Topic Covered in Todays Class

Unit 2:
Creating Android Applications and Activities

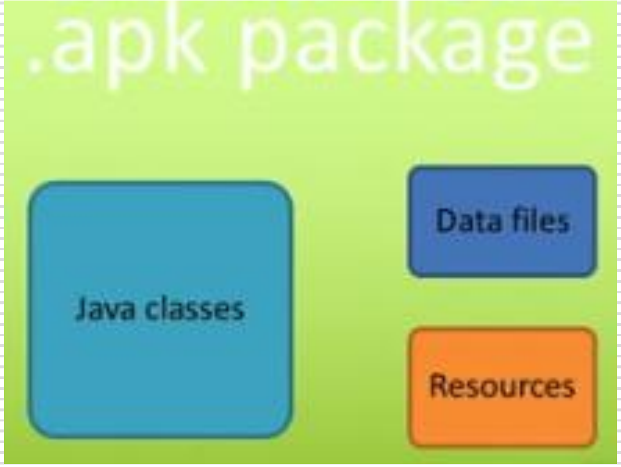
Creating your first “HelloWorld” Android app

Go through the slides uploaded on the course webpage under “Lab-Introduction-Starting-Android-Studio.pptx”

to create your first “HelloWorld” Android application

Android Application

- Android Applications are written in **Java**
- The compiled Java code (along with any data and resource files required by the application) is bundled by the aapt tool into an **Android package** (.apk)



Anatomy of Android Application

Before we run our app, we should be aware of a few directories and files in the Android project.

Each project module appears as a folder at the top level of the project hierarchy and contains these three elements at the top level:

java/ :

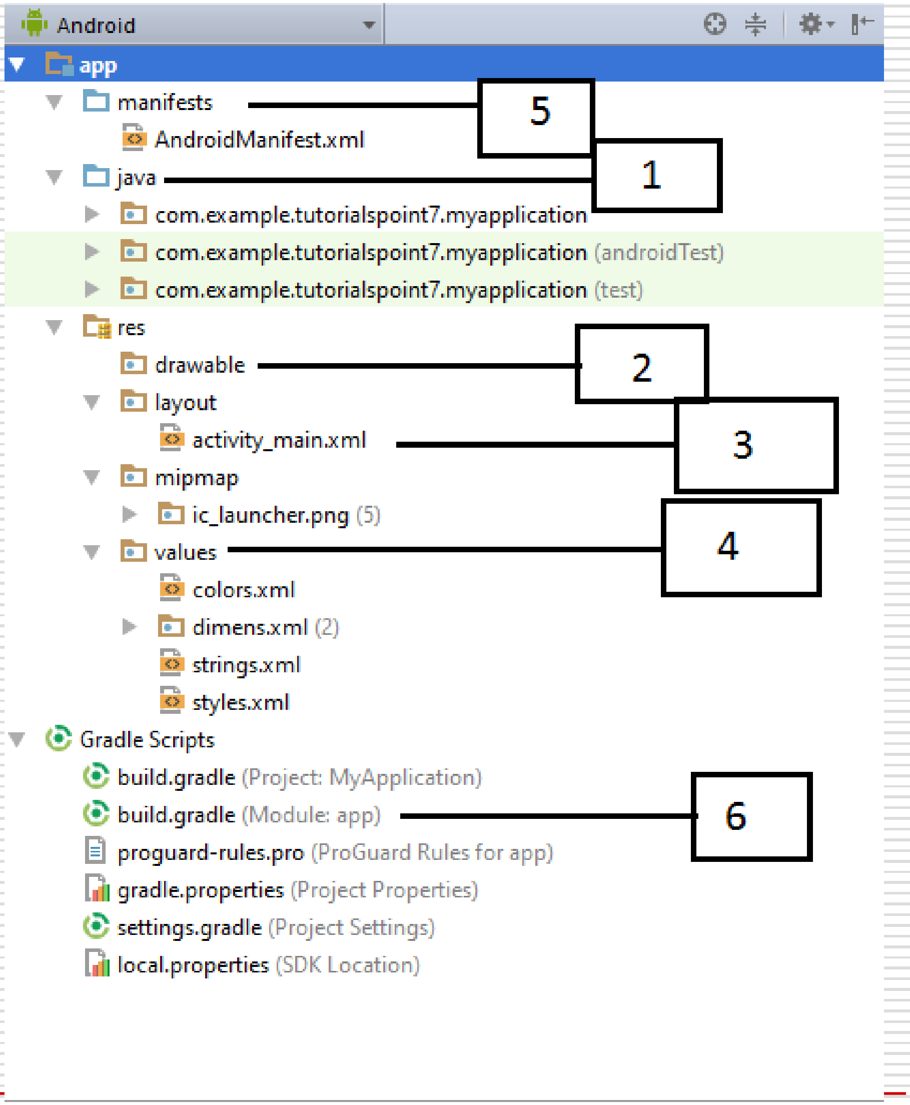
Source files for the module.

manifests/ :

Manifest files for the module

res/ :

Resource files for the module



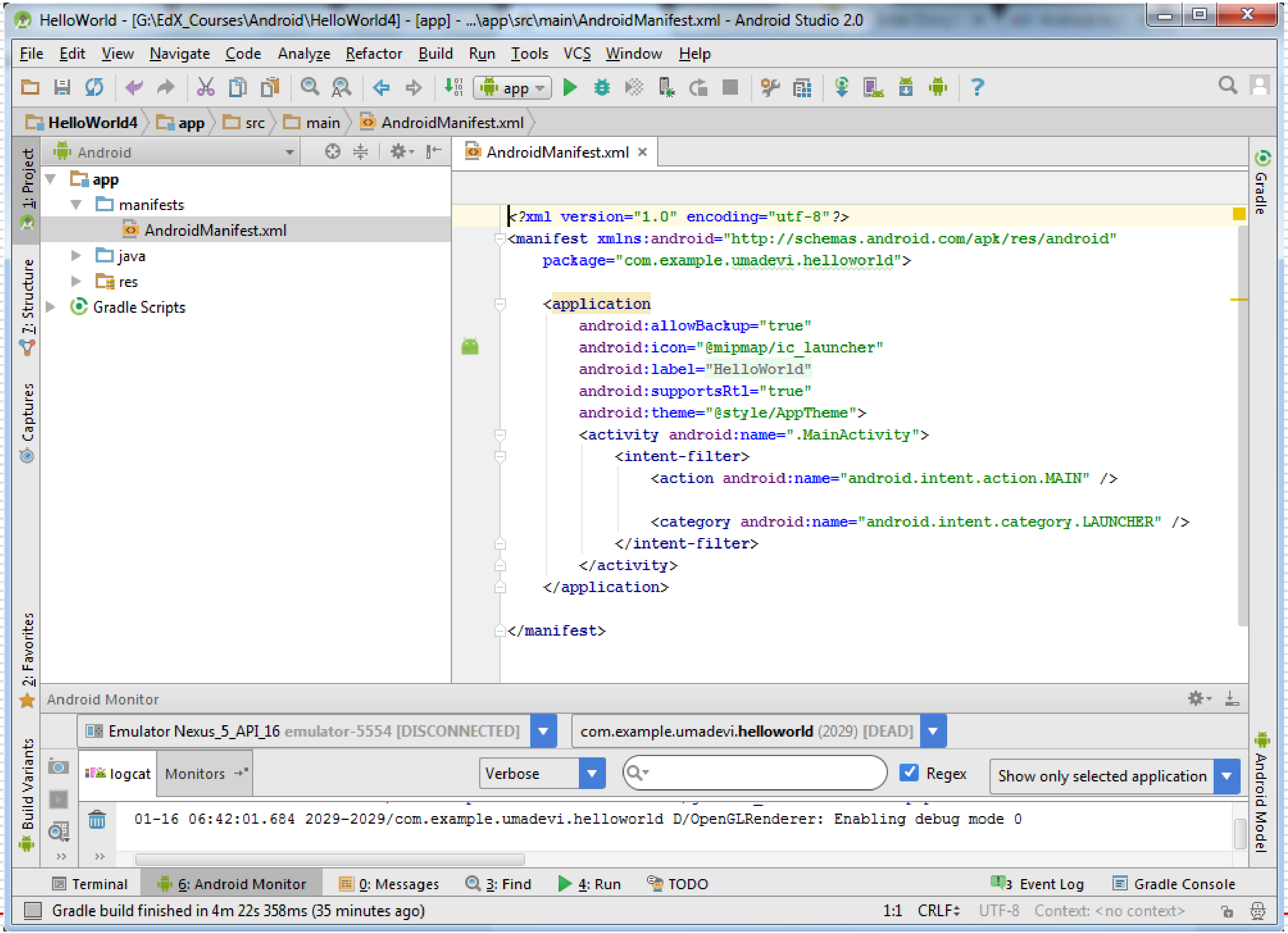
Android
app
manifests
AndroidManifest.xml
java
com.example.tutorialspoint7.myapplication
com.example.tutorialspoint7.myapplication (androidTest)
com.example.tutorialspoint7.myapplication (test)
res
drawable
layout
activity_main.xml
mipmap
ic_launcher.png (5)
values
colors.xml
dimens.xml (2)
strings.xml
styles.xml
Gradle Scripts
build.gradle (Project: MyApplication)
build.gradle (Module: app)
proguard-rules.pro (ProGuard Rules for app)
gradle.properties (Project Properties)
settings.gradle (Project Settings)
local.properties (SDK Location)
5
1
2
3
4
6

Anatomy of Android Application (Contd..)

Sl.No.	Folder, File & Description
1	**Java:** This contains the **.java** source files for your project. By default, it includes an *MainActivity.java* source file having an activity class that runs when your app is launched using the app icon.
2	**res/drawable-hdpi:** This is a directory for drawable objects that are designed for high-density screens.
3	**res/layout:** This is a directory for files that define your app's user interface.
4	**res/values:** This is a directory for other various XML files that contain a collection of resources, such as strings and colours definitions.
5	**AndroidManifest.xml:** This is the manifest file which describes the fundamental characteristics of the app and defines each of its components.
6	**Build.gradle:** This is an auto generated file which contains compileSdkVersion, buildToolsVersion, applicationId, minSdkVersion, targetSdkVersion, versionCode and versionName

App Fundamentals: “Hello World” App

AndroidManifest.xml



HelloWorld - [G:\EdX_Courses\Android\HelloWorld4] - [app] - ...\app\src\main\AndroidManifest.xml - Android Studio 2.0
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
app
HelloWorld4 app src main AndroidManifest.xml
Android
app
manifests
AndroidManifest.xml
java
res
Gradle Scripts
1: Project
2: Structure
Captures
2: Favorites
Build Variants
Gradle
Android Model
AndroidManifest.xml ×
Android Monitor
Emulator Nexus_5_API_16 emulator-5554 [DISCONNECTED]
com.example.umadevi.helloworld (2029) [DEAD]
logcat
Monitors
Verbose
☑ Regex
Show only selected application
Terminal
6: Android Monitor
0: Messages
3: Find
4: Run
TODO
3 Event Log
Gradle Console
1:1 CRLF÷ UTF-8 Context: <no context>

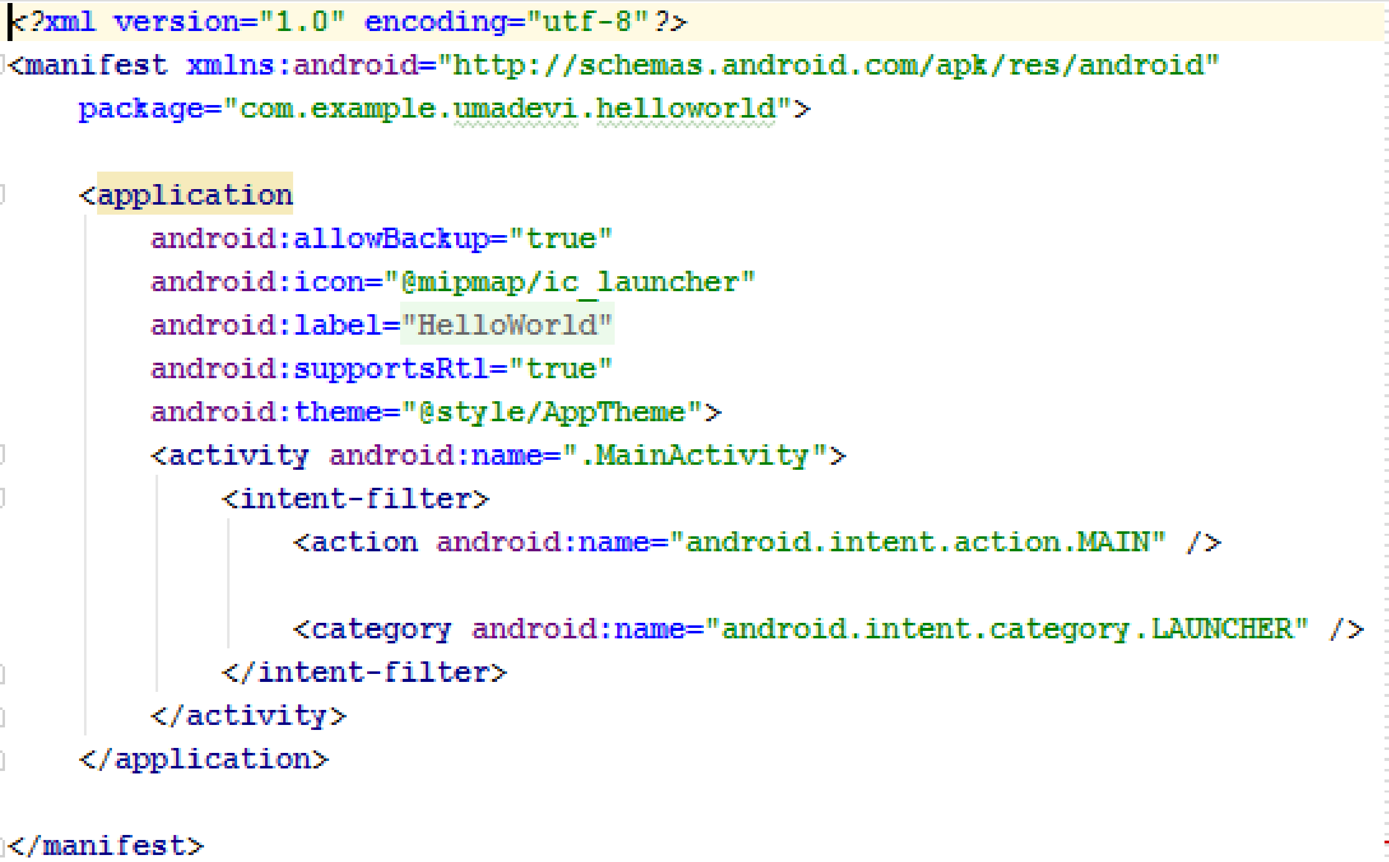
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.umadevi.helloworld">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="HelloWorld"
        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>
    </application>

</manifest>
```

```
01-16 06:42:01.684 2029-2029/com.example.umadevi.helloworld D/OpenGLRenderer: Enabling debug mode 0
```

Gradle build finished in 4m 22s 358ms (35 minutes ago)

AndroidManifest.xml

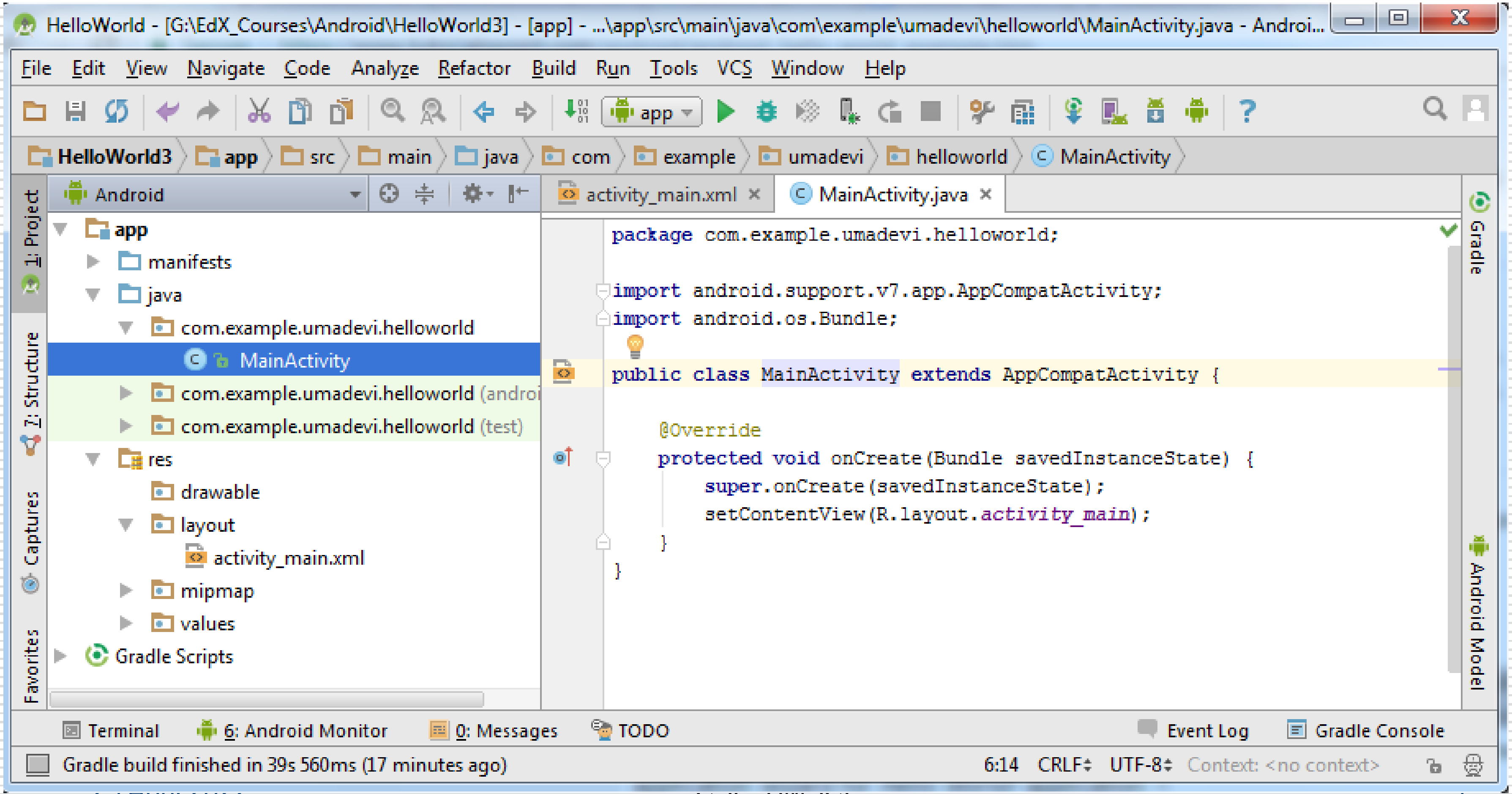
- Whatever component we develop as a part of the application, then all its components must be declared in a *manifest.xml* which resides at the root of the application project directory. This file works as an interface between **Android OS and application**, so if we do not declare the component in this file, then it will not be considered by the OS.



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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="HelloWorld"
        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>
    </application>

</manifest>
```

MainActivity.java

- The main activity code is a Java file **MainActivity.java**. This is the actual application file which ultimately gets converted to a Dalvik executable and runs your application.



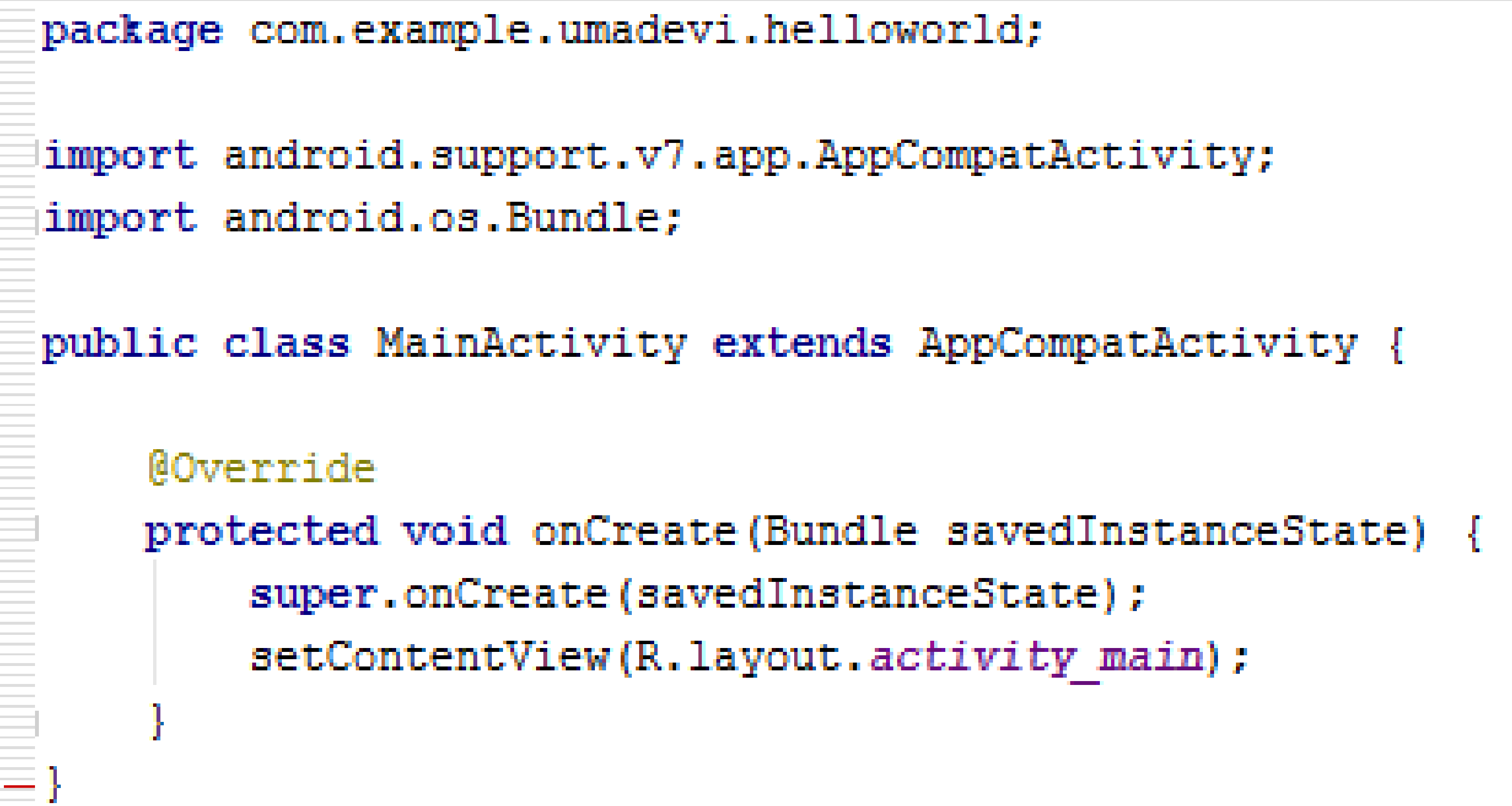
```
package com.example.umadevi.helloworld;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

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

Gradle build finished in 39s 560ms (17 minutes ago)

HelloWorld - [G:\EdX_Courses\Android\HelloWorld3] - [app] - ...\app\src\main\java\com\example\umadevi\helloworld\MainActivity.java - Androi...
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
app
HelloWorld3 app src main java com example umadevi helloworld MainActivity
Android
activity_main.xml
MainActivity.java
app
manifests
java
com.example.umadevi.helloworld
MainActivity
com.example.umadevi.helloworld (androi
com.example.umadevi.helloworld (test)
res
drawable
layout
activity_main.xml
mipmap
values
Gradle Scripts
1: Project
2: Structure
Captures
Favorites
Gradle
Android Model
Terminal
6: Android Monitor
0: Messages
TODO
Event Log
Gradle Console
6:14
CRLF
UTF-8
Context: <no context>

MainActivity.java

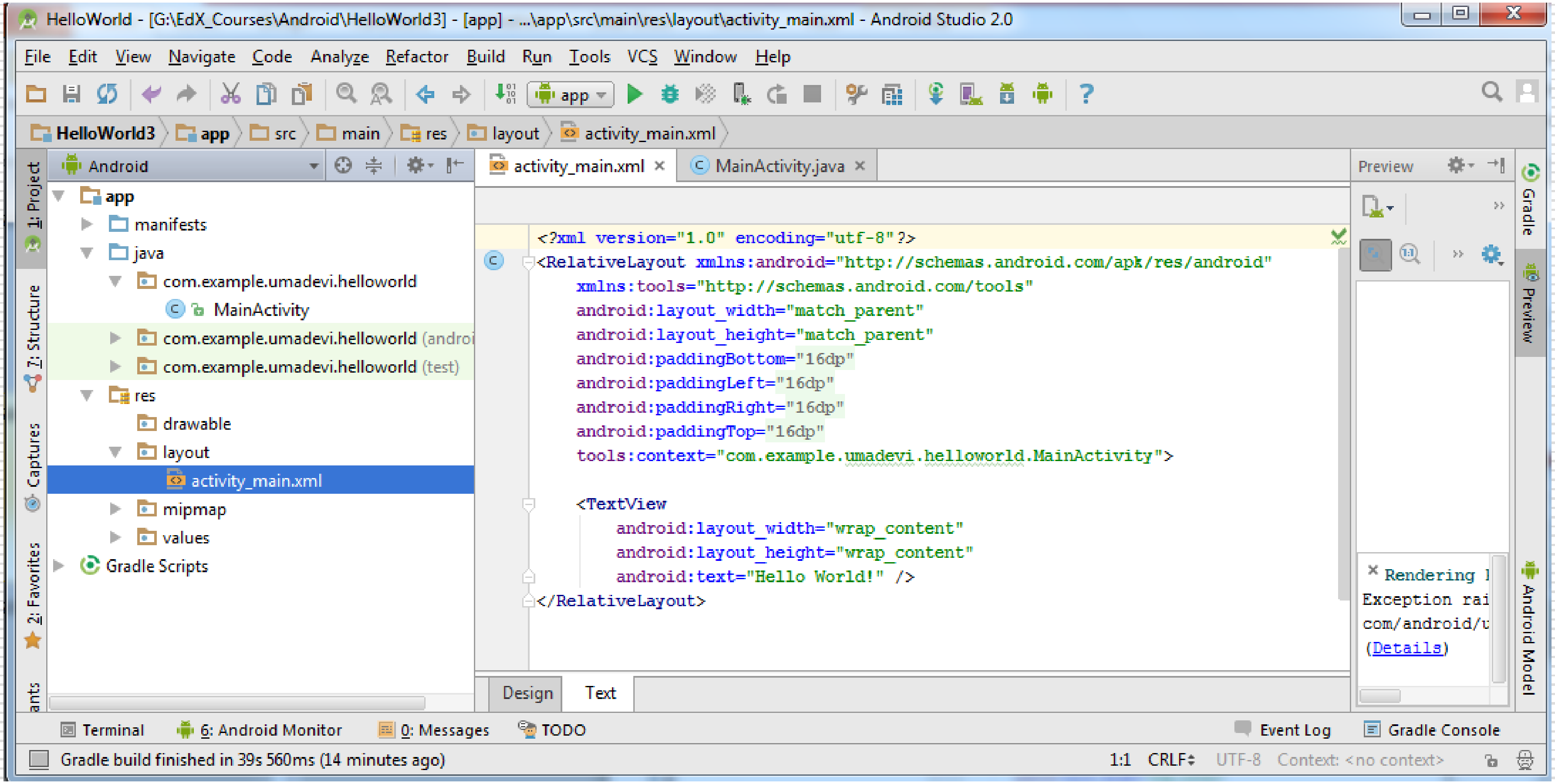
- Here, *R.layout.activity_main* refers to the *activity_main.xml* file located in the *res/layout* folder. The *onCreate()* method is one of many methods that are figured when an activity is loaded



```
package com.example.umadevi.helloworld;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

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

activity_main.xml

- The **activity_main.xml** is a layout file available in *res/layout* directory, that is referenced by your application when building its interface. You will modify this file very frequently to change the layout of your application.



HelloWorld - [G:\EdX_Courses\Android\HelloWorld3] - [app] - ...\app\src\main\res\layout\activity_main.xml - Android Studio 2.0

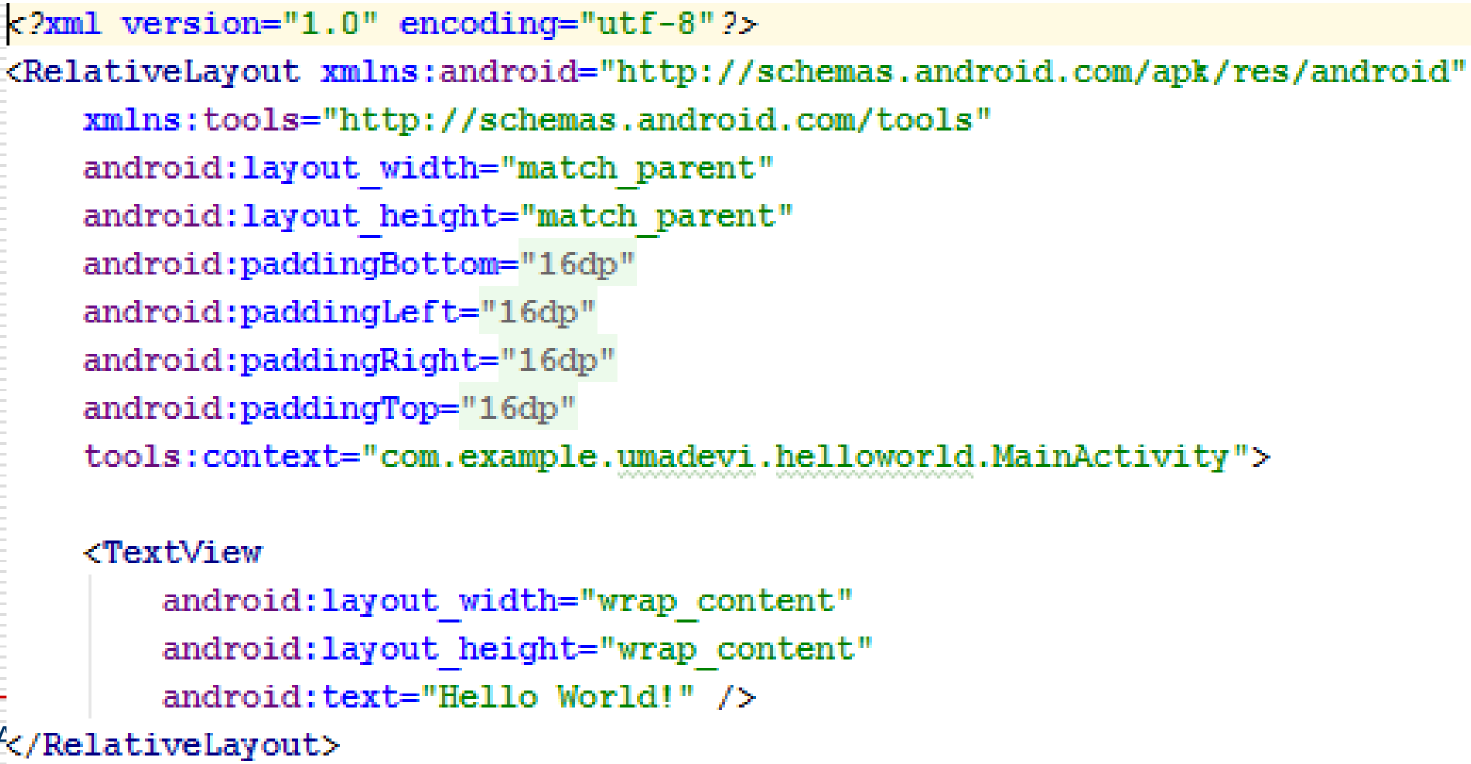
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    tools:context="com.example.umadevi.helloworld.MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!" />
</RelativeLayout>
```

```
× Rendering 
Exception rai
com/android/u
(Details)
```

Gradle build finished in 39s 560ms (14 minutes ago)   1:1 CRLF‡ UTF-8 Context: <no context>

activity_main.xml

- User interface is created for us in an XML file called **activity_main.xml**.
- *TextView* is an Android control used to build the GUI and it have various attributes like *android:layout_width*, *android:layout_height* etc which are being used to set its width and height etc..



