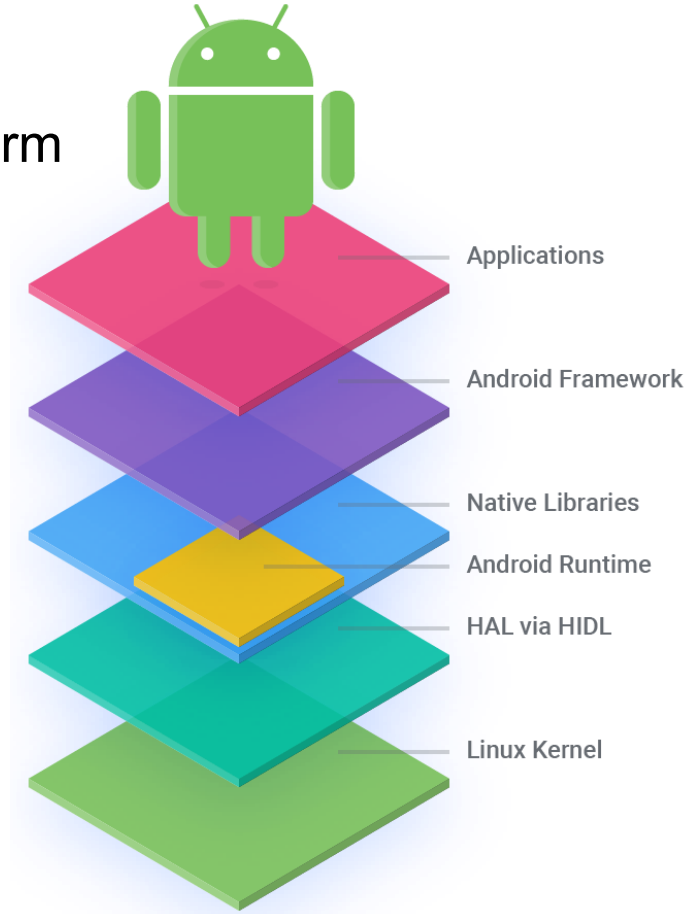
Unit 4:

Important Android Element - `Activity`

Mobile Application Development (MAD)
CCIT4059, 2025-2026

Outline

- Android Stack
- Android Studio Project for Android Platform
  - AndroidManifest.xml, the Manifest file
  - Java `Activity` Class
- Activity Class and Life Cycle
  - List of Activity's Callback Methods



** This unit briefly introduces "basic" Java and Android SDK for new beginners to start developing simple mobile apps. More details will be given in later unit(s).*

Android Stack

- Android is structured in the form of a software stack comprising applications, an operating system, run-time environment, middleware, services and libraries

- This architecture can be represented as a hierarchy of 5 layers:
    - Application (Apps)
    - Android (Java Application Programming Interface, API) Framework
    - Native (C/C++) Libraries and Runtime
    - Hardware Abstraction Layer (HAL)
    - Linux Kernel
- The Android Stack is the full set of layers that make up the Android operating system. E.g. Android Stack likes a five-layer cake, where each layer depends on the one below it.

Android Stack

The Android Stack is a layered structure that shows how Android works—from hardware at the bottom to apps at the top. (Below lists the 5 layers)

1. Linux Kernel (Bottom Layer)
   - Works as the foundation of Android.
   - OS related functions, connecting between device hardware and upper layers
   - Manages hardware like CPU, memory, battery, Wi-Fi, camera, etc.

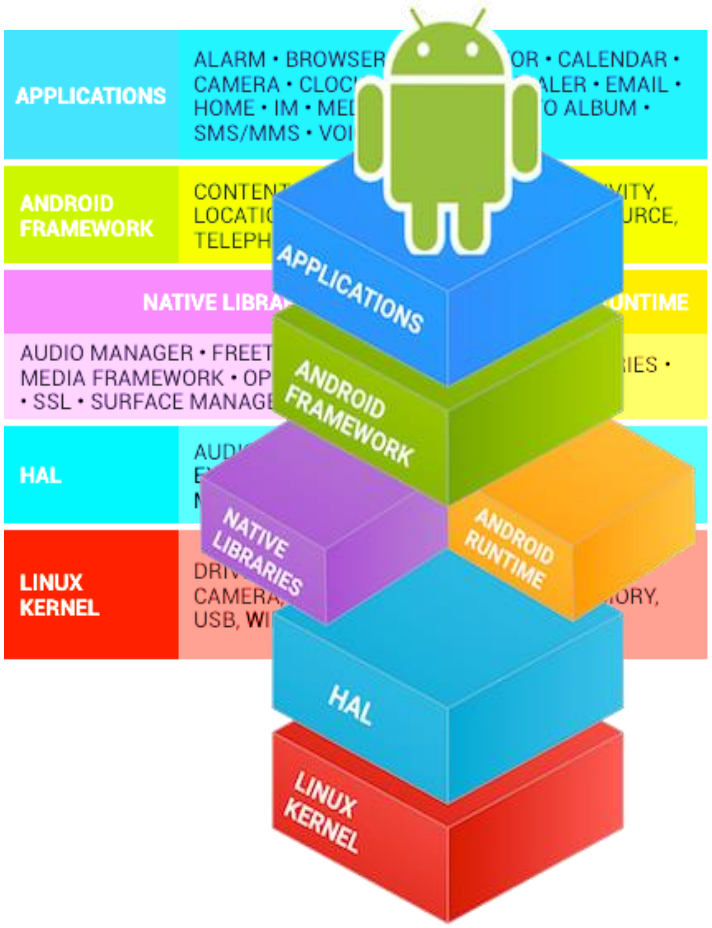
2. Hardware Abstraction Layer (HAL)
   - Acts as a translator between Android software and the phone's hardware.
   - Standard interfaces that expose device hardware capabilities to the higher-level Java API framework
   - Helps Android apps use hardware without needing to know how it works internally.

3. Native (C/C++) Libraries and Runtime Layer
   - Runs apps and keeps them stable and efficient.
   - Core native libraries & Android Runtime (ART)
   - Includes essential libraries (like Java libraries) that apps use to work.

Android Stack

4. Application Framework
   - Provides (building blocks) tools and services that apps creation rely on, e.g.
     - Activity Manager (Handles screens, layouts, elements, etc.)
     - Window Manager
     - Notification system
     - Location service

5. Applications (Top Layer)
   - The layer you interact with (e.g. GUI).
   - Includes system apps (Phone, Messages, Settings) and downloaded apps.
   - Apps for achieving specific goals (or purposes)



Linux and Linux Kernel

- To understand the Android Stack, it is important to first understand the foundation on which Android is built.
- Android is constructed on top of well established open source technologies—most importantly, Linux and the Linux Kernel.
- Linux is a family of open source operating systems used widely in servers, desktops, supercomputers, routers, smart devices, cars, and more.
- It includes many components such as:
  - The Linux Kernel
  - System utilities
  - Desktop environments
  - User applications

How Android Uses the Linux Kernel

- The Android operating system is NOT the same as traditional Linux distributions like Ubuntu. However, Android depends heavily on the Linux Kernel as its foundation.

- Android takes the Linux Kernel and adds:
  - Mobile specific features
  - Power management improvements
  - Android specific security models
  - Support for touchscreens, sensors, cameras, and mobile hardware
  - Android Runtime (ART)
  - Application Framework
  - Google Mobile Services (in most devices)

- **Android OS = Linux Kernel + Android software layers + Apps**

Why Android Uses the Linux Kernel

- Android developers chose the Linux Kernel because it is:
  - Open-source (free to use and modify)
  - Stable and secure, used in millions of systems
  - Efficient, suitable for mobile devices
  - Already supports a wide range of processors and hardware
  - Supported by a large global developer community

- Using Linux saved years of development work and gave Android a strong and secure foundation.

Android Studio Project **for Android Platform**

- Difference of using Java in Android and other Java platforms (e.g. Java SE)
  - Although native Android apps are often written in Java language, there are many differences between the original standard Java API and the Android API for developers
- Example APIs of for building desktop applications
  - Swing/AWT (Abstract Window Toolkit) It is a platform-dependent API to develop GUI (Graphical User Interface) or window-based applications in Java):
    - Provides a wide range of components like buttons, text fields, tables, and trees.
  - Android UI:
    - Offers a variety of widgets such as Button, TextView, EditText, RecyclerView, and ListView.

