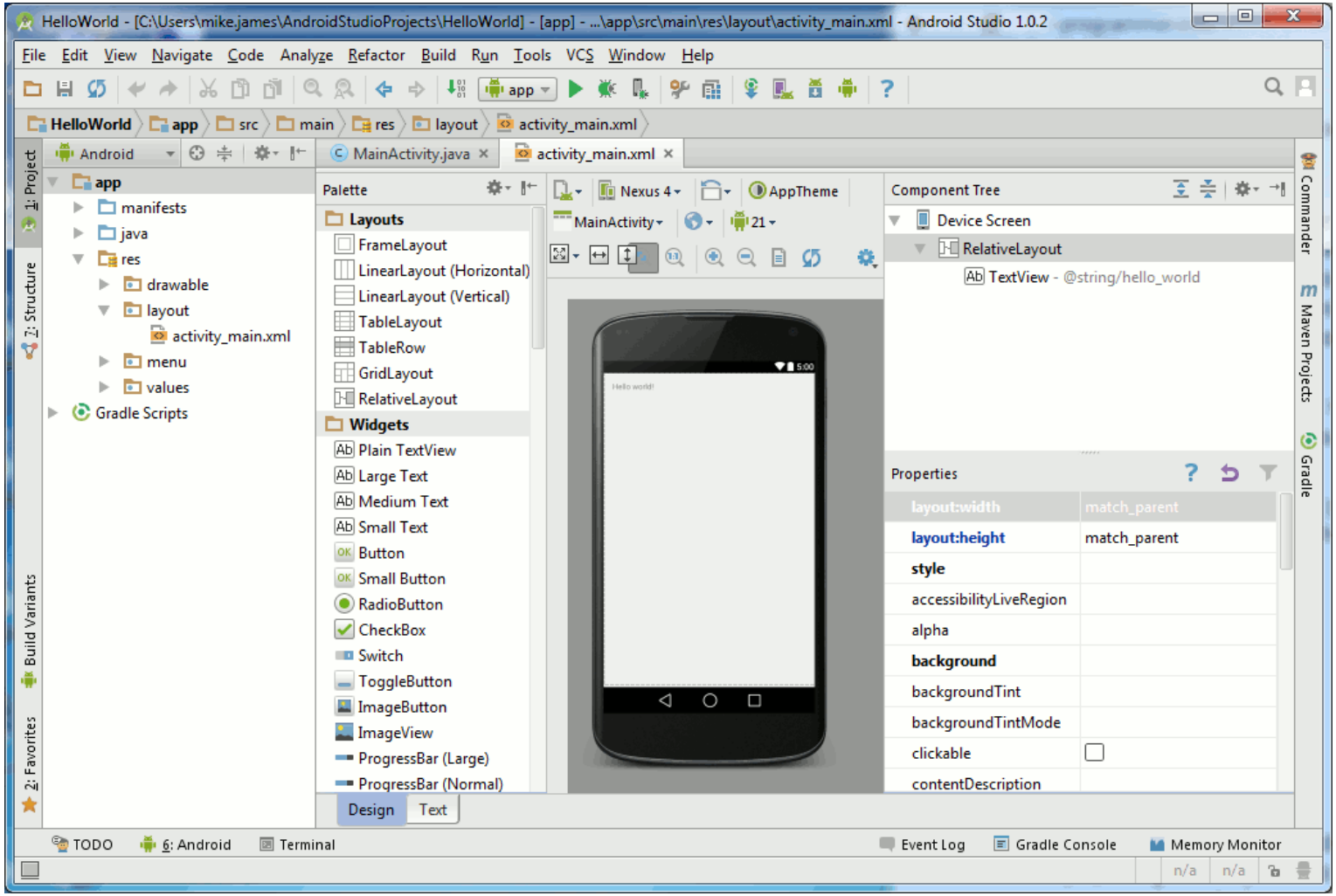
CET325

Advanced Mobile Development

Lecture 3B

Android Building Blocks

- Agenda
  - Anatomy of an Android Studio Project
  - Application Components
    - Activities
  - Event Listeners