```xml
<?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"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    tools:context="com.example.umadevi.helloworld.MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!" />
</RelativeLayout>
```

Question

What is Manifest.xml in android?

- □ It has information about layout in an application
- □ It has the information about activities in an application
- □ It has all the information about an application
- □ None of the above

Question: Answer

What is Manifest.xml in android?

- □ It has information about layout in an application
- □ It has the information about activities in an application
- □ **It has all the information about an application**
- □ None of the above

Explanation

- □ Manifest.xml is having information about application as number of components in your application, Activity information, service information, and icon about an application
- □ Each application has at least one Manifest file. Without manifest file we can't generate the APK file.

Question

What is activity_main.xml in android?

- □ It has information about layout in an application
- □ It has the information about activities in an application
- □ It has all the information about an application
- □ None of the above

Question: Answer

What is activity_main.xml in android?

- □ **It has information about layout in an application**
- □ It has the information about activities in an application
- □ It has all the information about an application
- □ None of the above

Question

What is MainActivity.java in android?

- □ It has information about layout in an application
- □ It has the information about activities in an application
- □ It has all the information about an application
- □ None of the above

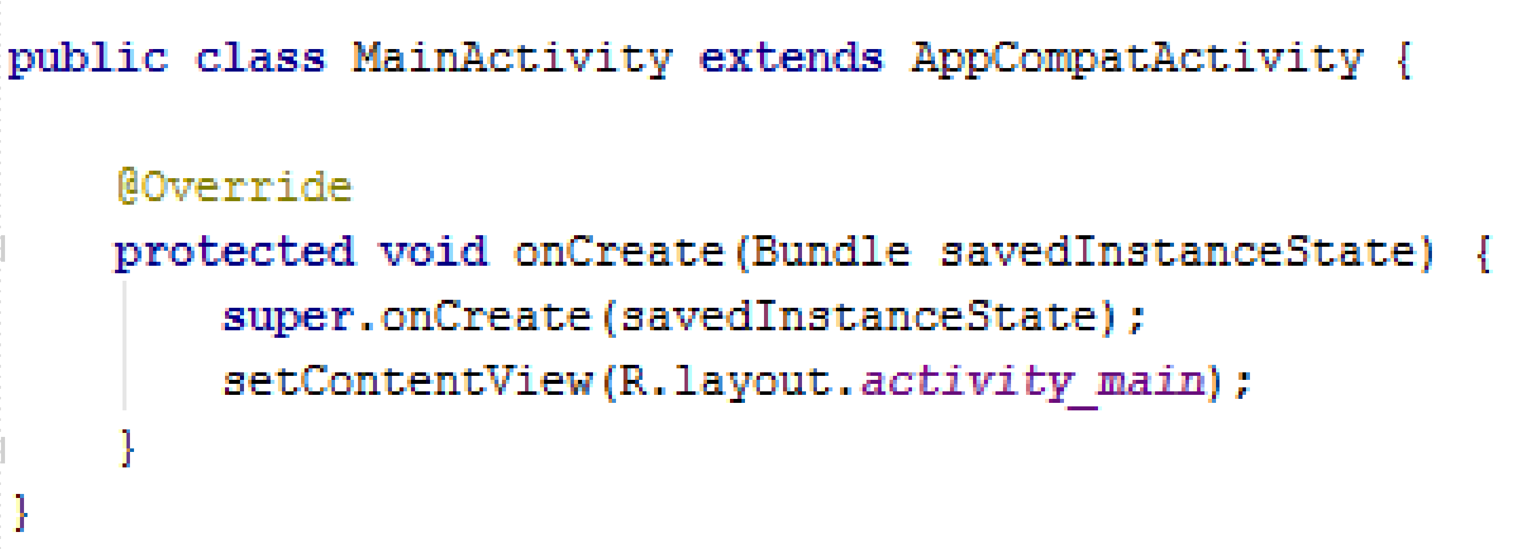
Question: Answer

What is MainActivity.java in android?

- □ It has information about layout in an application
- □ **It has the information about activities in an application**
- □ It has all the information about an application
- □ None of the above

What is Activity?

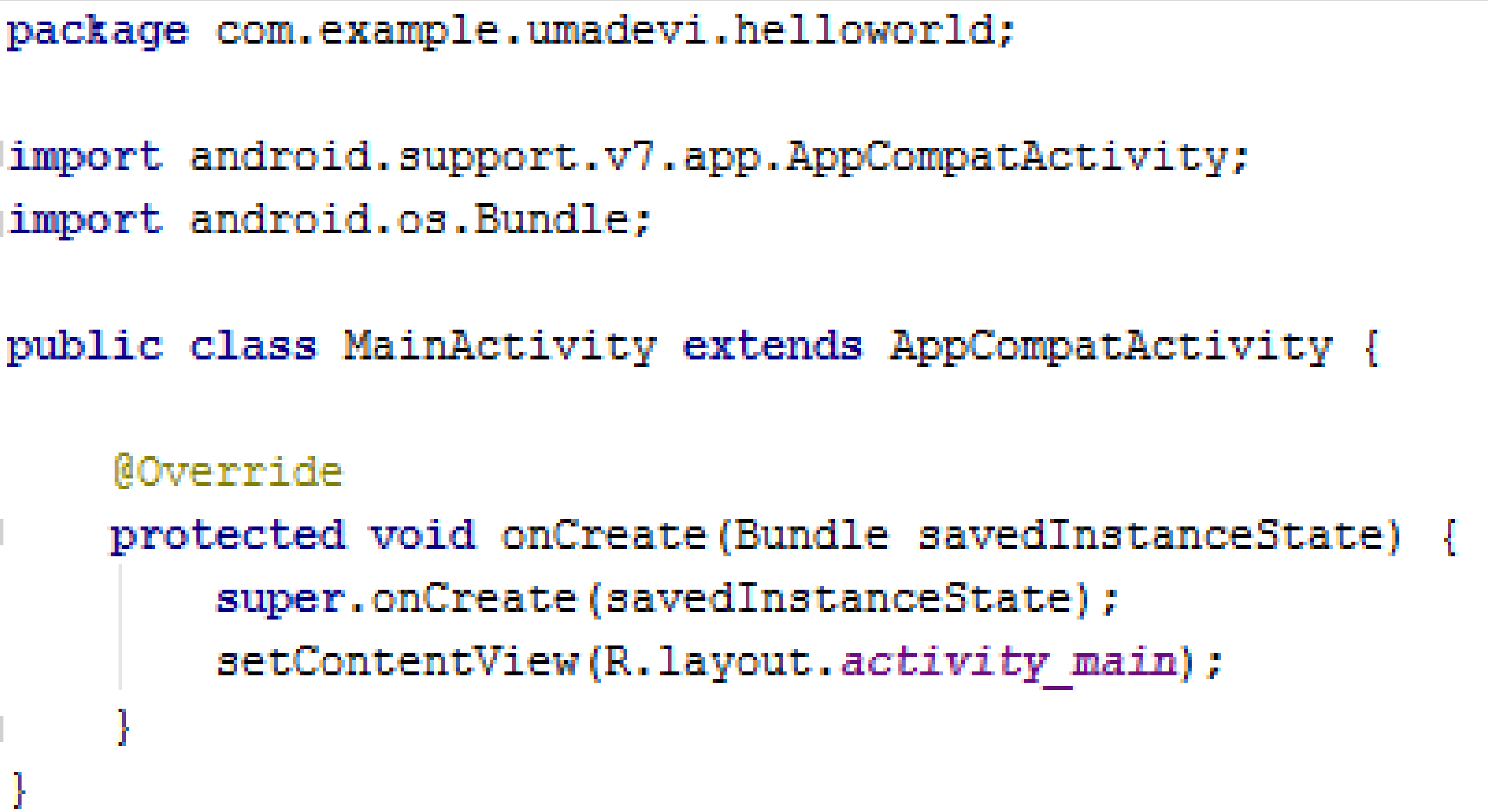
- **Activity** is a Java code that supports a screen or User Interface (UI). In other words, building block of the user interface is the activity. Activity class is a pre-defined class in Android and every application which has UI must inherit it to create window.
- For example, **MainActivity** class is subclassing a super class called as a app compact activity



```java
public class MainActivity extends AppCompatActivity {

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

MainActivity.java

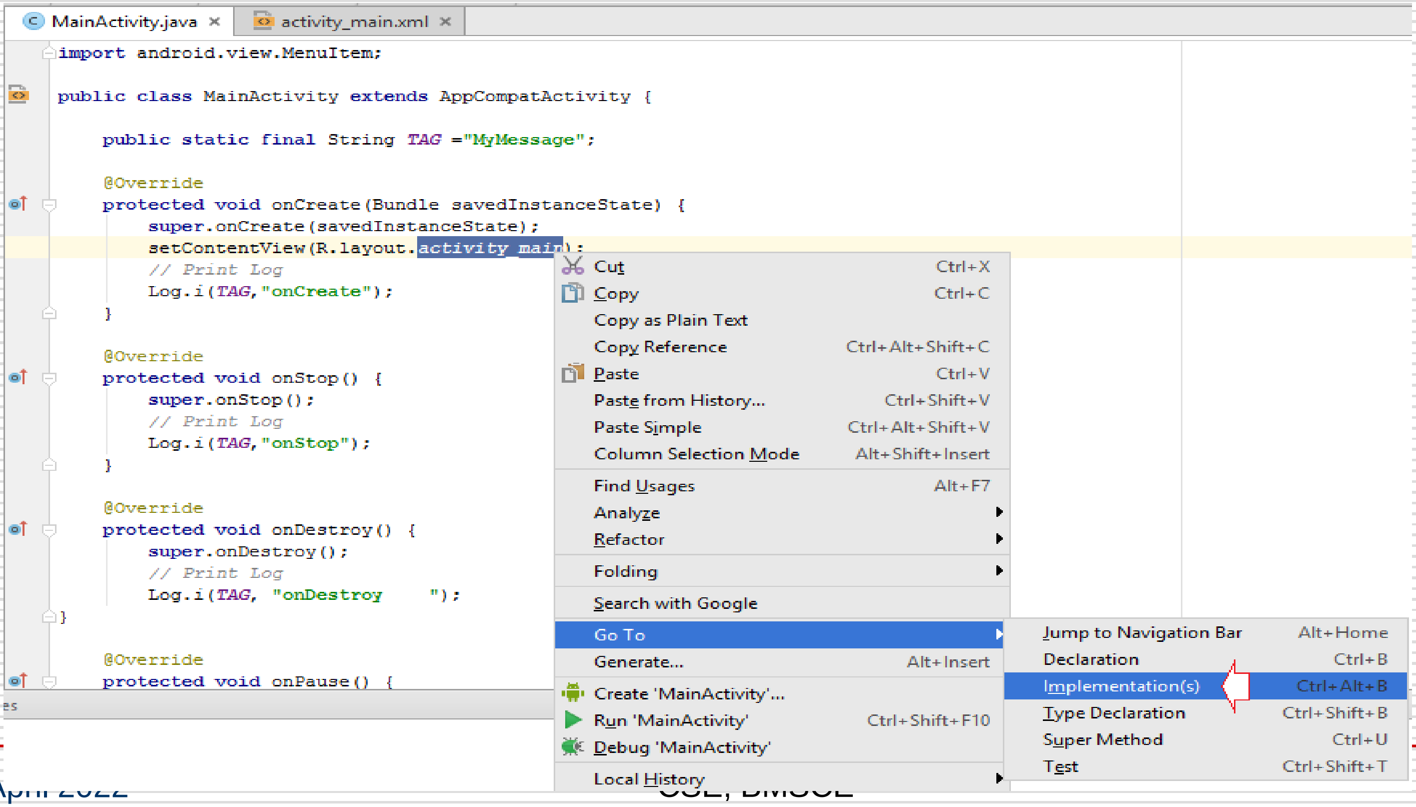
- Next we will learn about R.layout.activity_main in the file MainActivity.java



```
package com.example.umadevi.helloworld;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

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

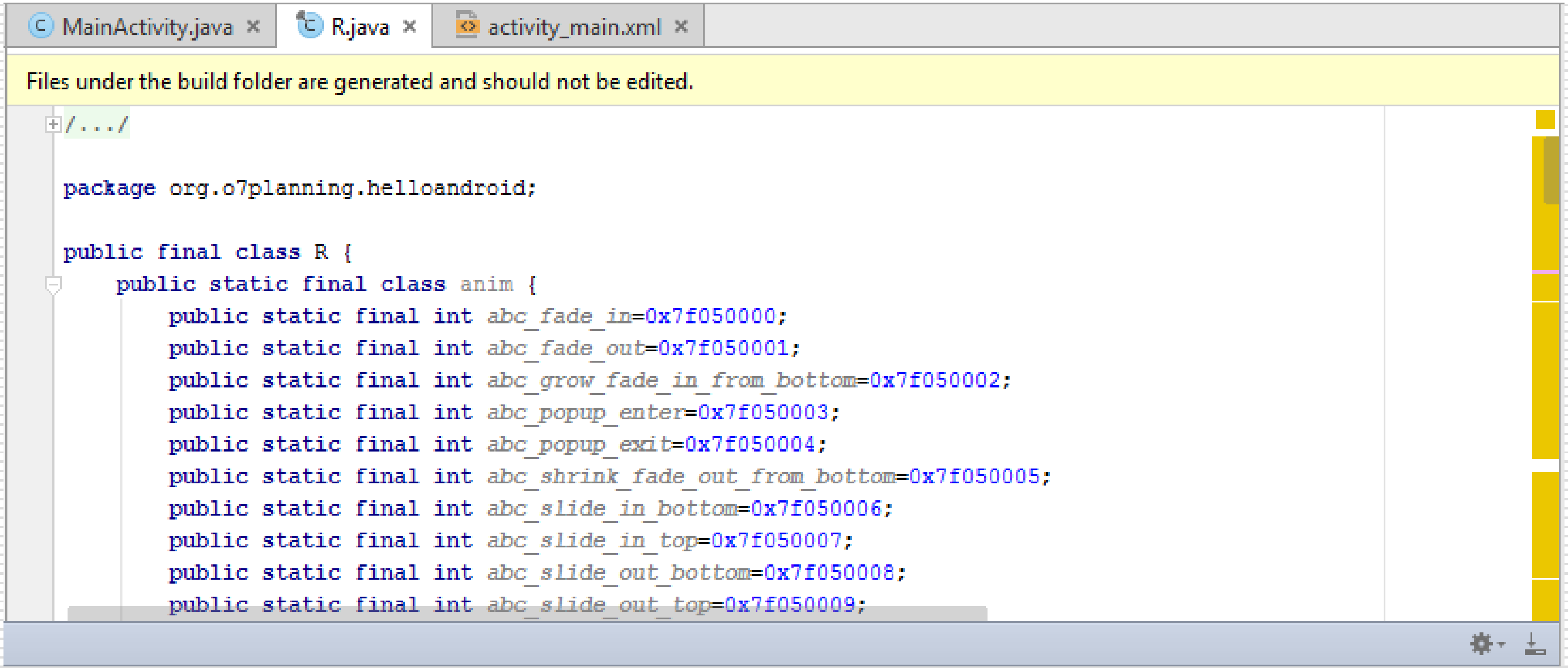
R.java file

- When we see on the Project window, not all components involved creating your application is visible. There are components that are generated automatically by the compiler program. And it does not display on Project window. For example, based on the structure of the source files in your project, the compiler creates a R.java source file that defines constants for the resources on the project.
- To view the **R.java** file on **Android Studio**, open **MainActivity** class, right click on ***R.layout.main_activity*** and select ***Go To/Implementation (s)***



```
import android.view.MenuItem;

public class MainActivity extends AppCompatActivity {

    public static final String TAG ="MyMessage";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // Print Log
        Log.i(TAG,"onCreate");
    }

    @Override
    protected void onStop() {
        super.onStop();
        // Print Log
        Log.i(TAG,"onStop");
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        // Print Log
        Log.i(TAG, "onDestroy     ");
}

    @Override
    protected void onPause() {
```

R.java file

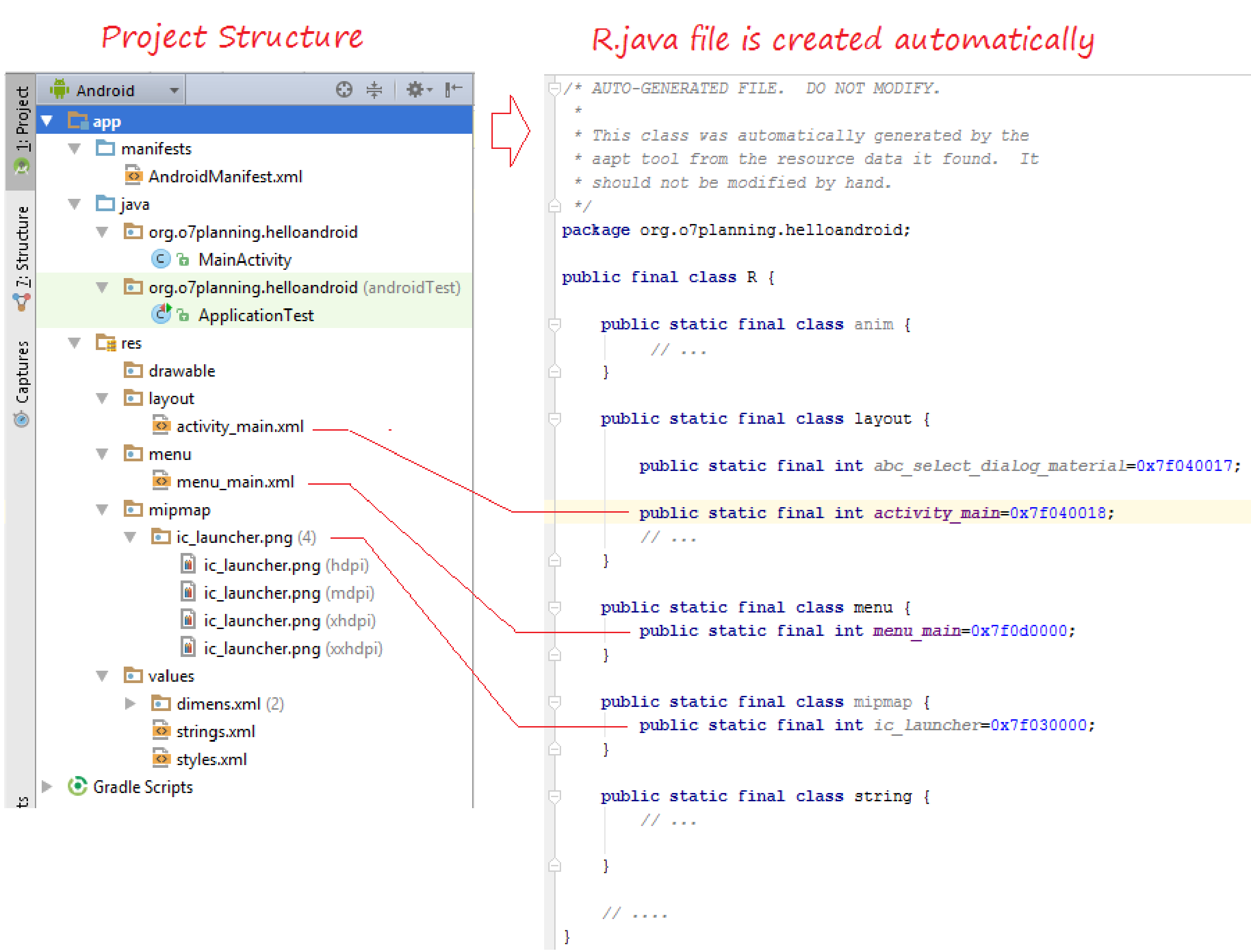


Files under the build folder are generated and should not be edited.

```
/.../

package org.o7planning.helloandroid;

public final class R {
    public static final class anim {
        public static final int abc_fade_in=0x7f050000;
        public static final int abc_fade_out=0x7f050001;
        public static final int abc_grow_fade_in_from_bottom=0x7f050002;
        public static final int abc_popup_enter=0x7f050003;
        public static final int abc_popup_exit=0x7f050004;
        public static final int abc_shrink_fade_out_from_bottom=0x7f050005;
        public static final int abc_slide_in_bottom=0x7f050006;
        public static final int abc_slide_in_top=0x7f050007;
        public static final int abc_slide_out_bottom=0x7f050008;
        public static final int abc_slide_out_top=0x7f050009;
```

R.java

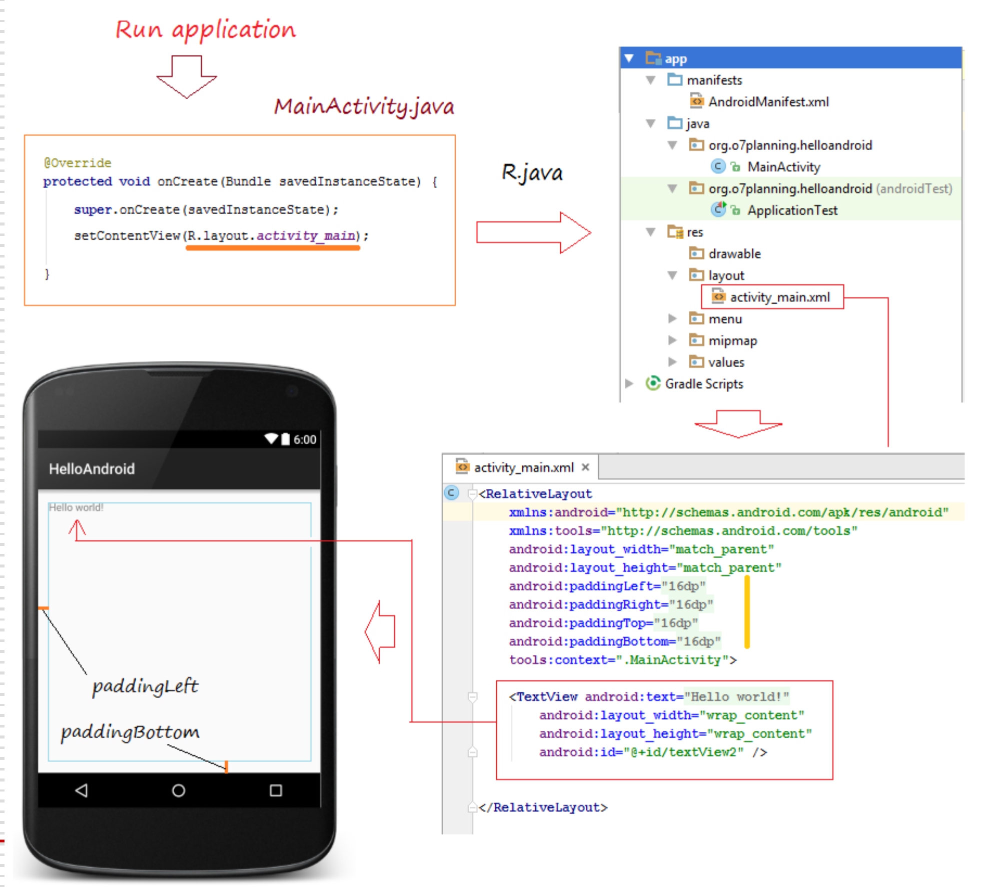
- The constants defined in class **R.java** are created corresponding to the resources on the Project:



```java
/* AUTO-GENERATED FILE.  DO NOT MODIFY.
 *
 * This class was automatically generated by the
 * aapt tool from the resource data it found.  It
 * should not be modified by hand.
 */
package org.o7planning.helloandroid;

public final class R {

    public static final class anim {
        // ...
    }

    public static final class layout {

        public static final int abc_select_dialog_material=0x7f040017;

        public static final int activity_main=0x7f040018;
        // ...
    }

    public static final class menu {
        public static final int menu_main=0x7f0d0000;
    }

    public static final class mipmap {
        public static final int ic_launcher=0x7f030000;
    }

    public static final class string {
        // ...

    }

    // ....
}
```

R.java

- So somewhere in the Project, you can use the constants to refer resources in the project. Such as, **R.layout.activity_main** is a constant which refers **activity_main.xml** file in the **res/layout** folder.



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

}
```

```xml
<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"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    tools:context=".MainActivity">

    <TextView android:text="Hello world!"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/textView2" />

</RelativeLayout>
```

Unit 2: What makes an Android Application

Components or Building blocks of Android Application

Android Application Components

Four Basic Components:

1. Activities
2. Services
3. Broadcast Receivers
4. Content Providers

Activities
Services
Application Components
Broadcast Receivers
Content Providers

Basic Android Application Components

Sl.No	Components & Description
1	**Activities:** They dictate the UI and handle the user interaction to the smart phone screen.
2	**Services:** They handle background processing associated with an application.
3	**Broadcast Receivers:** They handle communication between Android OS and applications.
4	**Content Providers:** They handle data and database management issues.

Complete list of Components or Building blocks of Android Application

- Activities
- Services
- Broadcast Receivers
- Content Providers
- Intents
- Widgets
- Notifications

Activities

- An activity represents a single screen with a user interface,in-short Activity performs actions on the screen.
- Activities are equivalent to Forms.
- For example, an **email** application might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails.
- An activity is implemented as a subclass of **Activity** class as follows −

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

Services

- A service is a component that runs in the background to perform long-running operations.
- For example, a service might play music in the background while the user is in a different application, or it might fetch data over the network without blocking user interaction with an activity.
- A service is implemented as a subclass of **Service** class as follows –

```
public class MyService extends Service { }
```

Broadcast Receivers

- □ Broadcast Receivers simply **respond to broadcast messages** from other applications or from the system.
- □ Broadcast receivers main job is to respond to incoming events. Example, if a server sends you a notification, the broadcast receives a response that in turn puts a notification in your notification bar of your Android device.
- □ For example, applications can also initiate broadcasts to let other applications know that some data has been downloaded to the device and is available for them to use, so this is broadcast receiver who will intercept this communication and will initiate appropriate action.
- □ A broadcast receiver is implemented as a subclass of **BroadcastReceiver** class and each message is broadcaster as an **Intent** object.

```
public class MyReceiver extends BroadcastReceiver {
 public void onReceive(context,intent){} }
```

Content Providers

- A content provider component supplies data from one application to others on request. Such requests are handled by the methods of the *ContentResolver* class. The data may be stored in the file system, the database or somewhere else entirely.
- A content provider is implemented as a subclass of **ContentProvider** class and must implement a standard set of APIs that enable other applications to perform transactions.

```
public class MyContentProvider extends ContentProvider
{ public void onCreate(){} }
```

Intents

- **Android Intents** are the communication medium .i.e., app components send messages to one another like you do with your friends.
- Intents are powerful interapplication message-passing framework.

Widgets

- A widget is a small gadget or control of your android application placed on the home screen.
- A special variation of a Broadcast Receiver, widgets enable you to create dynamic, interactive application components for users to embed on their home screens.
- Widgets can be very handy as they allow you to put your favourite applications on your home screen in order to quickly access them.
- You have probably seen some common widgets, such as music widget, weather widget, clock widget e.t.c

Notification

- Notifications enable you to alert users to application events without stealing focus or interrupting their current Activity.
- They’re the preferred technique for getting a user’s attention when your application is not visible or active, particularly from within a Service or Broadcast Receiver.
- For example, when a device receives a text message or an email, the messaging and Gmail applications use Notifications to alert you by flashing lights, playing sounds, displaying icons, and scrolling a text summary.

Question

List and explain various components/Building blocks of Android Application

Question

A Broadcast Receiver is used to

- □ respond to events
- □ design user interface
- □ deal with data storage
- □ do work in the background

Question

A Broadcast Receiver is used to

- □ **respond to events**
- □ design user interface
- □ deal with data storage
- □ do work in the background

Question

Activity is used to

- □ respond to events
- □ design user interface
- □ deal with data storage
- □ do work in the background

Question

Activity is used to

- □ respond to events
- □ **design user interface**
- □ deal with data storage
- □ do work in the background

Question

A Service is used to

□ respond to events

□ design user interface

□ deal with data storage

□ do work in the background

Question

A Service is used to

- □ respond to events
- □ design user interface
- □ deal with data storage
- □ **do work in the background**

Question

Content Providers are used to

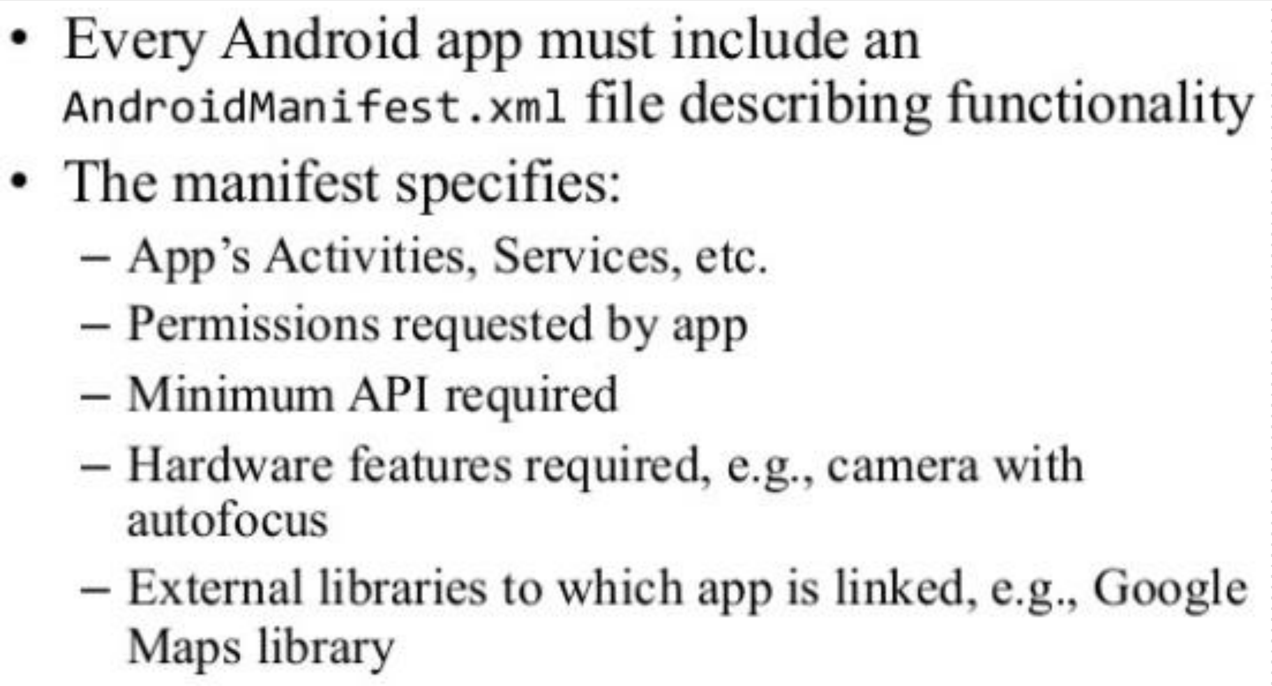
- □ respond to events
- □ design user interface
- □ deal with data storage
- □ do work in the background

Question

Content Providers are used to

- □ respond to events
- □ design user interface
- □ **deal with data storage**
- □ do work in the background

Unit 2: Introducing Application Manifest File



- Every Android app must include an `AndroidManifest.xml` file describing functionality
- The manifest specifies:
  - App's Activities, Services, etc.
  - Permissions requested by app
  - Minimum API required
  - Hardware features required, e.g., camera with autofocus
  - External libraries to which app is linked, e.g., Google Maps library

The manifest file

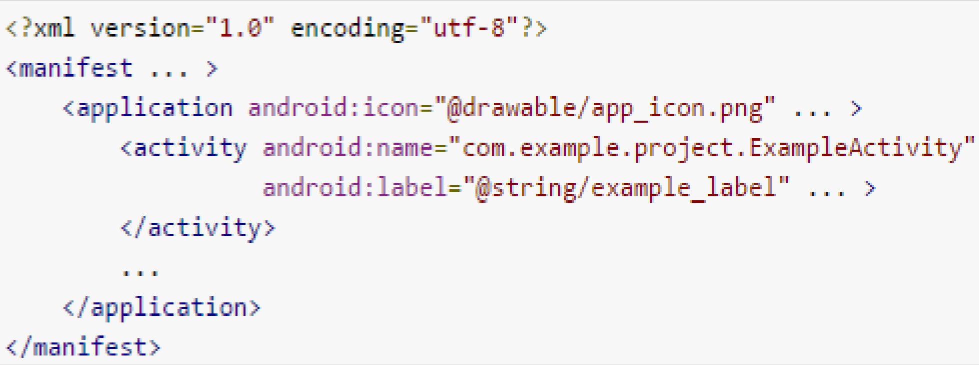
- Before the Android system can start an app component, the system must know that the component exists by reading the app's manifest file, **AndroidManifest.xml**. Your app must declare all its components in this file, which must be at the root of the app project directory.

The manifest does a number of things in addition to declaring the app's components, such as the following:

- Identifies any user permissions the app requires, such as Internet access or read-access to the user's contacts.
- Declares the minimum API Level required by the app, based on which APIs the app uses.
- Declares hardware and software features used or required by the app, such as a camera, bluetooth services, or a multitouch screen.
- Declares API libraries the app needs to be linked against (other than the Android framework APIs), such as the Google Maps library.

Manifest file: Declaring Components

- The primary task of the manifest is to inform the system about the app's components. For example, a manifest file can declare an activity as follows:



```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
    <application android:icon="@drawable/app_icon.png" ... >
        <activity android:name="com.example.project.ExampleActivity"
                  android:label="@string/example_label" ... >
        </activity>
        ...
    </application>
</manifest>
```

- In the <application> element, the android:icon attribute points to resources for an icon that identifies the app.
- In the <activity> element, the android:name attribute specifies the fully qualified class name of the Activity subclass and
  the android:label attribute specifies a string to use as the user-visible label for the activity.

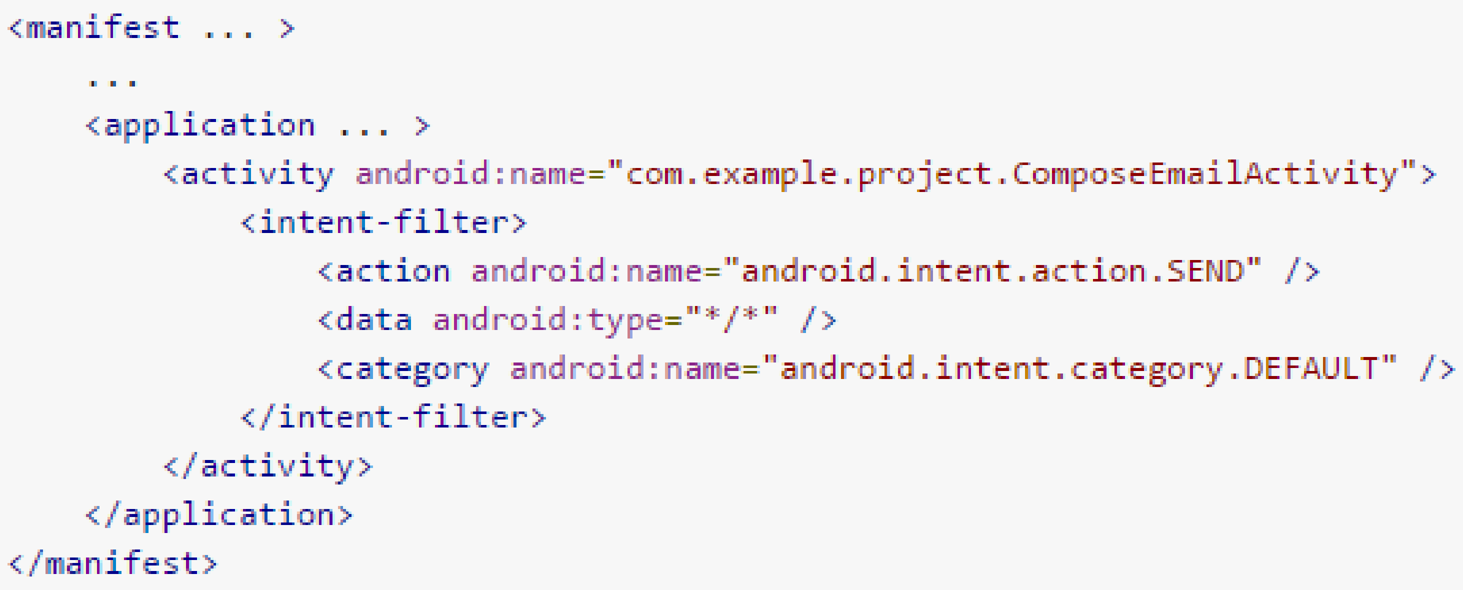
You must declare all app components using the following elements:

<activity> elements for activities. **<service>** elements for services.

<receiver> elements for broadcast receivers.**<provider>** elements for content providers.