Android Studio Project for Android Platform

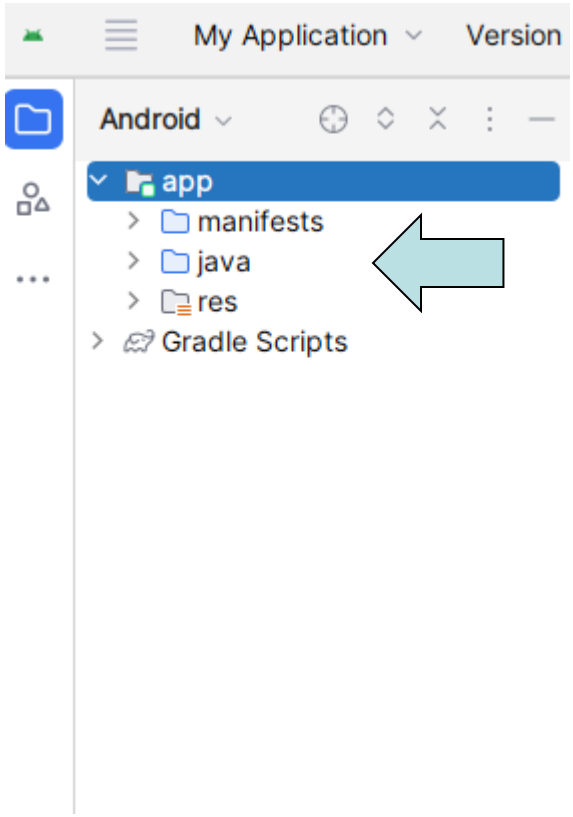
- Unlike a pure Java SE app which is entirely written with Java codes, Android apps are normally developed with Java codes and other resource files in a project

- In a typical Android project (such as one created with Android Studio), often there are many folders and resource files which contain major Java classes, XML files (manifest, layout, etc.), and other resources

- The Android SDK tools compile your code - along with any data and resource files - into an APK: an *Android package*, which is an archive file with an `.apk` suffix
  - One APK file contains all the contents of an Android app and is the file that Android-powered devices use to install the app

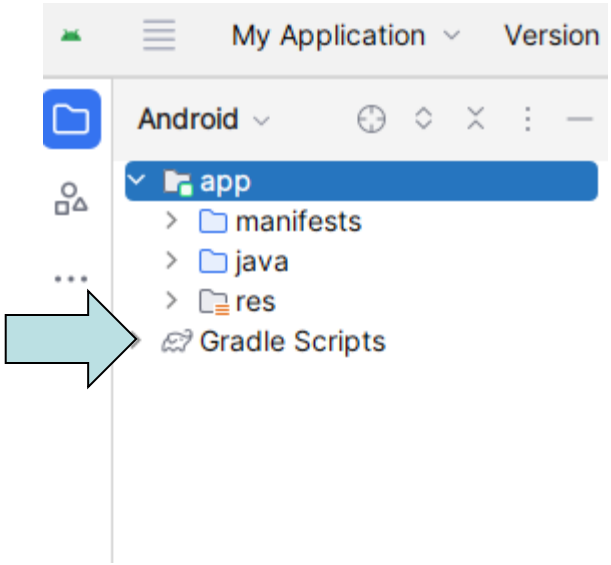
Android Studio Project for Android Platform

- Android Studio project of an app contains 3 important folders:
  - `manifests`: Contains the `AndroidManifest.xml` file which provides important information of an app such as listing the app components
  - `java`: Contains the Java source code files.
    - Such as activity Java file
  - `res`: Contains all non-code resources, such as XML layouts, UI strings, and bitmap images.
    - Such our activity layout xml file, associated to the activity Java file



Android Studio Project for Android Platform - Gradle Scripts

- Gradle Scripts in Android Studio are files that define how your Android project is built and managed.
  - It is crucial for transforming your source code and resources into a runnable Android application.
  - It automates and manages the entire build process, making it easier for developers to focus on writing code.



Android Studio Project for Android Platform (AndroidManifest.xml)

- A typical sample AndroidManifest.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="mad.m1hiworld">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/mad_ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/mad_ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.M1HiWorld">
        <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>
    </application>

</manifest>
```

Android Studio Project for Android Platform (`AndroidManifest.xml`)

- The example `AndroidManifest.xml` file below shows an application:
  - Contains a main "launchable" activity: **`MainActivity`**
    - The intent-filter indicates the activity starts as a main entry point, and displayed in the top-level launcher
  - `MainActivity` essentially is a Java Activity class
    - When app runs, it creates and starts the `MainActivity` activity

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="mad.m1hiworld">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/mad_ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/mad_ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.M1HiWorld">
        <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>
    </application>

</manifest>
```

Android Studio Project for Android Platform (`AndroidManifest.xml`)

- Before the Android system can start an app component, the system must know that the component exists with the "manifest" file, *`AndroidManifest.xml`*
- Android app must declare all its components in this `AndroidManifest.xml` file, which must be at the root of the app project directory
- The manifest file does several things, including to:
  - Declare the app's components:
  - Identify any user permissions the app requires, such as accessing Internet or user's contacts
  - Declare hardware and software features used or required by the app, such as a camera, Bluetooth services, or a multi-touch screen
  - Indicate API libraries the app needs to be linked against (other than Android framework APIs), such as the Google Maps library

Android Studio Project for Android Platform (Java `Activity` Class)

- Android apps are built as a combination of components that can be invoked individually
- An important component type (Java class) is `Activity`, which provides a *single screen for user interface*
  - There are other component types such as `Service` performs work in the background
- In Android, an activity is referred to as one screen in an application. It is very similar to a single window of any desktop application. An Android app consists of one or more screens or activities.
- Developers often use subclasses of class `Activity`, such as `AppCompatActivity`

**Remark: Earlier version uses `ActionBarActivity`, which is now deprecated.*

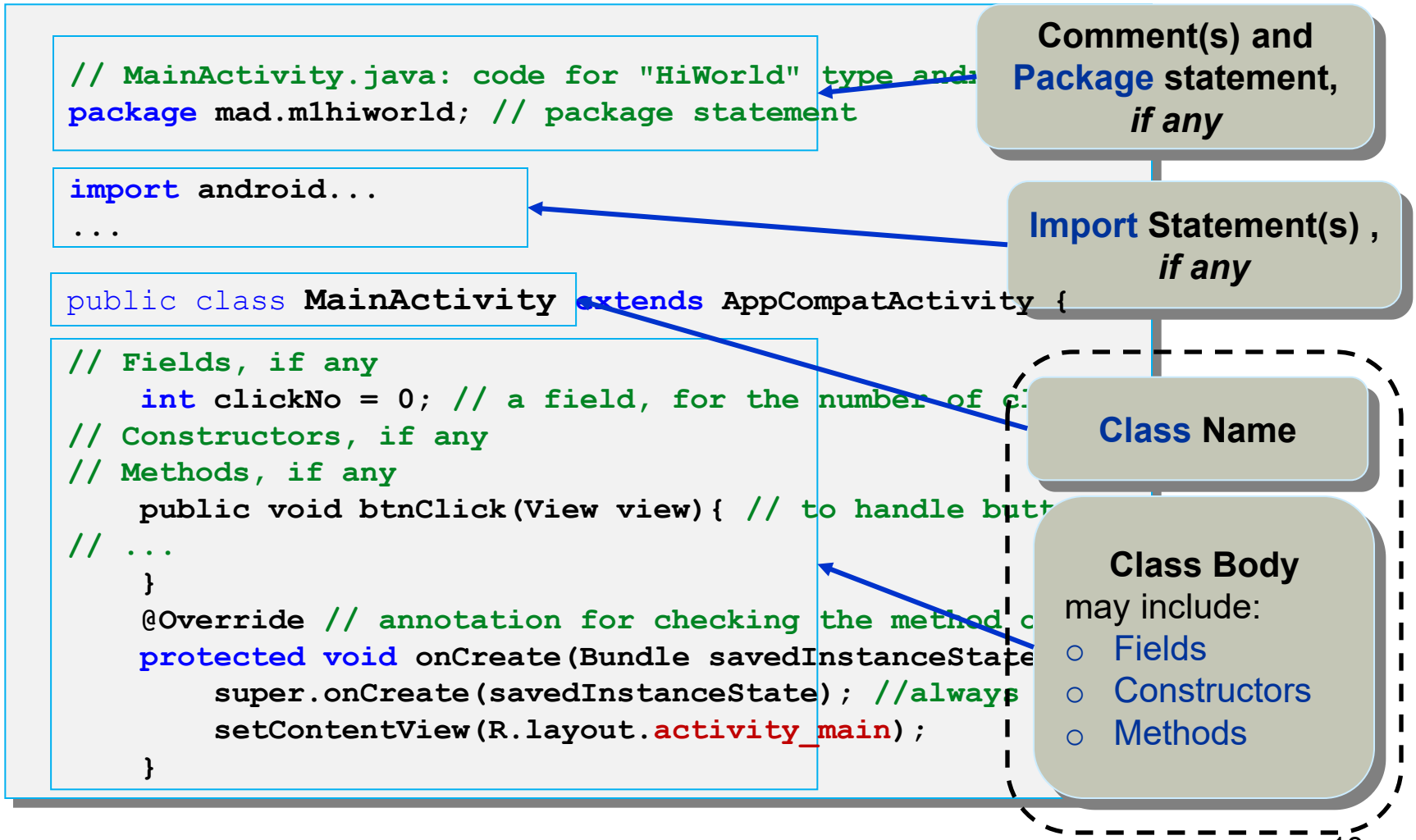
Android Studio Project for Android Platform (Sample Subclass of the Class `Activity`)

