SENG 440

Week 2: Activities and Layouts

Ben Adams, benjamin.adams@canterbury.ac.nz

Android App **Architecture**

- Made up of various **app components** defined in your App **manifest** file:
  - Activities
  - Fragments
  - Services
  - Content providers
  - Broadcast receivers

Activities and Fragments

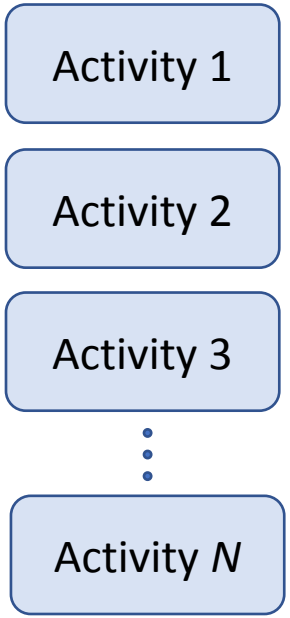
- Most fundamental component of the Android application model
- An **`Activity`** ~= a screen-ful of user interaction
- **Entry point** for user interaction into an app
  - “When one app invokes another, the calling app invokes an activity in the other app, rather than the app as an atomic whole.”
  - Provides window where app UI will be drawn
- You subclass `Activity` or a specialized type such as `AppCompatActivity`, `ComponentActivity`
- A **Fragment** is a re-usable piece of UI, must be owned by Activity

Other components (covered in detail later on)

- **Services**
  - Application component that performs long-running operations in background with no UI
  - For example, an application that automatically responds to texts when driving
- **Content Providers**
  - A bridge between applications to share data
  - For example, the device's Contacts information
  - We tend to use these, but not create new ones
- **Broadcast Receivers**
  - Component that responds to system wide announcements
  - For example, battery low, screen off, date changed
  - Also possible to initiate broadcasts from within an application

Activity Stack

Most recently created is at Top



User currently interacting with this

Pressing Back or destroying Activity 1 will bring this to the top

If Activities above this use too many resources, it will be destroyed!

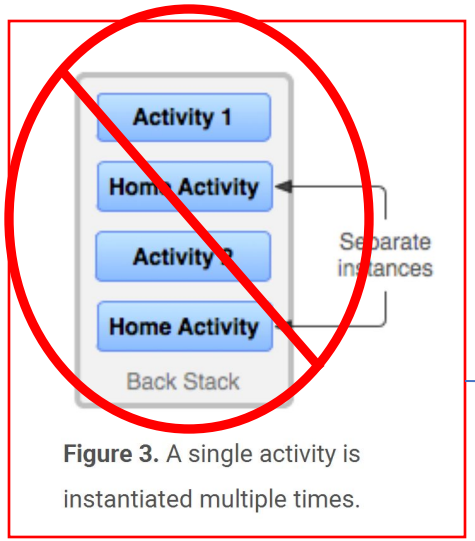
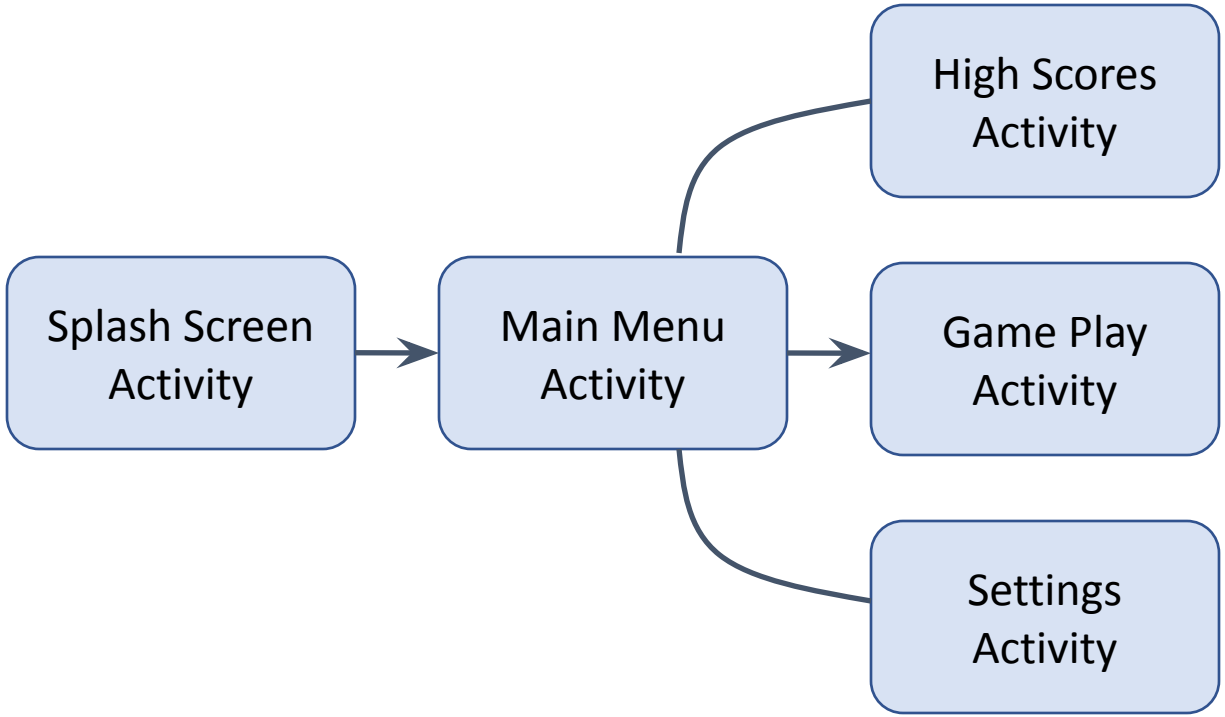