Manifest File: Declaring component capabilities

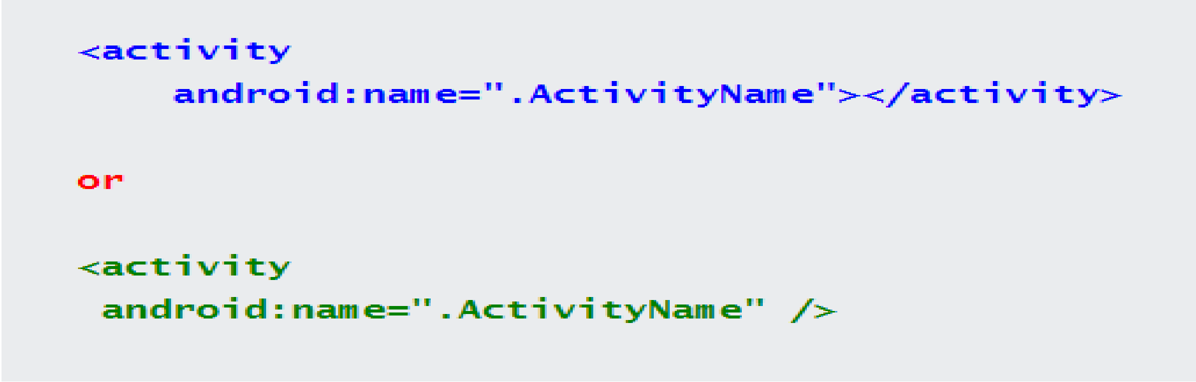
- When you declare an activity in your app's manifest, you can optionally include intent filters that declare the capabilities of the activity so it can respond to intents from other apps. You can declare an intent filter for your component by adding an <intent-filter> element as a child of the component's declaration element.
- For example, if you build an email app with an activity for composing a new email, you can declare an intent filter to respond to "send" intents (in order to send a new email), as shown in the following example:



```
<manifest ... >
    ...
    <application ... >
        <activity android:name="com.example.project.ComposeEmailActivity">
            <intent-filter>
                <action android:name="android.intent.action.SEND" />
                <data android:type="*/*" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

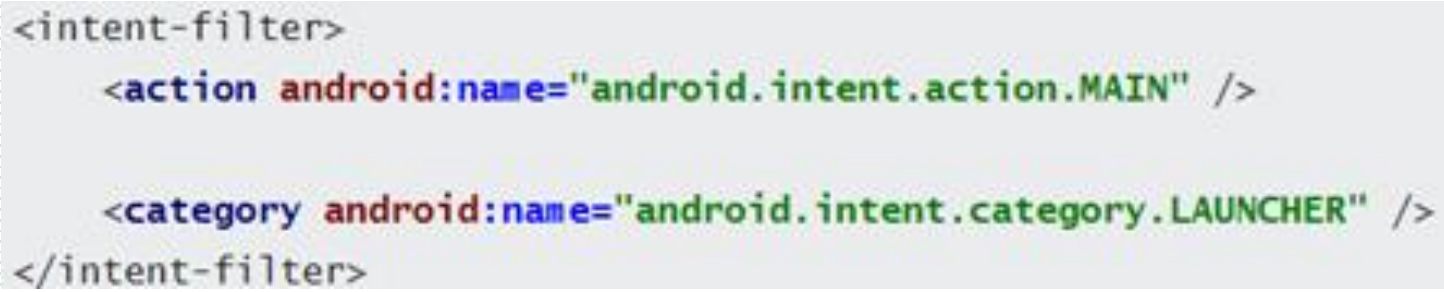
Understanding AndroidManifest.xml file

- Declaring Activity inside AndroidManifest.xml file



```
<activity
     android:name=".ActivityName"></activity>

or

<activity
 android:name=".ActivityName" />
```

- An activity should have the following code to make it as a LAUNCHER activity whose layout (UI) would be the first to load when an app is opened.



```
<intent-filter>
    <action android:name="android.intent.action.MAIN" />

    <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
```

Closer Look at the Application Manifest

- The following XML snippet shows a typical manifest node:

```
<manifest
xmlns:android=”http://schemas.android.com/apk/res/android”
package=“com.paad.myapp“
android:versionCode=“1“
android:versionName=“0.9 Beta“
android:installLocation=“preferExternal“>
[ ... manifest nodes ... ]
</manifest>
```

- The manifest tag can include nodes that define the application components, security settings, test classes, and requirements that make up your application. List of available manifest sub-node tags are: uses-sdk, uses-configuration, uses-feature, supports-screens, uses-permission, …………

Closer Look at the Application Manifest (Contd….)

- uses-configuration: The uses-configuration nodes specify each combination of input mechanisms are supported by your application. We can specify any combination of input devices that include the following:
- reqFiveWayNav — Specify true for this attribute if you require an input device capable of navigating up, down, left, and right and of clicking the current selection. This includes both trackballs and directional pads.
- reqHardKeyboard — If your application requires a hardware keyboard, specify true.
- reqKeyboardType — Lets you specify the keyboard type as one of nokeys, qwerty, twelvekey, or undefined.
- reqNavigation — Specify the attribute value as one of nonav, dpad, trackball, wheel, or undefined as a required navigation device.
- reqTouchScreen — Select one of notouch, stylus, finger, or undefined to specify the required touchscreen input.
- We can specify multiple supported configurations, for example, a device with a finger touchscreen, a trackball, and either a QUERTY or a twelve-key hardware keyboard, as shown here:

```
<uses-configuration android:reqTouchScreen=”finger”
android:reqNavigation=”trackball”
android:reqHardKeyboard=”true”
android:reqKeyboardType=”qwerty”/>
<uses-configuration android:reqTouchScreen=”finger”
android:reqNavigation=”trackball”
android:reqHardKeyboard=”true”
android:reqKeyboardType=”twelvekey”/>
```

Refer

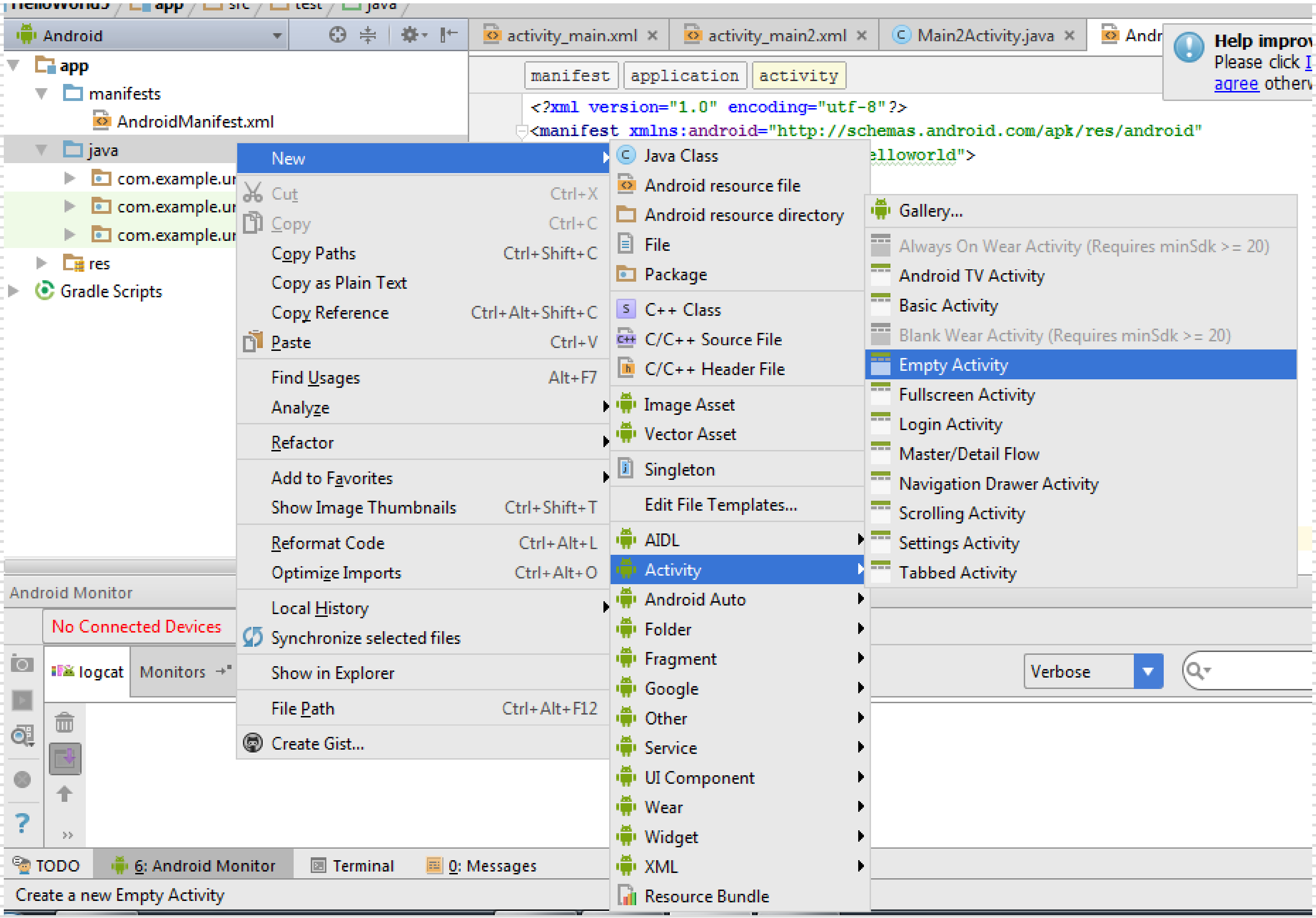
- Text book page numbers from 56 to 61

We will See in Action

- Create an ‘HelloWorld’ app

Add Second activity to 'Hello World' app

- Click on app -> java –> New -> Activity -> EmptyActivity

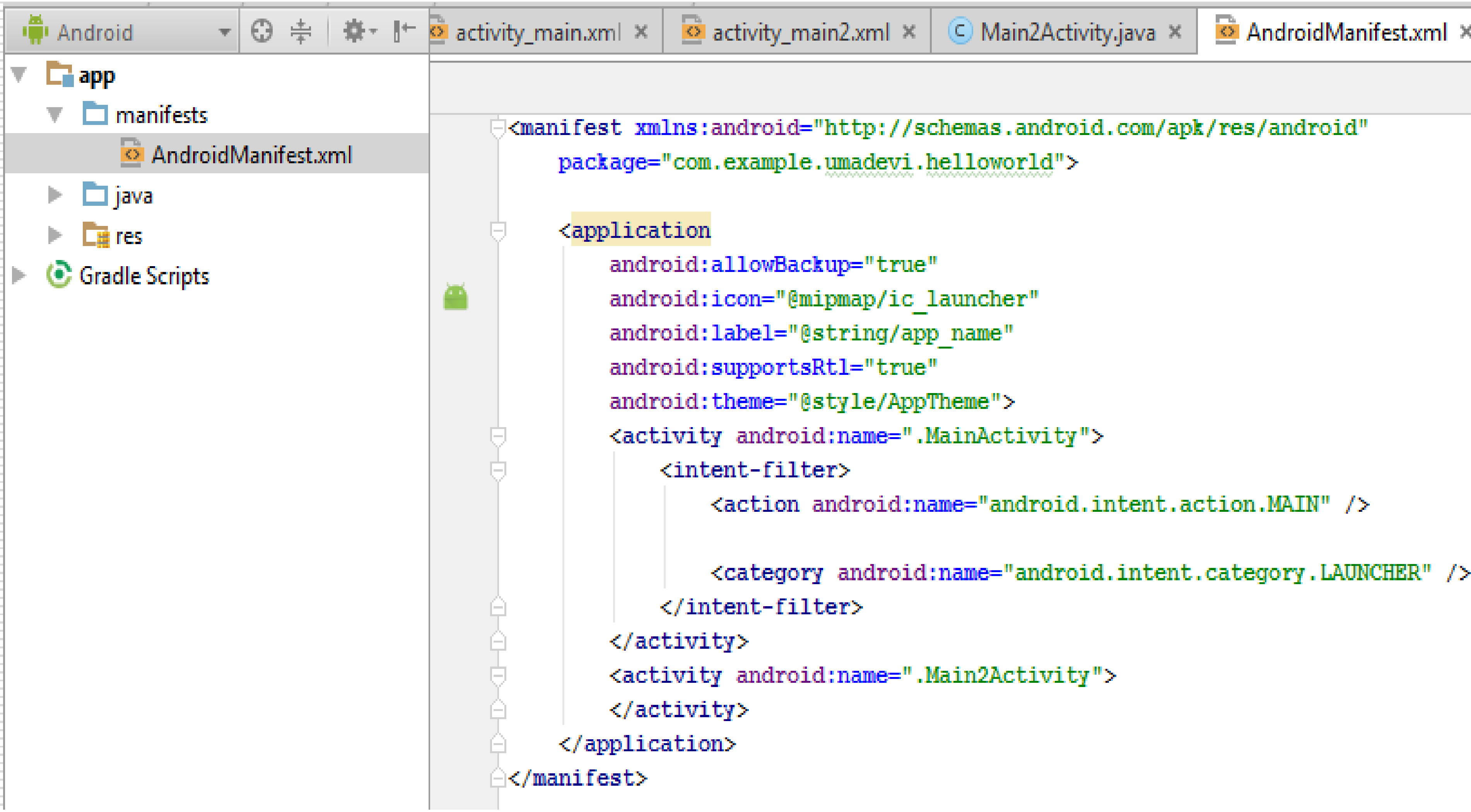


```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
elloworld">
```

No Connected Devices

Create a new Empty Activity

AndroidManifest.xml file

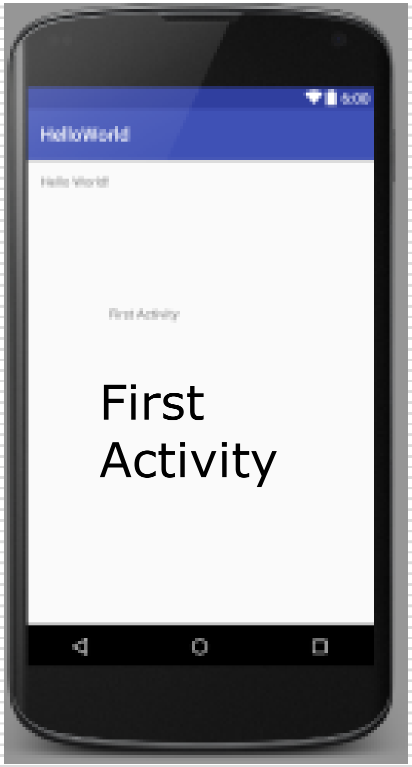


```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.umadevi.helloworld">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        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=".Main2Activity">
        </activity>
    </application>
</manifest>
```

Question

When we run the manifest shown in previous slide
Which screen will be shown first: Screen1 or Screen 2 ?

Screen 1



6:00

HelloWorld

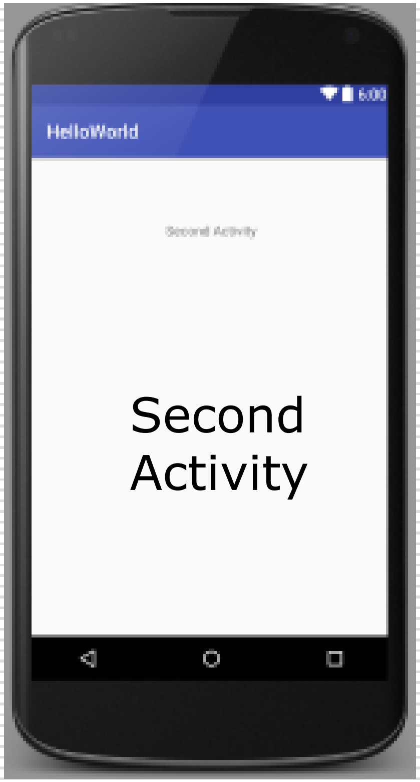
Hello World!

First Activity

First
Activity

MainActivity

Screen 2



6:00

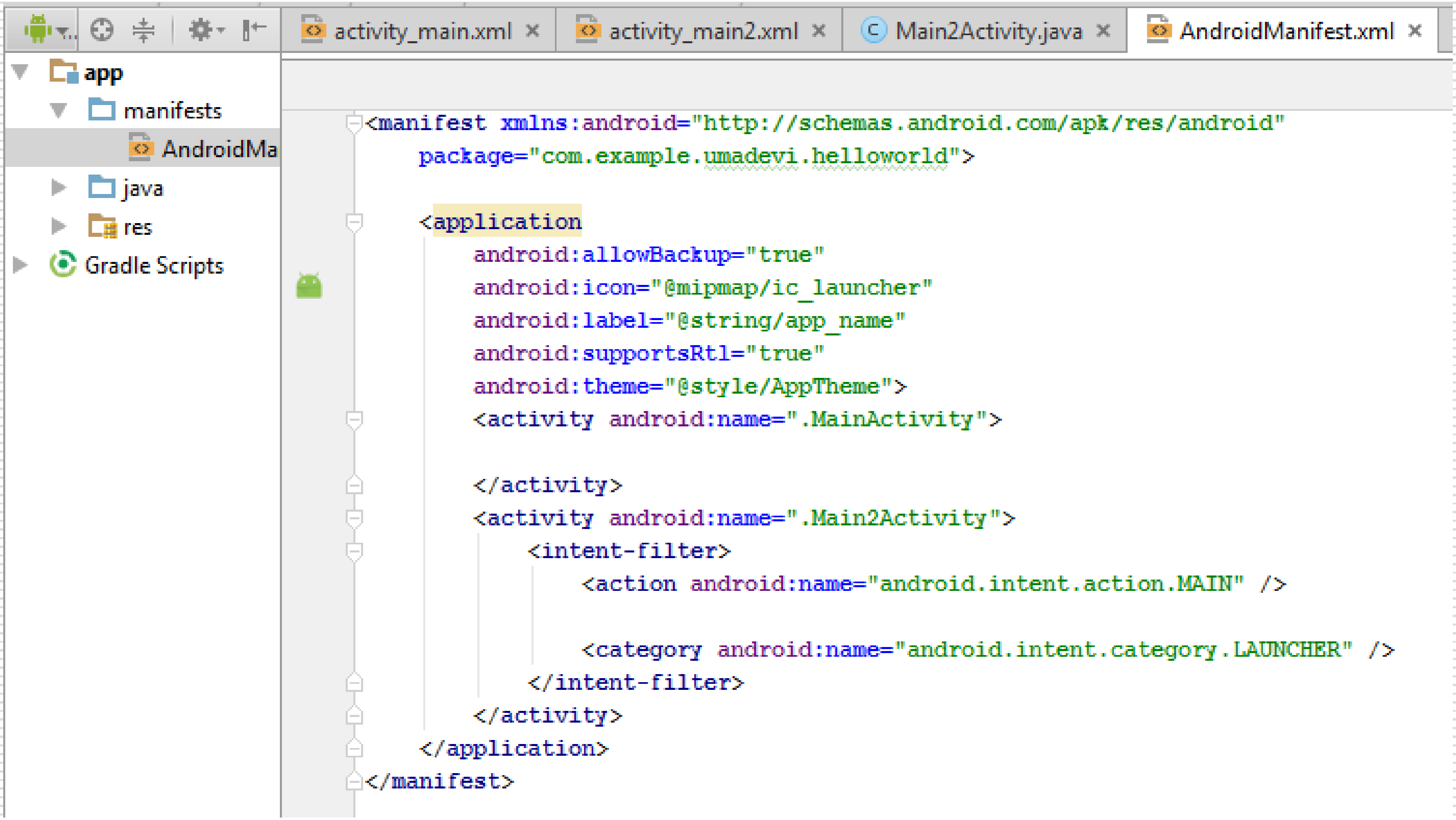
HelloWorld

Second Activity

Second
Activity

Main2Activity

AndroidManifest.xml file

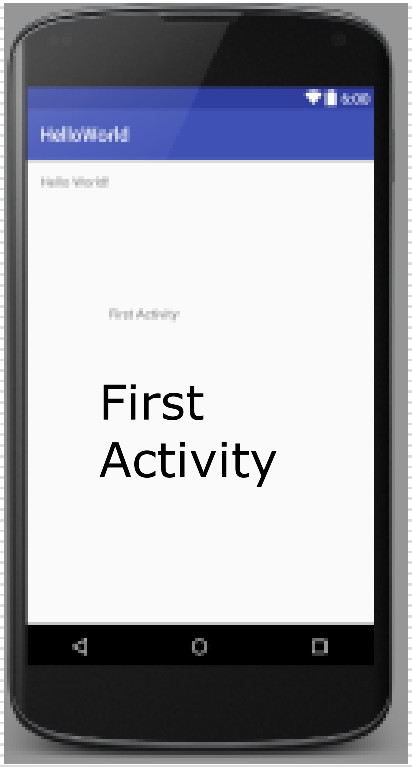


```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.umadevi.helloworld">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">

        </activity>
        <activity android:name=".Main2Activity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Question

When we run the manifest shown in previous slide
Which screen will be shown first: Screen1 or Screen 2 ?

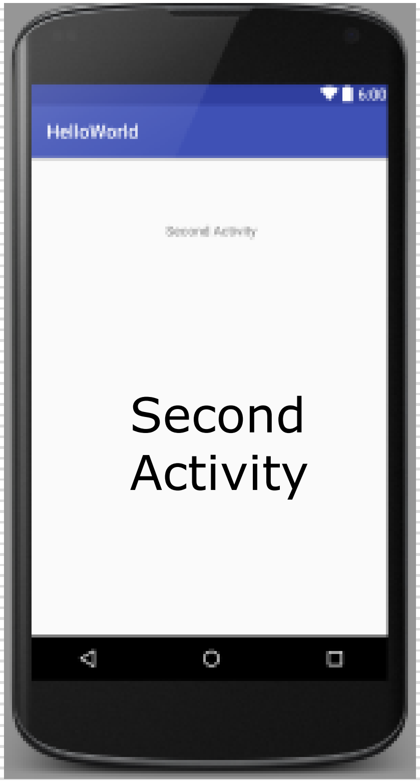
Screen 1



First
Activity

MainActivity

Screen 2



Second
Activity

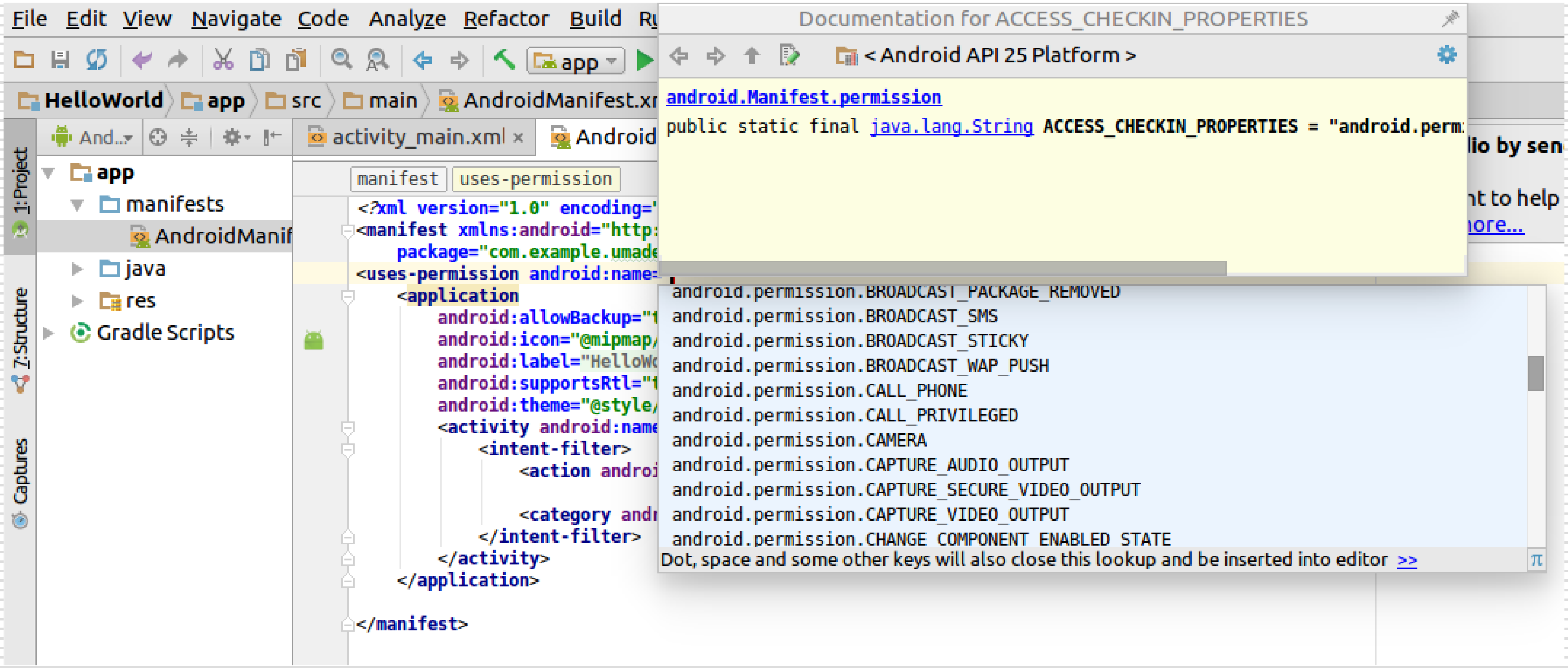
Main2Activity

AndroidManifest.xml file

android:allowBackup="true"	To backup application data
android:icon="@mipmap/ic_launcher"	App icon
android:label="@string/app_name"	App label
android:supportsRtl="true"	Declares whether your application is willing to support right-to-left (RTL) layouts.
android:theme="@style/AppTheme">	App theme

AndroidManifest.xml file

- Adding permissions inside manifest file
- <uses-permission android:name="android.permission.CAMERA"/>
- This permission is required to be able to access the camera device.



```
<?xml version="1.0" encoding='
<manifest xmlns:android="http:
    package="com.example.umade
<uses-permission android:name=
    <application
        android:allowBackup="t
        android:icon="@mipmap/
        android:label="HelloWo
        android:supportsRtl="t
        android:theme="@style/
        <activity android:name
            <intent-filter>
                <action androi

                <category andr
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Documentation for ACCESS_CHECKIN_PROPERTIES

< Android API 25 Platform >

android.Manifest.permission

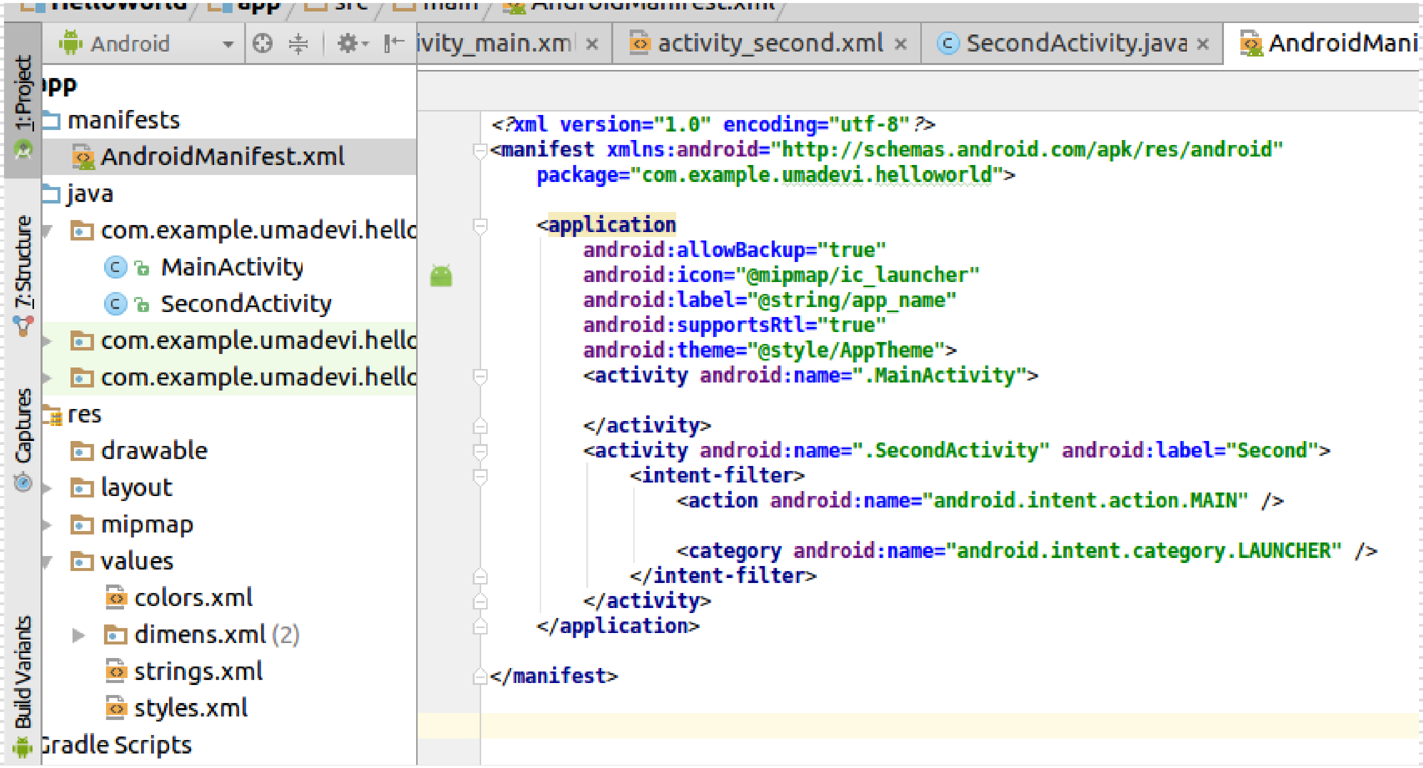
```
public static final java.lang.String ACCESS_CHECKIN_PROPERTIES = "android.perm:
```

```
android.permission.BROADCAST_PACKAGE_REMOVED
android.permission.BROADCAST_SMS
android.permission.BROADCAST_STICKY
android.permission.BROADCAST_WAP_PUSH
android.permission.CALL_PHONE
android.permission.CALL_PRIVILEGED
android.permission.CAMERA
android.permission.CAPTURE_AUDIO_OUTPUT
android.permission.CAPTURE_SECURE_VIDEO_OUTPUT
android.permission.CAPTURE_VIDEO_OUTPUT
android.permission.CHANGE_COMPONENT_ENABLED_STATE
```

Dot, space and some other keys will also close this lookup and be inserted into editor >>

AndroidManifest.xml file

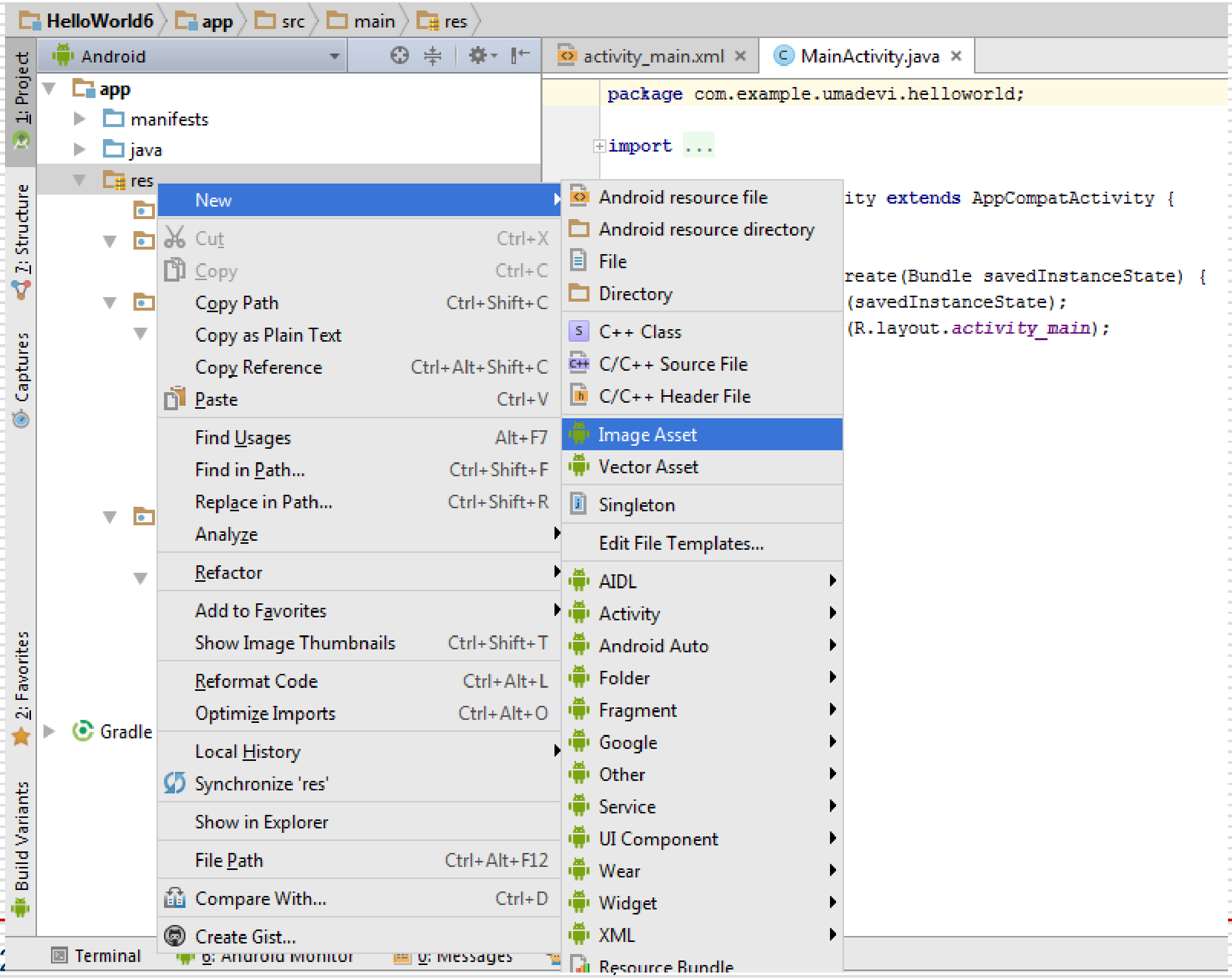
- Changing Activity label
- <activity android:name=".SecondActivity" android:label="Second">



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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">

        </activity>
        <activity android:name=".SecondActivity" android:label="Second">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

AndroidManifest.xml: Changing app icon

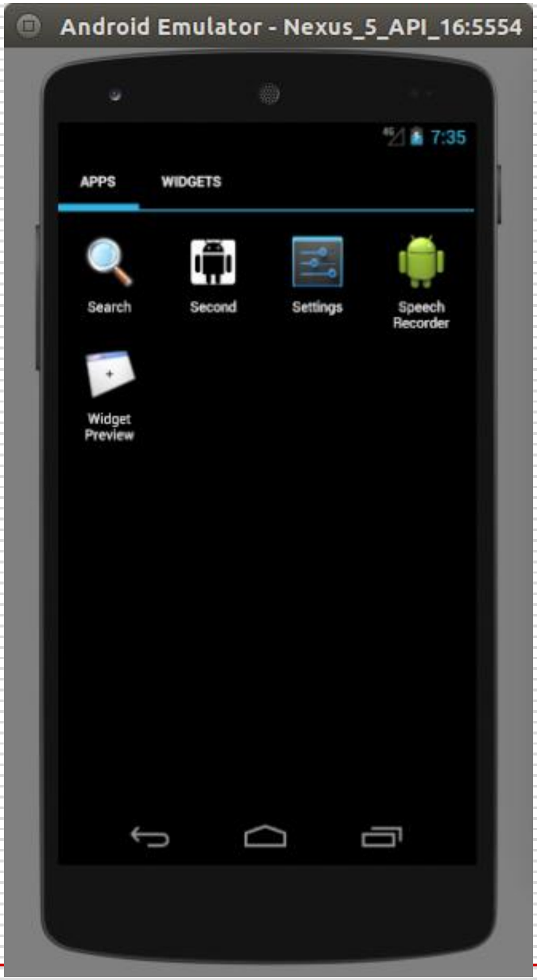
- res -> New -> Image Asset