- Code in a sample Java file "`MainActivity.java`"

```
// MainActivity.java: code for "HiWorld-type" android app
package mad.m1hiworld; // package statement

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.widget.Button;

// class (named MainActivity) inherits parent (AppCompatActivity )
public class MainActivity extends AppCompatActivity {
    int clickNo = 0; // a field, for the number of clicking
    public void btnClick(View view){ // to handle button click, defined in xml
        Button aBut = ((Button)findViewById(R.id.button)); // refers to button
        aBut.setText("MAD BUTTON CLICKED:\n" + ++clickNo);
    }

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

Android Studio Project for Android Platform (Sample Subclass of the Class `Activity`)

- Typical Form of an `Activity` Java Class



Android Studio Project for Android Platform (Sample Subclass of the Class `Activity`)

- We make use of inheritance mechanism frequently to create classes that inherit other classes, e.g. `MainActivity.java`:

```
public class MainActivity extends AppCompatActivity {
        // ...
}
```

- `MainActivity` is a subclass of `AppCompatActivity` (superclass)
  - Without specifying a superclass, the class's superclass would be the class `Object` in Java, the "Root" class in the inheritance hierarchy

Android Studio Project for Android Platform (Sample `MainActivity`)

- Typical Form of the Sample `MainActivity` Java Class

```
// MainActivity.java: code for "HiW
package mad.m1hiworld; // package s

import androidx.appcompat.app.AppCo
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

// class (named MainActivity) inher
public class MainActivity extends A
    int clickNo = 0; // a field, fo
    public void btnClick(View view)
        Button aBut = ((Button)find
        aBut.setText("MAD BUTTON CL
    }

    @Override
    protected void onCreate(Bundle
        super.onCreate(savedInstanc
        setContentView(R.layout.act
    }
}
```

Android `MainActivity` Class

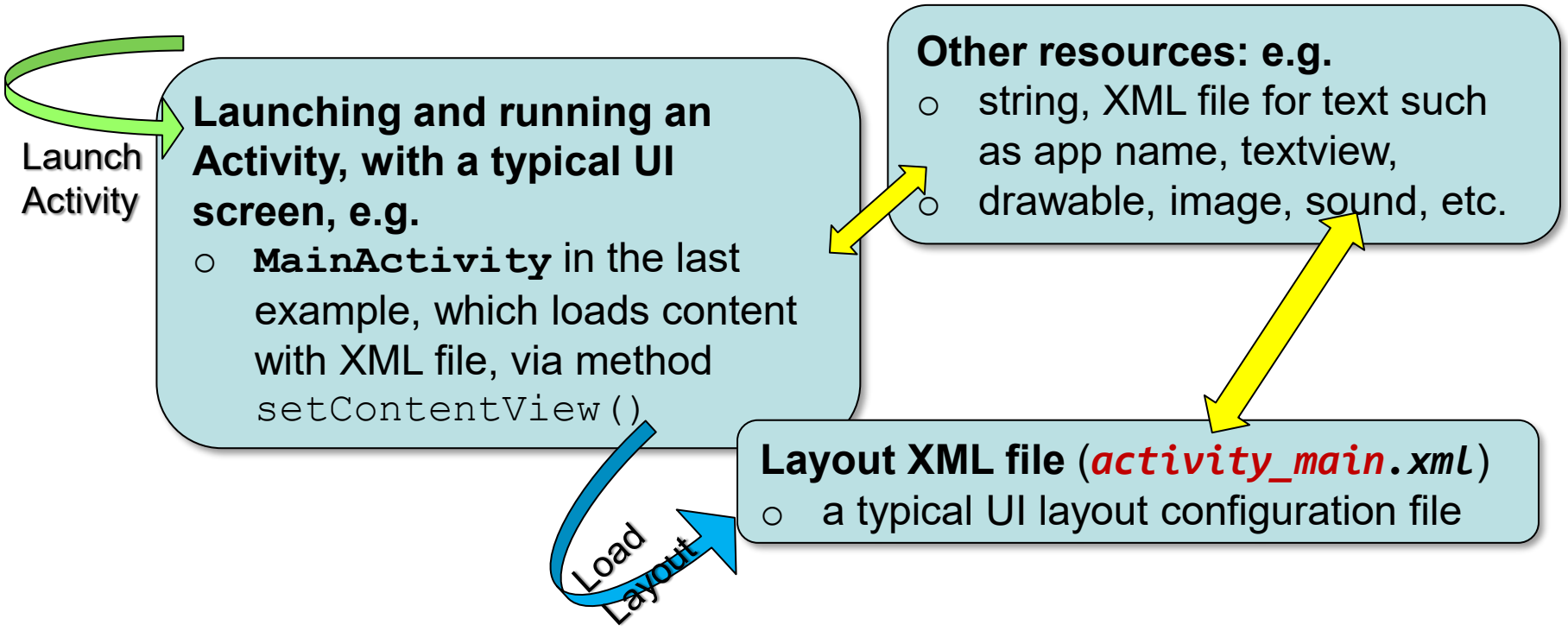
- **`package`** statement
- **`import`** statement(s)
- **Class declaration** (with superclass `AppCompatActivity`)

  `public class MainActivity extends AppCompatActivity {...}`
- **Field(s)**
  - `clickNo` of type `int`
- **Method(s)**
  - Method `btnClick()`
  - Overriding the same callback method of its superclass

  `protected void onCreate(Bundle savedInstanceState)`
- *Constructor(s)*
  - NO specified (a default “no-argument” constructor automatically given)

Android Studio Project for Android Platform (Flow of Executing Android App)

- It is less common to layout the activity with pure Java code. Instead, an associated XML layout file is used, for easier layout composition.
- When an Android app (with activity) is compiled, installed and started, a typical flow of execution is illustrated as below:



Android Studio Project for Android Platform (Flow of Executing a Simple Android App)

```
// MainActivity.java: code for "HiWorld-type" android app
package mad.m1hiworld; // package statement

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.widget.Button;

// class (named MainActivity) inherits parent (AppCompatActivity )
public class MainActivity extends AppCompatActivity {
    int clickNo = 0; // a field, for the number of clicking
    public void btnClick(View view){ // to handle b
        ton)findViewById(R.id.bu
        TTON CLICKED:\n" + ++cl

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setConten ew(R.layout.activity_main);
    }
}
```

0. `package` statement and `import` statements, for declaring a Java class

1. Method **`onCreate()`** is firstly called, like an entry point, when the activity starts (is invoked)

2. The (overridden) **`onCreate()`** method of superclass is normally called first

3. Call the method **`setContentView()`** to set the content of screen (activity), with a layout as argument: **`R.layout.activity_main`**

4. **`onCreate()`** method exits, the Android app keep running (often then jump to another stage)

Android Studio Project for Android Platform (Java `Activity` Class)

- Activity
  - Most apps include several different activities that allow the user to perform different actions
  - Whether an activity is the main activity created when the user clicks an app icon or a different activity that an app starts in response to a user action, the system creates a new instance of Activity and calls its `onCreate()` method

```
// ...
 @Override // annotation for checking the method onCreate
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState); //always call super
        setContentView(R.layout.activity_main);
```

Android Studio Project for Android Platform (Java `Activity` Class)

- Activity
  - We should implement (override) `onCreate()` method to perform basic application startup and setup logic that should happen only once for the entire life of the activity
  - In the `MainActivity` example, `onCreate()` method performs (overrides) its superclass `onCreate()` method, then sets UI (which is in an XML layout file in this case)

```
// ...
   @Override // annotation for checking the method onCreate
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState); // call onCreate of superclass
      setContentView(R.layout.activity_main); // set XML layout file
   }
```

Android Studio Project for Android Platform (Java `Activity` Class and XML Layout file)

- In order to reference and handling UI elements in Java files, we commonly add an ID value to an UI element in the resource file, such as XML layouts, with the syntax: **`android:id="@+id/eltname"`**.
  - We can then reference to and manipulate the component with this ID attribute in other files, such as in Java files.

- Example: In a layout xml file containing a TextView UI element object:

```
<TextView
      android:id="@+id/aTextView"
```

- In a related Java file, we can reference this TextView above with its ID to a corresponding TextView Java object:

