means for the user
  to accomplish one main goal.
- Represents a single screen with a user interface.
- An Android app can have zero or more activities.
- When a new activity starts,
  it is pushed onto the back stack and takes user focus.

What’s an Activity Cont’d?



- The back stack abides by the basic "last in, first out" queue mechanism, so, when the user is done with the current activity and presses the BACK key, it is popped from the stack (and destroyed) and the previous activity resumes.

What does an Activity do?

- Represents an activity,
  such as ordering groceries, sending emails, or getting directions.
- Handles user interactions,
  such as button clicks, text entries, or login verification.
- Can start other activities in the same or other apps.
- Has a lifecycle
  created, started, runs, paused, resumed, stopped, and destroyed.

Implement new activity (MainActivity.kt)

1. Define layout in XML
2. Define `Activity` Kotlin/Java class
    - you must create a subclass of Activity (or an existing subclass of it)

```
class MainActivity: AppCompatActivity()
```

Implement new activity (MainActivity.kt)

3. Connect Activity with Layout
    - Set content view in `onCreate()`

      implement callback methods that the system calls when the activity

      transitions between various states of its lifecycle,

      such as when the activity is being created, stopped, resumed, or destroyed.
4. Declare `Activity` in the Android manifest

1. Define layout in XML

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

2. Define Activity class

```
class MainActivity : AppCompatActivity() {

   override fun onCreate(savedInstanceState: Bundle?) {
       super.onCreate(savedInstanceState)
   }
}
```

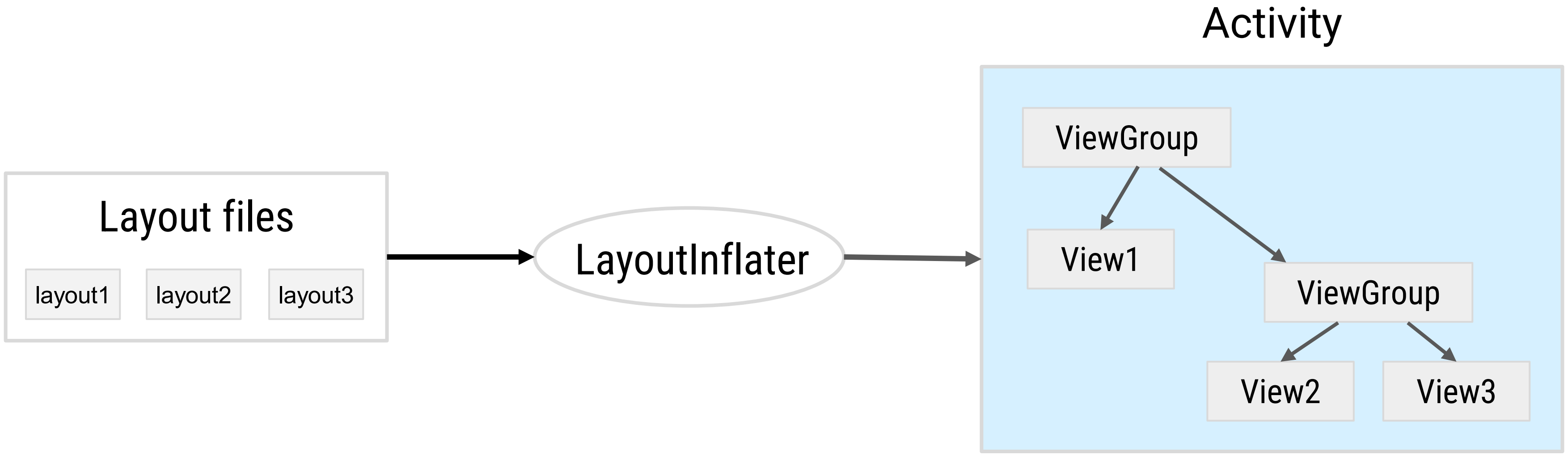
3. Connect Activity with Layout

```
class MainActivity : AppCompatActivity() {

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

}
```

Resource — is layout — in this XML file

- Called when the system creates your Activity
- Call setContentView() to define the layout for the activity's user interface.