Project window

Component Tree

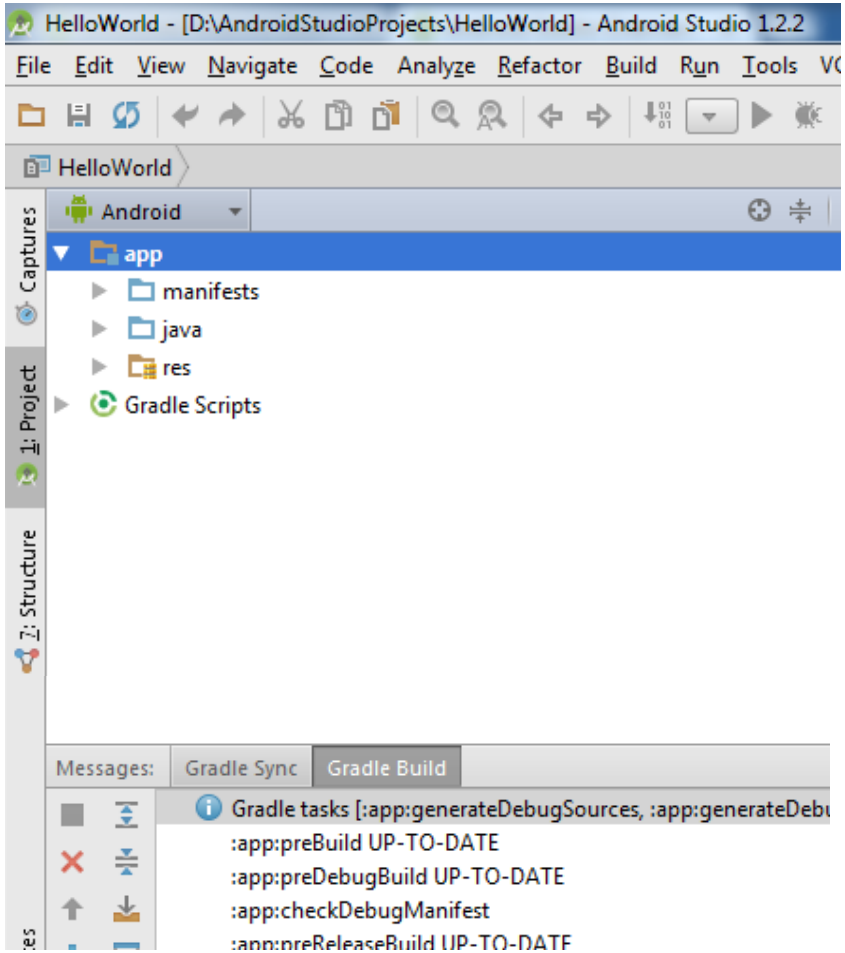
Properties

Designer (palette and layout window)

lifechanging

University of Sunderland

Android Project View



- Quick access to key files
  - **java**/ - Source files for the module.
  - **manifests**/ - Manifest files for the module.
  - **res**/ - Resource files for the module.
  - **Gradle** Scripts/ - Gradle build and property files.

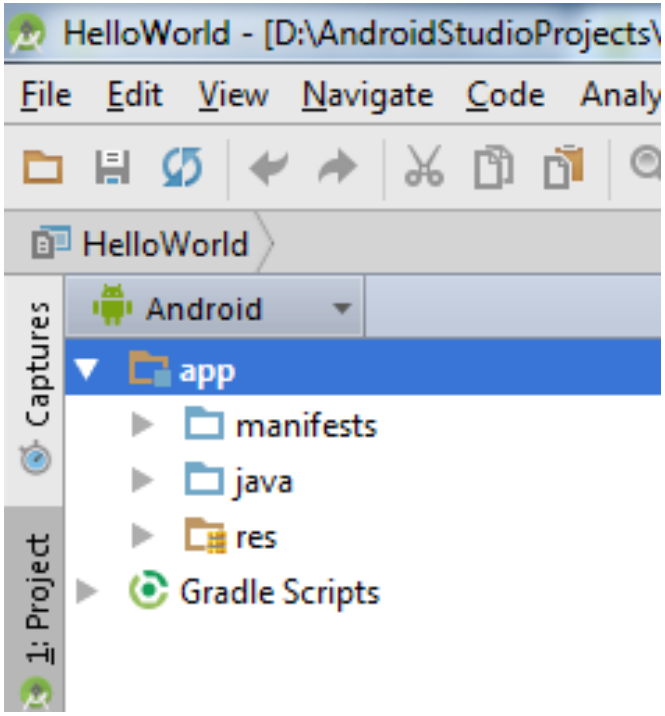
lifechanging University of Sunderland

Java Files - Source Code

- Core application logic.

