Mobile Devices & Android

Desenvolvimento de Software e Sistemas Móveis (DSSMV)

Licenciatura em Engenharia de Telecomunicações e Informática

LETI/ISEP

2025/26

Paulo Baltarejo Sousa

pbs@isep.ipp.pt

Disclaimer

Material and Slides

Some of the material/slides are adapted from various:

- Presentations found on the internet;
- Books;
- Web sites;
- ...

Outline

Android OS

Overview

- Android is a mobile OS:
  - Based on the Linux kernel
  - Developed by Google.
  - Designed primarily for mobile devices, such as smartphones and tablets, but nowadays there are specific versions for:
    - Televisions, Cars, and Wear (for wrist watches)

  - Variants of Android are also used on notebooks, game consoles, digital cameras, and other electronics.

Versions

- Android versions evolution

- Android compatibility [1]
  - It assures application's compatibility with one or more versions of the Android platform.
    - An android application is able to run in multiple Android versions

[1] https://source.android.com/compatibility/overview.html

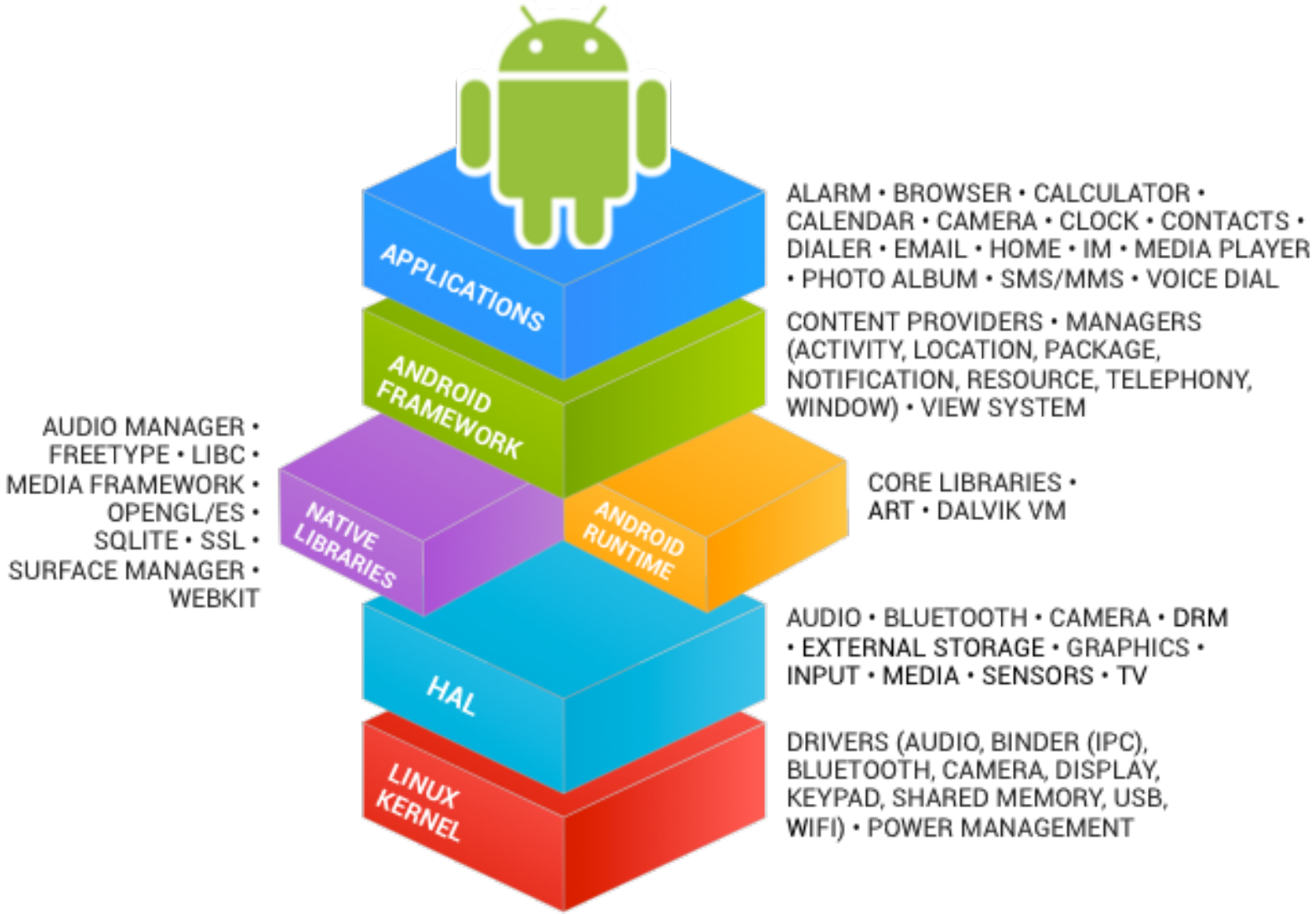
How to develop Android apps

- Android applications are written in the Java or Kotlin programming language.
  - During development the developer creates the Android specific configuration files and writes the app logic in the Java programming language.
  - The Android tooling converts these app files, transparently to the user, into an Android app (an `apk` file).

- When developers trigger the deployment in their Integrated Development Environment (IDE), the whole Android app is compiled, packaged, deployed and started.

Android architecture

Architecture



Runtime

- Each Android app runs in its own security sandbox:
- Each process has its own virtual machine (VM), so an app's code runs in isolation from other apps.
  - Every app runs in its own Linux process.
  - Android starts the process when a app is launched, then shuts down the process when it is no longer needed or when the system must recover memory for other applications.
  - Since Android 5.0 Lollipop system, the Dalvik VM has been officially replaced by a new runtime called ART (Android RunTime).