```
package com.example.umadevi.helloworld;

import ...

ity extends AppCompatActivity {

reate(Bundle savedInstanceState) {
(savedInstanceState);
(R.layout.activity_main);
```

AndroidManifest.xml

- Changing app icon

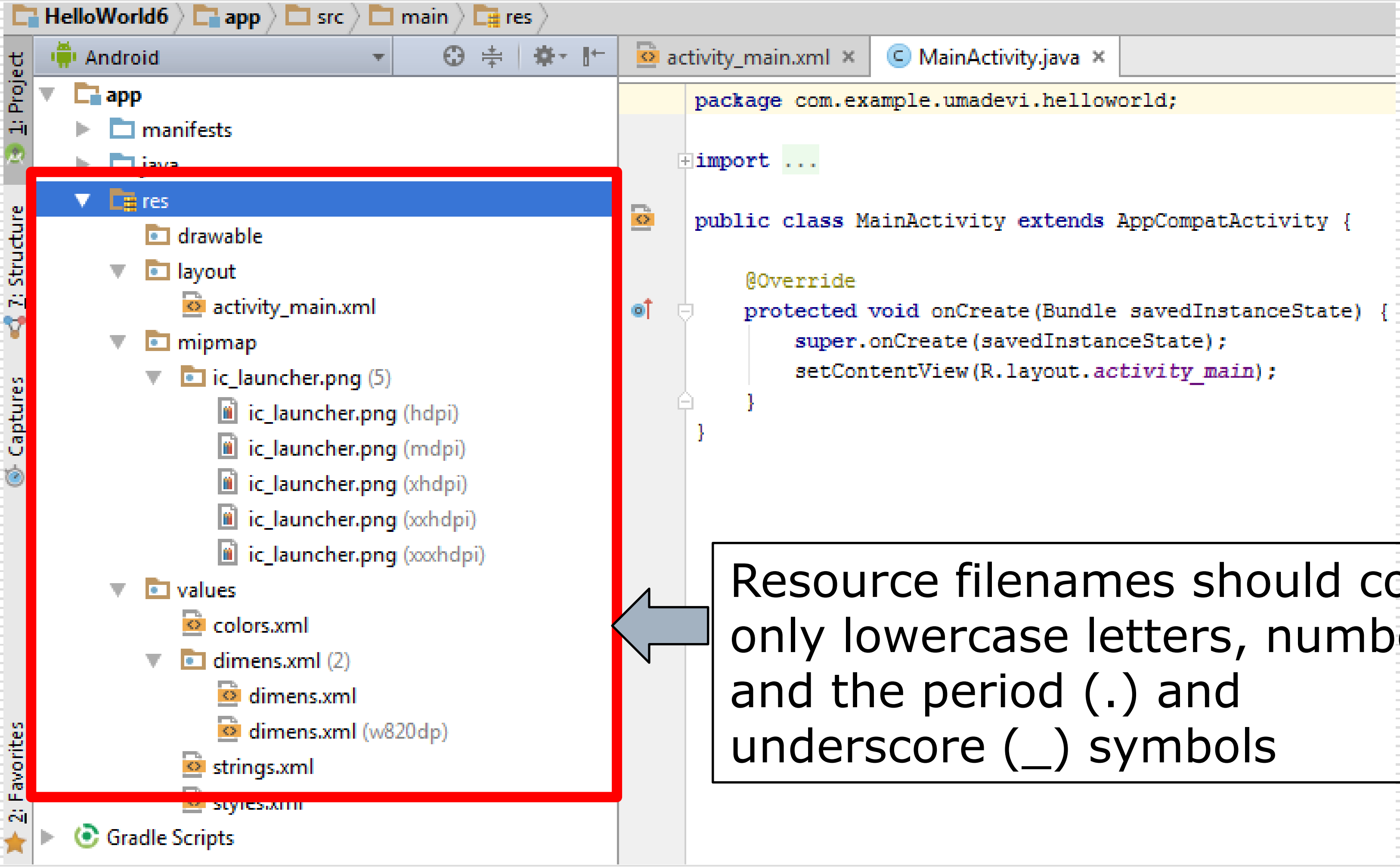


Android Emulator - Nexus_5_API_16:5554

EXTERNALIZING RESOURCES

Application resources such as String constants, Colors, Images, Animations, Themes, and menus are kept separate from under “**res**” folder. This will make them easier to maintain, update, and manage.

“res” folder



```
app
  manifests
  java
  res
    drawable
    layout
      activity_main.xml
    mipmap
      ic_launcher.png (5)
        ic_launcher.png (hdpi)
        ic_launcher.png (mdpi)
        ic_launcher.png (xhdpi)
        ic_launcher.png (xxhdpi)
        ic_launcher.png (xxxhdpi)
    values
      colors.xml
      dimens.xml (2)
        dimens.xml
        dimens.xml (w820dp)
      strings.xml
      styles.xml
Gradle Scripts
```

```java
package com.example.umadevi.helloworld;

import ...

public class MainActivity extends AppCompatActivity {

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

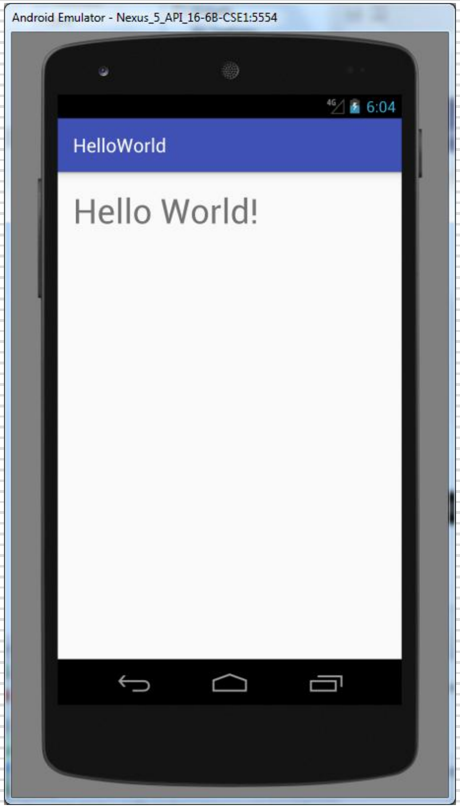
Resource filenames should contain only lowercase letters, numbers, and the period (.) and underscore (_) symbols

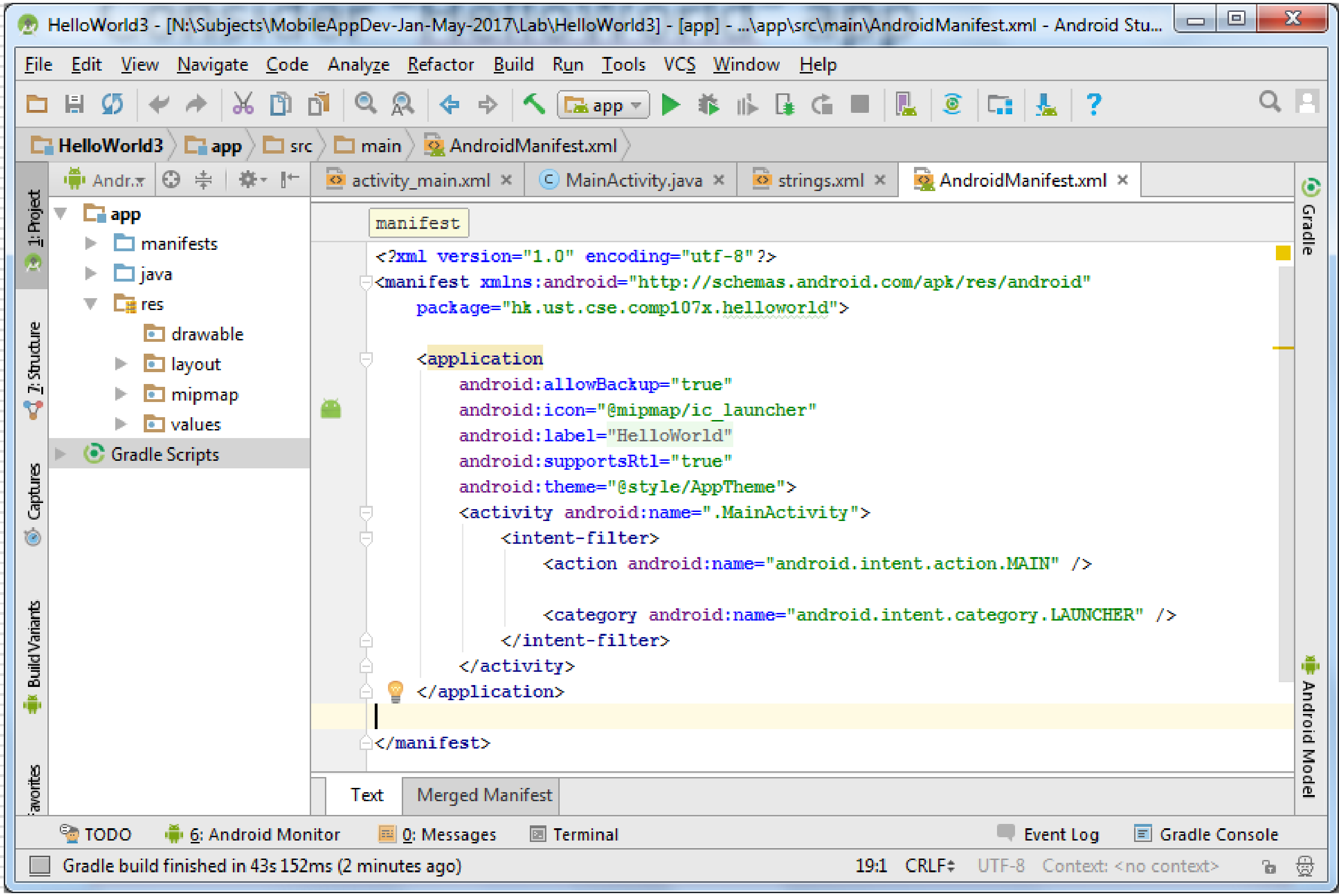
Various Resource types in Android under res folder

SL.	Directory & Resource Type
1	**values/** XML files that contain simple values, such as strings, integers, and colors. For example, here are some filename conventions for resources you can create in this directory – •arrays.xml for resource arrays, and accessed from the **R.array** class. •integers.xml for resource integers, and accessed from the **R.integer** class. •bools.xml for resource boolean, and accessed from the **R.bool** class. •colors.xml for color values, and accessed from the **R.color** class. •dimens.xml for dimension values, and accessed from the **R.dimen** class. •strings.xml for string values, and accessed from the **R.string** class. •styles.xml for styles, and accessed from the **R.style** class.
2	**color/** XML files that define a state list of colors. They are saved in res/color/ and accessed from the **R.color** class.
3	**drawable/** Image files like .png, .jpg, .gif or XML files that are compiled into bitmaps, state lists, shapes, animation drawable. They are saved in res/drawable/ and accessed from the **R.drawable** class.
4	**layout/** XML files that define a user interface layout. They are saved in res/layout/ and accessed from the **R.layout** class.
5	**anim/** XML files that define property animations. They are saved in res/anim/ folder and accessed from the **R.anim** class.
6	**menu/** XML files that define application menus, such as an Options Menu, Context Menu, or Sub Menu. They are saved in res/menu/ and accessed from the **R.menu** class.

Consider “HelloWorld” app

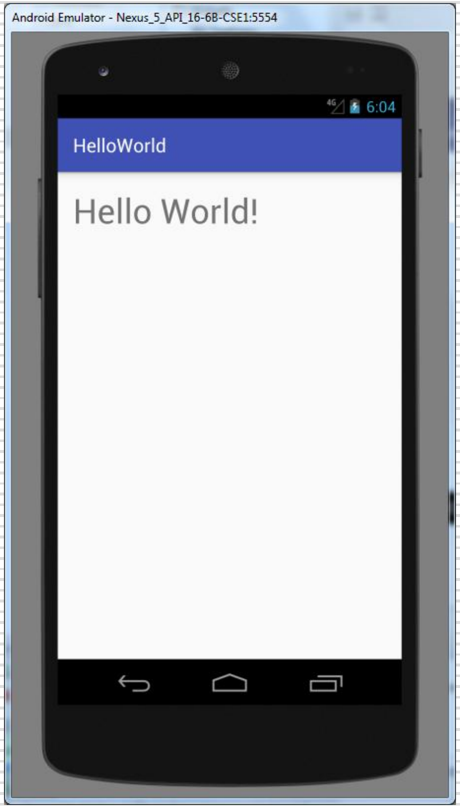
AndroidManiFest.xml file




HelloWorld3 - [N:\Subjects\MobileAppDev-Jan-May-2017\Lab\HelloWorld3] - [app] - ...\app\src\main\AndroidManifest.xml - Android Stu...
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
app
HelloWorld3 › app › src › main › AndroidManifest.xml
Andr.
activity_main.xml
MainActivity.java
strings.xml
AndroidManifest.xml
1: Project
7: Structure
Captures
Build Variants
Favorites
Gradle
Android Model
app
manifests
java
res
drawable
layout
mipmap
values
Gradle Scripts
manifest
Text
Merged Manifest
TODO
6: Android Monitor
0: Messages
Terminal
Event Log
Gradle Console
Gradle build finished in 43s 152ms (2 minutes ago)
19:1 CRLF UTF-8 Context: <no context>

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="hk.ust.cse.comp107x.helloworld">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="HelloWorld"
        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>
    </application>

</manifest>
```

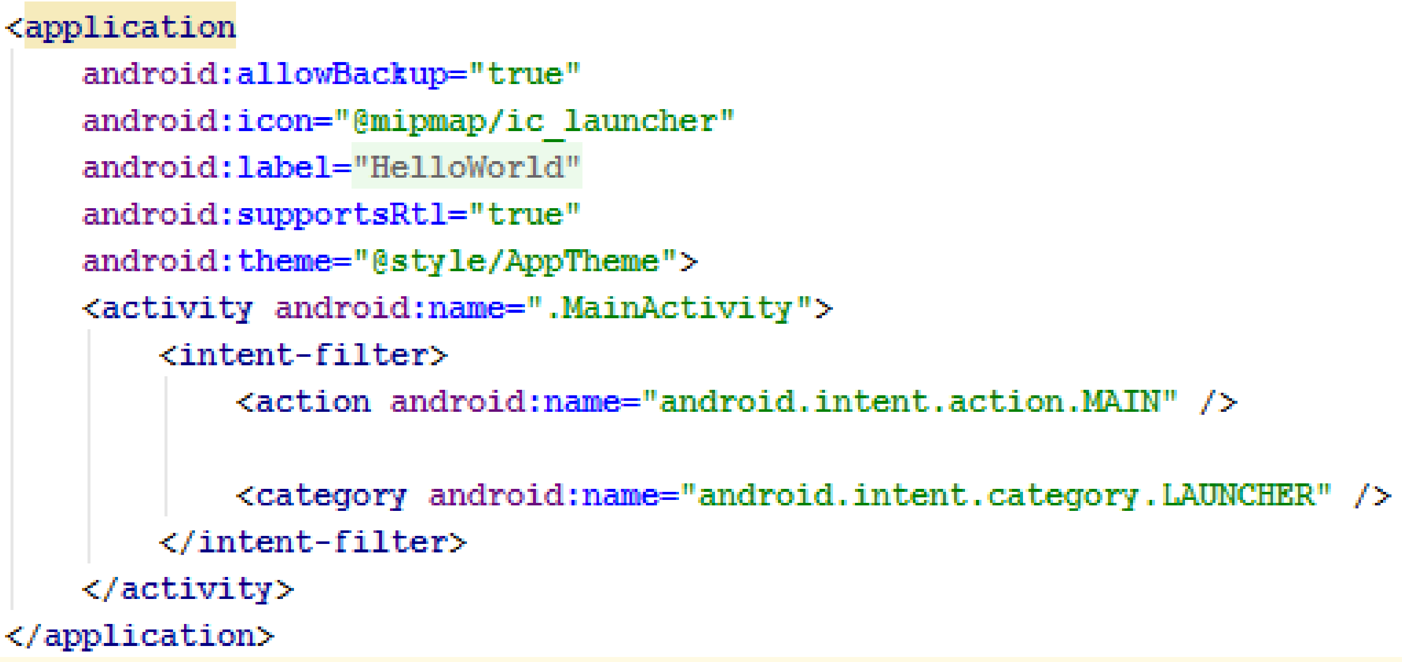
Consider “HelloWorld” app



HelloWorld

Hello World!

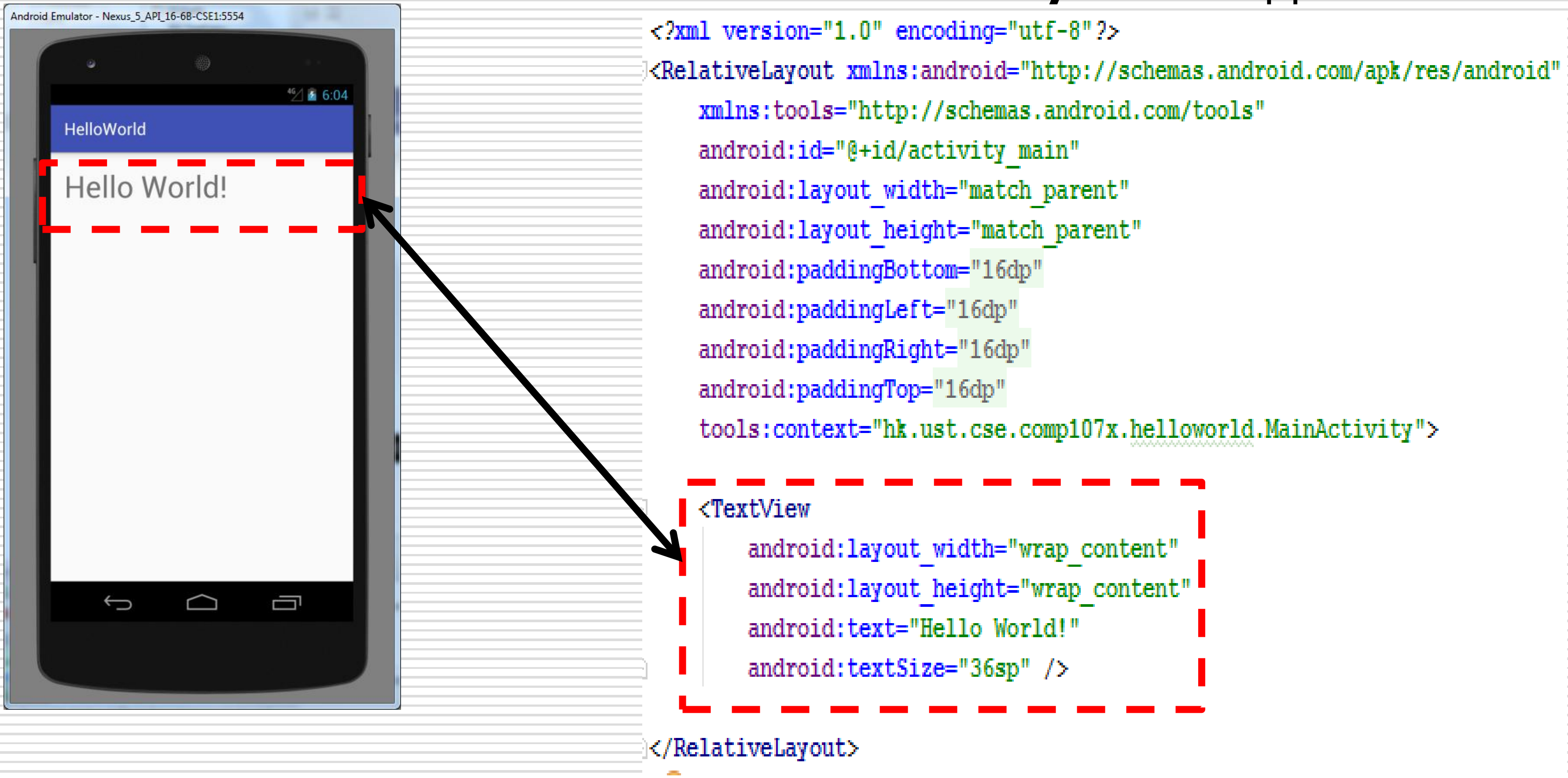
AndroidManiFest.xml file



```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="HelloWorld"
    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>
</application>
```

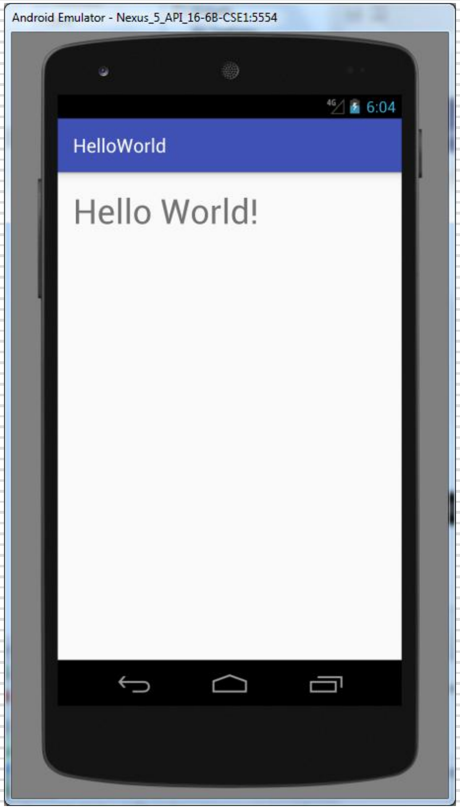
Consider “HelloWorld” app

activity_main.xml file
Create **layout** for app



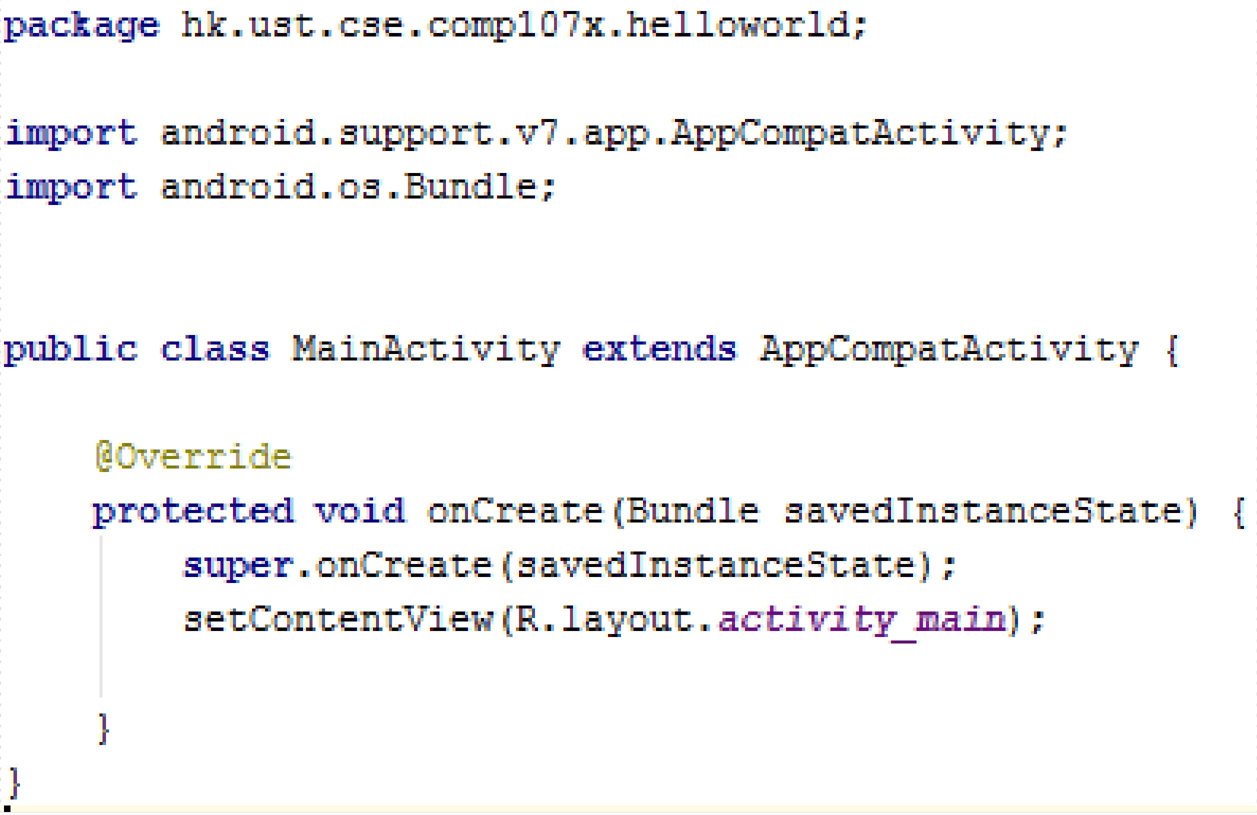
```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    tools:context="hk.ust.cse.comp107x.helloworld.MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        android:textSize="36sp" />

</RelativeLayout>
```

Consider “HelloWorld” app



MainActivity.java file

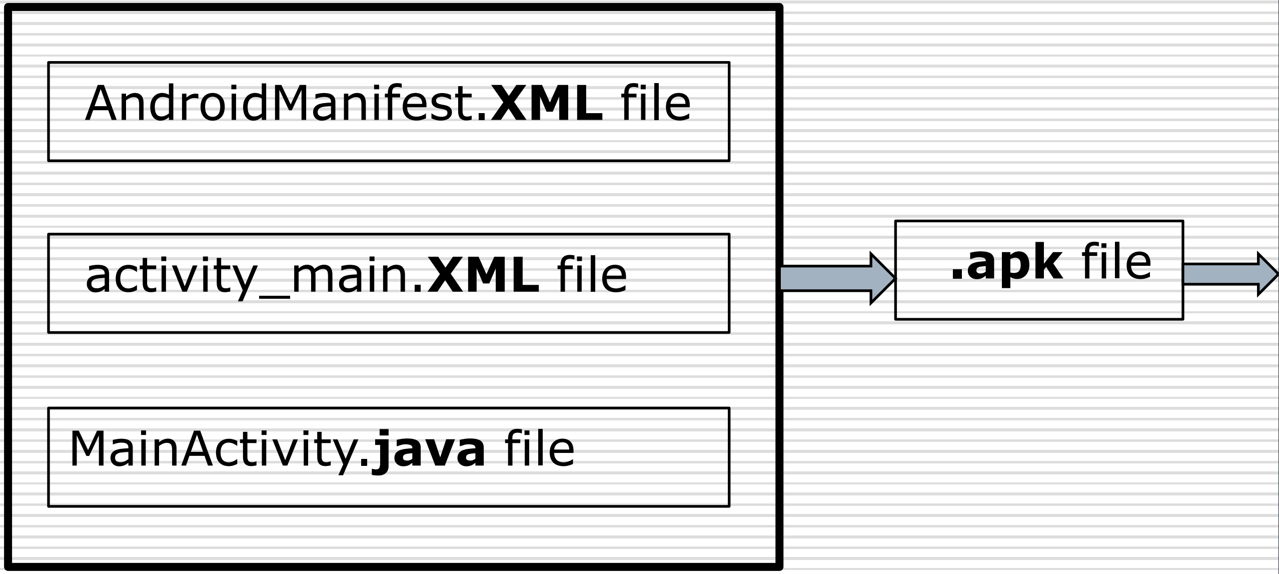
Handles User Interface and Events



```
package hk.ust.cse.comp107x.helloworld;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;


public class MainActivity extends AppCompatActivity {

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

    }
}
```

Million Rupee question

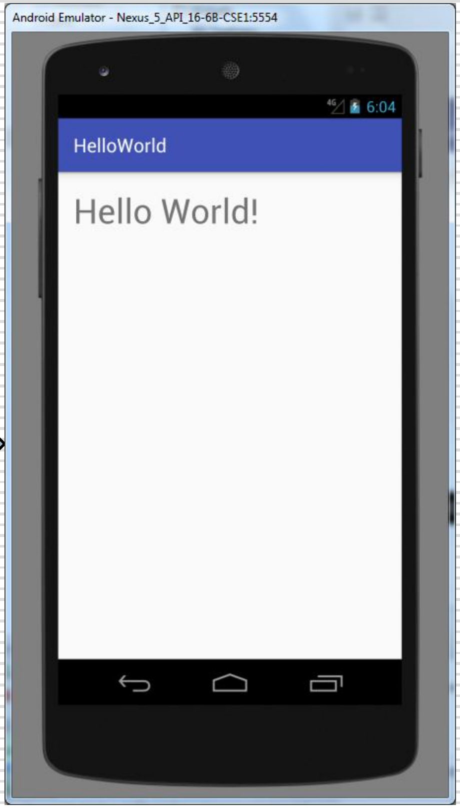
User Interface components have been created in the *activity_main.xml,* but the event handling code has to be written in pure Java *MainActivity.java*. There is a **Million Rupee question**; how does Java identify the XML user interface components?


AndroidManifest.**XML** file

activity_main.**XML** file

MainActivity.**java** file

.apk file


Android Emulator - Nexus_5_API_16-6B-CSE1:5554

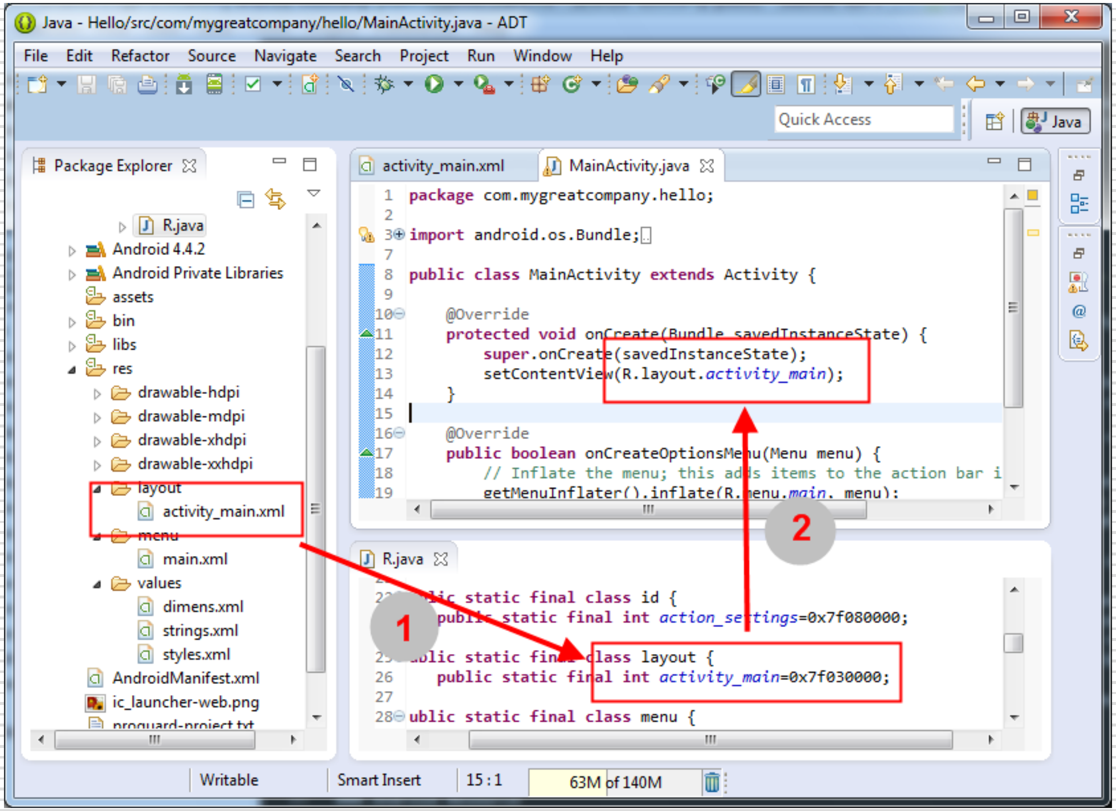
6:04

HelloWorld

Hello World!

R.java

- **R.java** file is the glue between the world of Java and the world of resources. It is an **automatically generated** file, and as such, you never modify it. It is recreated every time you change anything in the res directory, for example, when you add an image or add/edit XML file.

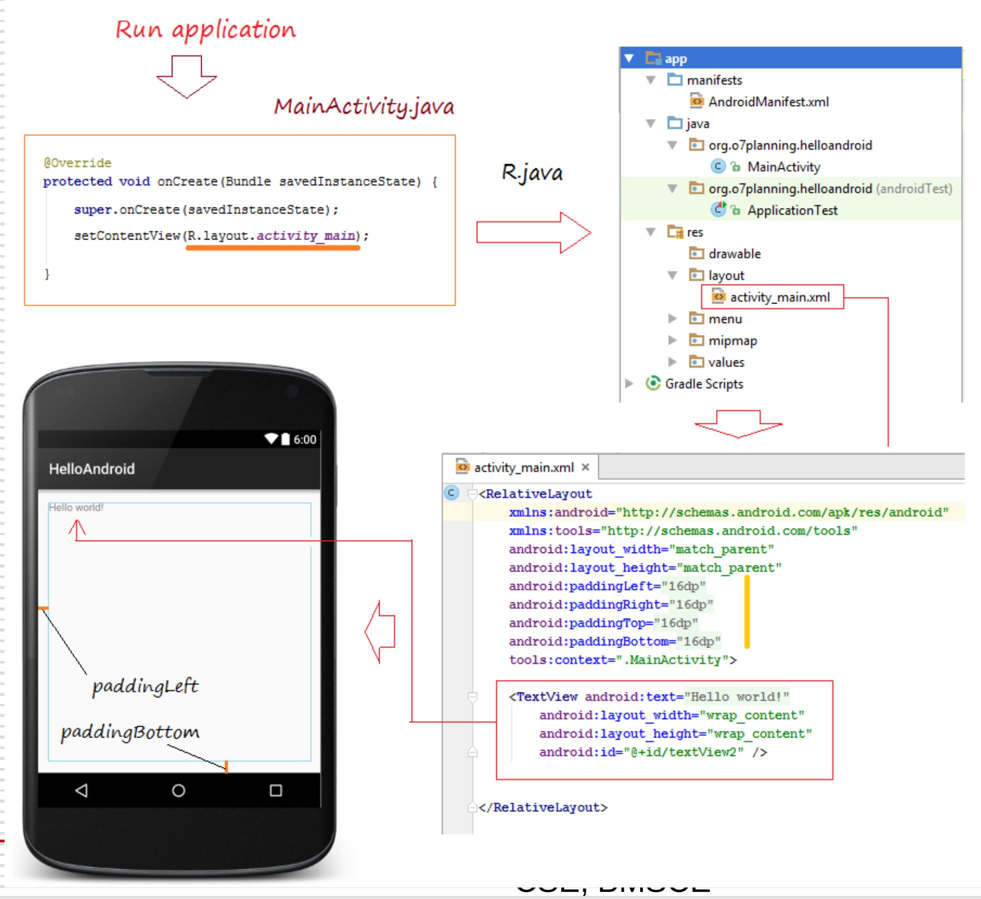