```
TextView textView = (TextView) findViewById(R.id.aTextView);
```

Android Studio Project for Android Platform (Java `Activity` Class and XML Layout file)

- Buttons are simple and popular UI elements for user action.
- It is also common to define an "**onClick**" action attribute in the layout xml file, and its related method in Java, similar to the syntax: **android:onClick="onClickMethod"**.
  - We can then define this method **onClickMethod()** in related Java file.
- Example: In a layout xml file containing a Button UI element object:

```
<Button
    android:id="@+id/button"
    android:onClick="btnClick"
```

- We should then define this method **btnClick()** in the related Java file:

```
public void btnClick(View view){ // method defined in xml
// action defined in method body if button is clicked
    Button aBut = ((Button)findViewById(R.id.button));
// ..
}
```

Android Studio Project for Android Platform (A Sample Layout File, **`activity_main.xml`**)

A sample layout XML file, that includes three UI components (2 textviews and 1 button) within a Layout "container"

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

    <TextView
        android:id="@+id/textView2"
...
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/textView"
...
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="btnClick"
        android:text="CLICK ME"
...
        app:layout_constraintVertical_bias="0.35" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Android Studio Project for Android Platform (Classes and Objects in Simple Android Apps)

- Apart from the class Activity, there are three 3 major types of app components:
  - Services
    - typically runs in the background to perform long-running operations
  - Content providers
    - manages a shared set of app data, stored in the file system, an SQLite database, etc.
  - Broadcast receivers
    - responds to system-wide broadcast announcements, such as a broadcast announcing that the screen has turned off, the battery is low, etc.

(Both will be introduced in the later chapters)

Activity Class and Life Cycle

- For a typical Android app, as users navigate through, out of, and back to the app, *activity* instances in the app transition between different states in their lifecycle, associated with various callback methods to be called

- Each activity goes through various stages or a lifecycle and is managed by activity stacks. So when a new activity starts, the previous one always remains below it.