Framework (I)

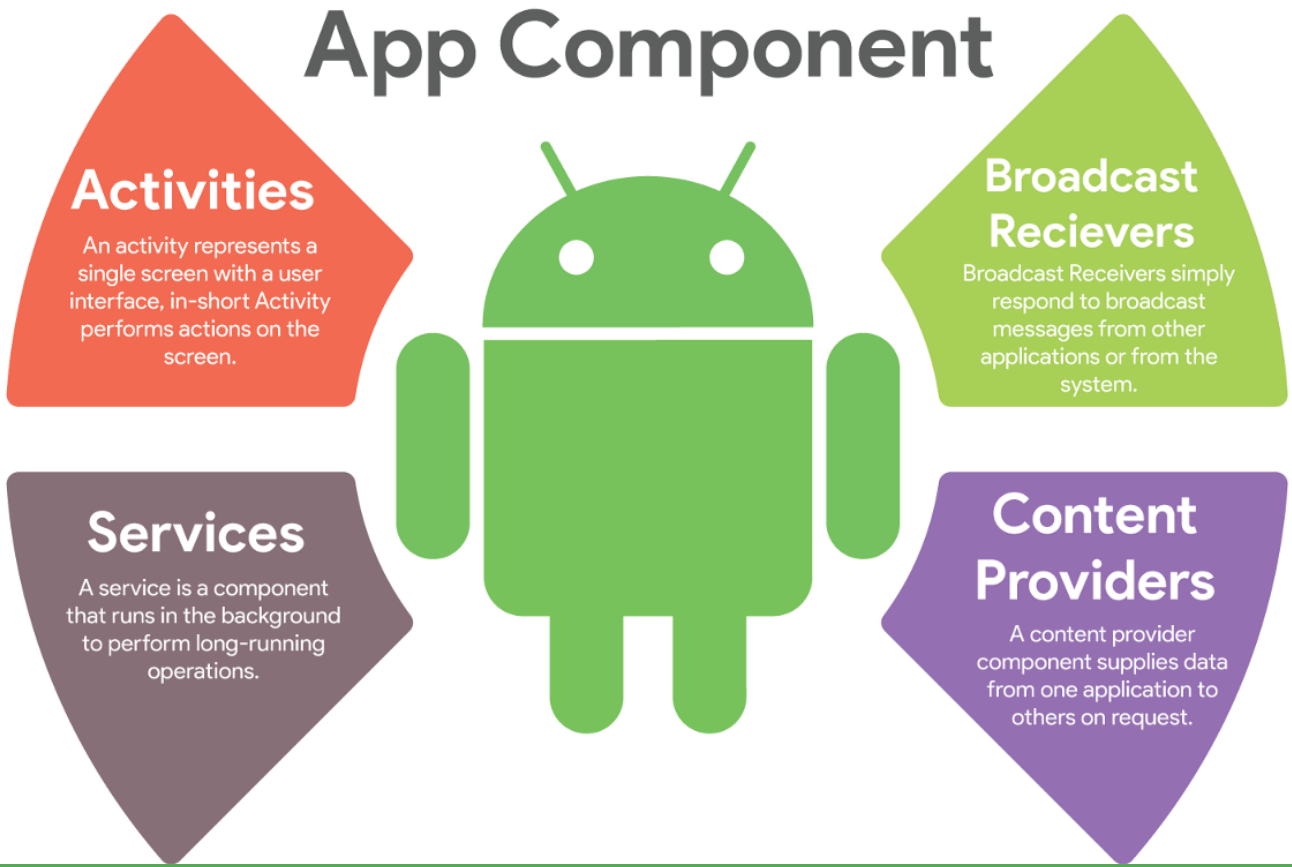
- **activities**: An activity is the key execution component. You need at least one activity in an app that deals with the User Interface (UI). An activity is implemented as a subclass of `Activity`.
- **fragments**: Fragments are relatively new to Android but very important in programming the UI. They are mini-activities. A fragment is implemented as a subclass of `Fragment`.
- **intents**: An Intent is a messaging object you can use to request an action from another app component. Although intents facilitate communication between components in several ways, there are three fundamental use-cases:
  - To start an activity
  - To start a service
  - To deliver a broadcast

Framework (II)

- **services**: Typically services are long running programs that do not need to interact with the UI. A service is implemented as a subclass of `Service`.
- **content providers**: Apps could share data. Content providers manage access to a structured set of data. A content provider is implemented as a subclass of `ContentProvider`.
- **broadcast receivers**: A Broadcast receiver allows you to register for system or application events. All registered receivers for an event are notified by the Android runtime once this event happens. A broadcast receiver is implemented as a subclass of `BroadcastReceiver` and each broadcast is delivered as an `Intent` object.

Android Application Components

- An android **component** is simply a piece of code that has a well defined life cycle e.g. Activity, BroadcastReceiver, Service etc.
- The core building blocks or fundamental components of android are activities, intents, services, content providers, and receivers.



Developing for Android

What is?

- Every application **must have an** `AndroidManifest.xml` file (with precisely that name) in its root directory.
- **It provides essential information** about your app to the **Android system**, which the system must have before it can run any of the app's code:
  - It **names the Java package** for the application.
    - The package name serves as a unique identifier for the application.
  - It describes the **components of the application**
    - Activities, services, broadcast receivers, and content providers that compose the application.
  - It **names the classes that implement each of the Android components and publishes their capabilities**, such as the Intent messages that they can handle.
    - These declarations inform the Android system of the components and the conditions in which they can be launched.
  - It **declares the permissions** that the application must have.
  - It declares the minimum level of the Android API that the application requires.