```
package com.mygreatcompany.hello;

import android.os.Bundle;

public class MainActivity extends Activity {

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

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar i
        getMenuInflater().inflate(R.menu.main, menu);
```

```
 lic static final class id {
    public static final int action_settings=0x7f080000;

ublic static final class layout {
    public static final int activity_main=0x7f030000;

ublic static final class menu {
```

R.java

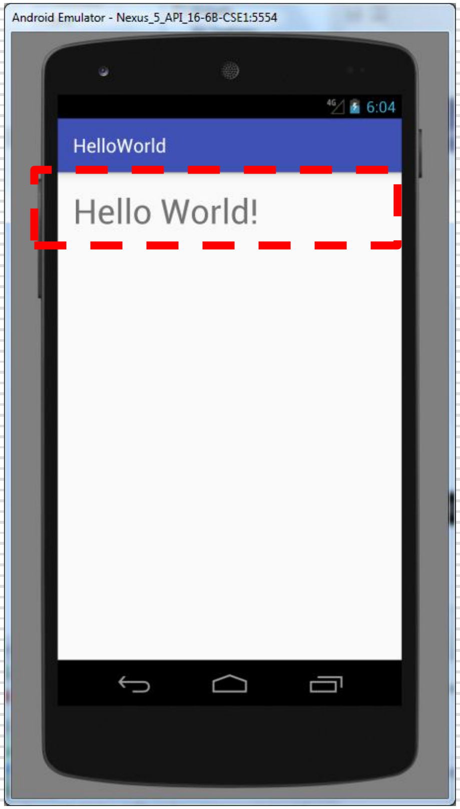
- So somewhere in the Project, you can use the constants to refer resources in the project. Such as, **R.layout.activity_main** is a constant which refers **activity_main.xml** file in the **res/layout** folder.



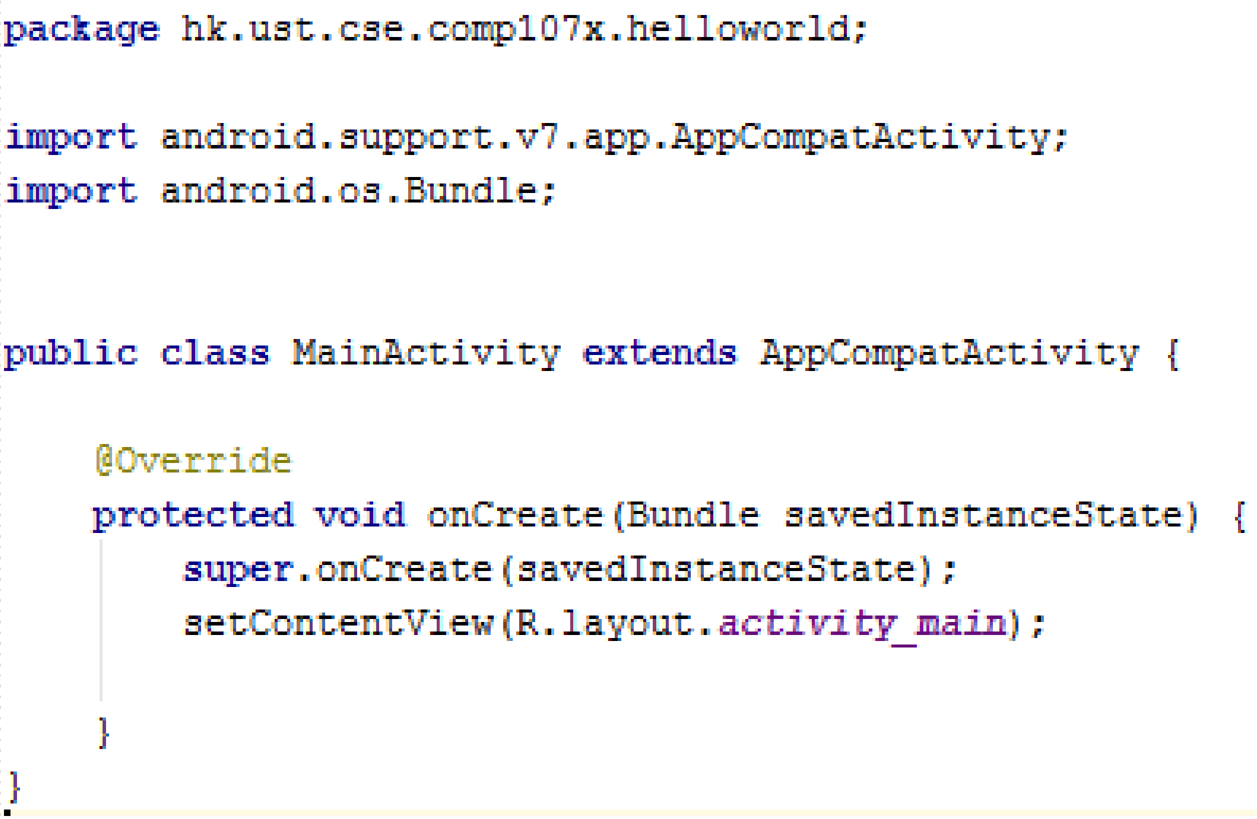
```java
@Override
protected void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);

    setContentView(R.layout.activity_main);

}
```

```xml
<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"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    tools:context=".MainActivity">

    <TextView android:text="Hello world!"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/textView2" />

</RelativeLayout>
```

Changing TextView content through MainActivity.java



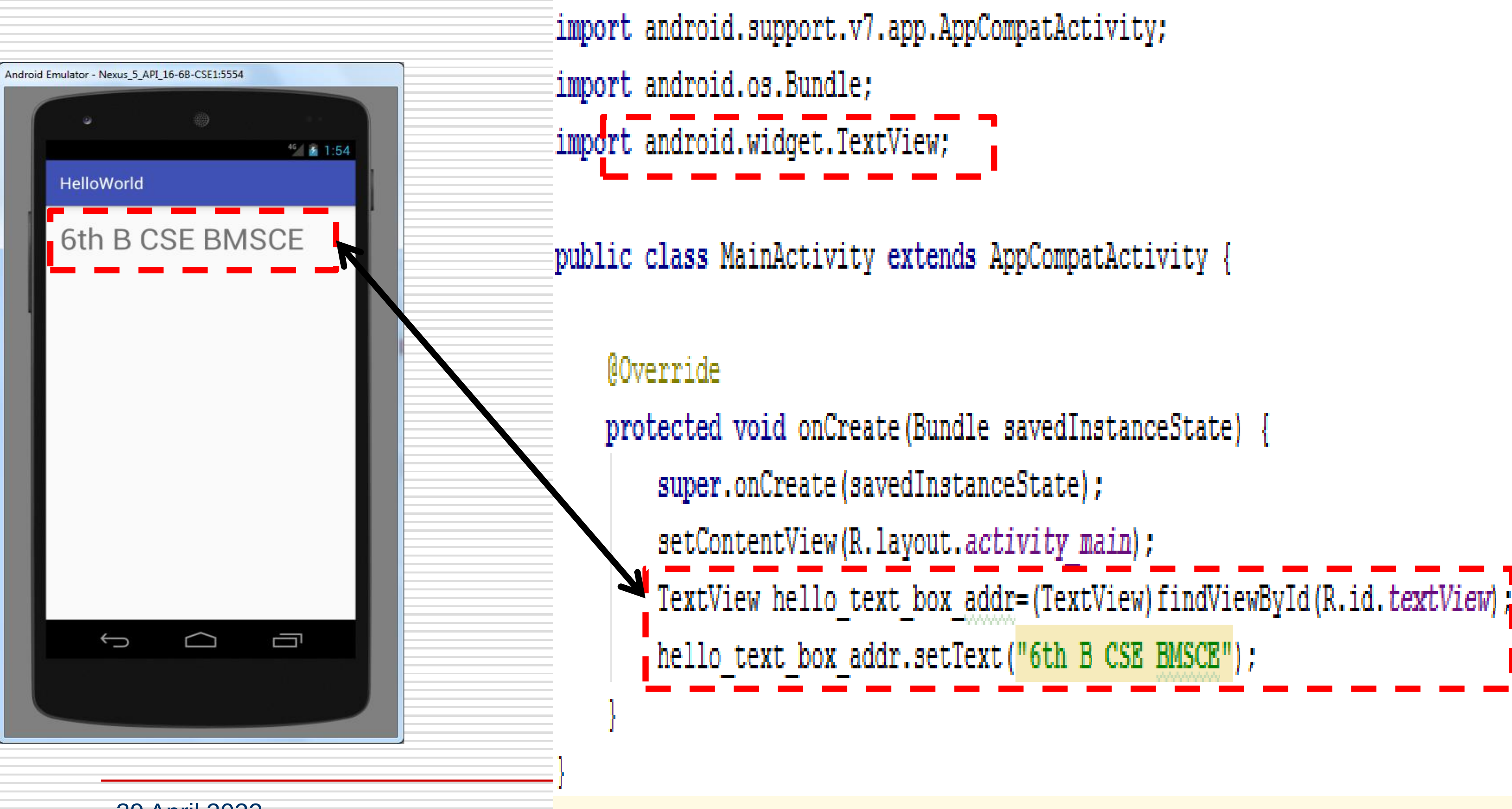
MainActivity.java file



```java
package hk.ust.cse.comp107x.helloworld;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;


public class MainActivity extends AppCompatActivity {

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

    }
}
```

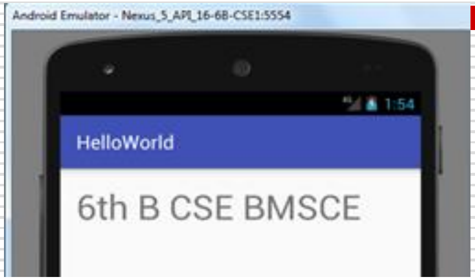
Changing TextView content through MainActivity.java

MainActivity.java file



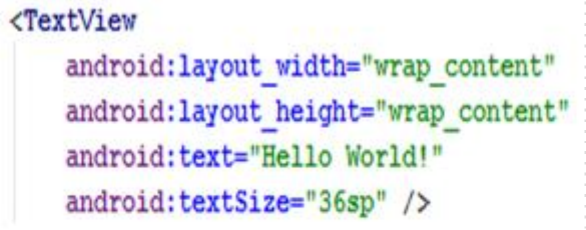
```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        TextView hello_text_box_addr=(TextView)findViewById(R.id.textView);
        hello_text_box_addr.setText("6th B CSE BMSCE");
    }
}
```

Changing TextView contents



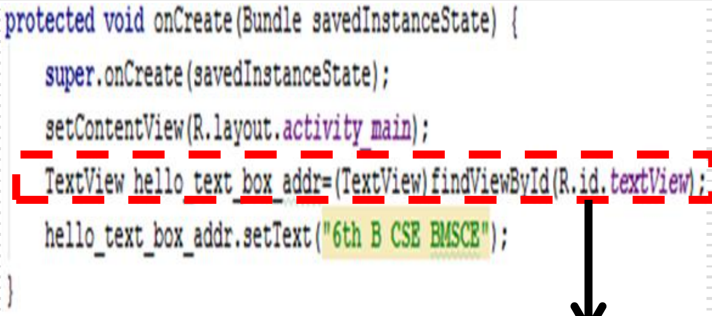
“HelloWorld” app

activity_main.xml file



```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!"
    android:textSize="36sp" />
```

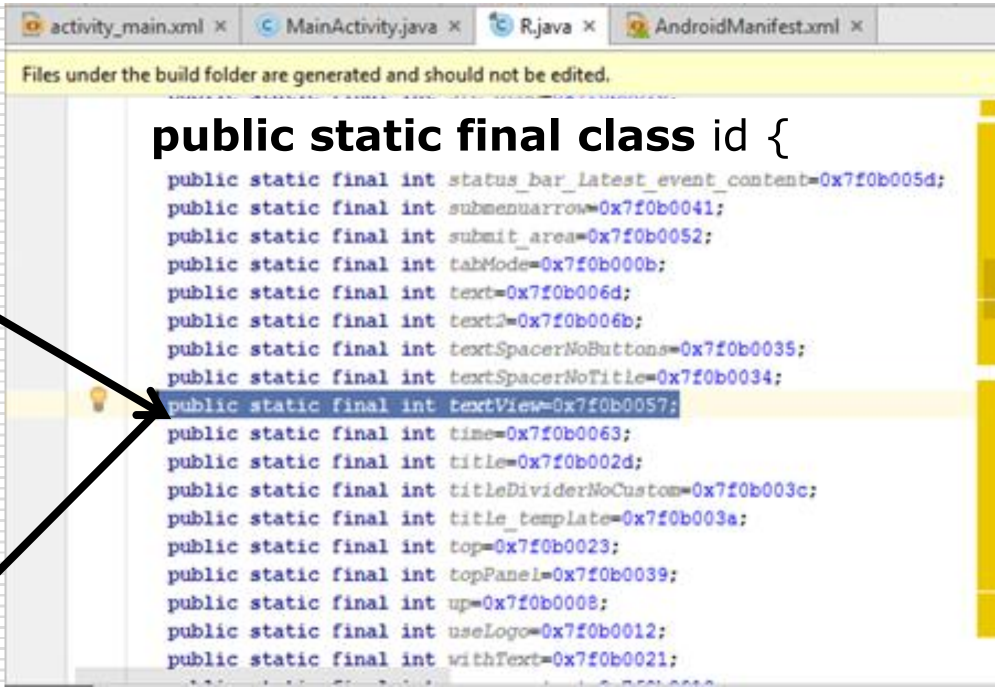
MainActivity.java



```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    TextView hello_text_box_addr=(TextView)findViewById(R.id.textView);
    hello_text_box_addr.setText("6th B CSE BMSCE");
}
```

findViewById(R.id.textView)

R.java



Files under the build folder are generated and should not be edited.

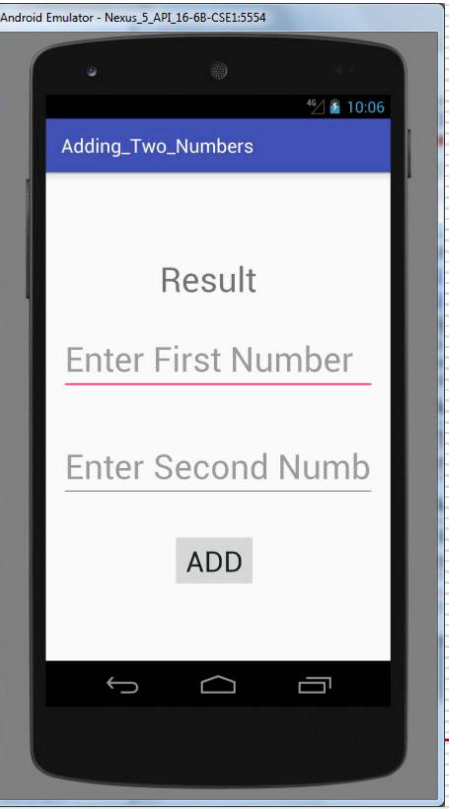
public static final class id {

```
public static final int status_bar_latest_event_content=0x7f0b005d;
public static final int submenuarrow=0x7f0b0041;
public static final int submit_area=0x7f0b0052;
public static final int tabMode=0x7f0b000b;
public static final int text=0x7f0b006d;
public static final int text2=0x7f0b006b;
public static final int textSpacerNoButtons=0x7f0b0035;
public static final int textSpacerNoTitle=0x7f0b0034;
public static final int textView=0x7f0b0057;
public static final int time=0x7f0b0063;
public static final int title=0x7f0b002d;
public static final int titleDividerNoCustom=0x7f0b003c;
public static final int title_template=0x7f0b003a;
public static final int top=0x7f0b0023;
public static final int topPanel=0x7f0b0039;
public static final int up=0x7f0b0008;
public static final int useLogo=0x7f0b0012;
public static final int withText=0x7f0b0021;
```

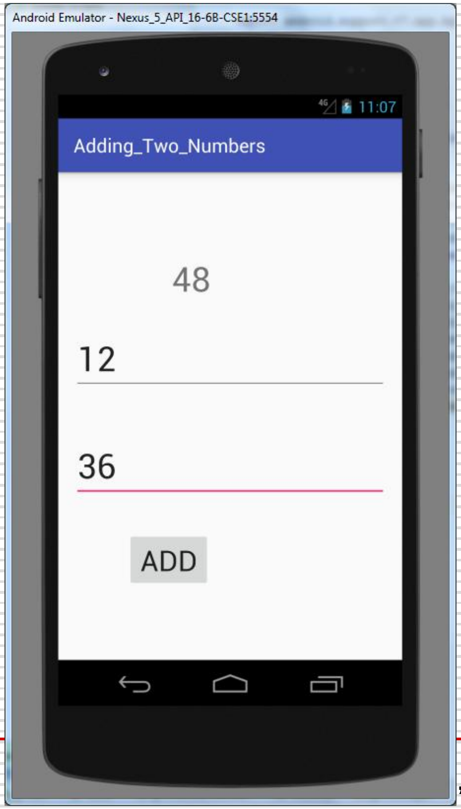
public static final int *textView*=0x7f0b0057;

To Do

- □ Consider on User Interface as shown in the diagram has been created. You should complete by writing the code in **MainActivity.java** which will add two numbers when user clicks on the button ADD.
- □ Given: ID of first number is ***editText,*** second number ***editText2***, result is ***textView****.* Hint: You should use *findViewById* , *getText* , *Integer.parseInt* to convert text to integer, and *setText*



Android Emulator - Nexus_5_API_16-6B-CSE1:5554



Android Emulator - Nexus_5_API_16-6B-CSE1:5554

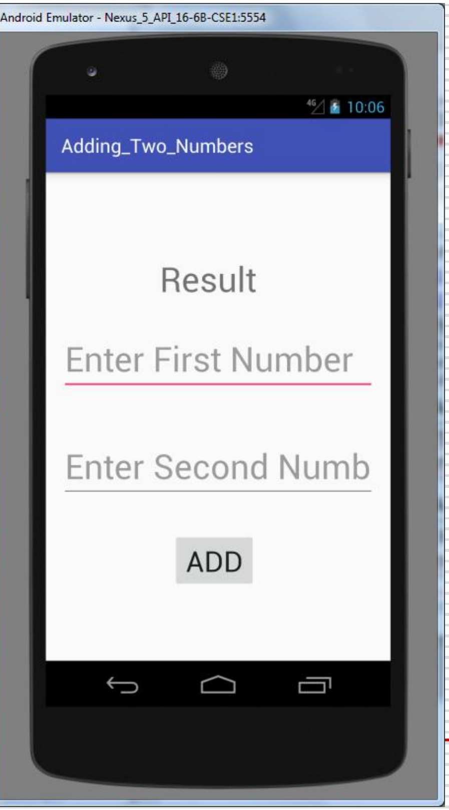
48

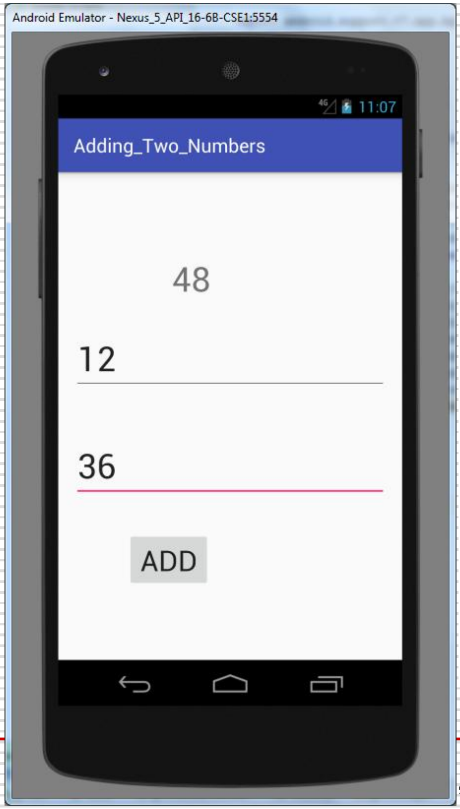
12

36

To Do

- Consider on User Interface as shown in the diagram has been created. You should complete by writing the code in **MainActivity.java** which will add two numbers when user clicks on the button ADD.
- Given: ID of first number is ***editText*,** second number ***editText2***, result is ***textView*.** Hint: You should use *findViewById* , *getText* , *Integer.parseInt* to convert text to integer, and *setText*

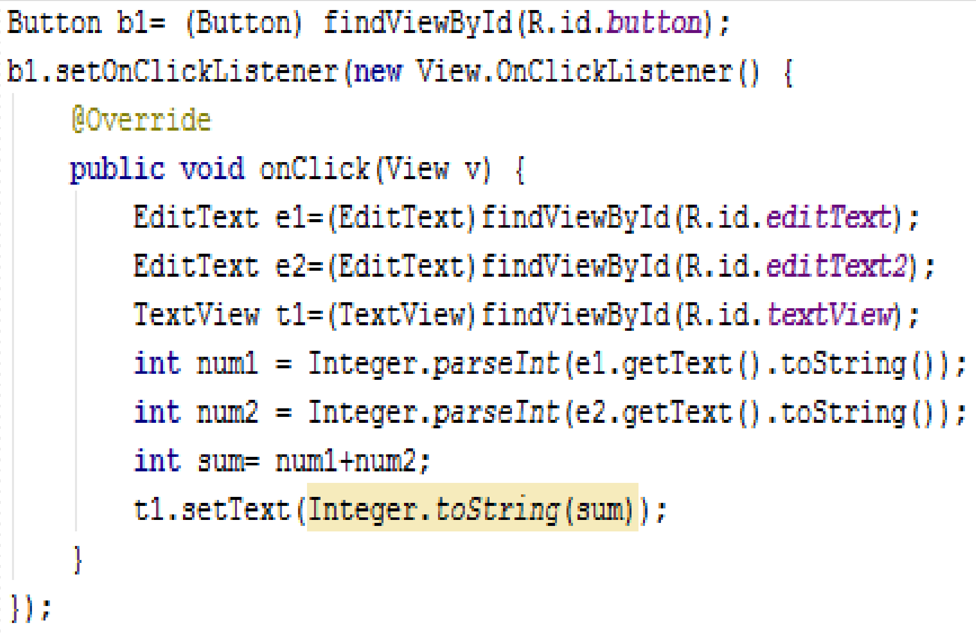




48

12

36



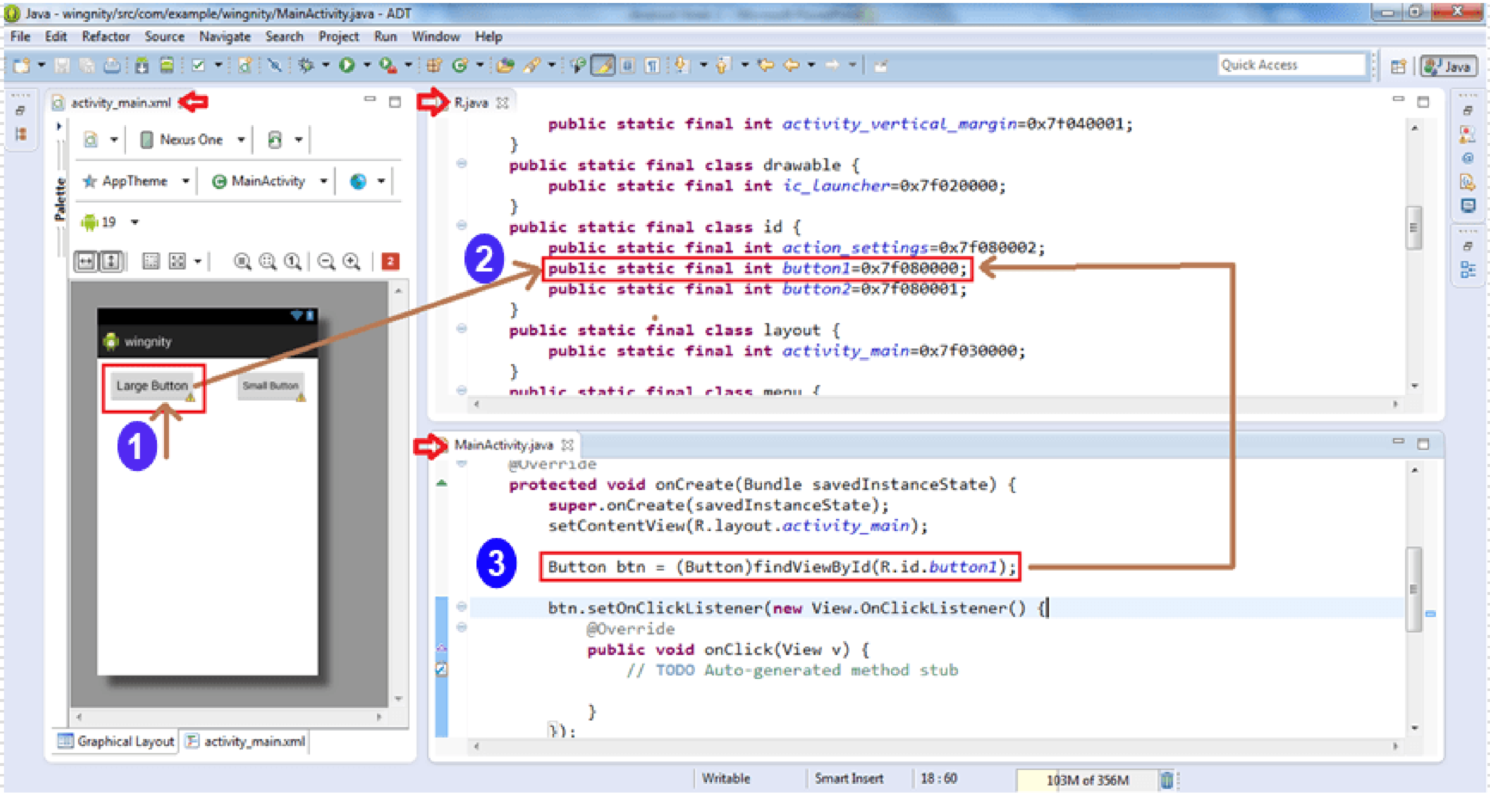
```
Button b1= (Button) findViewById(R.id.button);
b1.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        EditText e1=(EditText)findViewById(R.id.editText);
        EditText e2=(EditText)findViewById(R.id.editText2);
        TextView t1=(TextView)findViewById(R.id.textView);
        int num1 = Integer.parseInt(e1.getText().toString());
        int num2 = Integer.parseInt(e2.getText().toString());
        int sum= num1+num2;
        t1.setText(Integer.toString(sum));
    }
});
```

Question

Why designs are kept in **.XML** file ? Why not in **.java** file ?

Why XML files are needed ?

- It is easier to design better using XML files instead of doing the same via Java code. Also design can be seen instantly while in the latter case you may need to run the application first.



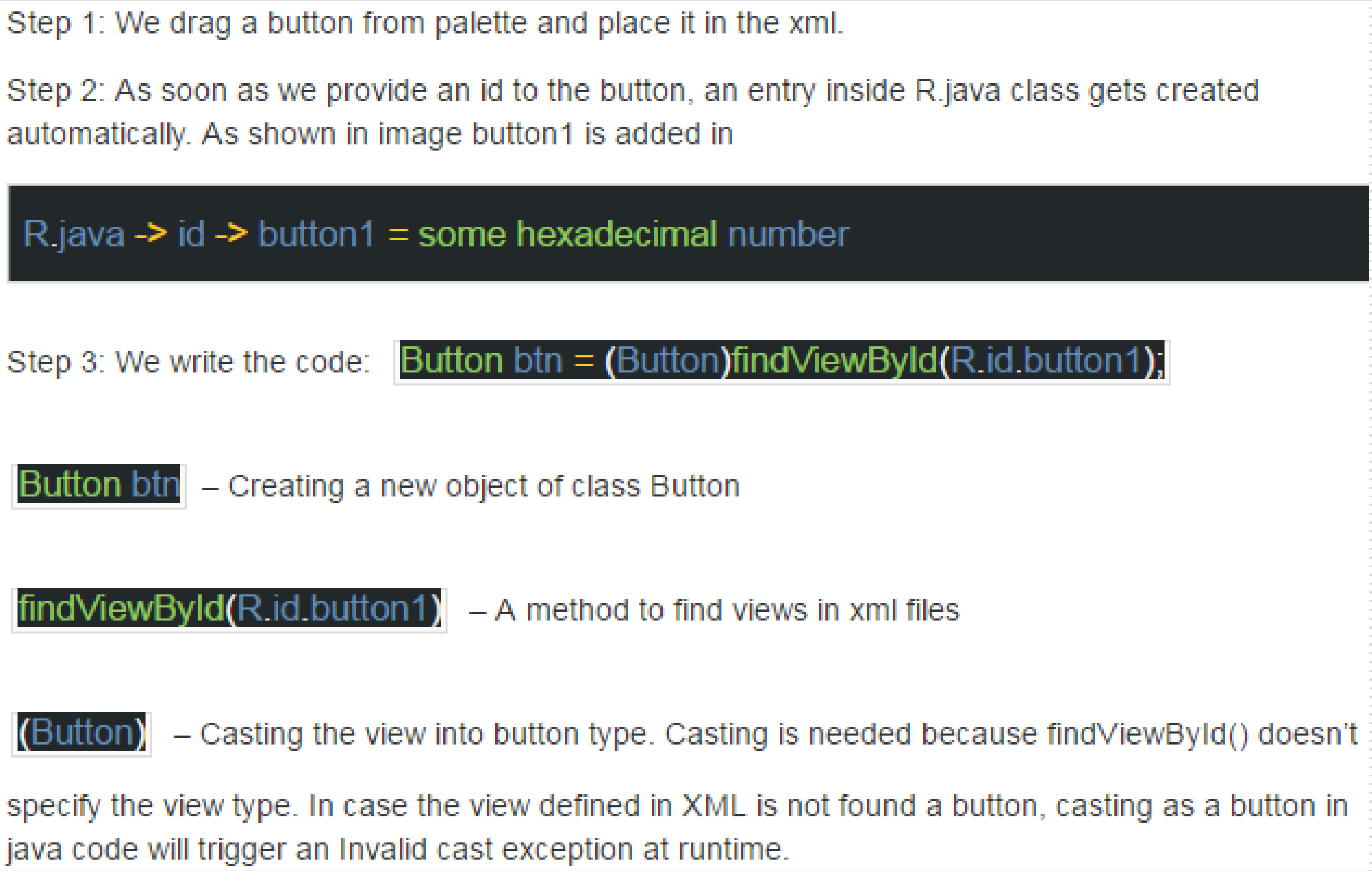
R.java

```
        public static final int activity_vertical_margin=0x7f040001;
    }
    public static final class drawable {
        public static final int ic_launcher=0x7f020000;
    }
    public static final class id {
        public static final int action_settings=0x7f080002;
        public static final int button1=0x7f080000;
        public static final int button2=0x7f080001;
    }
    public static final class layout {
        public static final int activity_main=0x7f030000;
    }
    public static final class menu {
```

MainActivity.java

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

        Button btn = (Button)findViewById(R.id.button1);

        btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // TODO Auto-generated method stub

            }
        });
```

How R.java works and bind XML and code files



Step 1: We drag a button from palette and place it in the xml.

Step 2: As soon as we provide an id to the button, an entry inside R.java class gets created automatically. As shown in image button1 is added in

```
R.java -> id -> button1 = some hexadecimal number
```

Step 3: We write the code: `Button btn = (Button)findViewById(R.id.button1);`

`Button btn` – Creating a new object of class Button

`findViewById(R.id.button1)` – A method to find views in xml files

`(Button)` – Casting the view into button type. Casting is needed because findViewById() doesn't specify the view type. In case the view defined in XML is not found a button, casting as a button in java code will trigger an Invalid cast exception at runtime.

List the different types of resources that can be created under **res** folder in Android. Write an example for each.

Simple Values, Strings, Colors, Dimensions, Styles and themes, Drawables, Layout, Animation, Menus

Strings

- The String resources provide the text that we can use in both Java code and UI layout.

string.xml under res folder

```
<resources>
<string name="app_name">AndroidResourcesLessons</string>
<string name="hello_world">Hello world!</string>
</resources>
```

In this file there are String resources with name app_name and hello_world. We can use these String resources in the layout by entering a text "@string/" and followed by the id of the resource that we want.