Layout inflation

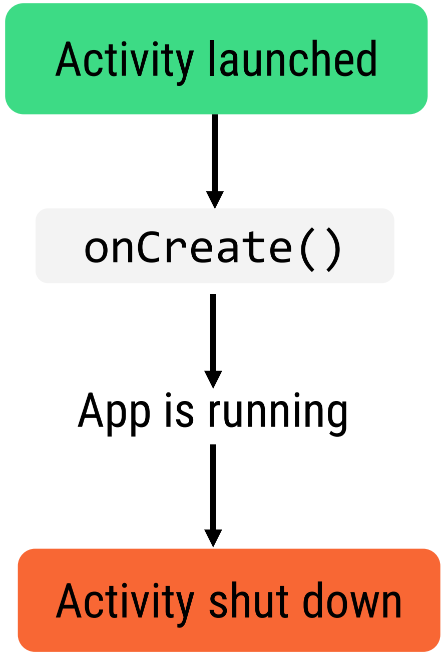


4. Declare activity in Android Manifest

- MainActivity needs to include an intent filter to start from the launcher.

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

How an Activity runs



Activity lifecycle

What is the Activity Lifecycle?

- The set of states an Activity can be in during its lifetime, from when it is created until it is destroyed

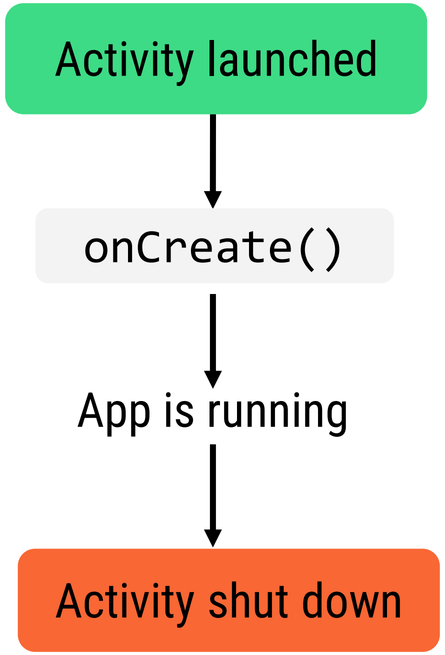
More formally:

A directed graph of all the states an Activity can be in, and the callbacks associated with transitioning from each state to the next one

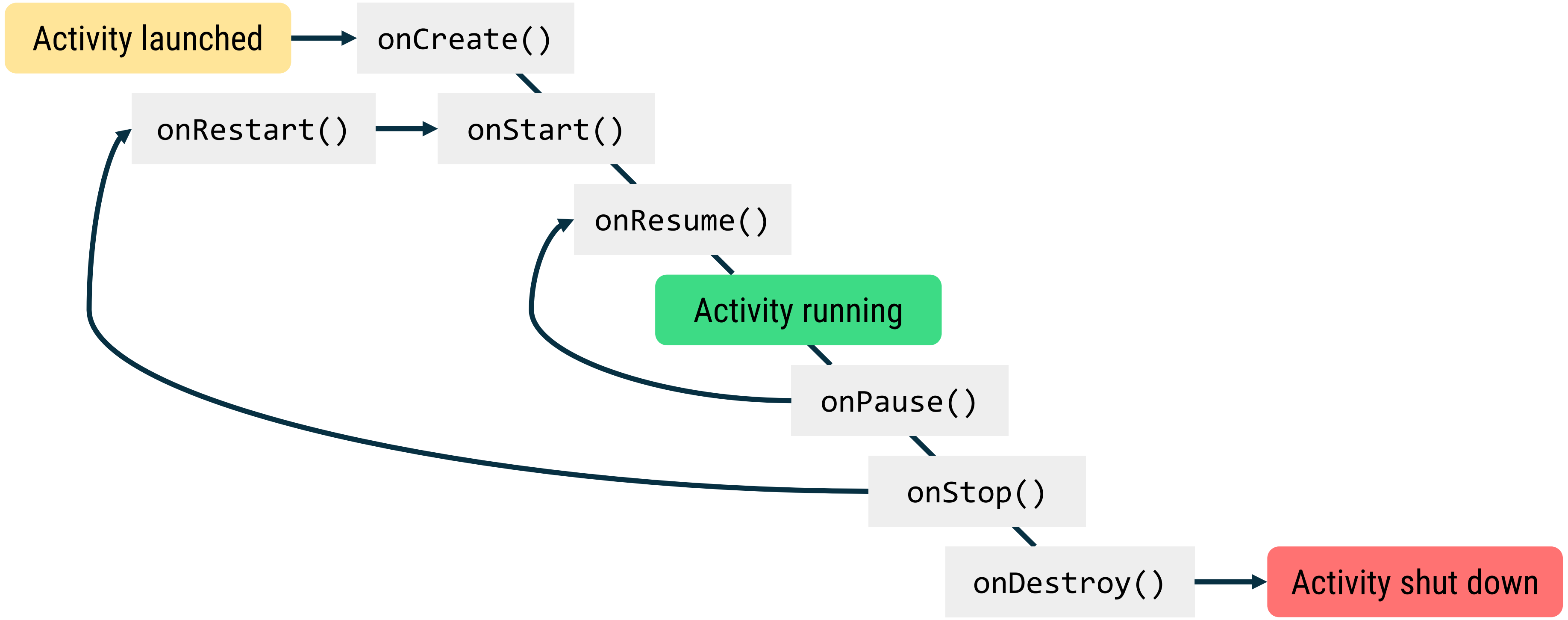
Why it matters?