Structure

```
<?xml version="1.0" encoding="utf
     -8"?>
<manifest>
<uses-permission />
<permission />
<permission-tree />
<permission-group />
<instrumentation />
<uses-sdk />
<uses-configuration />
<uses-feature />
<supports-screens />
<compatible-screens />
<supports-gl-texture />
<application>
<activity>
<intent-filter>
<action />
<category />
<data />
</intent-filter>
<meta-data />
</activity>
<activity-alias>
<intent-filter> . . . </intent-
     filter>
<meta-data />
</activity-alias>
<service>
<intent-filter> . . . </intent-
     filter>
<meta-data/>
</service>
<receiver>
<intent-filter> . . . </intent-
     filter>
<meta-data />
</receiver>
<provider>
<grant-uri-permission />
<meta-data />
<path-permission />
</provider>
<uses-library />
</application>
</manifest>
```

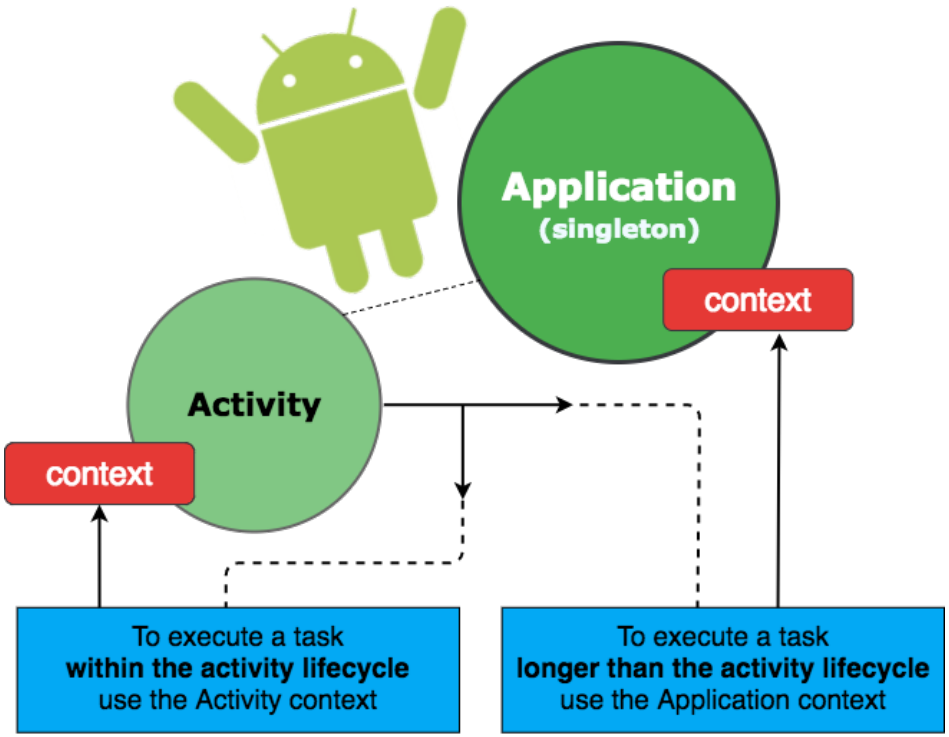
Example

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="pt.pbscaf.homecontrol">
  <uses-permission android:name="android.permission.BLUETOOTH"/>
  <uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
<application
  android:allowBackup="true"
  android:icon="@mipmap/ic_launcher"
  android:label="@string/app_name"
  android:roundIcon="@mipmap/ic_launcher_round"
  android:supportsRtl="true"
  android:theme="@style/AppTheme">
  <activity android:name=".MainActivity">
    <intent-filter>
    <action android:name="android.intent.action.MAIN"/>
    <category android:name="android.intent.category.LAUNCHER"/>
    </intent-filter>
  </activity>
  <activity android:name=".HomeControlActivity"/>
  <service android:name=".HomeControlService"/>
</application>
</manifest>
```

Context (I)

- Context is a **provider of access to a set of resources available to an application** such as application's environment and device settings.
  - It allows access to application-specific resources and classes, as well as up-calls for application-level operations such as launching activities, broadcasting and receiving intents, etc.