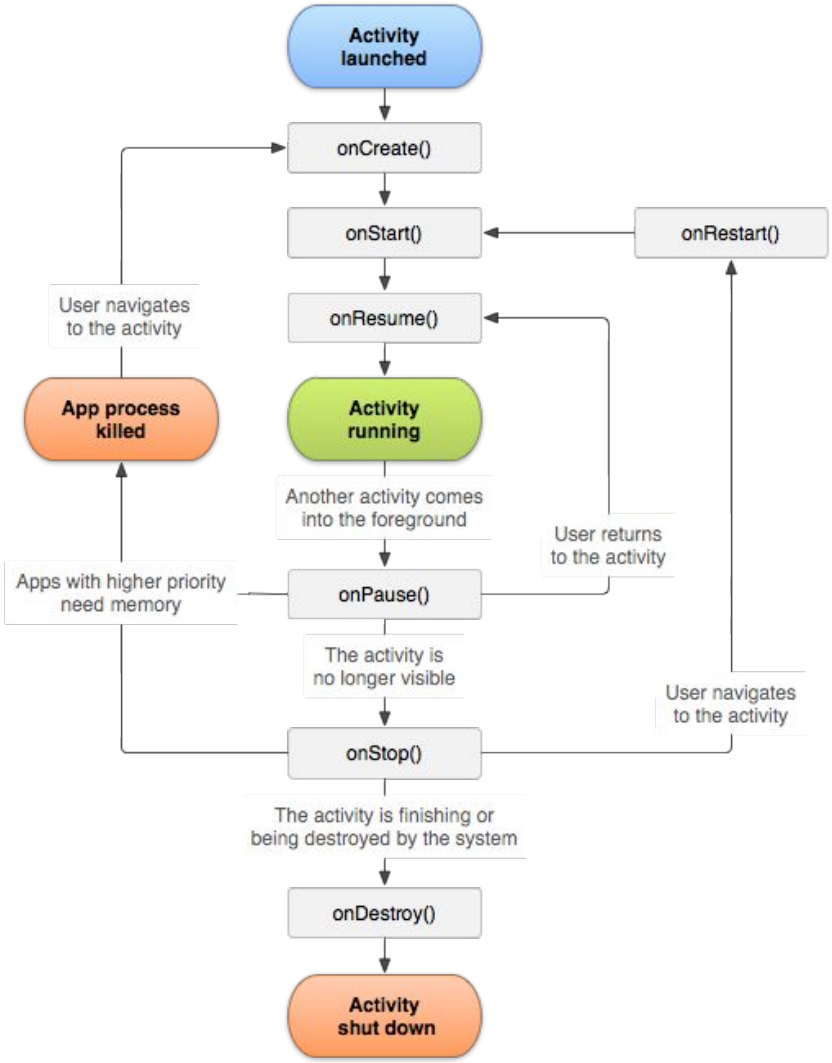
Figure 3. A single activity is instantiated multiple times.