- Preserve user data and state if:
    - User temporarily leaves app and then returns
    - User is interrupted (for example, a phone call)
    - User rotates device
- Avoid memory leaks and app crashes.

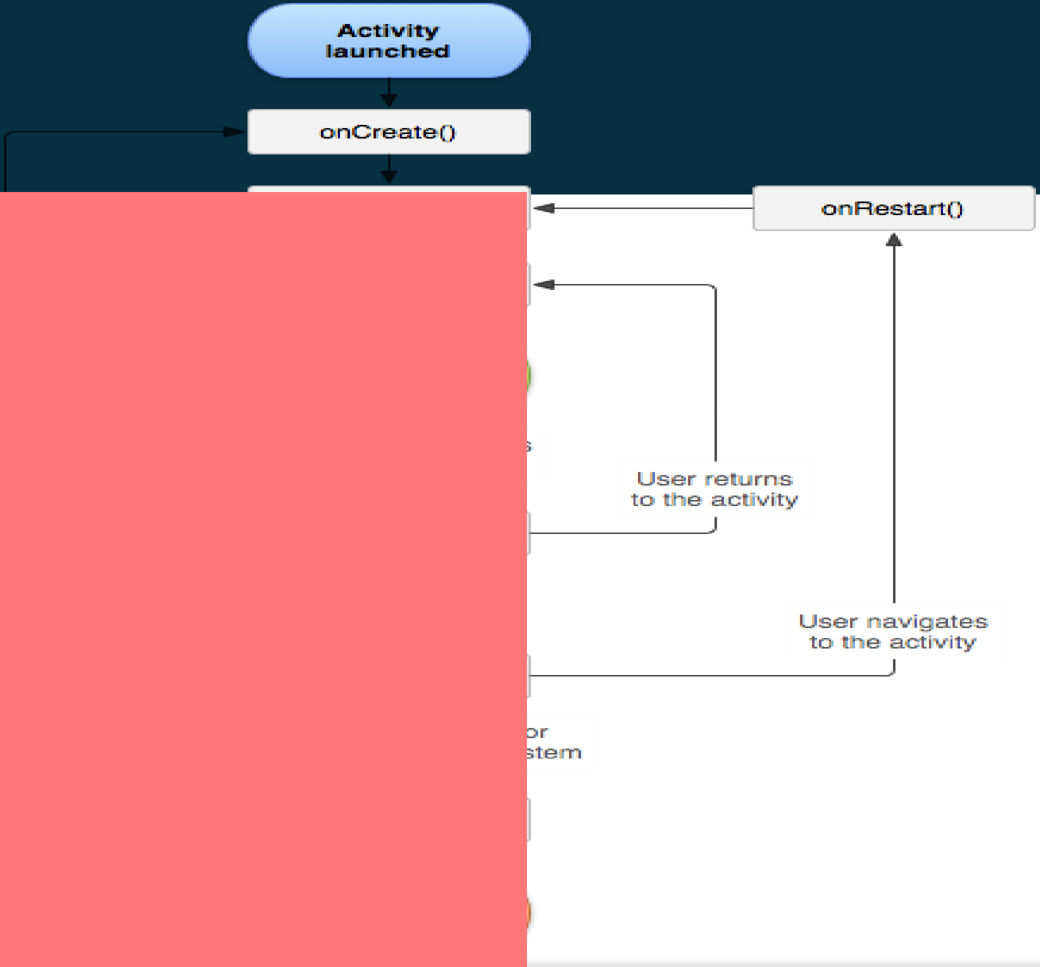
Simplified activity lifecycle



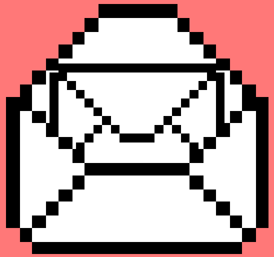
Activity lifecycle



Activity Lifecycle

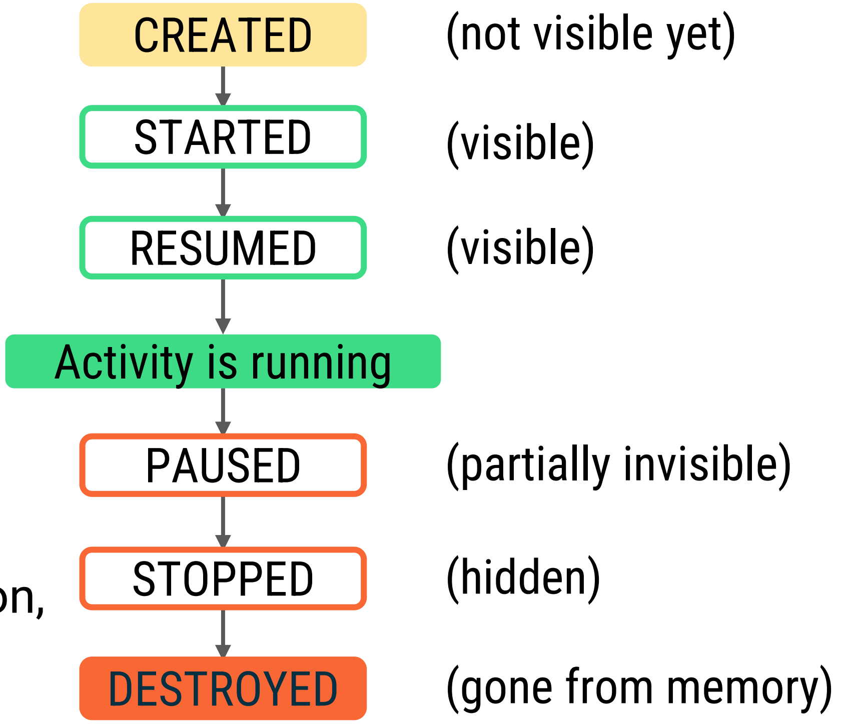


Activity State Restoring



Instance States

Activity states



State changes are triggered by
- user action,
- configuration changes such as device rotation,
- or system action.

Callbacks and when they are called

`onCreate(Bundle savedInstanceState)` - static initialization

`onStart()` - when Activity (screen) is becoming visible

`onRestart()` - called if Activity was stopped (calls onStart())

`onResume()` - start to interact with the user

`onPause()` - about to resume PREVIOUS Activity

`onStop()` - no longer visible, but still exists and all state info preserved

`onDestroy()` - final call before the Android system destroys Activity

Implementing and overriding callbacks

- Only onCreate() is required.
- Override the other callbacks
  to change the default behavior

onCreate()

- Activity is created and other initialization work occurs
- You must implement this callback
- Inflate activity UI and perform other app startup logic

onStart()

- Activity becomes visible to the user
- Called after activity:
    - `onCreate()`

      or
    - `onRestart()` if activity was previously stopped

onResume()

- Activity gains input focus:
    - User can interact with the activity
- Activity stays in resumed state until system triggers activity to be paused

onPause()

- Activity has lost focus (not in foreground)
- Activity is still visible, but user is not actively interacting with it
- Counterpart to `onResume()`

onStop()

- Activity is no longer visible to the user
- Release resources that aren’t needed anymore
- Save any persistent state that the user is in the process of editing so they don’t lose their work

onDestroy()

- Activity is about to be destroyed, which can be caused by:
    - Activity has finished or been dismissed by the user
    - Configuration change
- Perform any final cleanup of resources.
- Don’t rely on this method to save user data (do that earlier)

Summary of activity states

State	Callbacks	Description
Created	`onCreate()`	Activity is being initialized.
Started	`onStart()`	Activity is visible to the user.
Resumed	`onResume()`	Activity has input focus.
Paused	`onPause()`	Activity does not have input focus.
Stopped	`onStop()`	Activity is no longer visible.
Destroyed	`onDestroy()`	Activity is destroyed.