- Instances of the class `android.content.Context` provide the **connection to the Android system and actual device**.
  - Activities and services extend the `Context` class. Therefore, they can be directly used to access the `Context`.
    - `getApplicationContext()`
    - `getContext()`
    - `getBaseContext()`
    - `this` (when in the activity class)
- It gives access to the system and application resources and services.
  - For example, you can check the size of the current device display via it.

Context (II)



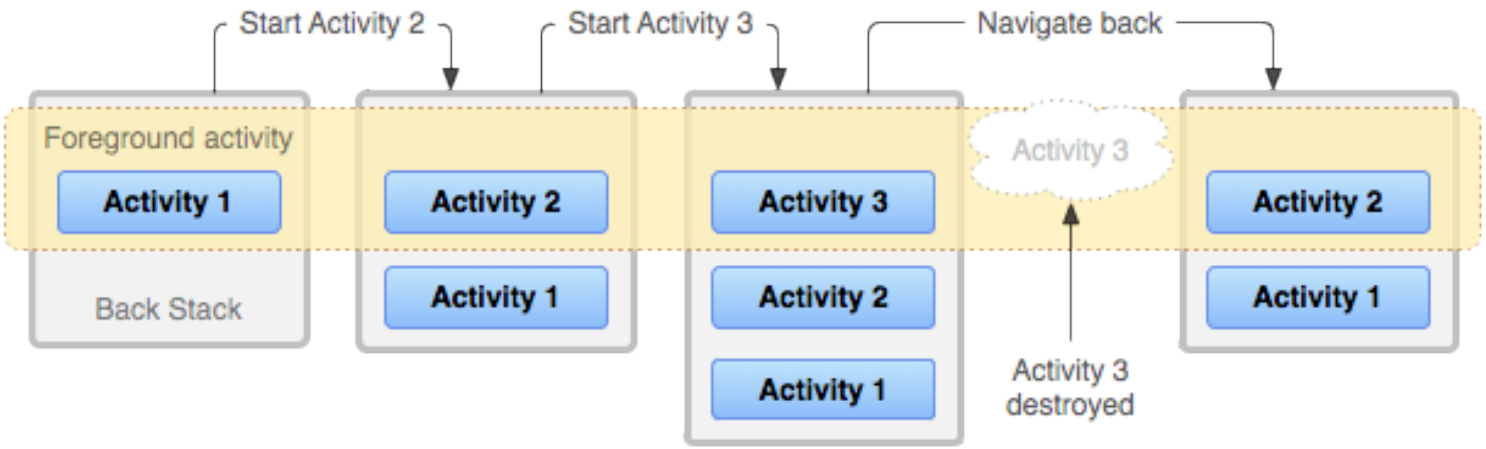
- **Application Context**: It is the application and we are present in Application
- **Activity Context**: It is the activity and we are present in Activity.

Activity

- An **activity provides a user interface** (UI) for a single screen in an app
  - Each activity is given a window in which is drawn its UI.
  - The window typically fills the screen, but may be smaller than the screen and float on top of other windows.
  - To create an activity, you must create a subclass of `Activity` (or an existing subclass of it.
- An app usually consists of multiple activities.
  - One activity in an app is specified as the "main" activity, which is presented to the user when launching the app for the first time (defined in the `AndroidManifest.xml` file).
  - An activity can start activities of the app (and also from other apps).
- **Each time a new activity starts, the previous activity is stopped**, but the system preserves the activity in a stack (Back Stack) .
  - Activities can move into the background and then be resumed with their state restored.

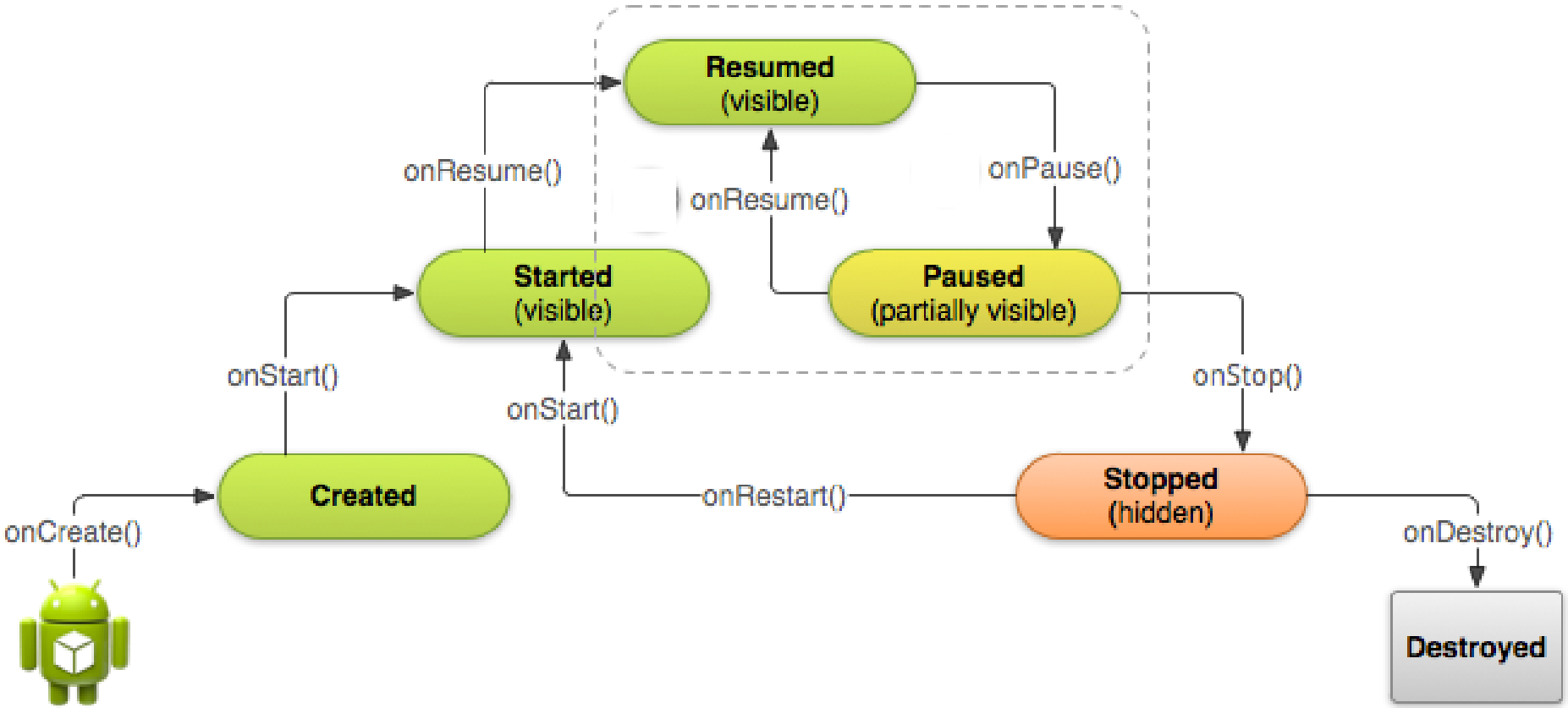
Back Stack

- **Activities are arranged in a stack** (Back Stack), in the order in which each activity is opened.

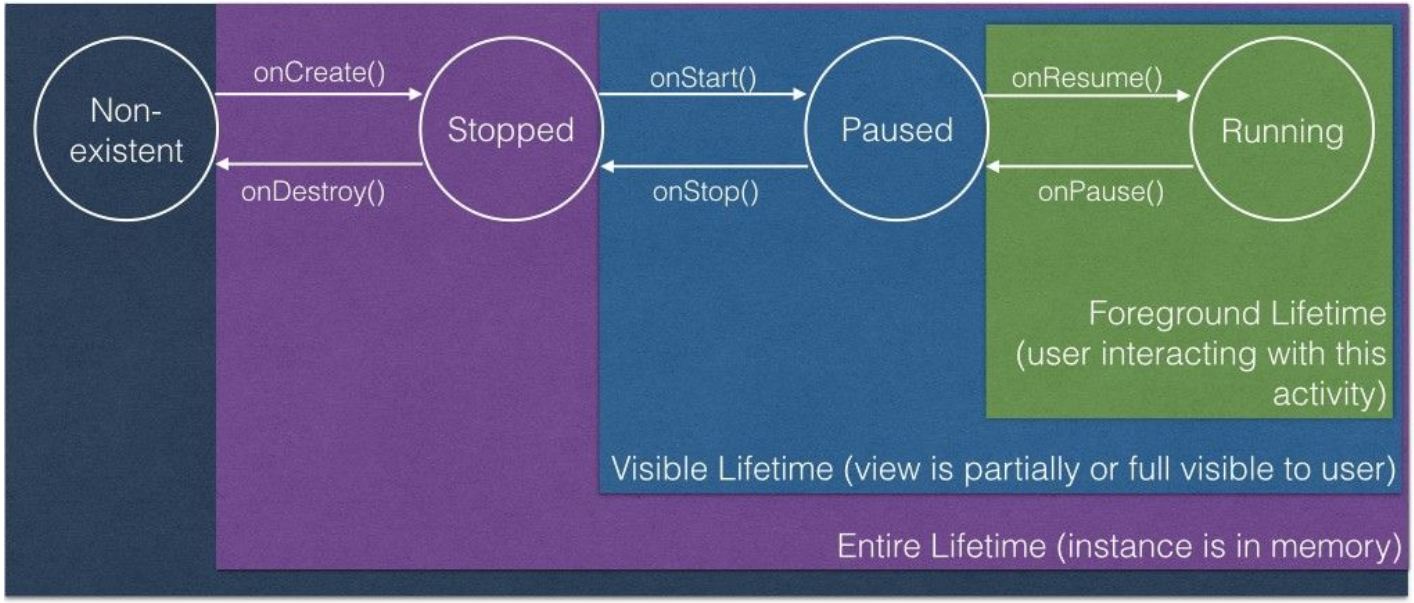


- Activities in the stack are pushed and popped from the stack on a Last-In First-Out (LIFO) basis.