MyActivity is a **subclass** which inherits the fields and methods from **superclass** Activity

```
public class MyActivity extends Activity {
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
// get layout resource object and make it the
// current view, i.e. display it on screen
setContentView(R.layout.main);
}
}
```



Java Files - Source Code

- Core application logic.

```
public class MyActivity extends Activity {
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
// get layout resource object and make it the
// current view, i.e. display it on screen
setContentView(R.layout.main);
}
}
```

onCreate method is called when your app is run and it is expected to create the view and do whatever the Activity is concerned with

onCreate event handler has as argument a Bundle object called **savedInstanceState**. This is intended to preserve state information between invocations of your app

Java Files - Source Code

- Core application logic.

```
public class MyActivity extends Activity {
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
// get layout resource object and make it the
// current view, i.e. display it on screen
setContentView(R.layout.main);
}
}
```

setContentView gets the resource object that represents the layout as defined by the XML file created by the designer and makes it the current **ContentView** i.e. it is what is displayed on the screen

The main role of the resource object **R** is to form a link between your Java code and the resources that have been created as XML files by the designer

Java Files - Source Code

- Core application logic.

```
public class MyActivity extends Activity {
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
// get layout resource object and make it the
// current view, i.e. display it on screen
setContentView(R.layout.main);
}
}
```

In this case **R.layout.activity_main** returns an integer value that allows the **setContentView** method to find the activity_main XML layout file

setContentView gets the resource object that represents the layout as defined by the XML file created by the designer and makes it the current **ContentView** i.e. it is what is displayed on the screen

The main role of the resource object **R** is to form a link between your Java code and the resources that have been created as XML files by the designer

app/src/main/java/com.mycompany.myfirstapp/ MyActivity.java

- This is the class definition for an activity called *activity_my.xml*.
- If it is the only activity in the application, it will be the LAUNCHER activity, and this will be stated in your manifest file.
- When you build and run the app, the Activity class starts the activity *onCreate()* method.

app/src/main/AndroidManifest.xml

- Describes the fundamental characteristics of the app
- Glues everything together, describes main building blocks, the API level, and application permissions.
- Details
  - The package under which the application is registered
  - The version of the app, both code and name
  - The Android SDK that it is targeting and requiring in order to run.
  - The permissions that it users in order to run (the user gets asked to grant them at install time)
  - Custom permissions that it declares and may require from other components.
  - The application and all its main building blocks: activities, services, providers and receivers.