Accessing string resource inside **.xml** file

```
<TextView
 android:text="@string/hello_world" />
```

Accessing string resource inside **.java** file

```
TextView XYZ=(TextView) findViewById (R.id.textview_hello_world);
XYZ.setText(R.string.hello_world);
```

Question: What will be displayed inside the app of the TextView

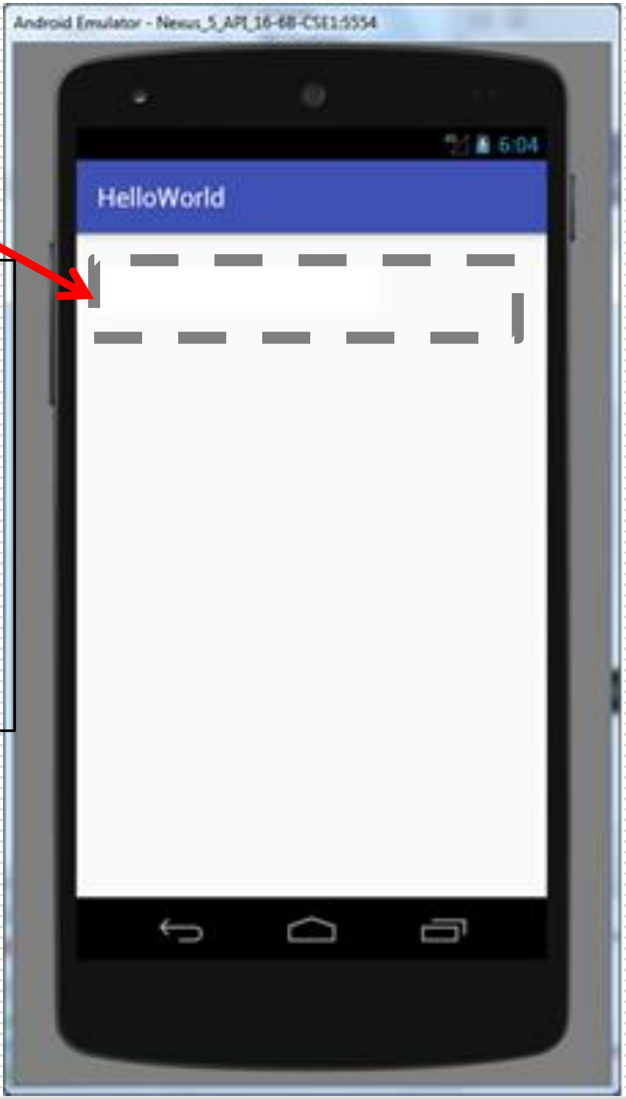
Consider an UI layout created with TestView by id **abc.**

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout  ...............>

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/cse" />
</RelativeLayout>
```

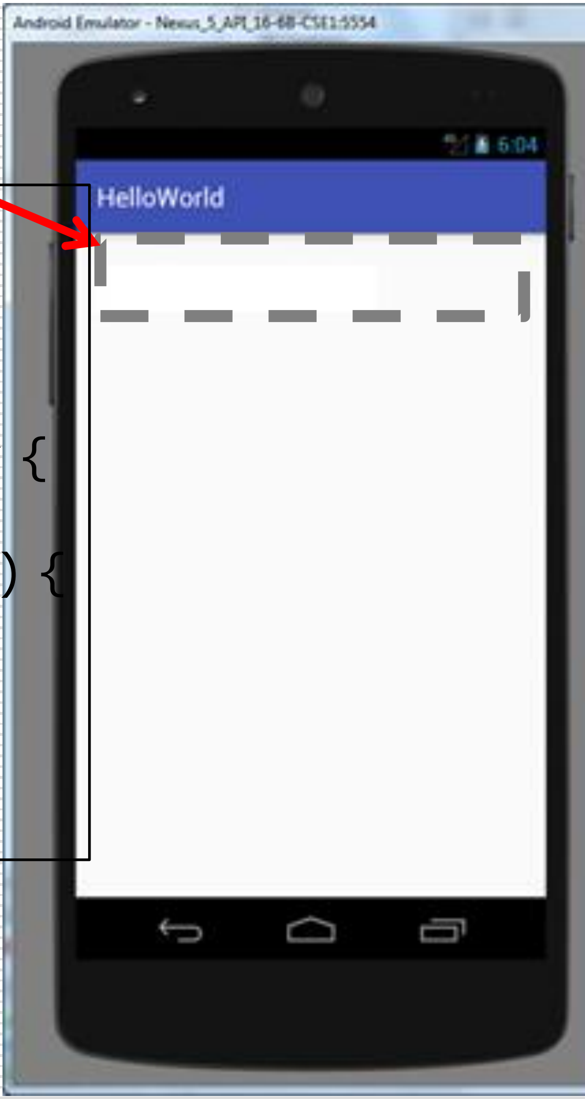
strings.xml

```
<resources>
<string name="cse">Utsav</string>
</resource>
```



Question: What will be displayed inside the app of the TextView

Consider an UI layout created with TestView by id **abc.**



MainActivity.java

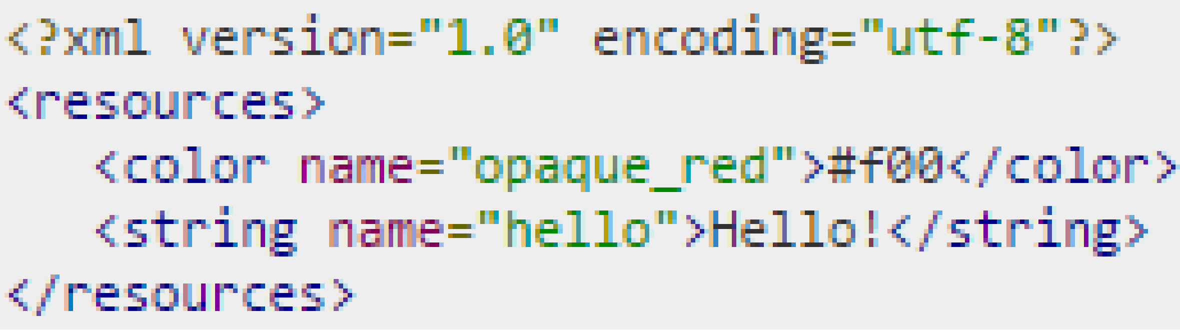
```
package com.example.umadevi.helloworld;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.TextView;
public class MainActivity extends AppCompatActivity {
 @Override
 protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_main);
 TextView XYZ=(TextView) findViewById (R.id.abc);
 XYZ.setText(R.string.cse);  } }
```

strings.xml

```
<resources>
<string name=“cse">Phase Shift</string>
</resource>
```

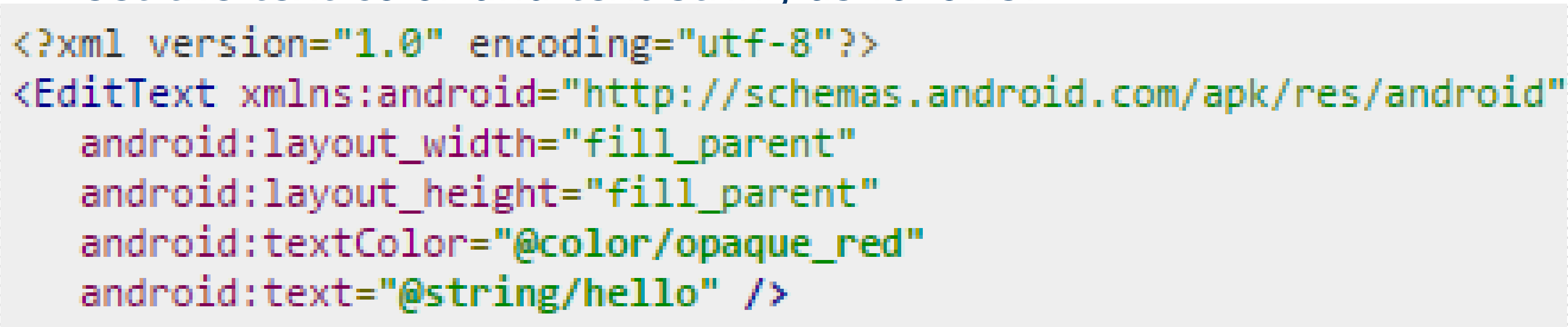
Colors

- □ Consider the following resource XML *res/values/strings.xml* file that includes a color resource and a string resource



```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <color name="opaque_red">#f00</color>
   <string name="hello">Hello!</string>
</resources>
```

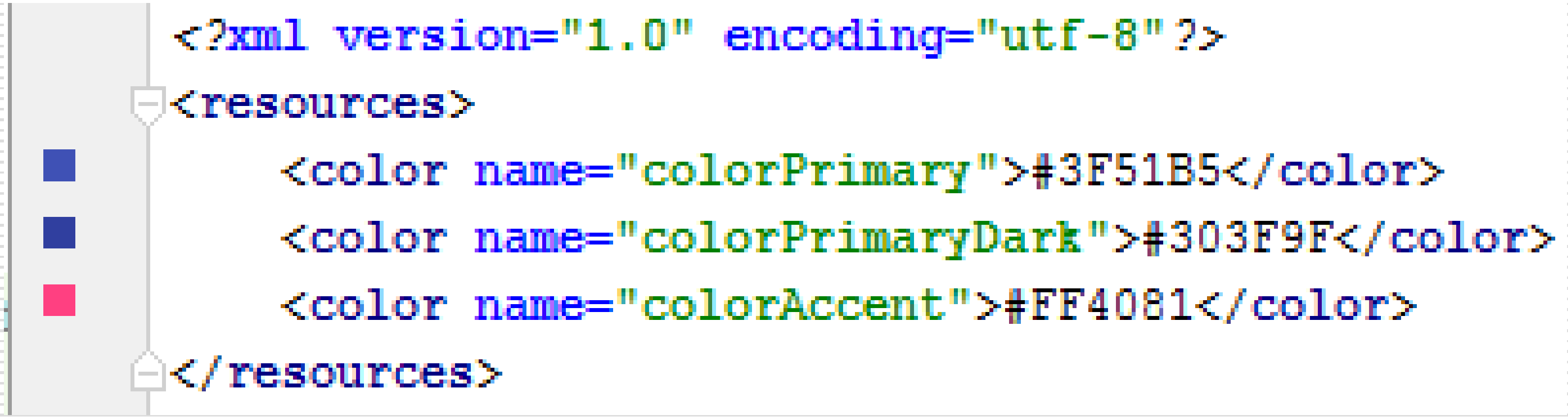
- □ Now we can use these resources in the following layout file to set the text color and text string as follows:



```xml
<?xml version="1.0" encoding="utf-8"?>
<EditText xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:textColor="@color/opaque_red"
   android:text="@string/hello" />
```

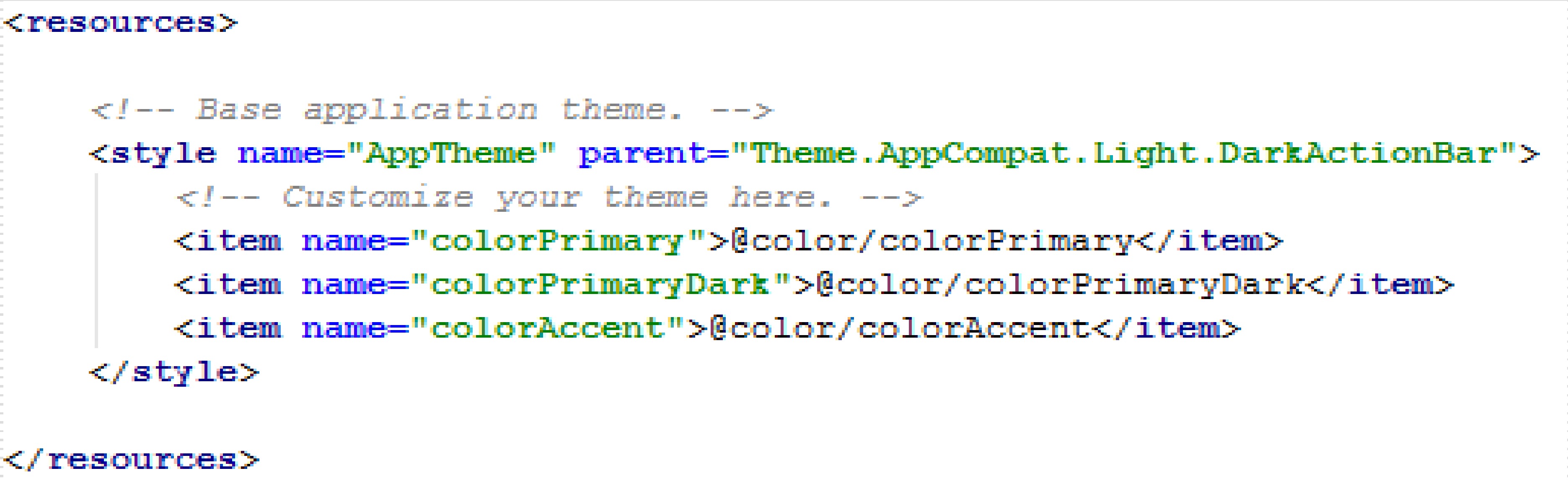
Colors

- Consider the following color resource XML ***res/values/colors.xml*** file



```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#3F51B5</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF4081</color>
</resources>
```

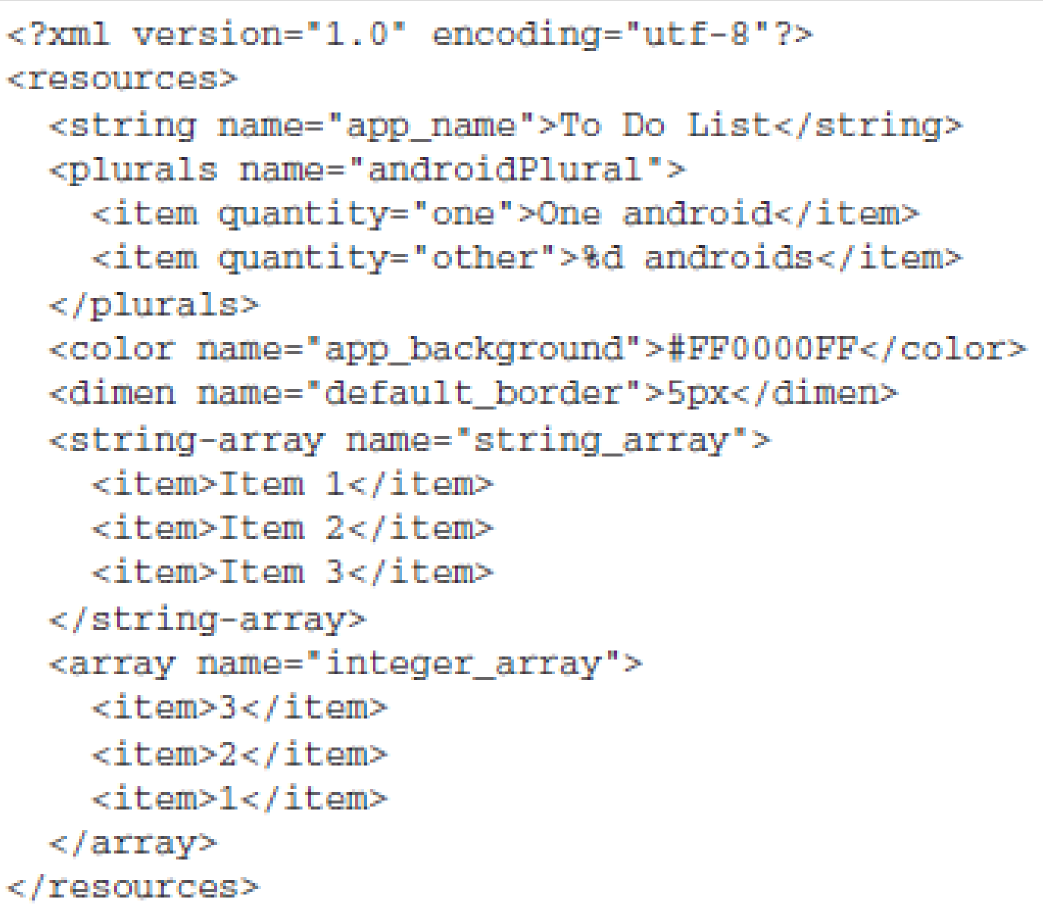
- Now we can use these resources in the ***res/values/styles.xml*** file as follows:



```
<resources>

    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>

</resources>
```

Simple Values

- Question: Identify how many resources are defined in the following resource file



```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">To Do List</string>
  <plurals name="androidPlural">
    <item quantity="one">One android</item>
    <item quantity="other">%d androids</item>
  </plurals>
  <color name="app_background">#FF0000FF</color>
  <dimen name="default_border">5px</dimen>
  <string-array name="string_array">
    <item>Item 1</item>
    <item>Item 2</item>
    <item>Item 3</item>
  </string-array>
  <array name="integer_array">
    <item>3</item>
    <item>2</item>
    <item>1</item>
  </array>
</resources>
```

Accessing and defining array of integers in XML.

- □ File location:res/values/*filename*.xml

```
<resources>
    <integer-array  name="integer_array_name">
        <item>int value</item>
    </integer-array>
</resources>
```

- □ Example:XML file saved at res/values/integers.xml:
- □ <?xml version="1.0" encoding="utf-8"?>

```
<resources>
    <integer-array name="bits">
        <item>4</item>
        <item>8</item>
        <item>16</item>
        <item>32</item>
    </integer-array>
</resources>
```

This application code retrieves the integer array:

- □ Resources res = getResources();

```
int[] bits = res.getIntArray(R.array.bits);
```

Dimensions

Dimensions are most commonly referenced within style and layout resources. They’re useful for creating layout constants, such as borders and font heights. To specify a dimension resource, use the dimen tag, specifying the dimension value, followed by an identifier describing the scale of your dimension:

- px (screen pixels)
- in (physical inches)
- pt (physical points)
- mm (physical millimeters)
- dp (density-independent pixels)
- sp (scale-independent pixels)

The following XML snippet shows how to specify dimension values for a large font size and a standard border:

- <dimen name=”standard_border”>5dp</dimen>
- <dimen name=”large_font_size”>16sp</dimen>

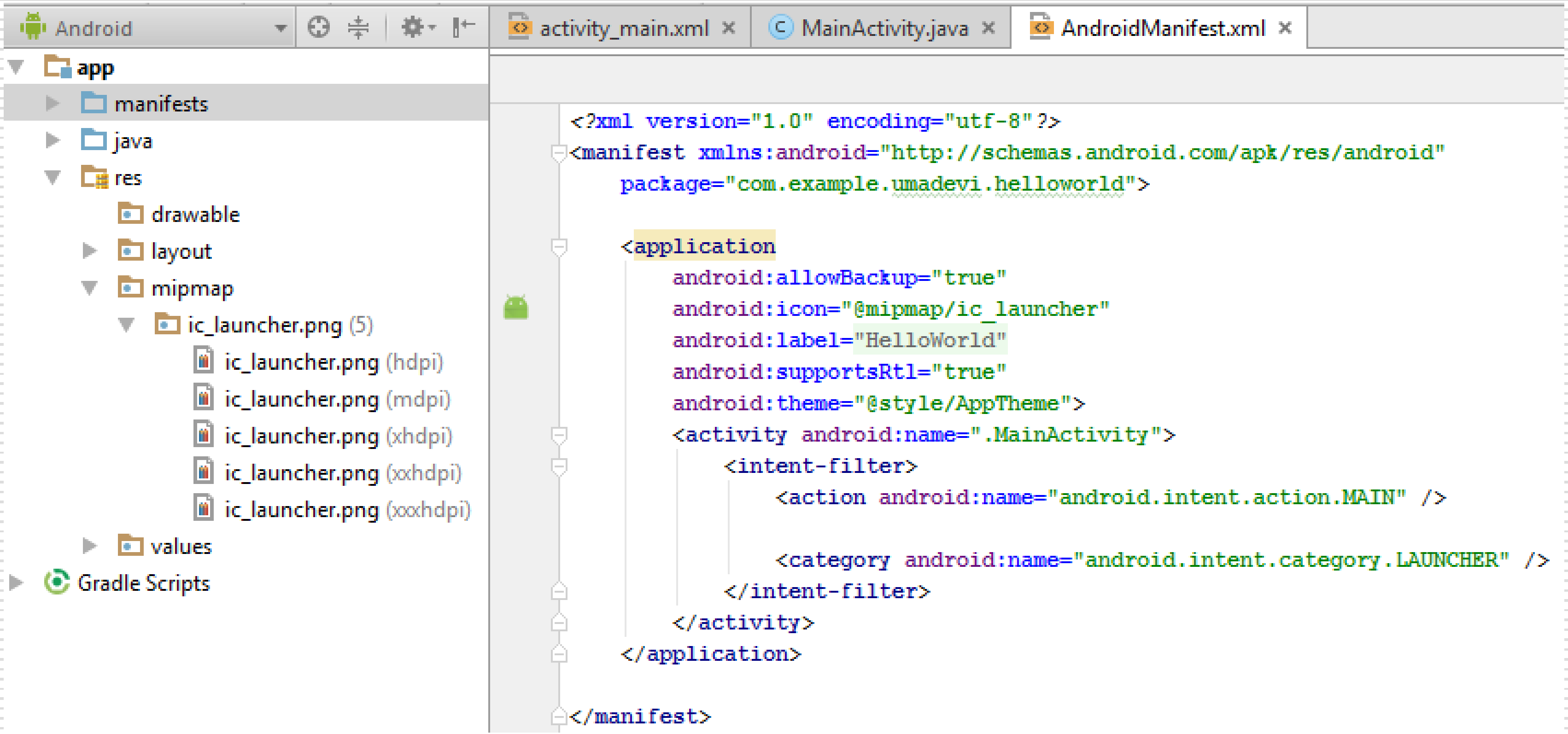
Styles and Themes

- To create a style, use a style tag that includes a name attribute and contains one or more item tags.

```
<resources>
<style name=”base_text”>
<item name=”android:textSize”>14sp</item>
<item name=”android:textColor”>#111</item>
</style>
</resources>
```

Drawables

- Drawable resources include bitmaps and NinePatches (stretchable PNG images).
- All Drawables are stored as individual files in the res/drawable folder. Note that it's good practice to store bitmap image assets in the appropriate drawable -ldpi, -mdpi, -hdpi, and -xhdpi folders



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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="HelloWorld"
        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>
    </application>

</manifest>
```

Drawables

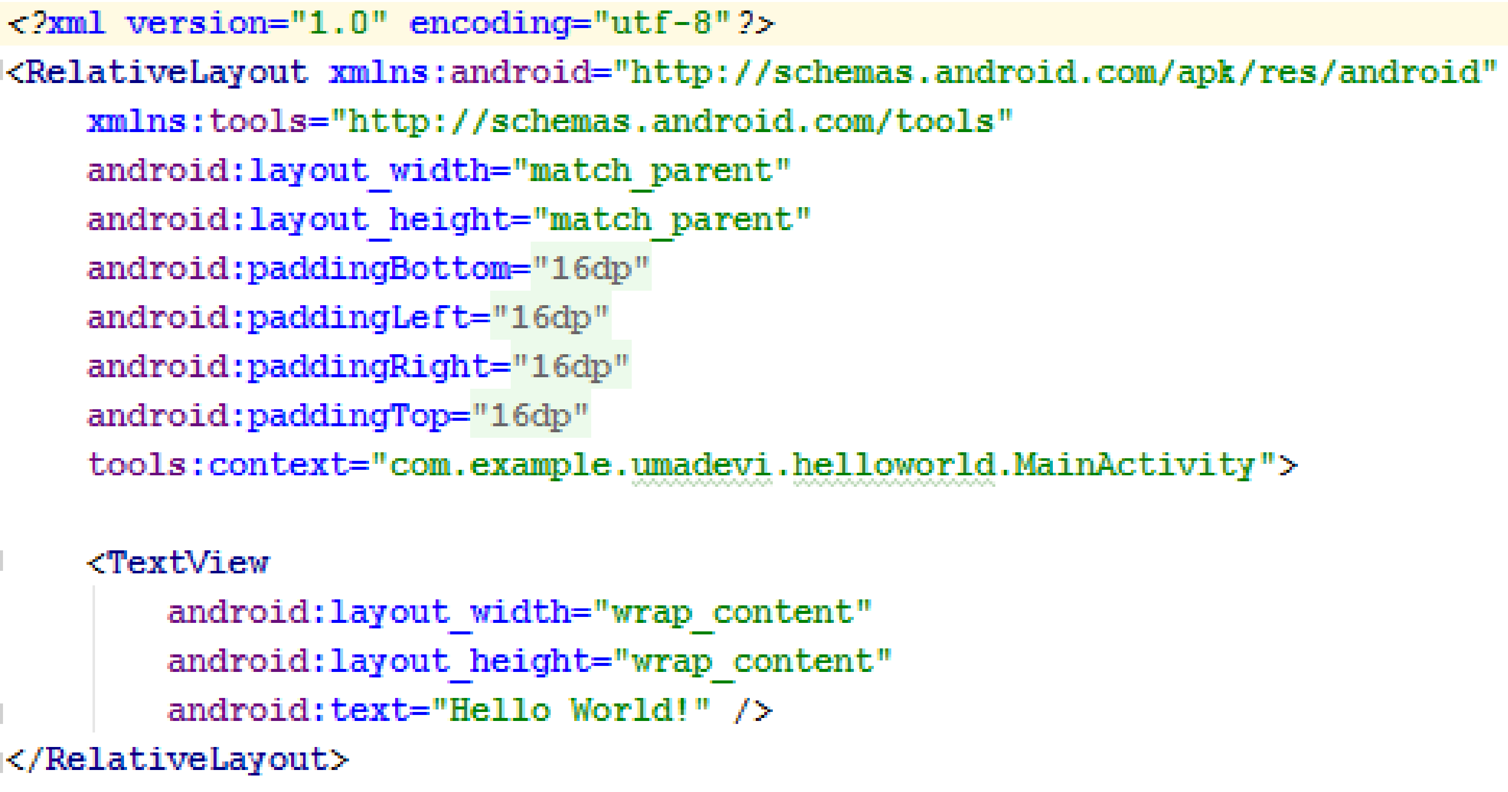
For Supporting Multiple Screens in Android different resolutions should be supported for images

- □ Low-density (Low-density Dots Per Inch)
- □ Medium-density (MDPI)
- □ High-density (HDPI)
- □ Extra high-density (XHDPI)
- □ Extra extra high-density (XXHDPI)
- □ Extra extra extra high-density (XXXHDPI)

Layout

- Layout resources enable to decouple your presentation layer from the business logic by designing UI layouts in XML rather than constructing them in code.

activity_main.xml



```
<?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"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    tools:context="com.example.umadevi.helloworld.MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!" />
</RelativeLayout>
```

Menues

- We can use menu resources to define both Activity and context menus within the application.
- Each menu definition is stored in a separate file, each containing a single menu, in the res/menu folder — the filename then becomes the resource identifier.

Example: Simple menu layout resource

```
<?xml version=”1.0” encoding=”utf-8”?>
<menu xmlns:android=”http://schemas.android.com/apk/res/android”>
<item android:id=”@+id/menu_refresh”
android:title=”@string/refresh_mi” />
<item android:id=”@+id/menu_settings”
android:title=”@string/settings_mi” />
</menu>
```

Animations

Android supports three types of animation:

- □ Property animations — A tweened animation that can be used to potentially animate any property on the target object by applying incremental changes between two values. This can be used for anything from changing the color or opacity of a View to gradually fade it in or out, to changing a font size, or increasing a character’s hit points.
- □ View animations — Tweened animations that can be applied to rotate, move, and stretch a View.
- □ Frame animations — Frame-by-frame “cell” animations used to display a sequence of Drawable images.

Tween Animation

- Tween Animation takes some parameters such as start value , end value, size , time duration , rotation angle e.t.c and perform the required animation on that object.
- Android has provided us with a class called Animation.
- In order to perform animation in android , we are going to call a static function **loadAnimation()** of the class **AnimationUtils**. We are going to receive the result in an instance of Animation Object.
- Its syntax is as follows −

```
Animation animation =
AnimationUtils.loadAnimation(getApplicationContext(),
R.anim.myanimation);
```

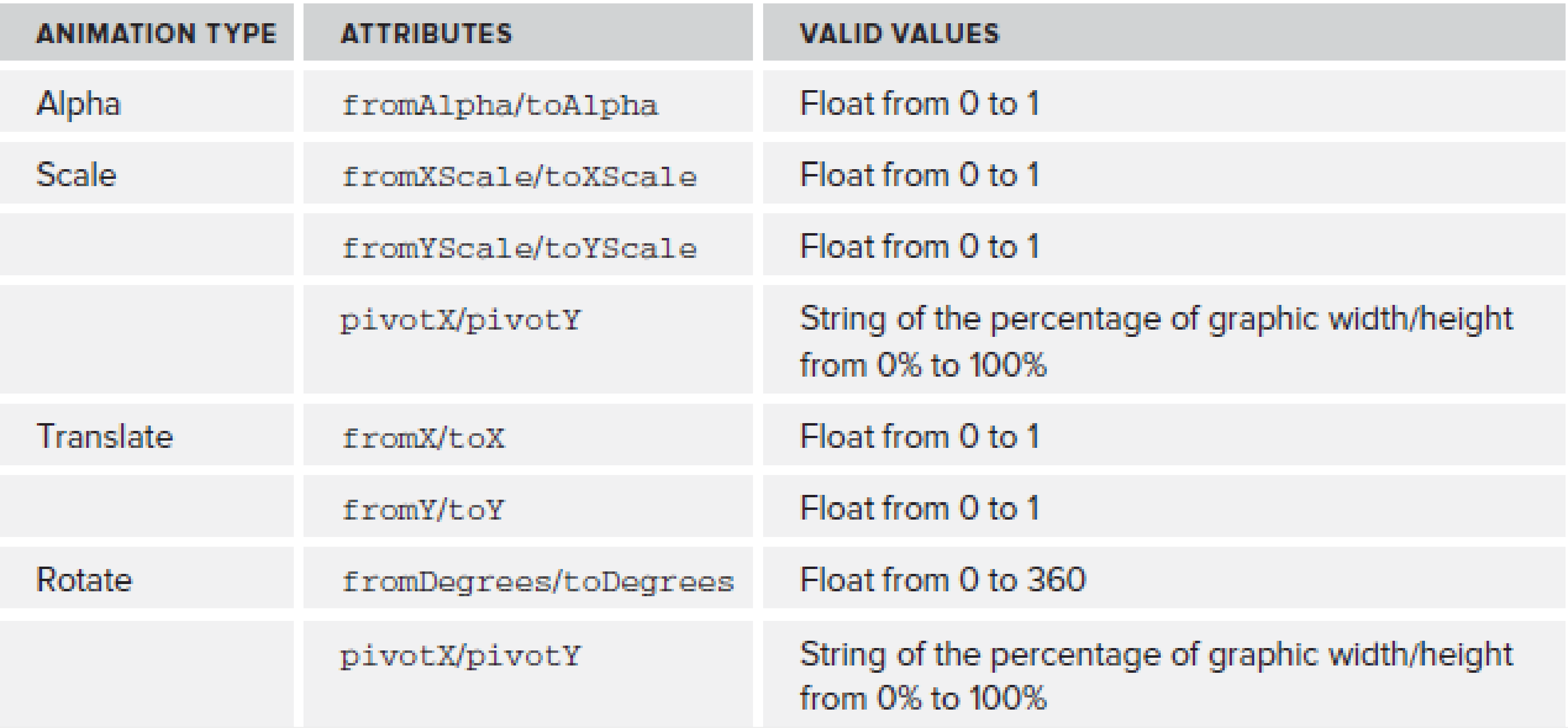
- Note the second parameter. It is the name of the our animation xml file. We have to create a new folder called **anim** under res directory and make an xml file under anim folder.

Android Animation class functions

Sl.No	Method & Description
1	**start()** This method starts the animation.
2	**setDuration(long duration)** This method sets the duration of an animation.
3	**getDuration()** This method gets the duration which is set by above method
4	**end()** This method ends the animation.
5	**cancel()** This method cancels the animation.

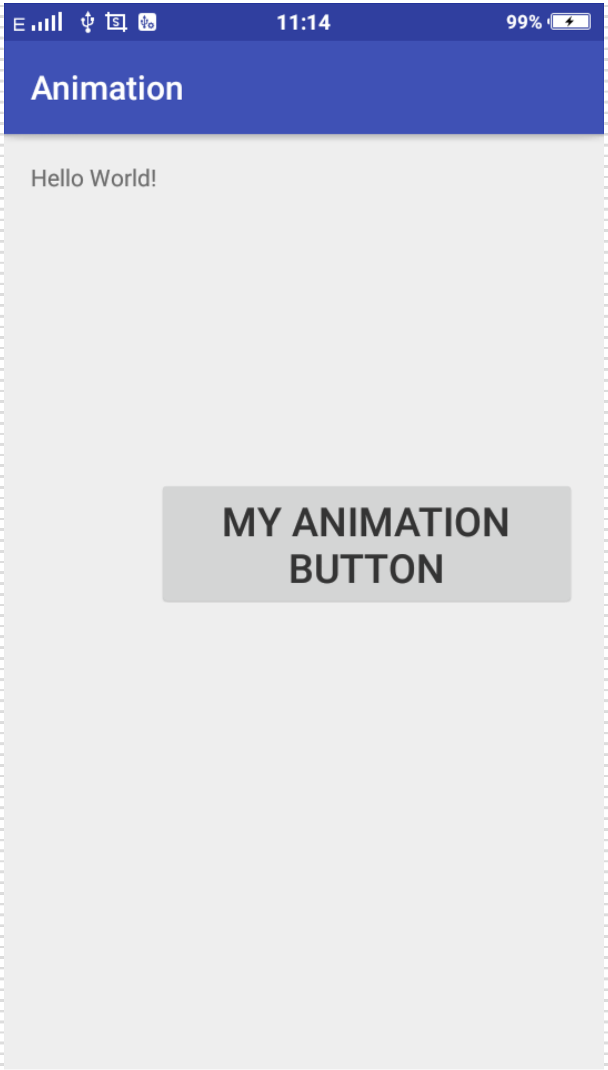
View Animation

- alpha (fading), scale (scaling), translate (movement),or rotate (rotation).
- Animation attributes



ANIMATION TYPE	ATTRIBUTES	VALID VALUES
Alpha	`fromAlpha/toAlpha`	Float from 0 to 1
Scale	`fromXScale/toXScale`	Float from 0 to 1
	`fromYScale/toYScale`	Float from 0 to 1
	`pivotX/pivotY`	String of the percentage of graphic width/height from 0% to 100%
Translate	`fromX/toX`	Float from 0 to 1
	`fromY/toY`	Float from 0 to 1
Rotate	`fromDegrees/toDegrees`	Float from 0 to 360
	`pivotX/pivotY`	String of the percentage of graphic width/height from 0% to 100%

Alpha (fading) animation

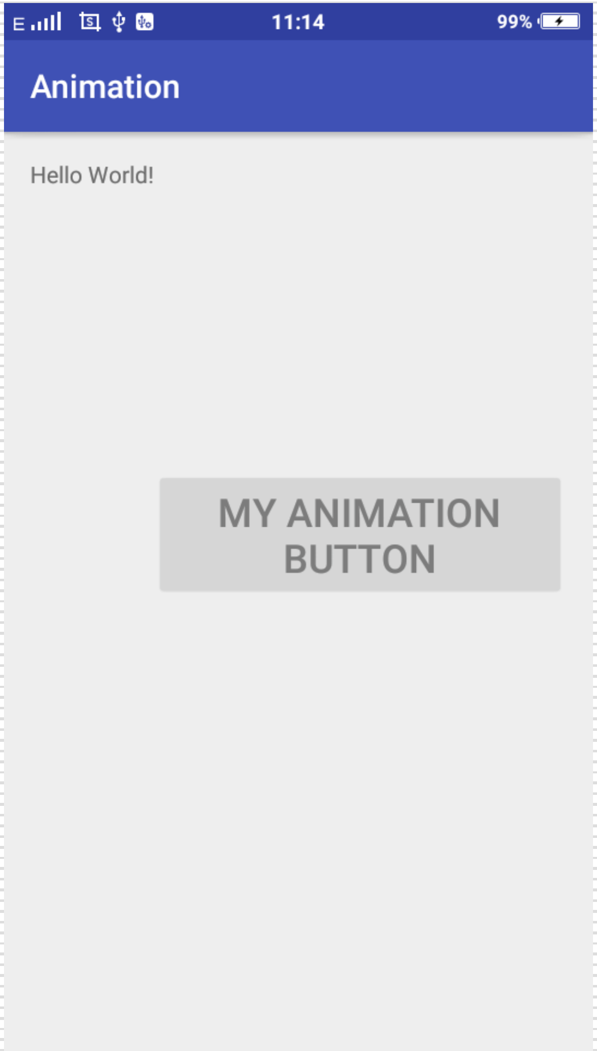


11:14

99%

Animation

Hello World!