  - Pushed onto the stack when started by the current activity.
  - Popped off when the user leaves it using the Back button.

Activity States



Activity Lifetime (I)



- **The entire lifetime** of an activity happens between the call to `onCreate` and the call to `onDestroy`.
  - Perform setup of "global" state (such as defining layout) in `onCreate`, and release all remaining resources in `onDestroy`.
  - For example, if your activity has a thread running in the background to download data from the network, it might create that thread in `onCreate` and then stop the thread in `onDestroy`.

Activity Lifetime (II)

- **The visible lifetime** of an activity happens between the call to `onStart` and the call to `onStop`.
  - The user can see the activity on-screen and interact with it.
  - The system might call `onStart` and `onStop` multiple times during the entire lifetime of the activity, as the activity alternates between being visible and hidden to the user.
- **The foreground lifetime** of an activity happens between the call to `onResume` and the call to `onPause`.
  - The activity is in front of all other activities on screen and has user input focus.
  - An activity can frequently transition in and out of the foreground
    - For example, `onPause` is called when the device goes to sleep or when a dialog appears.
    - The code in these two methods should be fairly lightweight in order to avoid slow transitions that make the user wait.

Using lifecycle callbacks (I)

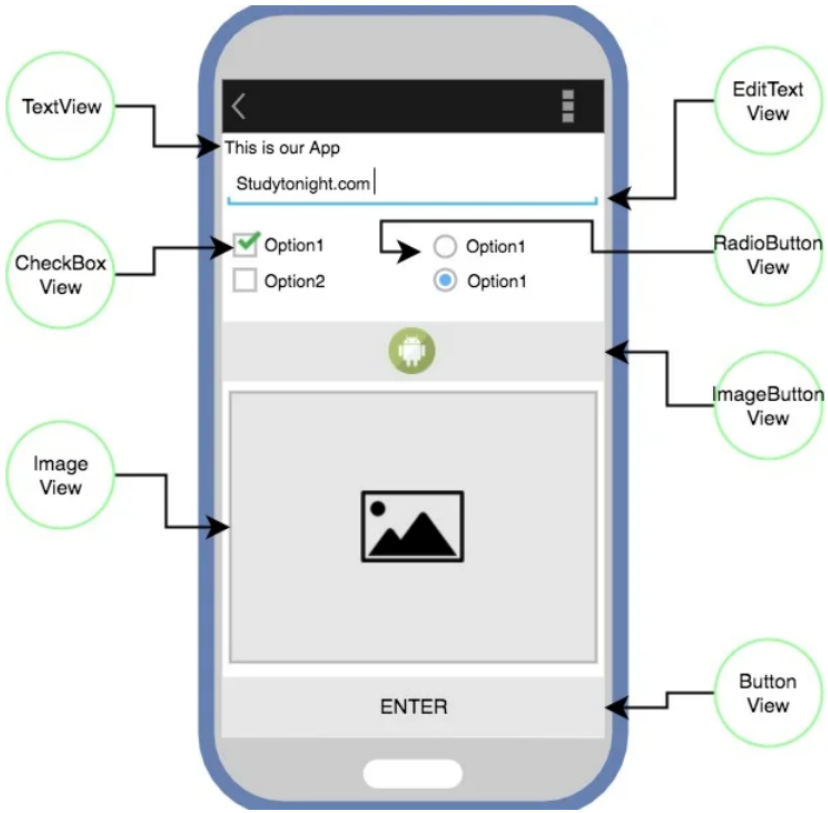
- Use the `onCreate` callback to:
  - Initialize the essential components of your activity.
  - (Most important) call `setContentView` method to define the layout for the activity.
- Use the `onPause` callback to:
  - Stop animations or other ongoing actions that could consume CPU.
  - Commit unsaved changes, but only if users expect such changes to be permanently saved when they leave (such as a draft email).