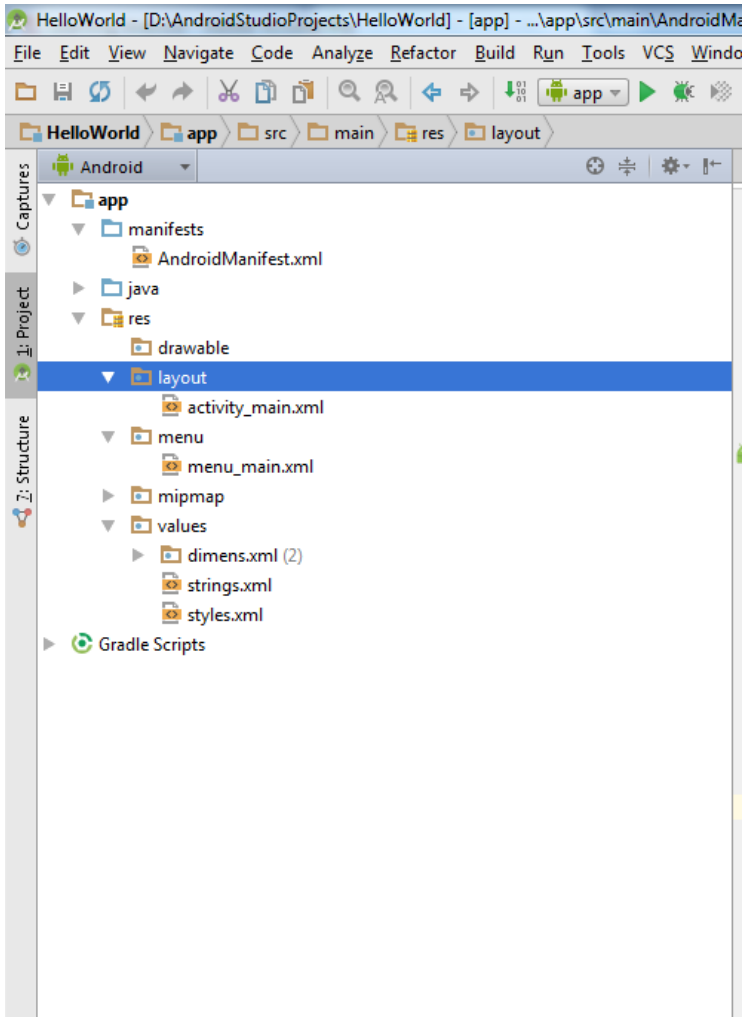
Android Manifest File

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

    <uses-sdk android:minSdkVersion="8" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Resource Files (/res subdirectories)

- Layout
  - Activity XML files (UI View)
- Menu
  - Menu xml files
- String Resources
  - Press ALT + Enter to automatically generate resource strings
- Colour Definitions
- Drawables (Graphics)

Resource Files



- Strings.xml – contains all the text that your application uses.
  - Think of it as a series of name-value pairs
  - Easily extend to multiple languages, by providing multiple values for the same string resource.

lifechanging University of Sunderland

Resource Files - Layouts

- Defines the visual structure for a users interface.
- You can declare a layout in two ways
  - Declare UI elements in XML
  - Instantiate layout elements at runtime (Create view objects and manipulate their properties programmatically)
- Declaring UI elements in XML allows better separation of your application's presentation from the code that controls application behaviour.

Resource Files - Example Layout

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
                xmlns:tools="http://schemas.android.com/tools"
                android:layout_width="fill_parent"
                android:layout_height="match_parent"
              tools:context=".MainActivity">

    <EditText
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:id="@+id/editTextInput"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="64dp"/>

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:id="@+id/textViewResult"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        android:layout_marginBottom="103dp"/>

    <Button
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/mm_to_cm"
        android:id="@+id/buttonMmCm"
        android:layout_centerVertical="true"/>
</RelativeLayout>
```

Resource Files - Layout Attributes

- Each object you create within the layout supports their own variety of XML attributes.
- Some attributes are common irrespective of the type of component you add
  - others are considered ‘layout parameters’
- Example attributes:
  - ID, eg `android:id="@+id/my_button"`
  - Layout Parameters – XML layout attributes named layout_something, eg layout_width and layout_height
  - position
  - Size, padding, margins

Resource Files - Layout Attributes

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

- The **id** allows you to uniquely identify the view within the tree
- The **@** 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 **+** symbol means that this is a new resource name that must be created and added to our resources in the **R.java** file.

Build.gradle

- Android Studio uses Gradle to compile and build applications.
- There is a *build.gradle* file for each module of your project, as well as one for the entire project.
- This is where your application's build dependencies are set, including defaultConfig settings:
  - `compiledSdkVersion`
  - `applicationId`
  - `minSdkVersion`
  - `targetSdkVersion`

Build.gradle (app)

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 23
    buildToolsVersion '24.0.3'

    defaultConfig {
        applicationId "com.example.paolo.myapplication"
        minSdkVersion 17
        targetSdkVersion 23
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    dependencies {
        compile fileTree(dir: 'libs', include: ['*.jar'])
        compile 'com.android.support:appcompat-v7:23.4.0'
    }
}
```

Application Components

- We know how the IDE structures an application.
- We have met ‘Activities’
  - These are known as an **app component**.
  - Components are the essential building blocks of an Android app.
  - Each component is a different point through which the system can enter your app (Although not all are entry points for the user)
  - Components help define your application’s overall behaviour.

Application Components

- Types of core component:
  - **Activities** (represents a single screen in an app)
  - **Services** (performs operations in the background without UI)
  - **Content Providers** (manages a shared set of app data)
  - **Broadcast Receivers** (responds to system-wide broadcast announcements)
- Also of interest:
  - **Intents** (Messages sent among the building blocks)
- Components are usually declared in the application manifest.