  - For instance, when your activity starts for the first time, it comes to the foreground of the system and receives user focus

- During this process, the Android system calls a series of lifecycle methods on the activity in which we may set up the user interface and other components

Activity Class and Life Cycle

- There are four stages of an activity.

1. If an activity is in the foreground of the screen i.e. at the top of the stack (layer), then it is said to be active or running. This is usually the activity that the user is currently interacting with.

2. If an activity has lost focus and a non-full-sized or transparent activity has focused on top of your activity.
   - In such a case either another activity has a higher position in multi-window mode or the activity itself is not focusable in the current window mode. Such activity is completely alive.

Activity Class and Life Cycle

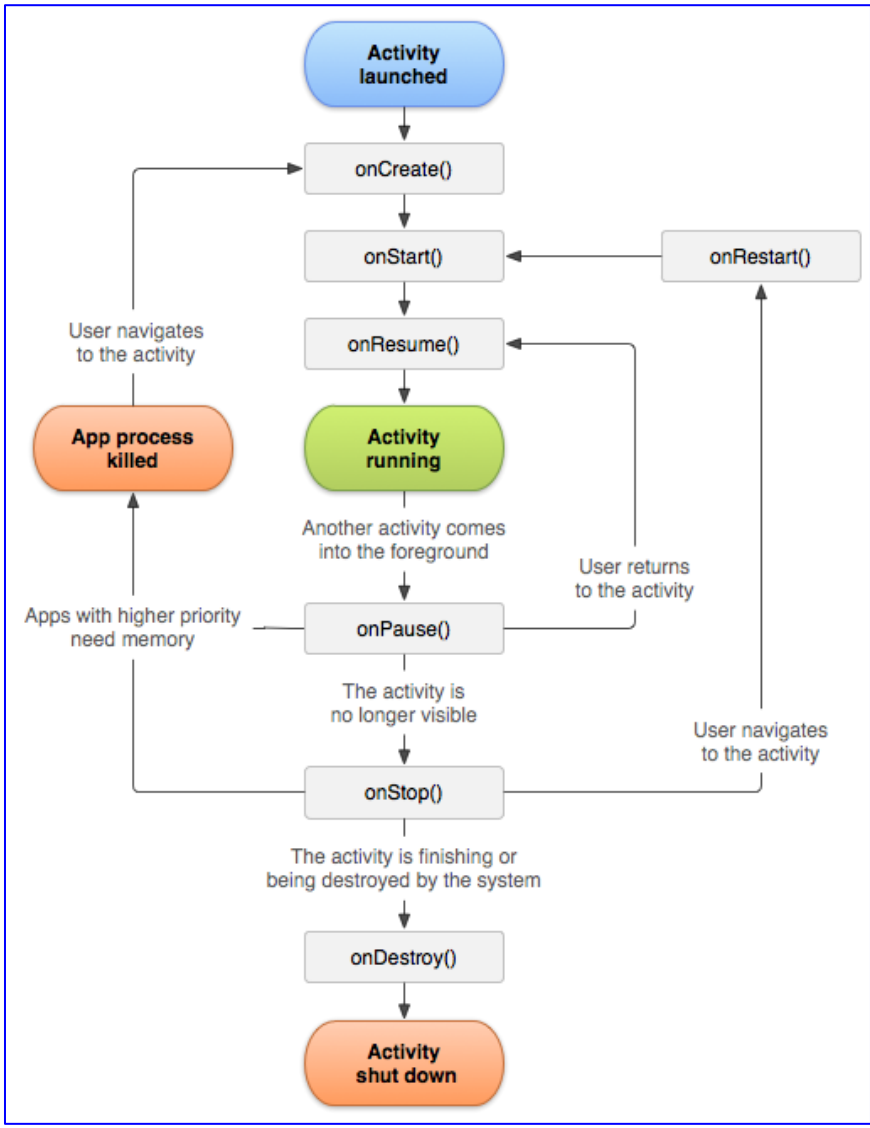
3. If an activity is completely hidden by another activity, it is stopped or hidden. It still retains all the information, and as its window is hidden thus it will often be killed by the system when memory is needed elsewhere.

4. The system can destroy the activity from memory by either asking it to finish or simply killing its process. When it is displayed again to the user, it must be completely restarted and restored to its previous state.

- For each stage, android provides a set of 7 callback methods that have their own significance for each stage in the life cycle.

Activity Class and Life Cycle

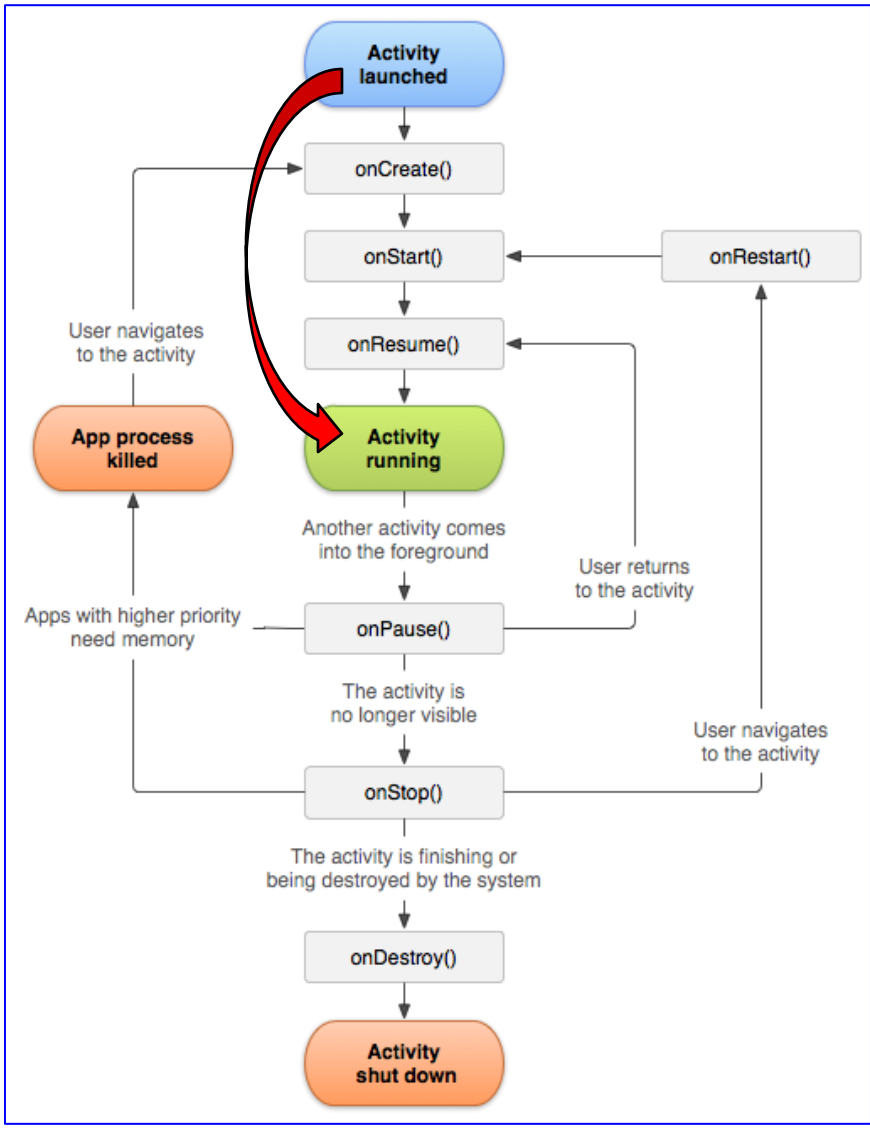
- Unlike programming in official Java SE platform which most apps are launched with a `main()` method, the Android system initiates code in an `Activity` instance by invoking specific callback methods that correspond to specific stages of its lifecycle.



A *simplified* illustration of the activity lifecycle

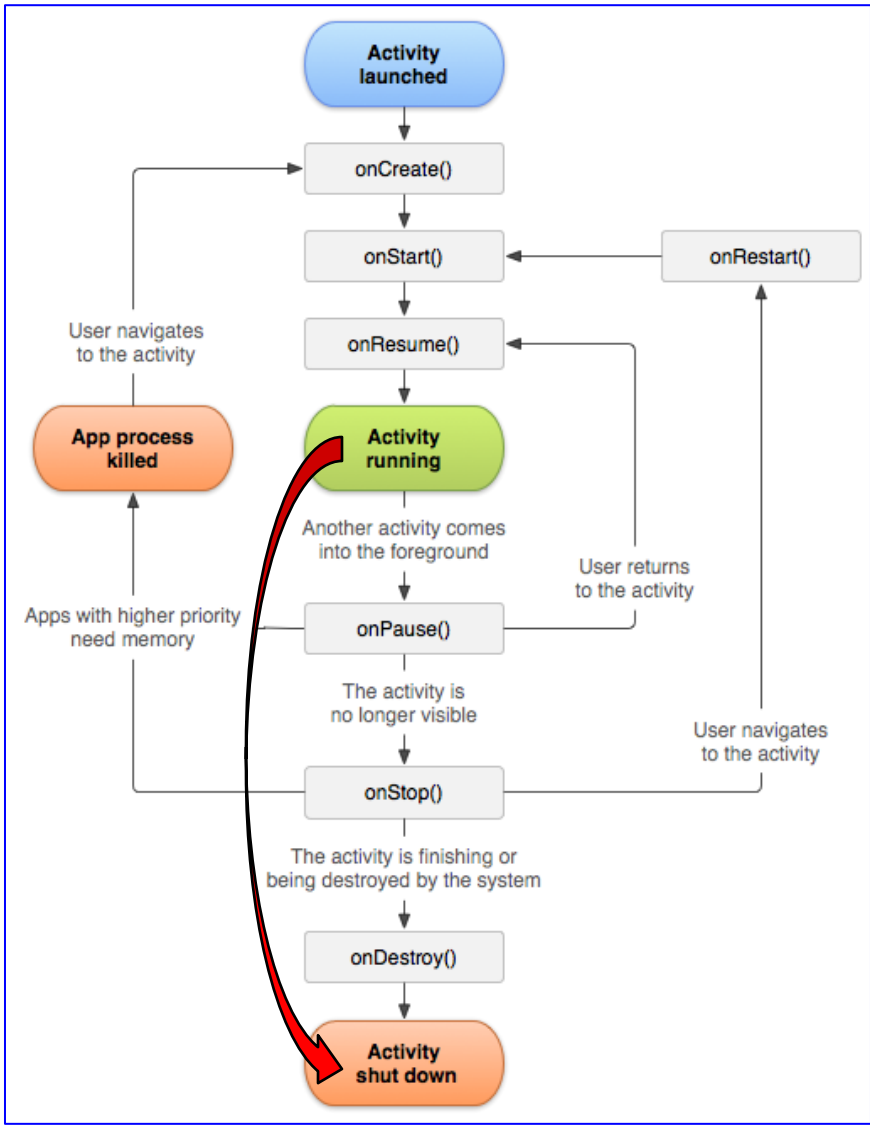
Activity Class and Life Cycle

- As the system creates a new activity instance, each *callback method* moves the activity state one step toward the point at which the activity is running (active) in the foreground and the user can interact with it



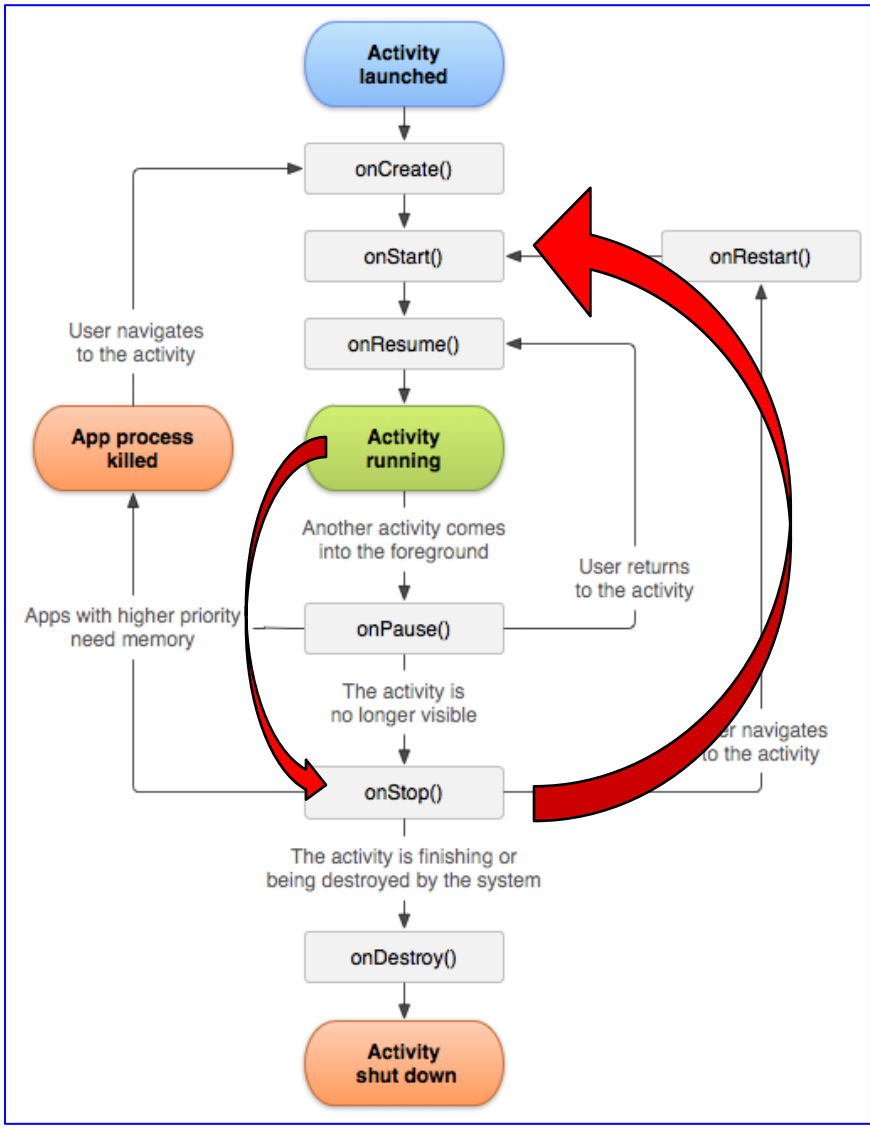
Activity Class and Life Cycle

- As user begins to leave the activity, the system calls other methods that move the activity state to dismantle the activity and destroy it



Activity Class and Life Cycle

- In some cases, the activity will move only part way down and wait (pause/stop), such as when the user switches to another app