Beware having multiple instance of the same activity on the stack

Typical Game



Conder & Darcey (2010), Fig 4.1, p. 74


Activity launched
onCreate()
onStart()
onRestart()
onResume()
User navigates to the activity
App process killed
Activity running
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

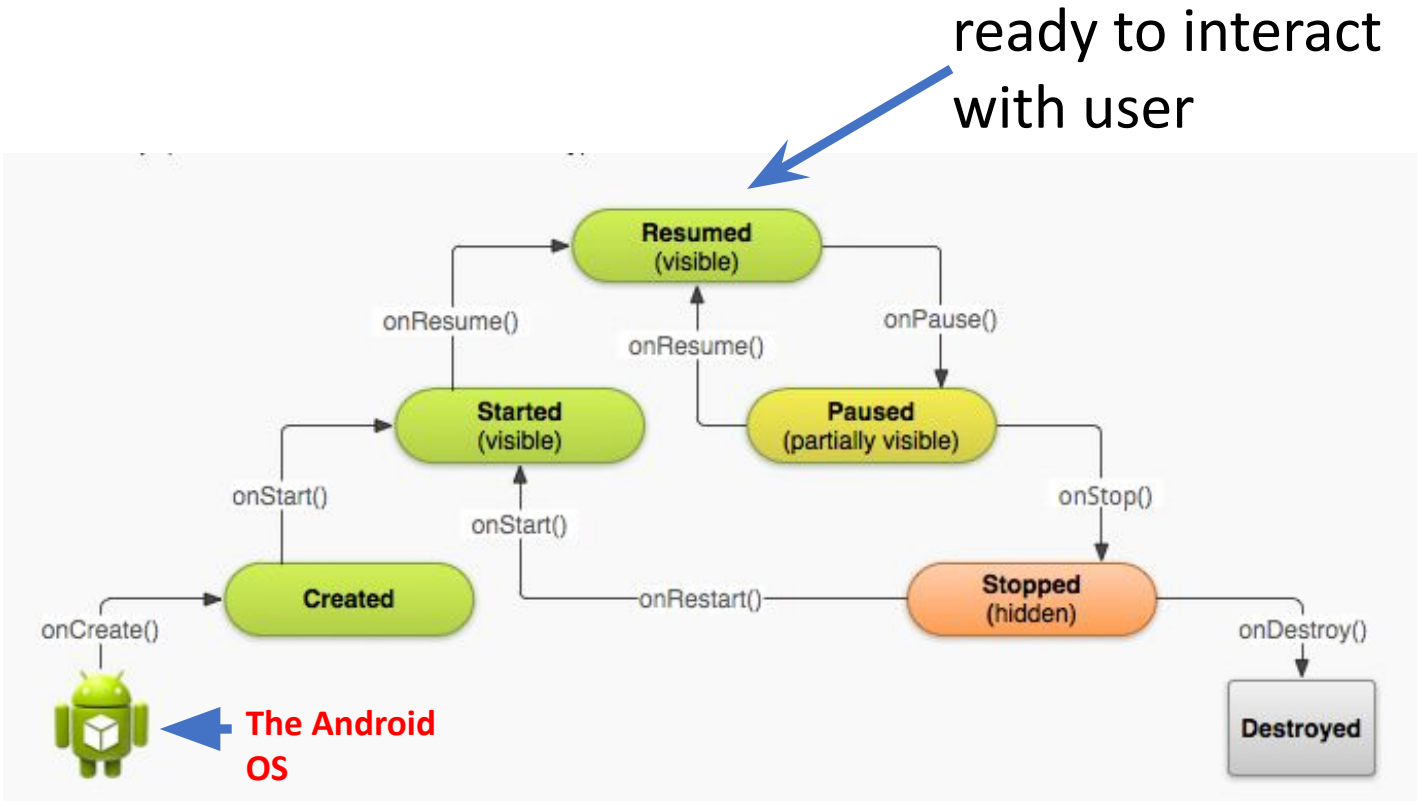
Starting Activities

- Android applications don't start with a call to

  fun main(args: Array<String>)

- A series of **callback methods** are invoked by the Android OS
- Each corresponds to specific stage of the Activity / application lifecycle:
  - `onCreate()`,`onStart()`,`onResume()`,`onPause()`,`onStop()`, `onRestart()`,`onDestroy()`
- Callback methods also used to tear down Activity / application

Activity Lifecycle



Understanding the Lifecycle

- Need to **overload callback methods** so your app behaves well:
  - App **should not crash** if the user receives a phone call or **switches to another app** while using your app.
  - App **should not consume valuable system resources** when the user is not actively using it.
  - App **should not lose the user's progress** if they leave your app and return to it at a later time.
  - App **should not crash** or lose the user's progress **when the screen rotates** between landscape and portrait orientation.