Activity

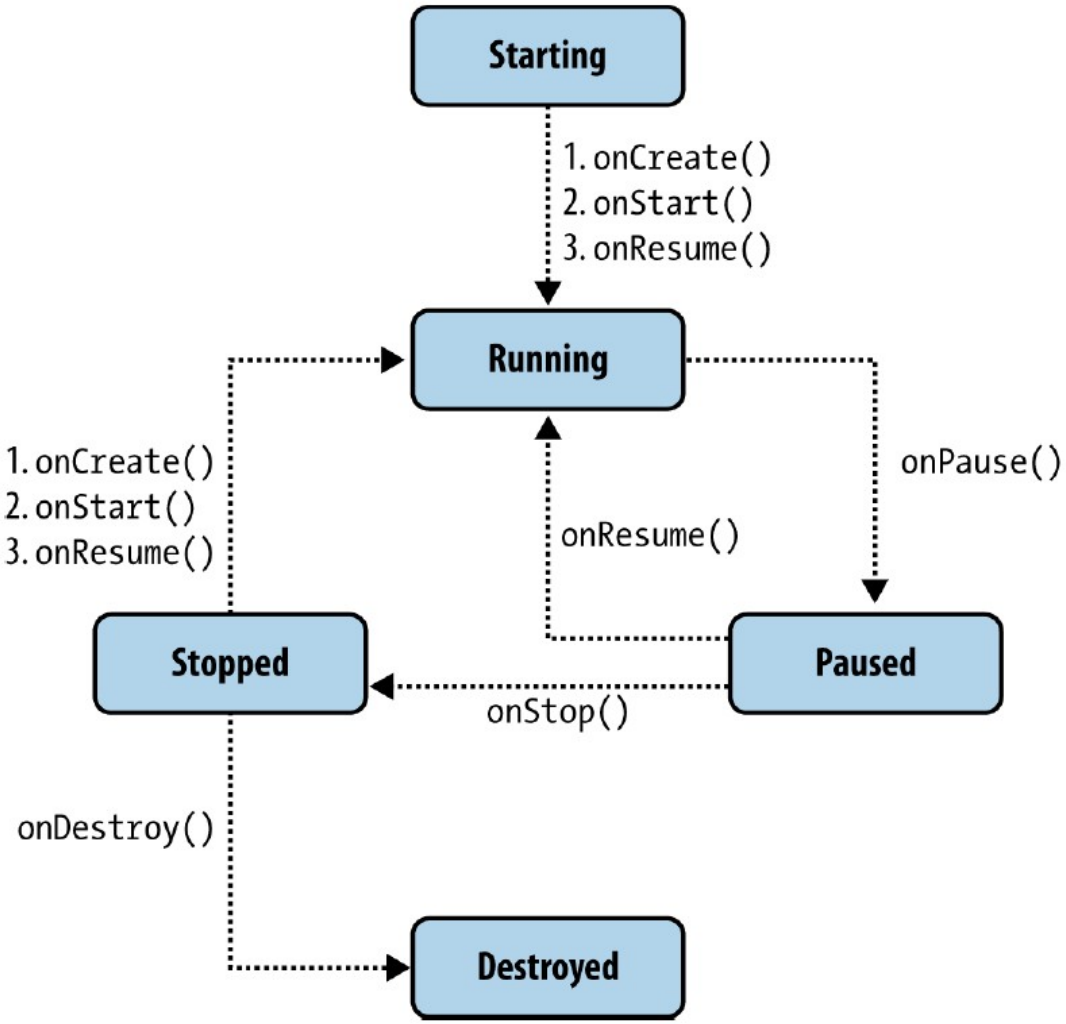
- An Activity is a single, focused thing that the user can do.
  - Equivalent to a Frame or Window in GUI toolkits.
  - Almost all activities interact with the user
- Each activity is given a default window to draw in
  - Activity class takes care of creating that window in which you can place your UI
    - setContentView(View)
- Written as a single class in Java
  - Applications main class extends the Activity Class.
- Activities are independent of each other
  - Allowed to work together with one activity initiating another.