  - the activity may later move back to foreground running / active (if the user returns to the activity) and resume where the user left off

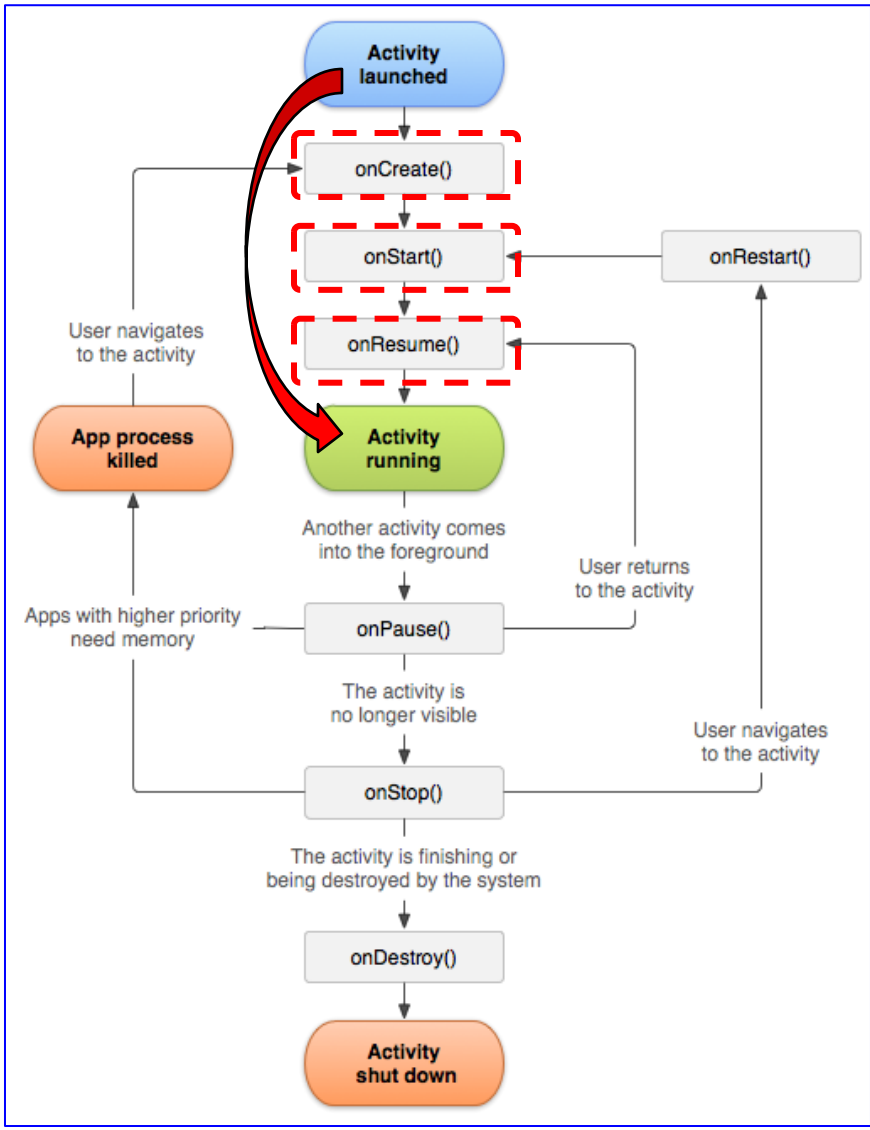


Activity Class and Life Cycle

- In general, different states in activity lifecycle can be associated to the different processes, often including:
  - Starting an Activity
    - user launches an app, and perform activity creation
  - Pausing and Resuming an Activity
    - activity is paused (e.g. partially obscured) & resumed
  - Stopping and Restarting an Activity
    - user completely leaves the activity, then returns to it
  - Recreating an Activity
    - activity is destroyed, and then rebuilt if necessary

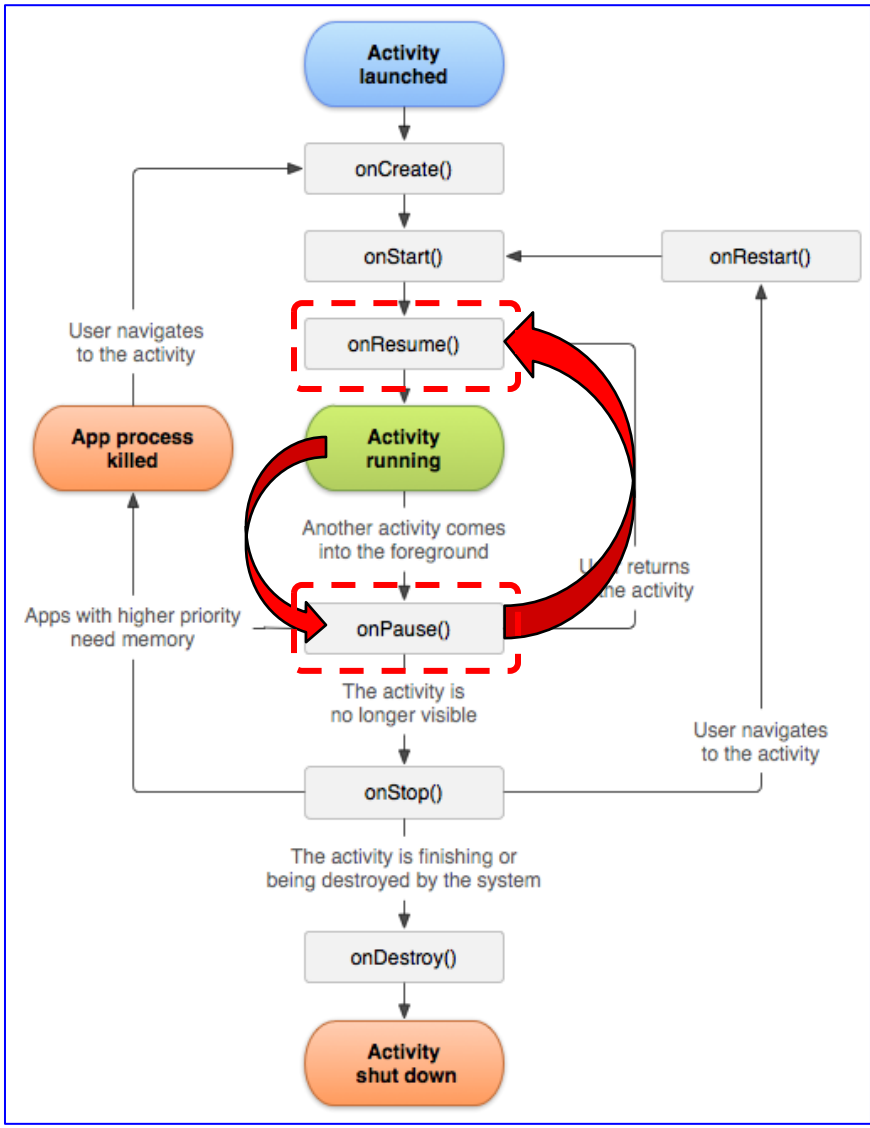
Activity Class and Life Cycle (Creating and Starting an Activity)

- As the system creates and starts a new activity instance, each *callback method* moves the activity state toward running (active)
- System calls in sequence when creating a new instance of the activity:
  - 1) **`onCreate()`**,
  - 2) **`onStart()`**, and
  - 3) **`onResume()`**



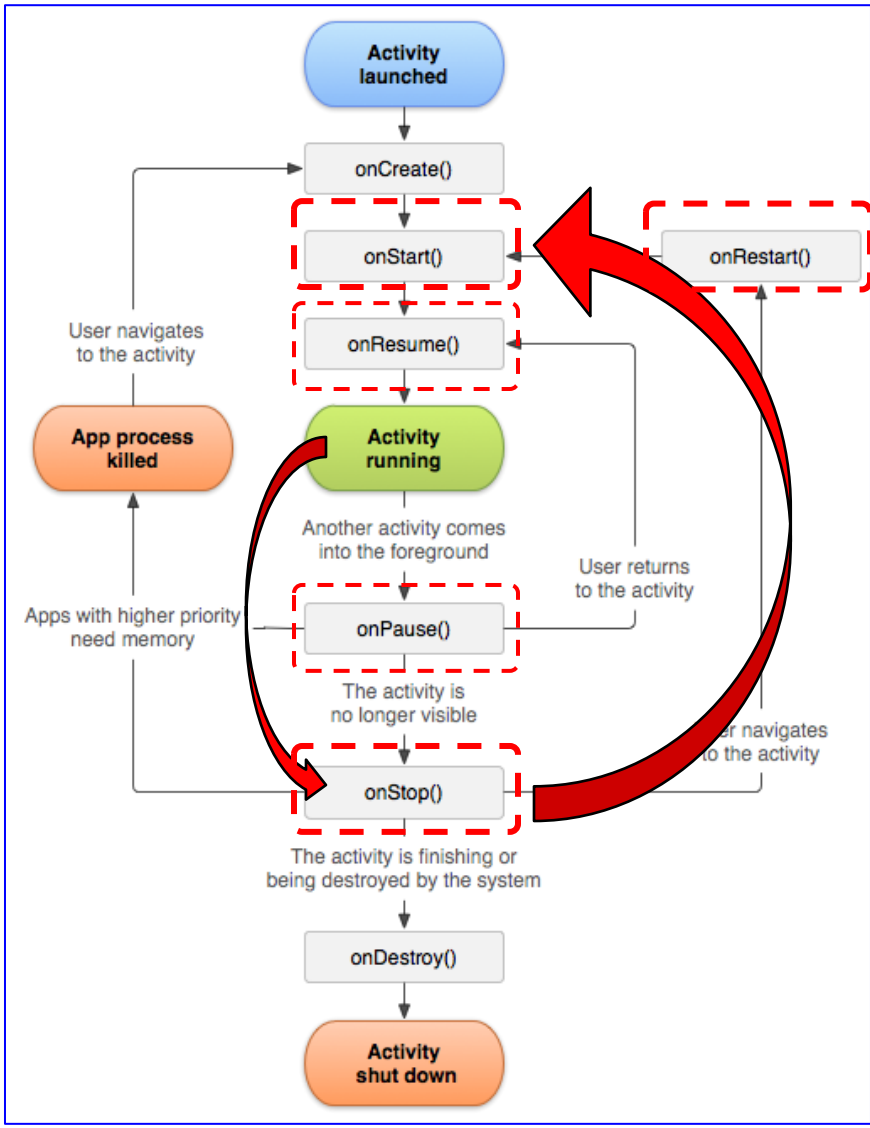
Activity Class and Life Cycle (Pausing and Resuming an Activity)

- In certain situations, the user is leaving the active activity, and activity is no longer in the foreground
  - System calls **`onPause()`** and the activity waits.
  - If the user returns to the activity while it's still paused, the system calls **`onResume()`.**



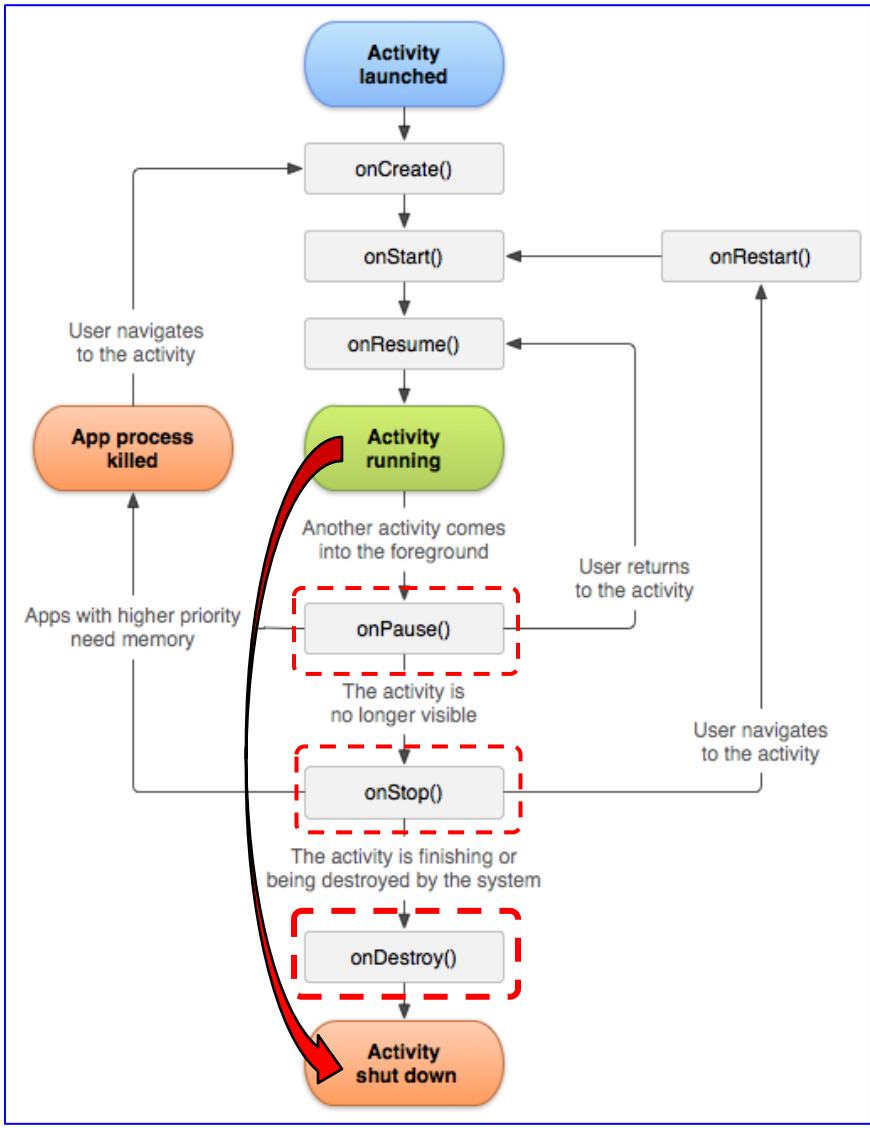
Activity Class and Life Cycle (Stopping and Restarting an Activity)

- When activity is no longer visible to the user (e.g. user presses HOME button), it has entered the Stopped state, and system invokes the **`onStop()`** callback.
- If activity comes back to interact with the user again, the system invokes **`onRestart()`** then **`onStart()`** toward running state again



Activity Class and Life Cycle (Destroying an Activity)

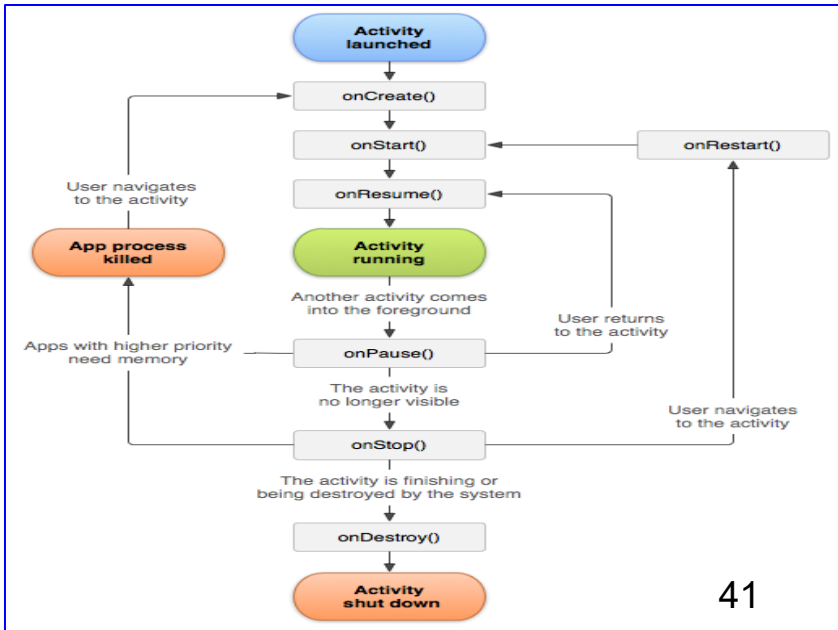
- While activity's first callback is `onCreate()`, its last is **`onDestroy()`**. E.g.
  - If user presses the BACK button
  - If activity signals its own destruction by calling its `finish()` method
  - System may destroy an activity to recover memory, if foreground process requires more resources



Activity Class and Life Cycle (List of Activity's Callback Methods)

- The entire lifecycle of an activity is defined by the following Activity callback methods.
  - Developers ***should always call up to their superclass first*** when implementing / overriding them.