Save state

User expects UI state to stay the same after a config change or if the app is terminated when in the background.

- Activity is destroyed and restarted, or app is terminated and activity is started.
- Store user data needed to reconstruct app and activity Lifecycle changes:
    - Use `Bundle` provided by `onSaveInstanceState()`.
    - `onCreate()` receives the `Bundle` as an argument when activity is created again.

When does config change?

Configuration changes invalidate the current layout or other resources in your activity when the user:

- Rotates the device
- Chooses different system language, so locale changes
- Enters multi-window mode (from Android 7)

What happens on configs change?

On configuration change, Android:

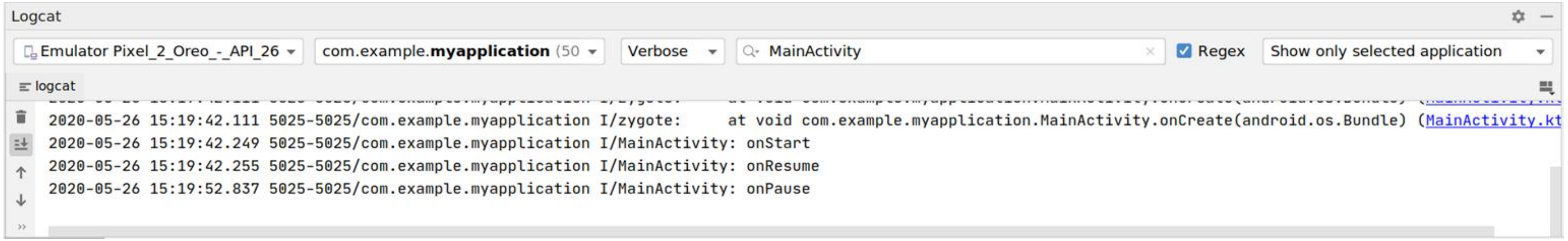
1. Shuts down Activity by calling:
   - onPause()
   - onStop()
   - onDestroy()

2. Starts Activity over again by calling:
   - onCreate()
   - onStart()
   - onResume()

Logging

Logging in Android

- Monitor the flow of events or state of your app.
- Use the built-in `Log` class or third-party library.
- Example `Log` method call: `Log.d(TAG, "Message")`


Logcat
Emulator Pixel_2_Oreo_-_API_26
com.example.myapplication (50
Verbose
MainActivity
Regex
Show only selected application
logcat

```
2020-05-26 15:19:42.111 5025-5025/com.example.myapplication I/zygote:     at void com.example.myapplication.MainActivity.onCreate(android.os.Bundle) (MainActivity.kt
2020-05-26 15:19:42.249 5025-5025/com.example.myapplication I/MainActivity: onStart
2020-05-26 15:19:42.255 5025-5025/com.example.myapplication I/MainActivity: onResume
2020-05-26 15:19:52.837 5025-5025/com.example.myapplication I/MainActivity: onPause
```

Adding logging to your app

- As the app runs, the **Logcat** pane shows information
- Add logging statements to your app that will show up in the Logcat pane
- Set filters in the **Logcat** pane to see what's important to you
- Search using tags

Logging statement

```
import android.util.Log

// use the class name as a TAG
private val TAG = MainActivity::class.java.simpleName

// show message in Android Monitor, logcat pane
// Log.<log-level>(TAG, "message")
Log.d(TAG, "onCreate(): ")
```

Log leveles

- **Verbose:** All verbose log statements and comprehensive system
- **Debug:** All debug logs, variable values, and debugging notes
- **Info:** Status info, such as database connection
- **Warning:** Unexpected behavior, non-fatal issues
- **Error:** Serious error conditions, exceptions, crashes only

Write logs

Priority level	**Log method**
Verbose	`Log.v(String, String)`
Debug	`Log.d(String, String)`
Info	`Log.i(String, String)`
Warning	`Log.w(String, String)`
Error	`Log.e(String, String)`

Demo

Assignment

Create Hello World Android Project and check the activity lifecycle.