  - Release system resources, such as broadcast receivers, handles to sensors (like GPS), or any resources that may affect battery life while your activity is paused and the user does not need them.

Using lifecycle callbacks (II)

- Use `onResume` callback to:
  - Initialize components that you release during `onPause`
  - Perform any other initializations that must occur each time the activity enters the Resumed
- Use `onStop` callback to:
  - Perform larger, more CPU intensive shut-down operations, such as writing information to a database.

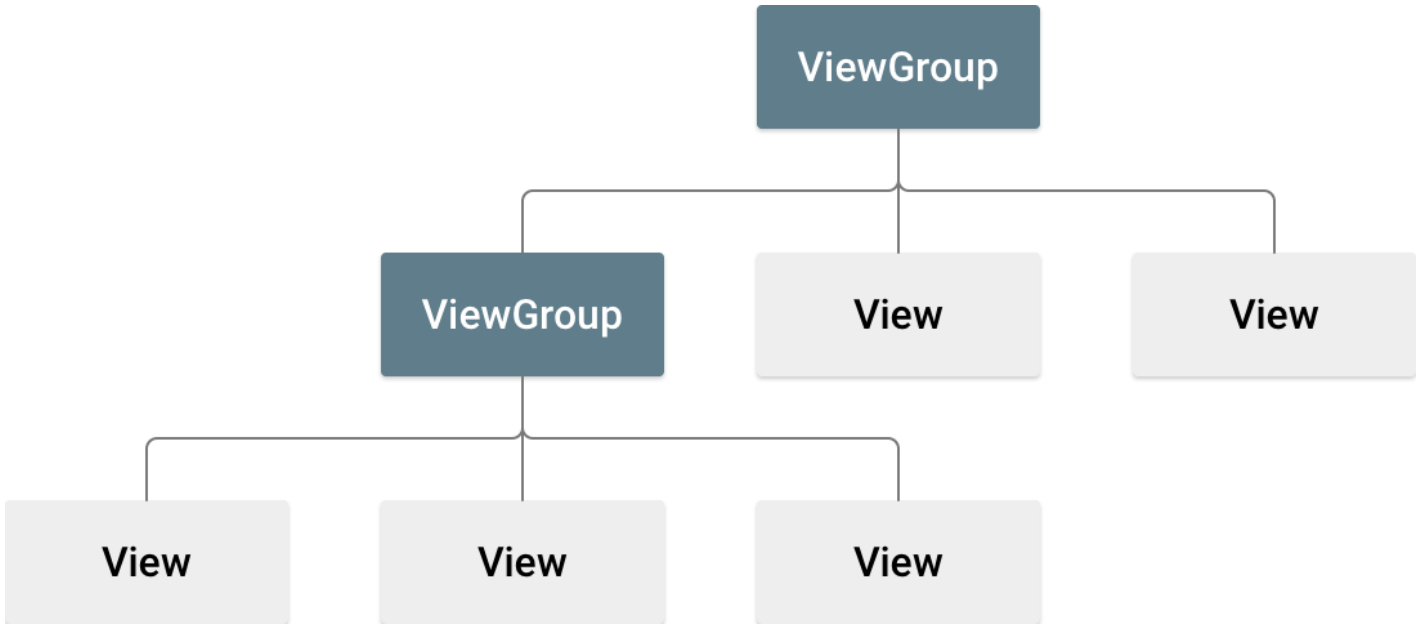
Views

- A View is a simple building block of a user interface.
- It is a small rectangular box that can be `TextView`, `EditText`, or even a `Button`.
- It occupies the area on the screen in a rectangular area and is responsible for drawing and event handling.
- `View` is a superclass of all the graphical user interface components.



ViewGroup

- `ViewGroup` provides an invisible container to hold the views or layouts.
- `ViewGroup` and `View` instances work together as a container for Layouts.



Layout

- A layout **defines the visual structure** for a user interface, such as the UI for an `Activity`.
- Layout basically refers to the arrangement of `View` elements.

  - Declare UI elements in XML (**Recommended**).
    - Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses, such as those for widgets and layouts.
  - Instantiate layout elements at runtime.
    - Your application can create View and ViewGroup objects (and manipulate their properties) programmatically.

Layouts: Atribute ID

- Any `View` object may have **an integer ID** associated with it, to uniquely identify the View within the tree.
- When the app is compiled, this ID is referenced as an integer, but the ID is typically assigned in the layout XML file as a string, in the `id` attribute.