```
public class Activity extends ApplicationContext {
    protected void onCreate(Bundle savedInstanceState);
    protected void onStart();
    protected void onResume();
    protected void onPause();
    protected void onStop();
    protected void onRestart();
    protected void onDestroy();
}
```



Activity Class and Life Cycle (List of Activity's Callback Methods)

- Sample code of the class "`MainActivity.java`"
  - Should ***always call up to their superclass of callback methods first*** when overriding them

```
// MainActivity.java: code for "HiWorld-type" android app
package mad.m1hiworld; // package statement
// import statements

// class (named MainActivity) inherits parent (AppCompatActivity )
public class MainActivity extends AppCompatActivity {
    int clickNo = 0; // a field, for the number of clicking
    public void btnClick(View view){
        //..
    }

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

Activity Class and Life Cycle (List of Activity's Callback Methods)

- **`onCreate()`:** Called when the system creates a new activity. This is where we should do normal static set up.
  - creates views, binding data to lists, etc.
  - provides a Bundle containing previous frozen state (if worked previously).

- **`onStart()`:** Called when the activity becomes visible to the user. Called after `onCreate()` (first started), or after `onRestart()` when the activity had been stopped but is now again being displayed.

- **`onResume()`:** Called just before the activity starts interacting with the user. Called after `onStart()`, or `onPause()`, for activity to start interacting with the user (The activity is at the top of the activity stack, with a user interacting with it).