11:14

99%

Animation

Hello World!

Alpha (fading) animation

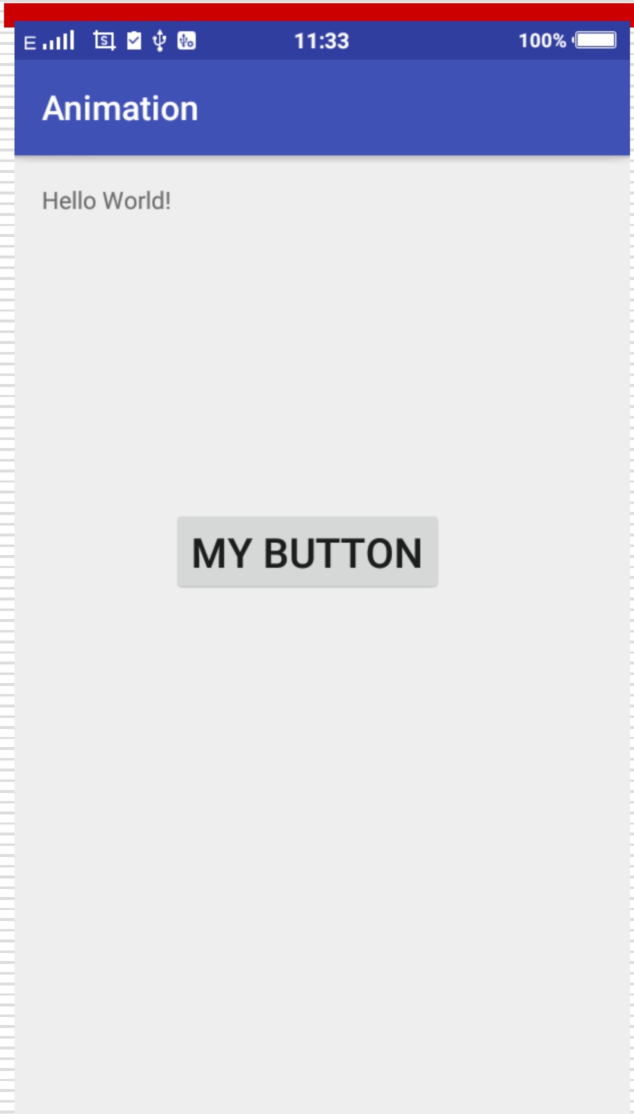
My_animation.xml

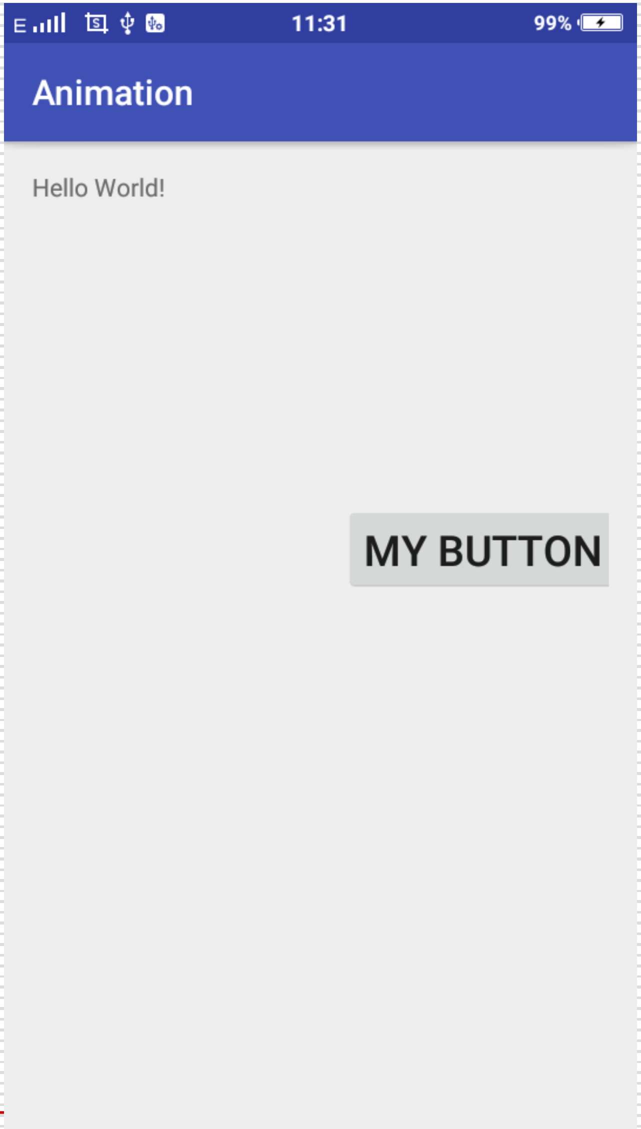
```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
<alpha
    android:fromAlpha="1.0"
    android:toAlpha="0.0"
    android:duration="2000"
    android:repeatCount="infinite"
    android:repeatMode="reverse"    />
</set>
```

MainActivity.java

```
package com.example.umadevi.animation;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {
    Button btn1;
    Animation anim1;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        btn1=(Button)findViewById(R.id.button);
        anim1= AnimationUtils.loadAnimation(this,R.anim.my_animation);
        btn1.startAnimation(anim1);     } }
```

Translate animation





Translate animation

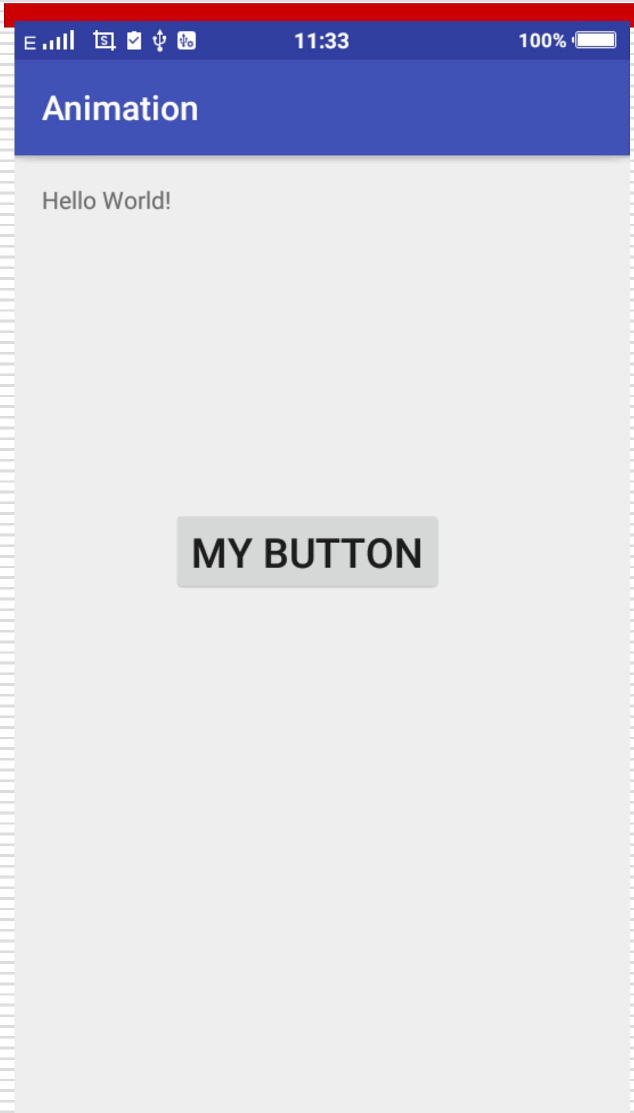
My_animation.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
<translate
        android:fromXDelta="0%"
        android:toXDelta="80%"
        android:duration="2000"
        android:repeatMode="reverse"
        android:repeatCount="infinite"    />
</set>
```

MainActivity.java

```
package com.example.umadevi.animation;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {
    Button btn1;
    Animation anim1;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        btn1=(Button)findViewById(R.id.button);
        anim1= AnimationUtils.loadAnimation(this,R.anim.my_animation);
        btn1.startAnimation(anim1);     } }
```

Rotate animation



Rotate animation

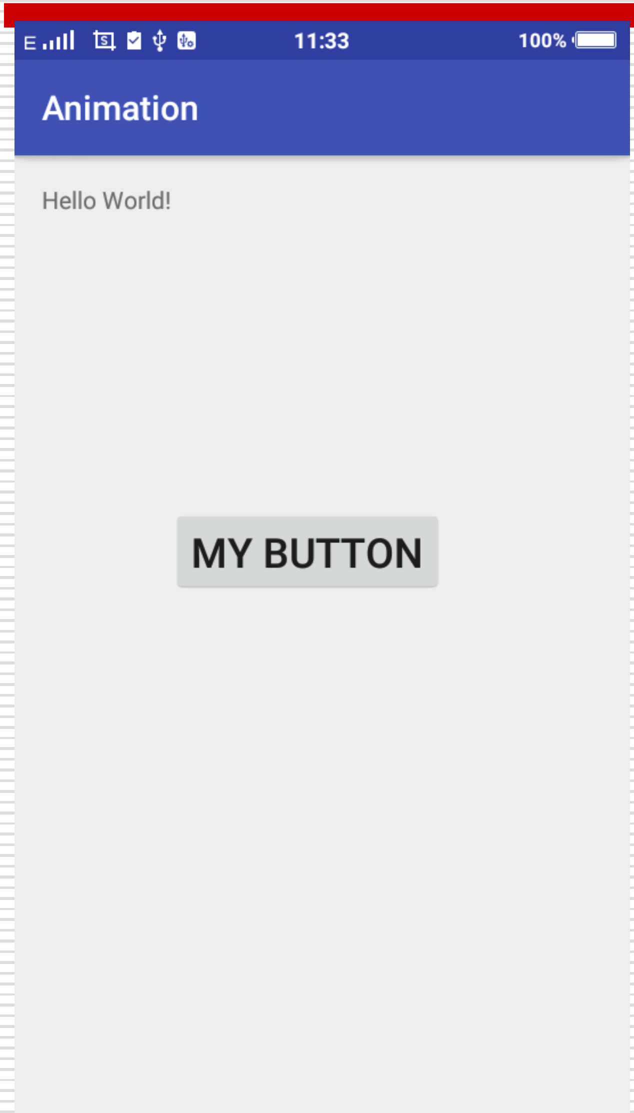
My_animation.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <rotate
        android:fromDegrees="0"
        android:toDegrees="360"
        android:pivotX="50%"
        android:pivotY="50%"
        android:duration="1000"
        android:repeatCount="infinite"
        android:repeatMode="restart“  />
</set>
```

MainActivity.java

```
package com.example.umadevi.animation;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {
    Button btn1;
    Animation anim1;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        btn1=(Button)findViewById(R.id.button);
        anim1= AnimationUtils.loadAnimation(this,R.anim.my_animation);
        btn1.startAnimation(anim1);     } }
```

Scaling animation



Scaling animation

My_animation.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
<translate
        android:fromXDelta="0%"
        android:toXDelta="80%"
        android:duration="2000"
        android:repeatMode="reverse"
        android:repeatCount="infinite"    />
</set>
```

MainActivity.java

```
package com.example.umadevi.animation;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {
    Button btn1;
    Animation anim1;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        btn1=(Button)findViewById(R.id.button);
        anim1= AnimationUtils.loadAnimation(this,R.anim.my_animation);
        btn1.startAnimation(anim1);     } }
```

Thanks for Listening

Weekend Assignment

This is will ensure check whether you made Android Studio setup ready on your personal laptops

To Do

- □ Using Android Studio, Create a User Interface which has one TextView (to display greeting message), EditText (to accept <friends name>) and Button element.

Write java code inside MainActivity.java which does the following

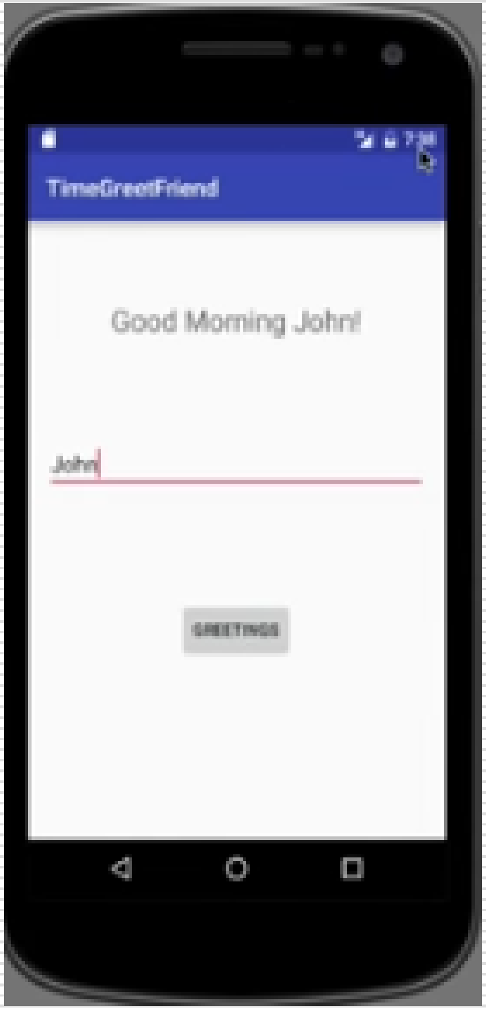
When user clicks on the button to show the greeting based on the time of the day as listed below:

- □ Between 6 am and 12 noon it should show "Good Morning <friend's name>!"
- □ Between 12 noon and 5 pm it should show "Good Afternoon <friend's name>!"
- □ Between 5 pm and 9 pm it should show "Good Evening <friend's name>!"
- □ Between 9 pm and 6 am it should show "Good Night <friend's name>!"

Hint: The following code snippet will give you the hour of the day. You can use it to implement your solution:

```
 Date date = new Date();
 Calendar cal = Calendar.getInstance();
 cal.setTime(date);   int hour = cal.get(Calendar.HOUR_OF_DAY);
```

Weekend Assignment: Screenshot



Good Morning John!

John

Resource Types

Resource Type	Description	Saved In	Accessed from
Animation Resources	Define pre-determined animations.	Tween animations are saved in res/anim/ and Frame animations are saved in res/drawable/.	Accessed from the R.anim class Accessed from the R.drawable class
Color State List Resource	Define a color resources that changes based on the View state.	res/color/	R.color
Drawable Resources	Define various graphics with bitmaps or XML.	res/drawable/	R.drawable class
Layout Resource	Define the layout for your application UI.	res/layout/	R.layout
Menu Resource	Define the contents of your application menus.	res/menu/	R.menu
String Resources	Define strings, string arrays, and plurals (and include string formatting and styling).	res/values/	R.string, R.array, and R.plurals classes
Style Resource	Define the look and format for UI elements.	res/values/	R.style

Accessing Resources

When Android application is compiled, aapt generates the R class, which contains resource IDs for all the resources in the res/ directory. For each type of resource, there is an R subclass (for example, R.drawable for all drawable resources), and for each resource of that type, there is a static integer (for example, R.drawable.icon).

A resource ID is always composed of:

- The *resource type*: Each resource is grouped into a "type," such as string, drawable, and layout.
- The *resource name*, which is either: the filename, excluding the extension; or the value in the XML android:name attribute, if the resource is a simple value (such as a string).

There are two ways you can access a resource:

- **In code:** Using a static integer from a sub-class of your R class, such as: R.string.hello string is the resource type and hello is the resource name.
- **In XML:** Using a special XML syntax that also corresponds to the resource ID defined in your R class, such as:@string/hello string is the resource type and hello is the resource name.

Accessing String Resources: Example

/res/values/**srings.xml** file

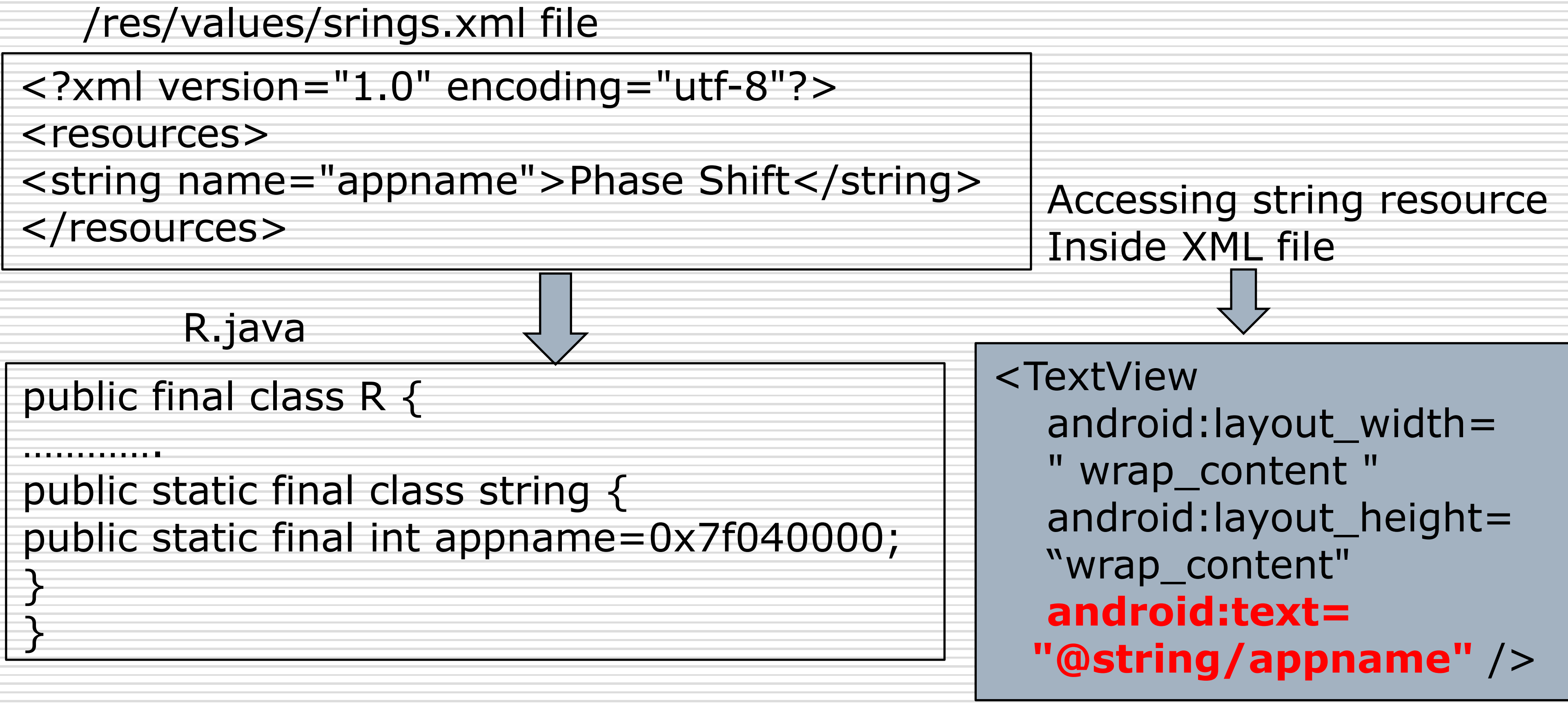
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="appname">Phase Shift</string>
</resources>
```

Accessing string resource
Inside XML file

R.java

```
public final class R {
………….
public static final class string {
public static final int appname=0x7f040000;
}
}
```

Accessing String Resources: Example



/res/values/srings.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="appname">Phase Shift</string>
</resources>
```

R.java

```
public final class R {
………….
public static final class string {
public static final int appname=0x7f040000;
}
}
```

Accessing string resource
Inside XML file

```
<TextView
   android:layout_width=
   " wrap_content "
   android:layout_height=
   “wrap_content"
   android:text=
  "@string/appname" />
```

Accessing String Resources: Example

/res/values/**srings.xml** file

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="appname">Phase Shift</string>
</resources>
```

Accessing string resource
Inside **java** file

R.java

```
public final class R {
…………
public static final class string {
public static final
int appname=0x7f040000;
}
}
```

Accessing String Resources: Example

/res/values/**srings.xml** file

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name=“appname">Phase Shift</string>
</resources>
```

R.java

```
public final class R {
………….
public static final class string {
public static final
int appname=0x7f040000;
}
}
```

Accessing string resource
Inside **java** file

```
String string =
getString(R.string.appname);

OR
String string =
(String)getResources().getText(R.string.app
name);
OR
TextView msgTextView = (TextView)
findViewById(R.id.msg);
msgTextView.setText(R.string.appname);
```

String Resources

There are three types of resources that can provide your application with strings:

- **String** XML resource that provides a single string.
- **String Array** XML resource that provides an array of strings.
- **Quantity Strings (Plurals)** XML resource that carries different strings for pluralization.

String Arrays

- XML file saved at **res/values/strings.xml**:

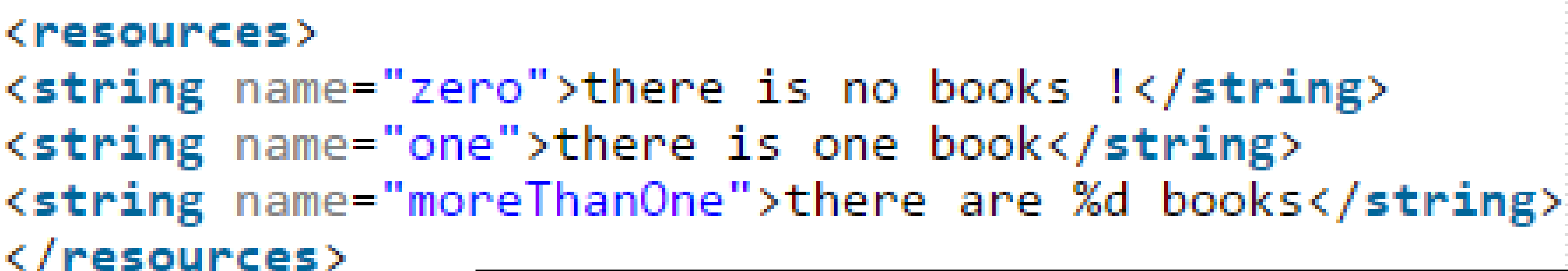
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="planets_array">
        <item>Mercury</item>
        <item>Venus</item>
        <item>Earth</item>
        <item>Mars</item>
    </string-array>
</resources>
```

This application code retrieves a string array:

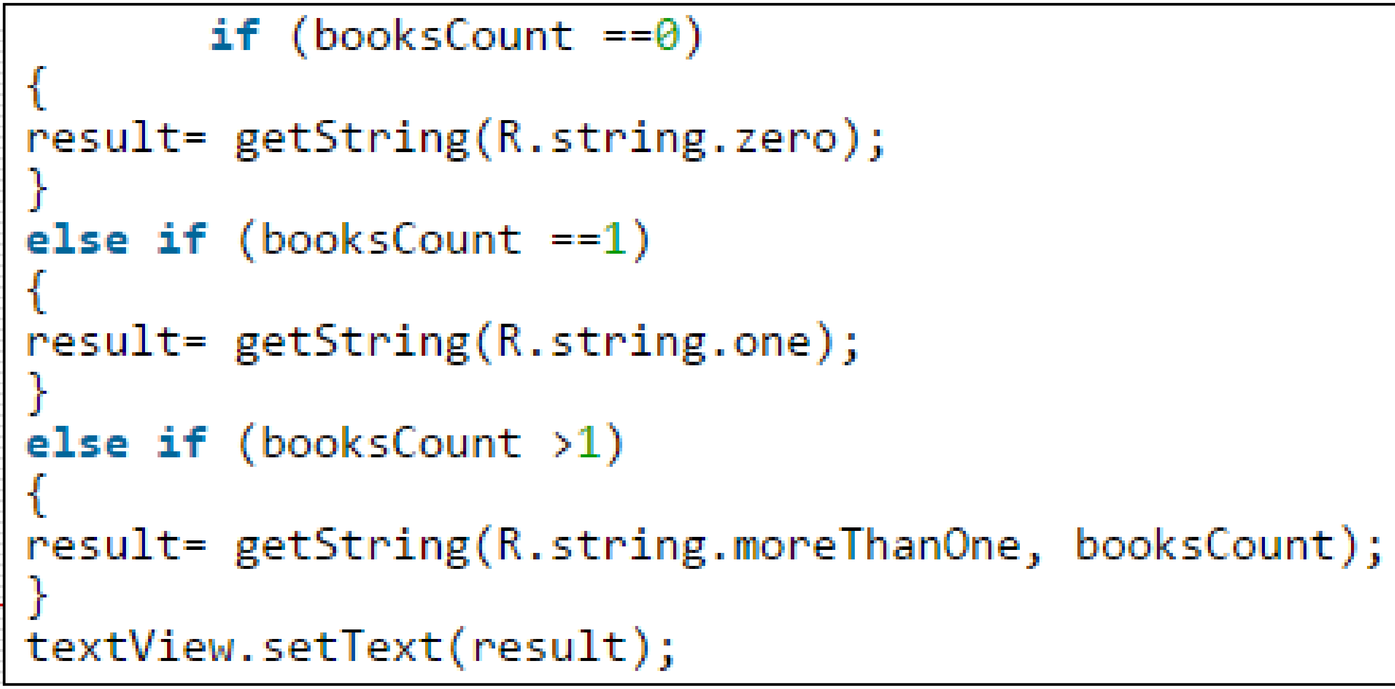
```
Resources res = getResources();
String[] planets =
res.getStringArray(R.array.planets_array);
```

Quantity Strings (Plurals)

□ Plurals are a type of resource, we use it very often to deal with plurality of Strings, what does plurality of Strings means ? Well suppose you are counting books and you want to print out the result, you will have to write at least two Strings like:

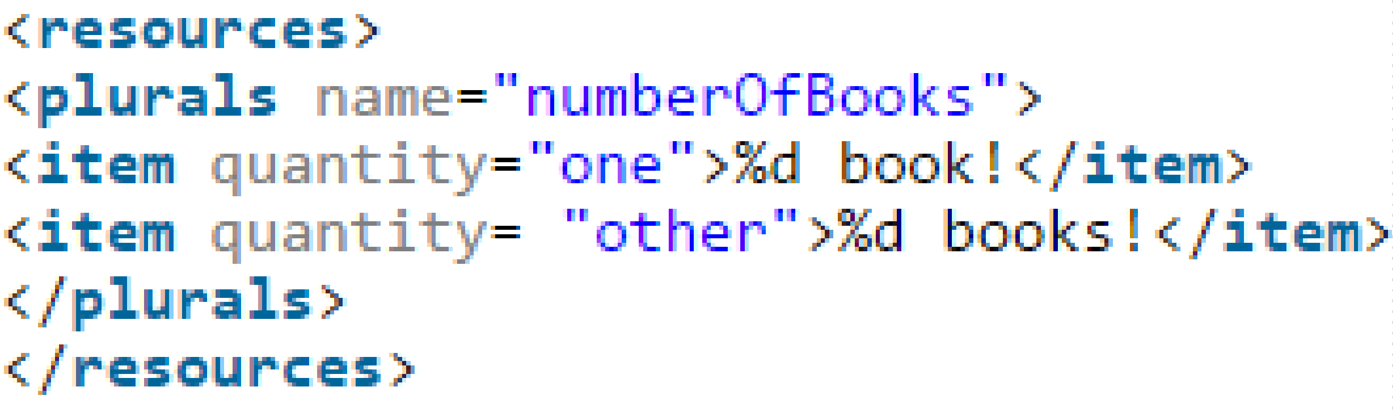


```
<resources>
<string name="zero">there is no books !</string>
<string name="one">there is one book</string>
<string name="moreThanOne">there are %d books</string>
</resources>
```



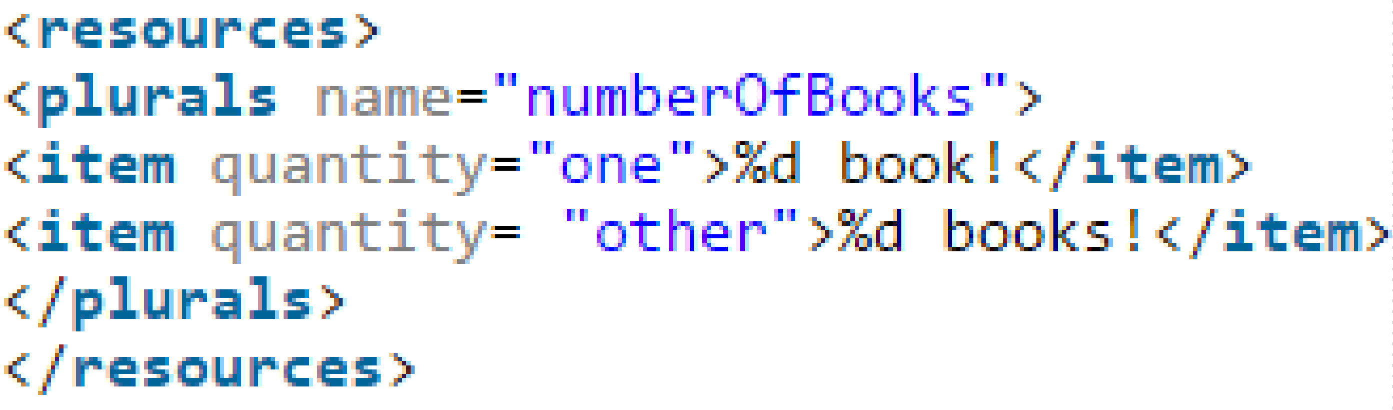
```
      if (booksCount ==0)
{
result= getString(R.string.zero);
}
else if (booksCount ==1)
{
result= getString(R.string.one);
}
else if (booksCount >1)
{
result= getString(R.string.moreThanOne, booksCount);
}
textView.setText(result);
```

Quantity Strings (Plurals)



```
<resources>
<plurals name="numberOfBooks">
<item quantity="one">%d book!</item>
<item quantity= "other">%d books!</item>
</plurals>
</resources>
```

Quantity Strings (Plurals)



```
<resources>
<plurals name="numberOfBooks">
<item quantity="one">%d book!</item>
<item quantity= "other">%d books!</item>
</plurals>
</resources>
```

```
int booksCount= 20;
String result =
getResources().getQuantityString(R.plurals.numberOfBooks,
booksCount,booksCount);

textView.setText(result);
```

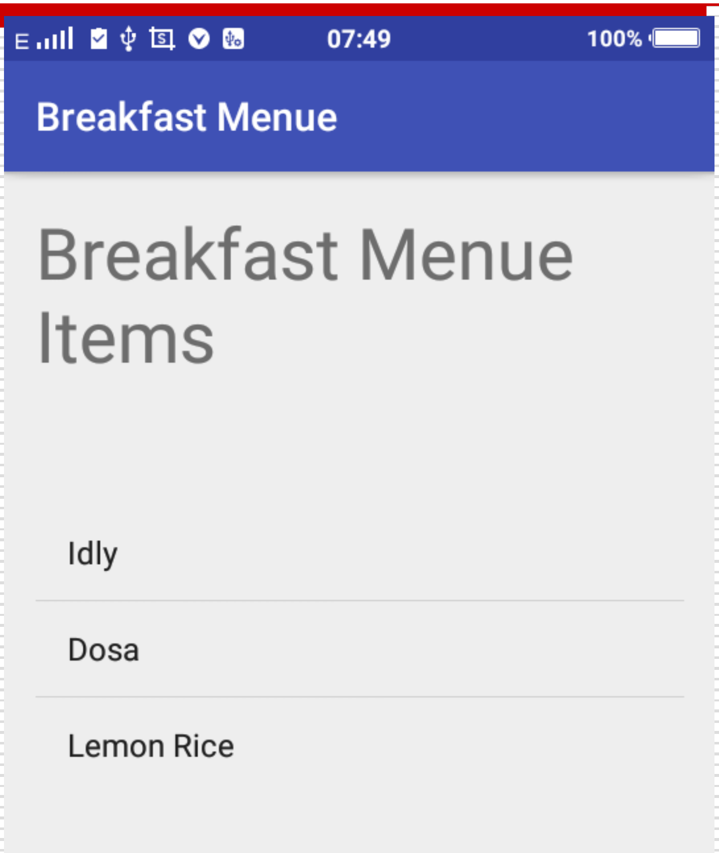
Note: When using the getQuantityString() method, you need to pass the count twice if your string includes string formatting with a number. For example, for the string %d songs found, the first count parameter selects the appropriate plural string and the second count parameter is inserted into the %d placeholder. If your plural strings do not include string formatting, you don't need to pass the third parameter to getQuantityString

Quantity Strings (Plurals)

Different languages have different rules for grammatical agreement with quantity. In English, for example, the quantity 1 is a special case. We write "1 book", but for any other quantity we'd write "*n* books". This distinction between singular and plural is very common, but other languages make finer distinctions. The full set supported by Android is zero, one, two, few, many, and other.

Value	Description
zero	When the language requires special treatment of the number 0 (as in Arabic).
one	When the language requires special treatment of numbers like one (as with the number 1 in English and most other languages; in Russian, any number ending in 1 but not ending in 11 is in this class).
two	When the language requires special treatment of numbers like two (as with 2 in Welsh, or 102 in Slovenian).
few	When the language requires special treatment of "small" numbers (as with 2, 3, and 4 in Czech; or numbers ending 2, 3, or 4 but not 12, 13, or 14 in Polish).
many	When the language requires special treatment of "large" numbers (as with numbers ending 11-99 in Maltese).
other	When the language does not require special treatment of the given quantity (as with all numbers in Chinese, or 42 in English).