https://developer.android.com/guide/components/activities/activity-lifecycle

Primary States

- **Active**
  - Activity is in the foreground and user can interact with it
- **Paused**
  - Activity partially obscured by another activity and user cannot interact with it (e.g., when working with a menu or dialog)
- **Stopped**
  - Activity completely hidden and not visible to user. It is in the background.
  - Activity instance and variables are retained but no code is being executed by the activity
- Dead, activity terminated (or never started)
- Two other states, **Created** and **Started**, but they are transitory
  onCreate -> onStart -> onResume

AndroidManifest.xml

All Activities that are part of application must be registered in Manifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="scott.examples.lifeCycleTest"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="10" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".LifeCycleTestActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
                <activity
            android:name=".NameGetter"
            android:label="@string/getName"/>
    </application>
```

Specify Activity to start with

Purpose of **Lifecycle Phases**

- **Entire lifetime**: onCreate / onDestroy
  - Load UI
  - Could start and stop threads that should always be running
- **Visible lifetime**: onStart / onStop
  - Access or release resources that influence UI
  - Write info to files if necessary
- **Foreground lifetime**: onResume / onPause
  - Restore state and save state
  - Start and stop audio, video, animations

Pausing – onPause() callback

- When activity **paused** you should
  - **Stop** animations of other **CPU intensive tasks**
  - **Release resources** such as broadcast receivers (app stops listening for broadcast info) and handles to sensors such as GPS device or handles to the camera
  - **Stop audio and video**, if appropriate (bad UX to keep running)

Stopping - onStop() callback

- **Many scenarios** cause an Activity to be stopped:
    - User performs action in activity that starts another activity in the application
    - User opens Overview window and starts a new application
    - User receives phone call

- Well-behaved apps **save progress and restart seamlessly**

- Use onStop to **release all resources and save information** (persistence)
    - We will discuss data persistence in more detail in a later lecture

How to stop an Activity yourself?

- ***Generally, don't!***
- "**Note:** In most cases, you should not explicitly finish an activity using these methods… the Android system manages the life of an activity for you, so you do not need to finish your own activities. Calling these methods could adversely affect the expected user experience and should only be used when you absolutely do not want the user to return to this instance of the activity."