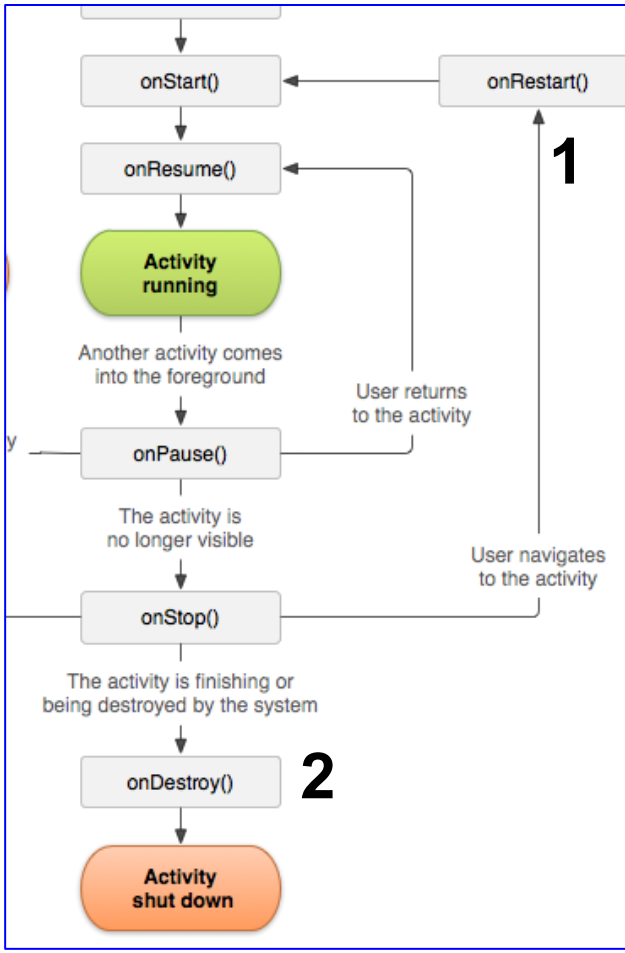
Activity Class and Life Cycle (List of Activity's Callback Methods)

- **`onPause()`:** Called as the activity loses focus (no more active) when it is going into the background but not killed; e.g. when the system is about to start resuming a previous activity.
  - Implementations of this method must be very quick because the next activity will not be resumed until this method returns.

Activity Class and Life Cycle (List of Activity's Callback Methods)

- **`onStop()`:** Called when the activity is no longer visible to the user, e.g. because another activity has been resumed and is covering this one.
  1. It is followed by **`onRestart()`** when the activity is revoked from the background,
  2. It is followed by **`onDestroy()`** when the activity is closed or finished, and nothing when the activity remains on the background only.

- This method may never be called, in low memory situations where the system does not have enough memory to keep the activity's process running after its **`onPause()`** is called.



Activity Class and Life Cycle (List of Activity's Callback Methods)

- **`onRestart()`:** Called after `onStop()` when activity is being re-displayed to user (e.g. user has navigated back to it). Always followed by `onStart()`.

- **`onDestroy()`:** Called before an activity is destroyed. This is the final call we receive before our activity is destroyed. Perform any final cleanup before an activity is destroyed.
  - This can happen either because the activity is finishing (someone called `finish()` on it), or because the system is temporarily destroying this instance of the activity to save space.

Activity Class and Life Cycle (List of Activity's Callback Methods)

The following is the sample program to demonstrate the life cycle of an Activity in Android Studio.

```
package com.example.myapplication;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toast.makeText(getApplicationContext(), "onCreate Called",
Toast.LENGTH_LONG).show();
    }

    protected void onStart() {
        super.onStart();
       Toast.makeText(getApplicationContext(), "onStart Called",
Toast.LENGTH_LONG).show();
    }
```

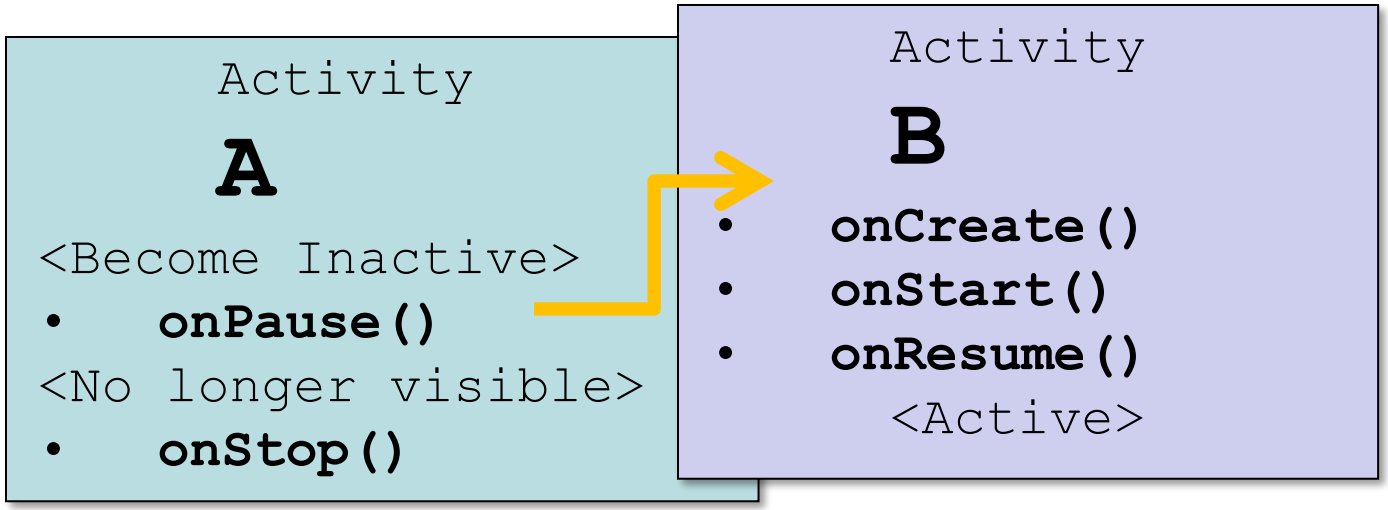
(Continue on next page)

Activity Class and Life Cycle (List of Activity's Callback Methods)

```
@Override
    protected void onRestart() {
        super.onRestart();
        Toast.makeText(getApplicationContext(), "onRestart Called",
Toast.LENGTH_LONG).show();
    }

    protected void onPause() {
        super.onPause();
        Toast.makeText(getApplicationContext(), "onPause Called",
Toast.LENGTH_LONG).show();
    }

    protected void onResume() {
        super.onResume();
        Toast.makeText(getApplicationContext(), "onResume Called",
Toast.LENGTH_LONG).show();
    }

    protected void onStop() {
        super.onStop();
        Toast.makeText(getApplicationContext(), "onStop Called",
Toast.LENGTH_LONG).show();
    }

    protected void onDestroy() {
        super.onDestroy();
        Toast.makeText(getApplicationContext(), "onDestroy Called",
Toast.LENGTH_LONG).show();
    }
}
```

Activity Class and Life Cycle (Navigating Between Activities)

- An activity may start another activity at some point, e.g. via `Intent`.
- When one activity starts another, they both experience lifecycle transitions. Below shows a typical order of operations (including callback methods) that occur when Activity **A** starts Activity **B**:
  - Activity **A**'s `onPause()` method executes.
  - Activity **B**'s `onCreate()`, `onStart()`, and `onResume()` methods execute in sequence. (Activity **B** now has user focus.)
  - Then, if Activity **A** is no longer visible on screen, its `onStop()` method executes.



References

- Part of this slide set is prepared and extracted from the followings:
  - Friesen, J. (2014) *'Learn Java for Android Development'*, 3ed, Apress Media
  - Gargenta, M. and Nakamura, M. (2014) *'Learning Android: Develop Mobile Apps Using Java and Eclipse'*, 2ed, O'Reilly Media, Inc.
  - Gok, N. and Khanna, N. (2013) *'Building Hybrid Android Apps with Java and JavaScript'*, O'Reilly Media, Inc.
  - Nolan, G., Cinar, O. and Truxall, D. (2014) *'Android Best Practices'*, Apress Media
  - Smyth, N. (2014) 'Android Studio Development Essentials', CreateSpace (Web: http://www.techotopia.com/index.php/Android_Studio_Development_Essentials)
  - Wolfson, M. (2013) *'Android Developer Tools Essentials'*, O'Reilly Media, Inc.
  - Google Inc. 'Website for Android Developers', http://developer.android.com/
  - Google Inc. 'Website for Android Sources', https://source.android.com/
  - Apple Inc. 'Website for Apple Developers', Website: https://developer.apple.com/
  - Microsoft Corp. 'Website for Windows Developers', https://developer.microsoft.com/en-us/windows
  - Wikipedia Website: http://en.wikipedia.org
- This set of slides is for teaching purpose only