```
android:id="@+id/my_button"
```

  - The at-symbol (@) at the beginning of the string indicates that the XML parser should parse and expand the rest of the ID string and identify it as an ID resource.
  - The plus-symbol (+) means that this is a new resource name that must be created and added to our resources (in the `R.java` file).

Linking a Layout to an Activity

- An `Activity` class loads all the UI component using the XML file available in `res/layout` folder of the project by invoking `setContentView` method.
  - `.../MainActivity.java`

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

  - `res/layout/activity_main.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
  ...
  android:layout_width="match_parent"
  android:layout_height="match_parent">
    <ListView
      android:layout_width="match_parent"
      android:layout_height="wrap_content"/>
</RelativeLayout>
```

Mapping a UI component on Java Variable

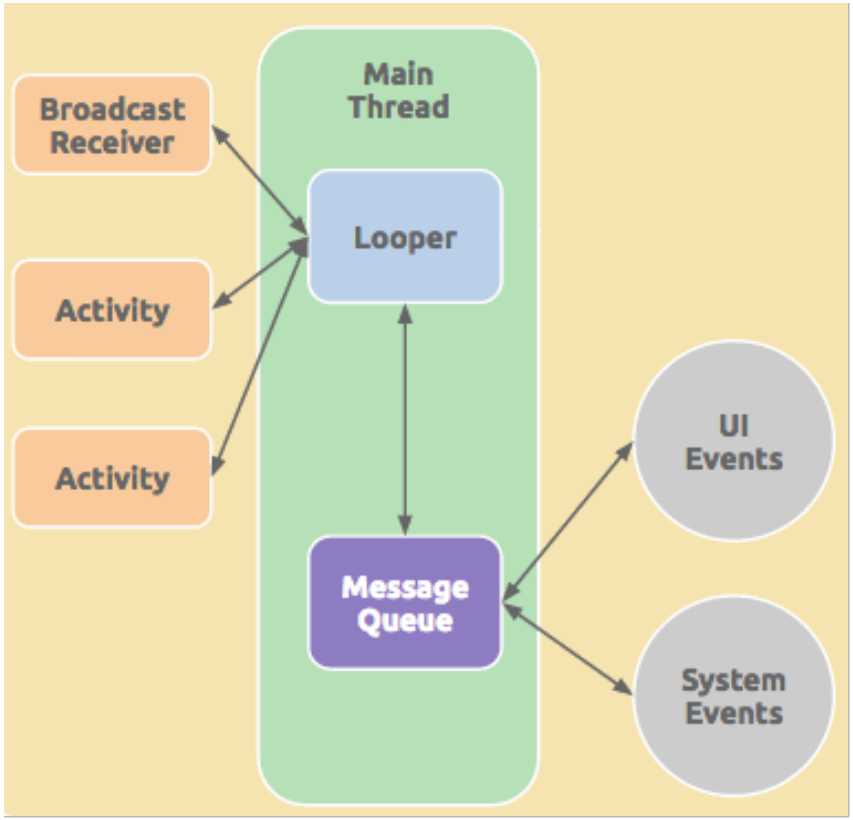
- `findViewById(int)`
  - Retrieves the UI component (loaded by `setContentView`).
  - Finds a view that was identified by the `id` attribute from the XML.