- methods: `finish()`, `finishActivity()`

Activity Destruction

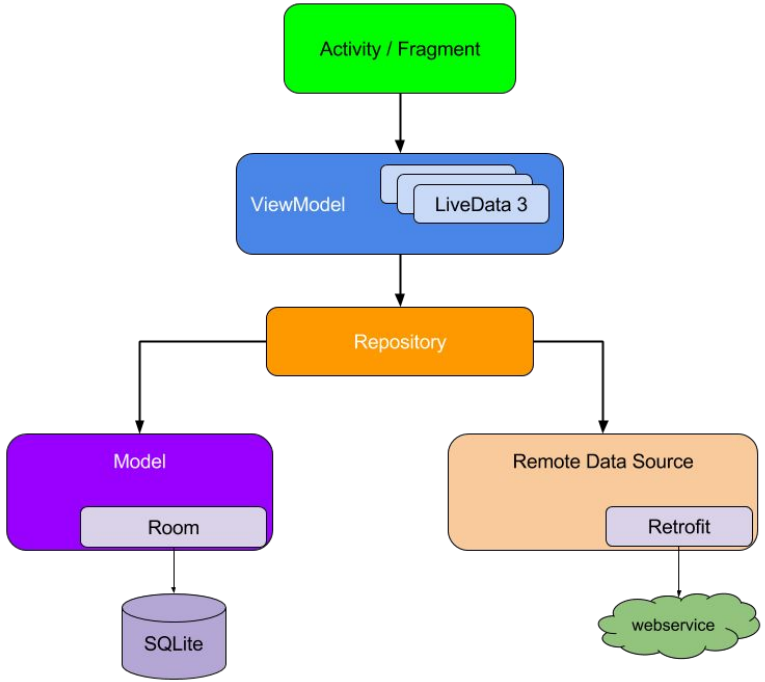
- App may be **destroyed** under normal circumstances:
  - On its own by calling finish or user pressing the back button to navigate away from app
  - Normal lifecycle methods handle this:
    `onPause() -> onStop() -> onDestroy()`
- If the system must destroy, the activity must be able to recreate `Activity`.
- System will destroy the Activity
  - To **recover resources**
  - On an **orientation change**

Shift in App architecture practice

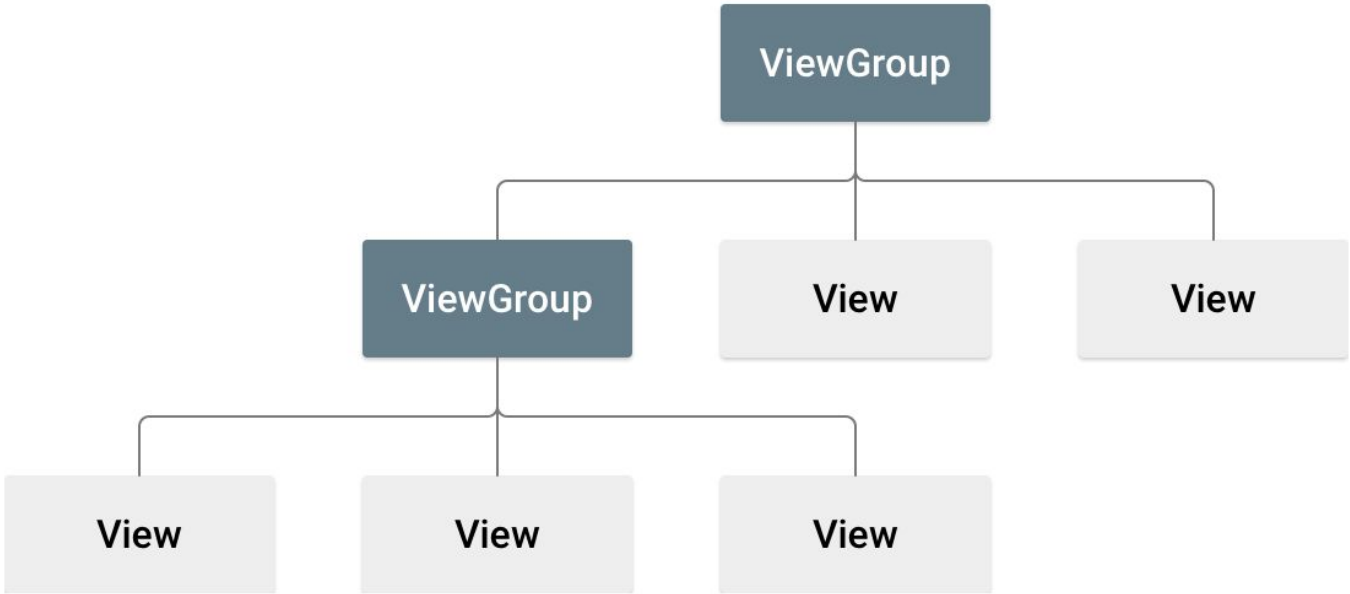
- **Multiple Activity** -> **Single Activity-Multiple Fragment** architecture
  - With new **`Navigation`** component jetpack library introduced in 2018, Google started recommending **Single Activity-Multiple Fragment** architecture.
  - **Simplifies data sharing** between different screens in your application using a **SharedViewModel**.
  - **Simplifies AndroidManifest.XML** file
- Next week we will introduce `Jetpack Compose` which has implements navigation as well

Architectural principles

- **Separation of concerns**
  - Modularity, encapsulation, etc.
- **Model-View design**
  - Create persistent model independent of view
  - Persistence prevents data loss in cases when App is destroyed or otherwise changes its lifecycle state
  - Allows for multiple views on same data (e.g. differently sized devices)
- Every **component** has its own **lifecycle**



Views, ViewGroups and Layouts



Views, ViewGroups and Layouts

- A **View** is a visible, interactive UI element (widget) such as a Button, TextView, etc.
- A **ViewGroup** is an invisible container used to hierarchically and spatially organize Views and other ViewGroups.
- ViewGroups are usually called **Layouts**, and implement defined layout structures, such as `LinearLayout` or `ConstraintLayout`.
- Layouts can be declared as **XML resources**
  - Separates presentation from code that handles behavior (event listeners)
  - Can easily create layouts for differently-sized devices (phones, tablets, etc.)
  - Can use Android Studio Layout editor

Todo before next class

- Work through Build Your First App tutorial. Run the example on a real device if you have one, but the emulator is fine, too.
- Watch the video Building Interfaces with ConstraintLayout.
- Read Understand the Activity Lifecycle.
- Create a new Android Studio project for your term 1 project and push it to a remote repository on eng-git (or GitHub if you prefer). Add me (`bta47`) as a member with 'Reporter' access or higher.
- Before the beginning of next class, write down any questions, concerns, or observations you encountered during your reading, watching, and project planning, and post to Slack channel

Additional Android resources

- Introduction to activities
- Layouts
- Guide to app architecture