Activity

- There are two methods almost all subclasses of Activity will implement.
  - *onCreate(Bundle)* – initialise activity
  - onPause() – add logic to deal with the user leaving your activity. If you have data persistence, this is where you will commit any changes which has been made

- All activity classes must have a corresponding <activity> declaration in their package's AndroidManifest.xml

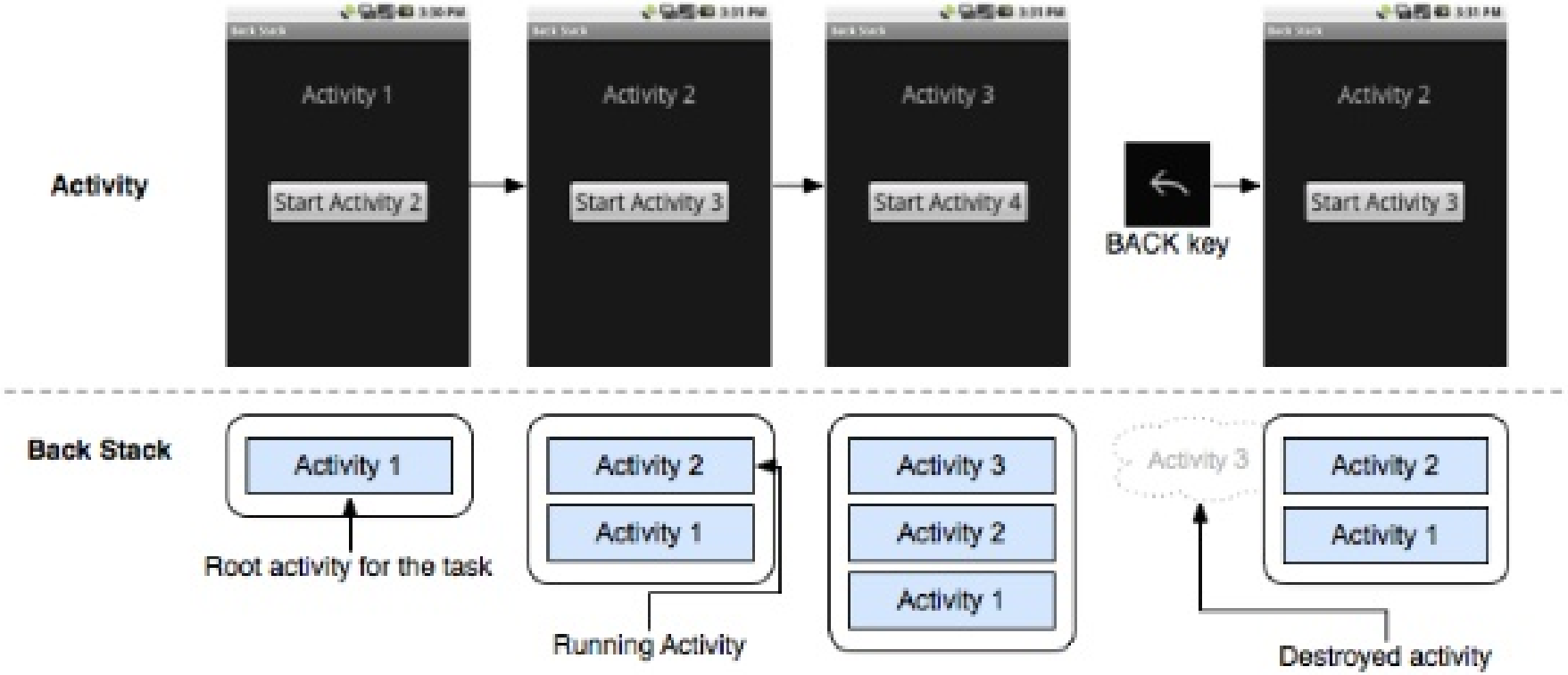
Activity



Activity Stack

- Android keeps navigation history of activities that the user has visited:
  - Activity Stack, or Back Stack
- Pressing back displays the previous Activity!
- User cannot go further than the last visit of home.

Activity Stack



Event Listeners - Tutorial

- Create UI component in the layout file (Within the activity, create a variable to represent that component)

```
TextView myTextView=(TextView) findViewById(R.id.textView1);
Button button = (Button) findViewById(R.id.button1);
EditText editText = (EditText) findViewById(R.id.editText1);
```

- The next step is to implement logic which would be executed when the user interacted with that component Android Event Handling

Event Listeners - Tutorial

```
public class MainActivity extends AppCompatActivity
implements View.OnClickListener

@Override
public void onClick(View v) {

    int id = v.getId();
    if(id == R.id.button1){
        String name = editText.getText().toString();
        textView.setText(name);
    }
}
```