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

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
  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">
    <ListView
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:id="@+id/listView" />
</RelativeLayout>
```

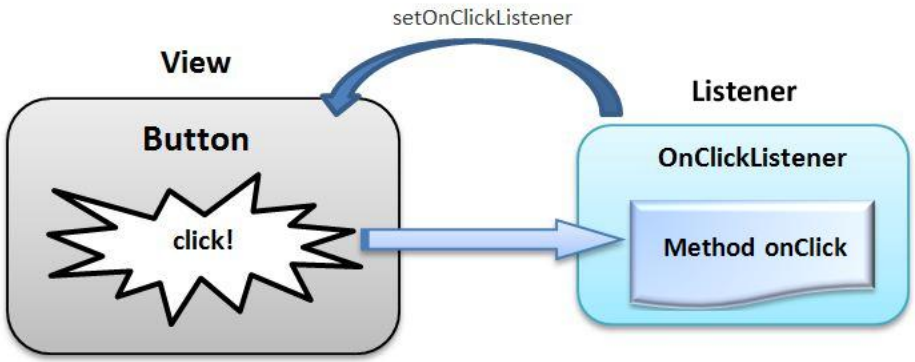
Handling events and messages

- The Android framework maintains an event/message queue into which events are placed as they occur.
- Events/messages are removed from the queue on a First-In, First-Out (FIFO) basis.



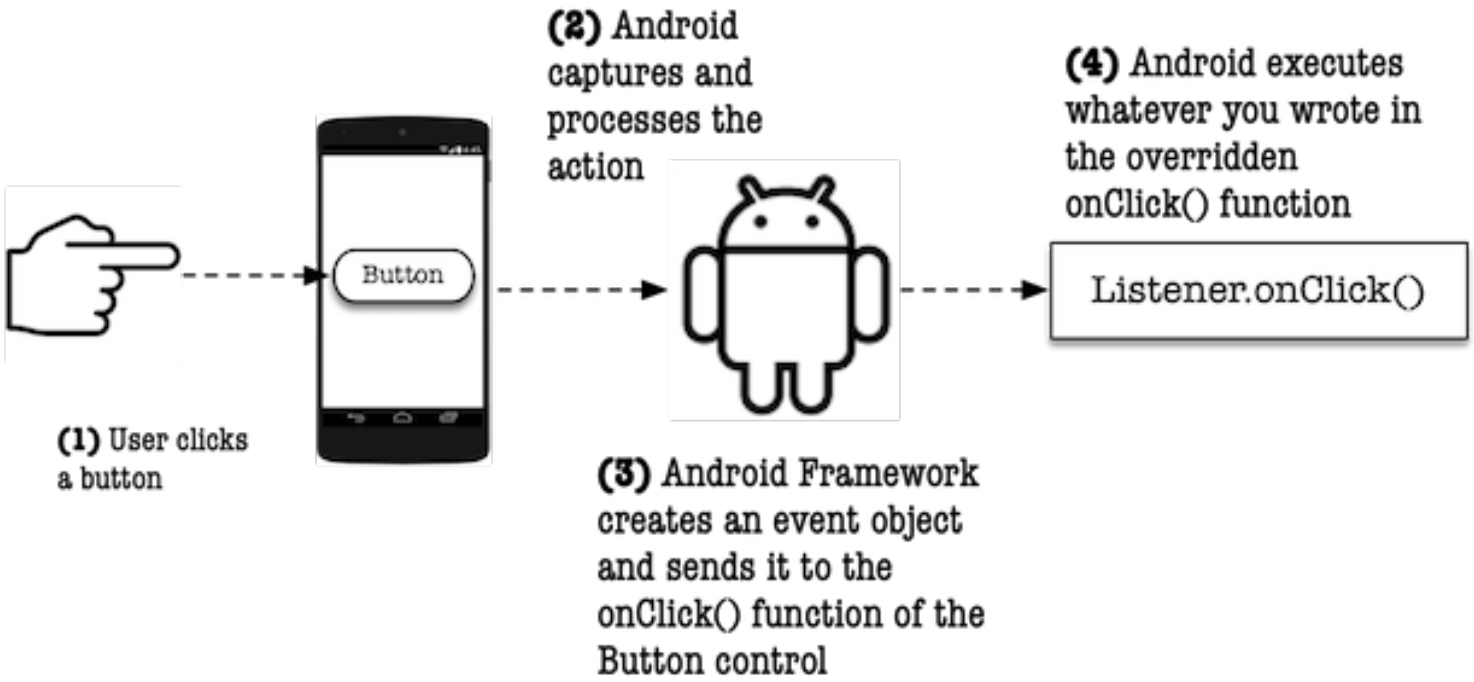
Event listeners

- **Event Listeners**
  - It is an interface in the `View` class that contains a callback method.
  - Callback method is invoked when the `View` is triggered by user interaction.
- **Event Listeners Registration**
  - It is the process by which an Event Handler gets registered with an Event Listener so that the handler is called when the Event Listener fires the event.
- **Event Handlers**
  - It is the callback method that actually handles the event.



Event Handling (I)

- The user interacts with an app by touching, clicking, swiping or typing something.
- The Android framework captures, stores, processes and sends these actions to the app as event objects.

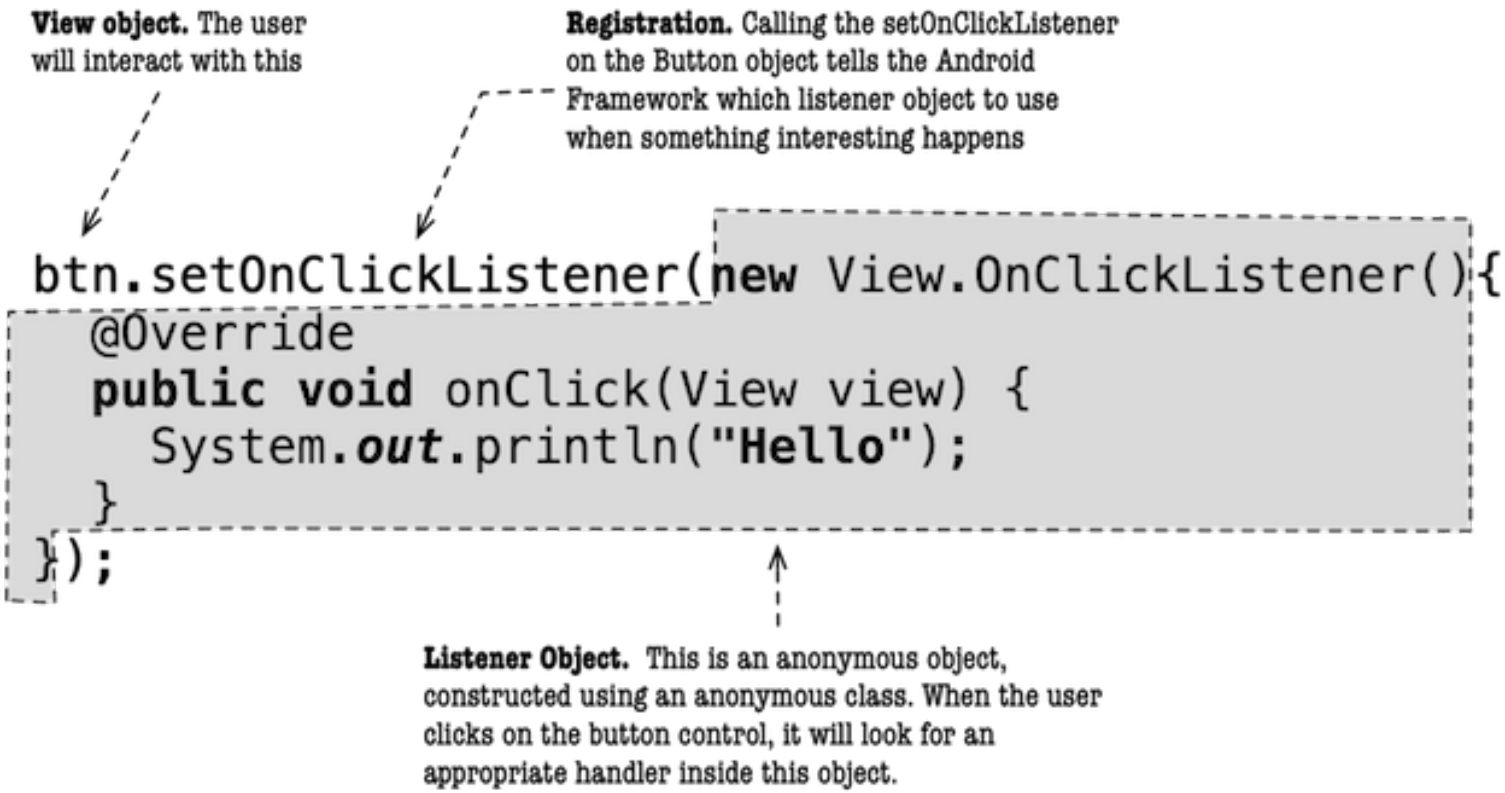


Event Handling (II)

1. Create a listener object that is appropriate for the event you want the app to react to;
   - If you want to respond to a click event, then use an the `OnClickListener` object.
2. Override the abstract methods of the listener.
   - In case of the `OnClickListener`, the abstract method you need to override is the `onClick()` method
3. Bind the listener object to a `View` object, like a `Button`

```
Button btn = (Button) findViewById(R.id.button);
btn.setOnClickListener(new View.OnClickListener(){
  @Override
  public void onClick(View view) {
    // statement;
  }
});
```

Event Handling (III)



Bibliography

Resources

- "Mastering Android Application Development", by Antonio Pachon Rui, 2015
- `https://developer.android.com/index.html`
- `http://simple.sourceforge.net/home.php` `http://simple.sourceforge.net/download/stream/doc/tutorial/tutorial.php`