ListView + String Arrays



Breakfast Menue

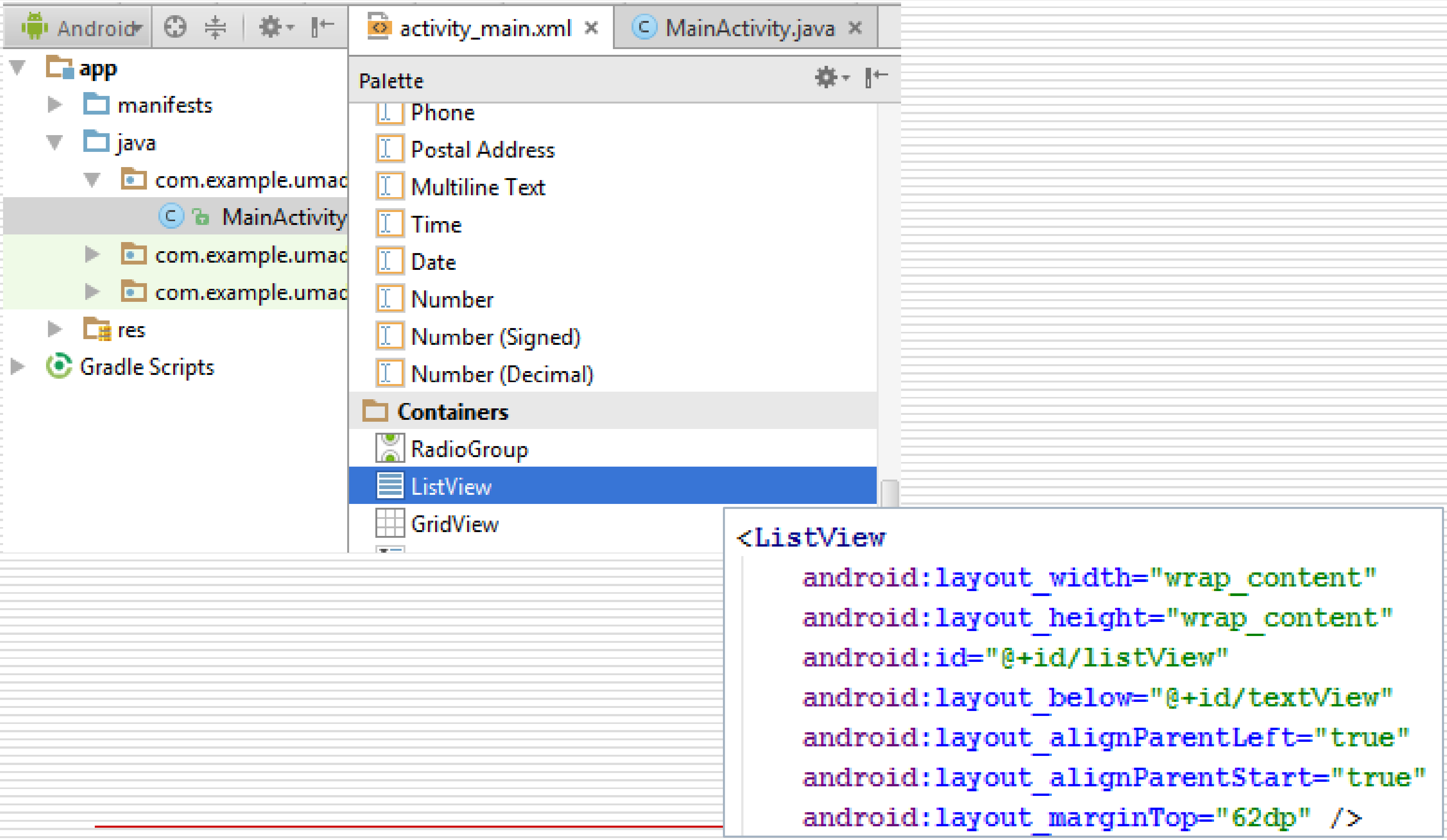
Breakfast Menue Items

Idly

Dosa

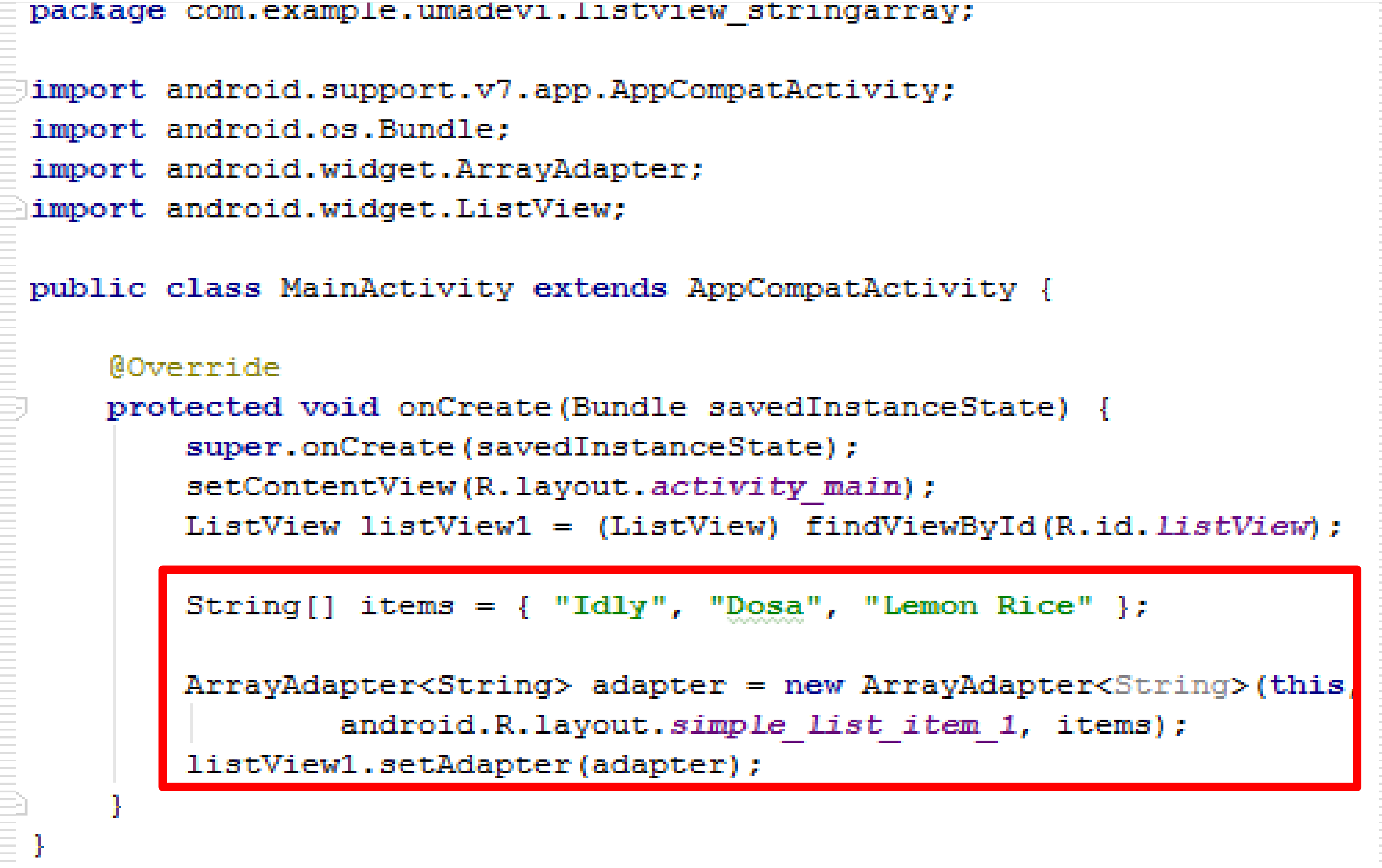
Lemon Rice

Create Listview



```
<ListView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/listView"
    android:layout_below="@+id/textView"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_marginTop="62dp" />
```

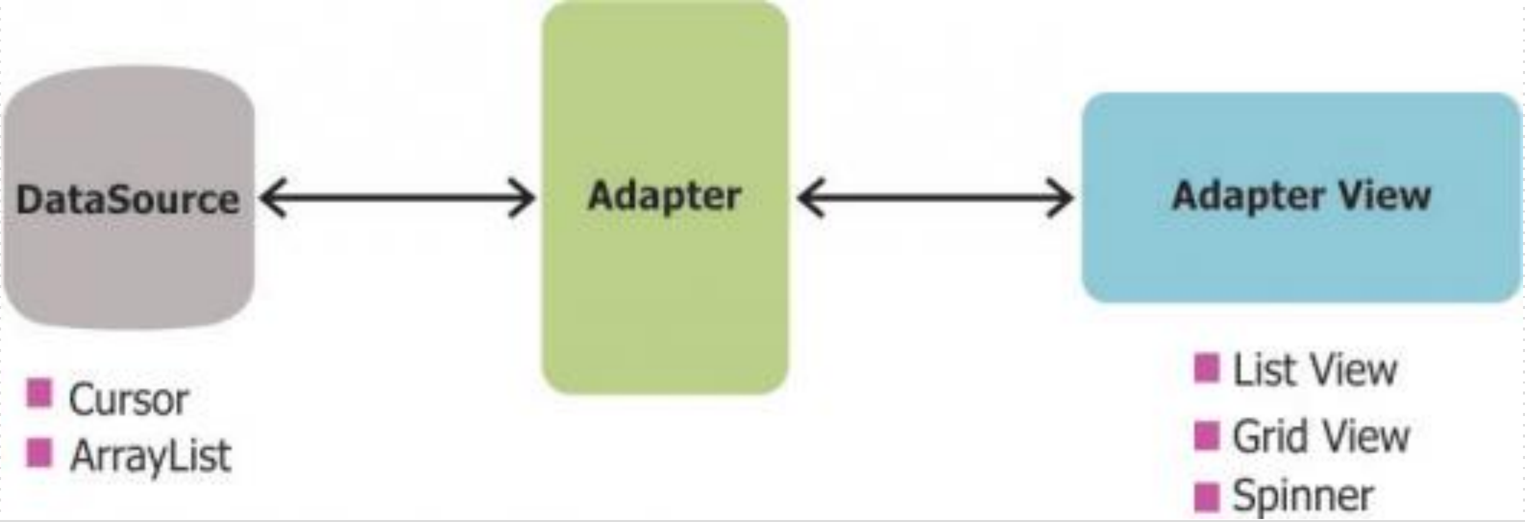
MainActivity.java



```
package com.example.umadevi.listview_stringarray;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.ListView;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ListView listView1 = (ListView) findViewById(R.id.listView);

        String[] items = { "Idly", "Dosa", "Lemon Rice" };

        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
                android.R.layout.simple_list_item_1, items);
        listView1.setAdapter(adapter);
    }
}
```

Android Array Adapters

- Adapters in Android are a bridge between the Adapter View (e.g. ListView) and the underlying data



To Do

To Do

□ Open your Mobile Phone

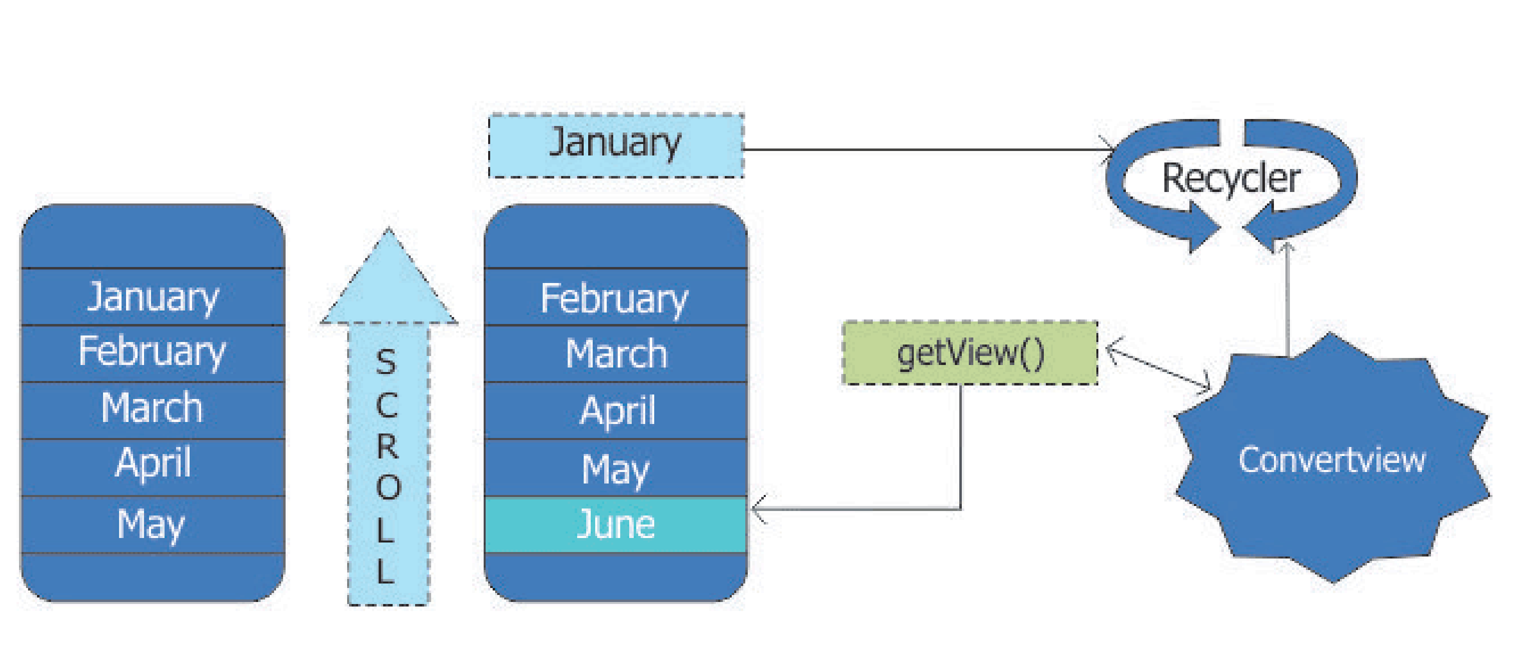
□ Click on Contacts app

□ Scroll up and down the contacts

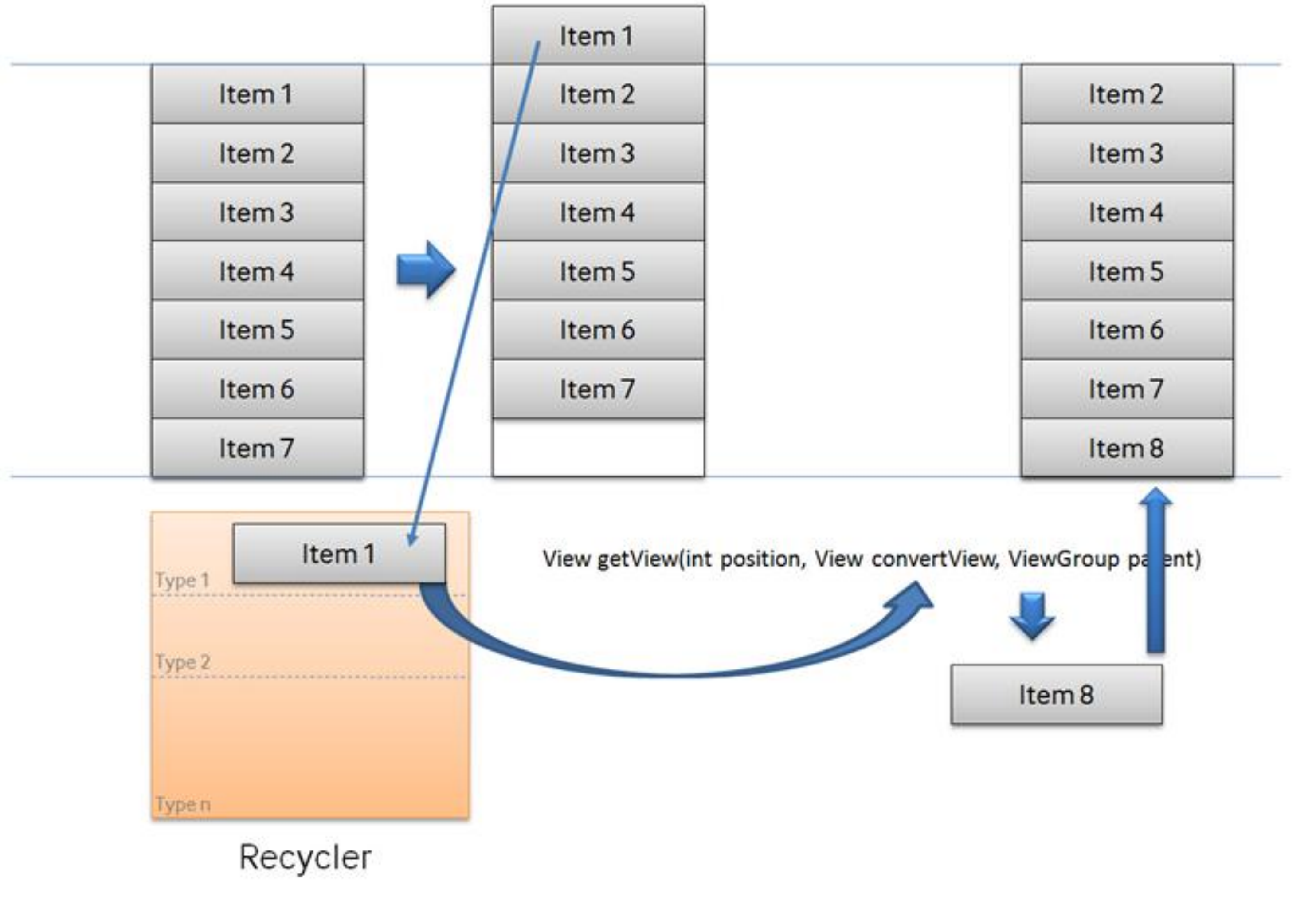
□ Observe, at a given time how contacts were **displayed** on your phone screen

ListView + ArrayAdapters

- □ Adapters call the **getView()** method which returns a view for each item within the adapter view.



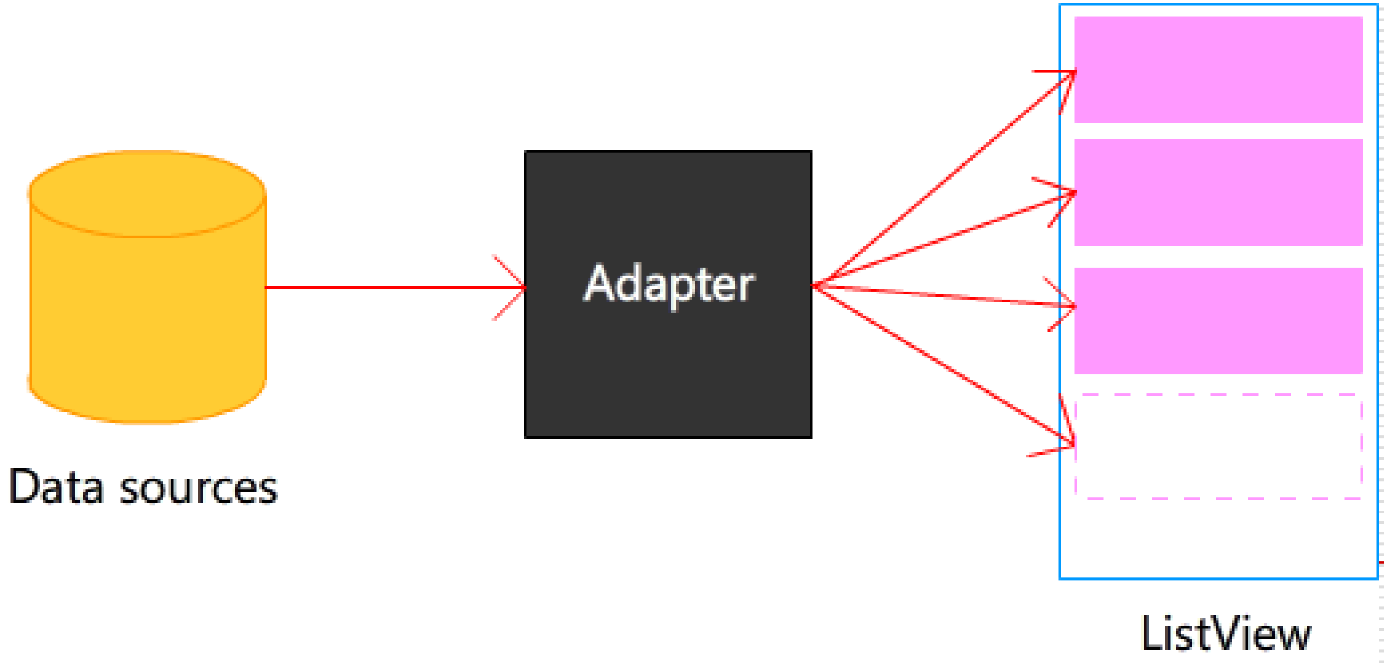
ListView + ArrayAdapters: Row View Recycling



ListView + Adapters

Define ArrayAdapter

- ☐ ArrayAdapter Assumes that one row of data is represented bt TextView
- ☐ ArrayAdapter<String> adapter = new ArrayAdapter (Context C, int layoutID, String[])
- ☐ setListAdapter(adapter)



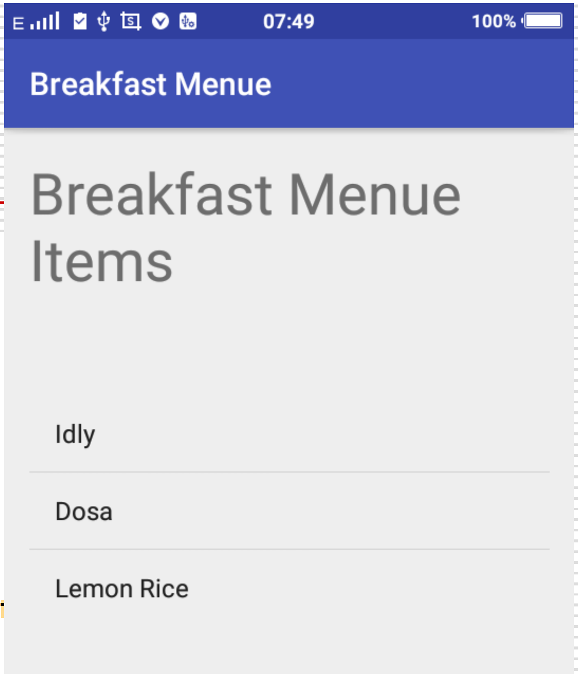
Array Adapter

- ArrayAdapter (Context context,
  int resource, T[] objects)

Parameters	
context	Context: The current context.
resource	int: The resource ID for a layout file containing a TextView to use when instantiating views.
objects	T: The objects to represent in the ListView

MainActivity.java

```
package com.example.umadevi.listview_stringarray;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.ListView;

public class MainActivity extends AppCompatActivi

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ListView listView1 = (ListView) findViewById(R.id.listView);

        String[] items = { "Idly", "Dosa", "Lemon Rice" };

        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
                android.R.layout.simple_list_item_1, items);
        listView1.setAdapter(adapter);
    }
}
```



Breakfast Menue

Breakfast Menue Items

Idly

Dosa

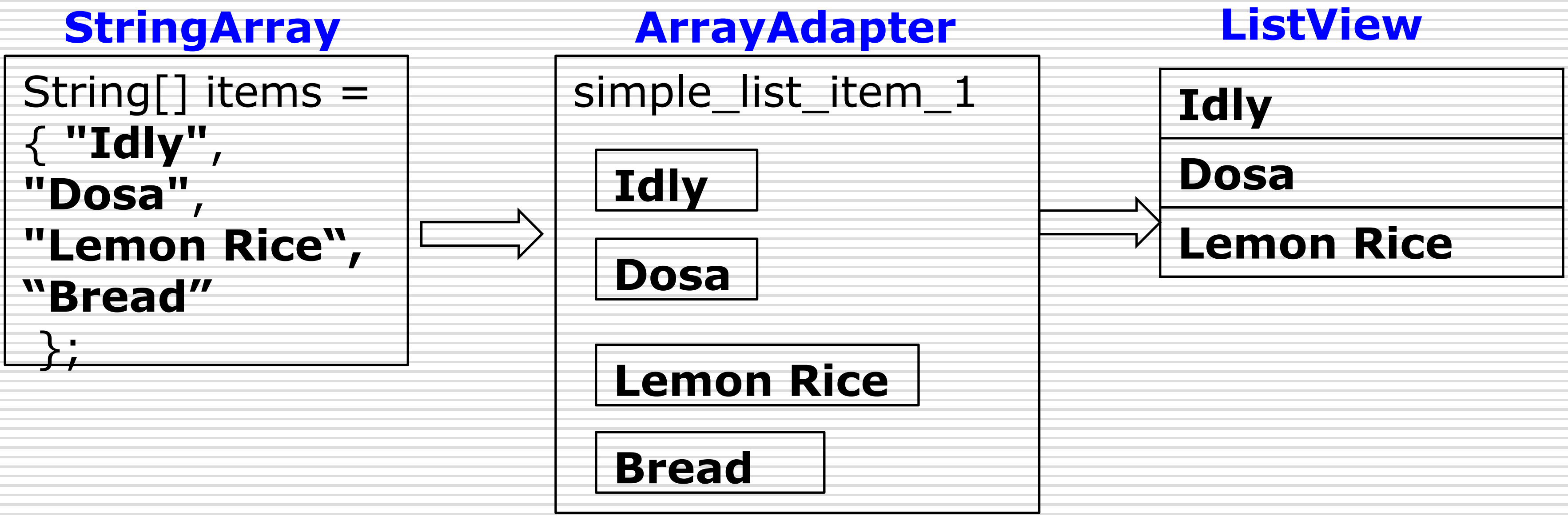
Lemon Rice

simple_list_item_1

- android.R.layout.simple_list_item_1: reference to an built-in XML layout document that is part of the Android OS, rather than one of your own XML layouts.
- <TextView xmlns:android="http://schemas.android.com/apk/res/android" android:id="@android:id/text1" android:layout_width="match_parent" android:layout_height="wrap_content" android:textAppearance="?android:attr/textAppearanceLarge" android:gravity="center_vertical" android:paddingLeft="6dip" android:minHeight="?android:attr/listPreferredItemHeight" />
- The android.R.layout.simple_list_item_1 is an inbuilt layout resource and it displays a single string. If you want to use your own layout file then you can use
- setListAdapter(new ArrayAdapter<String>(this, R.layout.<your layout filename>, titles));

StringArray+ArrayAdapter+ListView

```
ListView listView1 = (ListView) findViewById(R.id.listView);

String[] items = { "Idly", "Dosa", "Lemon Rice“, “Bread” };

ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
                                      android.R.layout.simple_list_item_1, items);
listView1.setAdapter(adapter);
```



StringArray

String[] items =
{ **"Idly"**,
"Dosa",
"Lemon Rice“,
“Bread”
};

ArrayAdapter

simple_list_item_1

Idly

Dosa

Lemon Rice

Bread

ListView

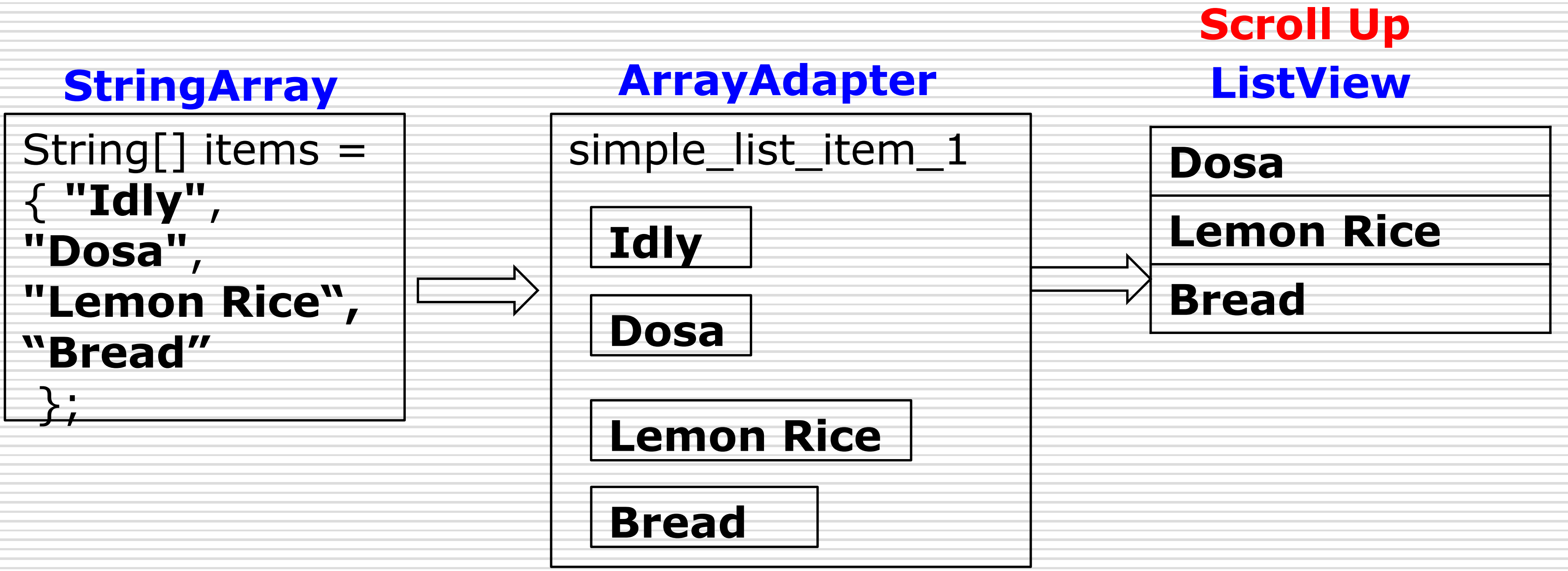
Idly

Dosa

Lemon Rice

StringArray+ArrayAdapter+ListView

```
ListView listView1 = (ListView) findViewById(R.id.listView);

String[] items = { "Idly", "Dosa", "Lemon Rice“, “Bread” };

ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
                              android.R.layout.simple_list_item_1, items);
listView1.setAdapter(adapter);
```



Scroll Up

StringArray

String[] items =
{ **"Idly"**,
"Dosa",
"Lemon Rice“,
“Bread”
};

ArrayAdapter

simple_list_item_1

Idly

Dosa

Lemon Rice

Bread

ListView

Dosa

Lemon Rice

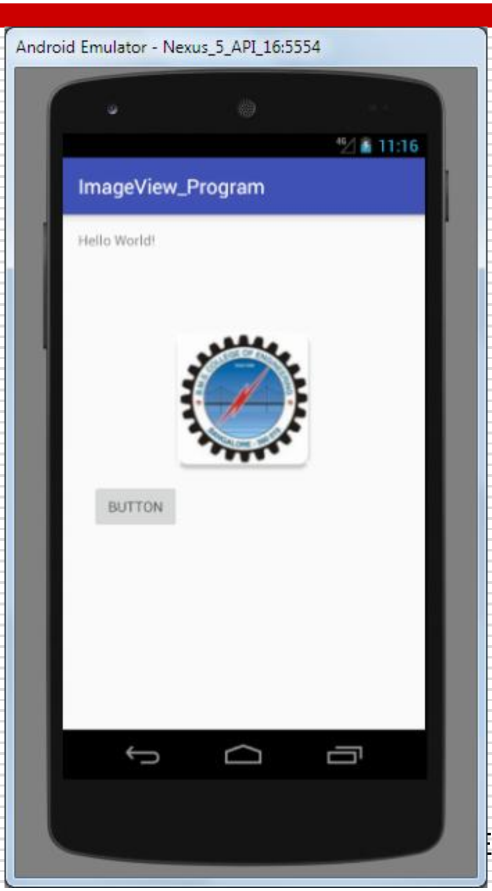
Bread

Watch this Video

- Link

https://www.youtube.com/watch?v=2lcoB5-PCCw

Image Resources



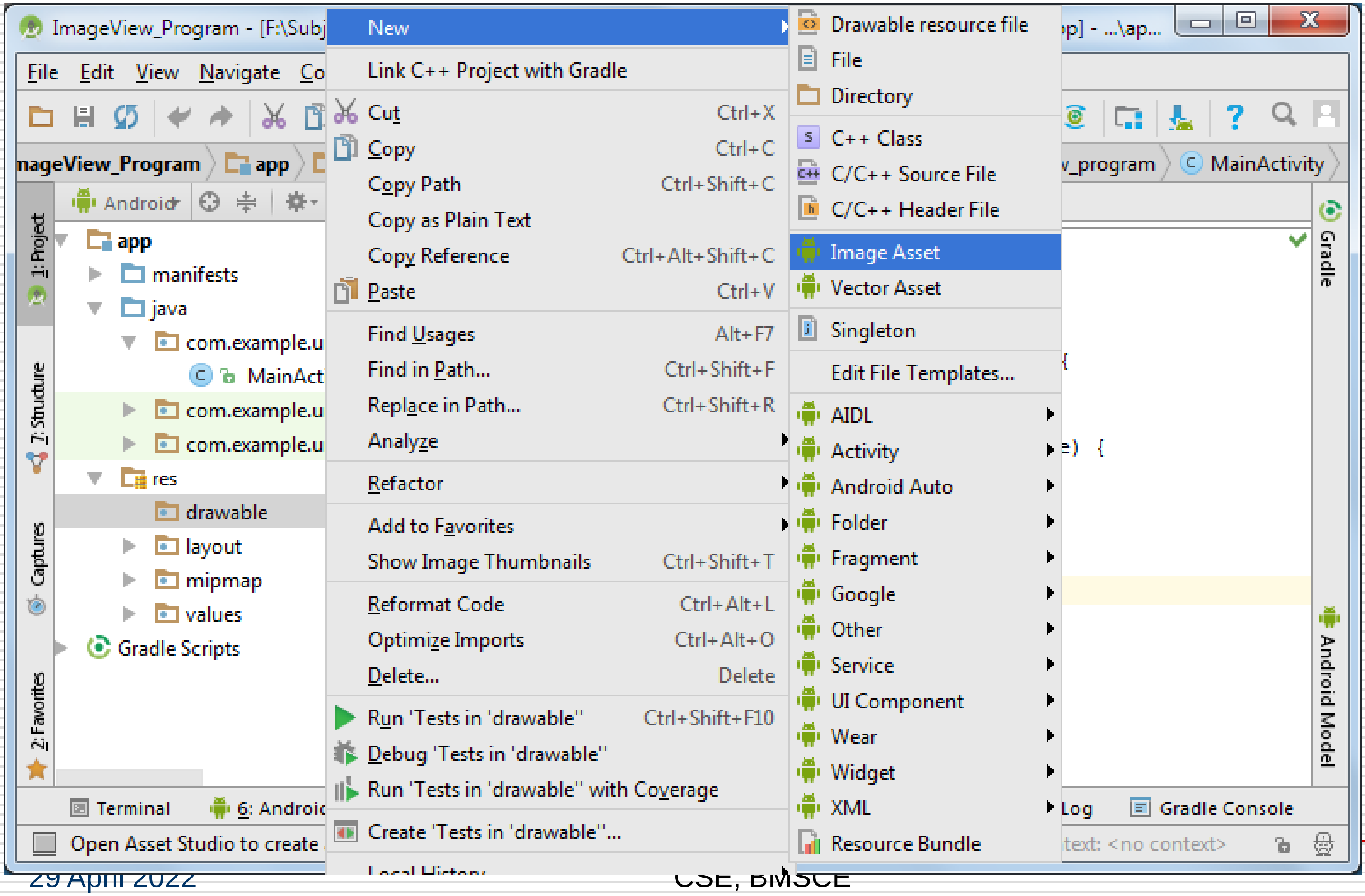
Android Emulator - Nexus_5_API_16:5554

ImageView_Program

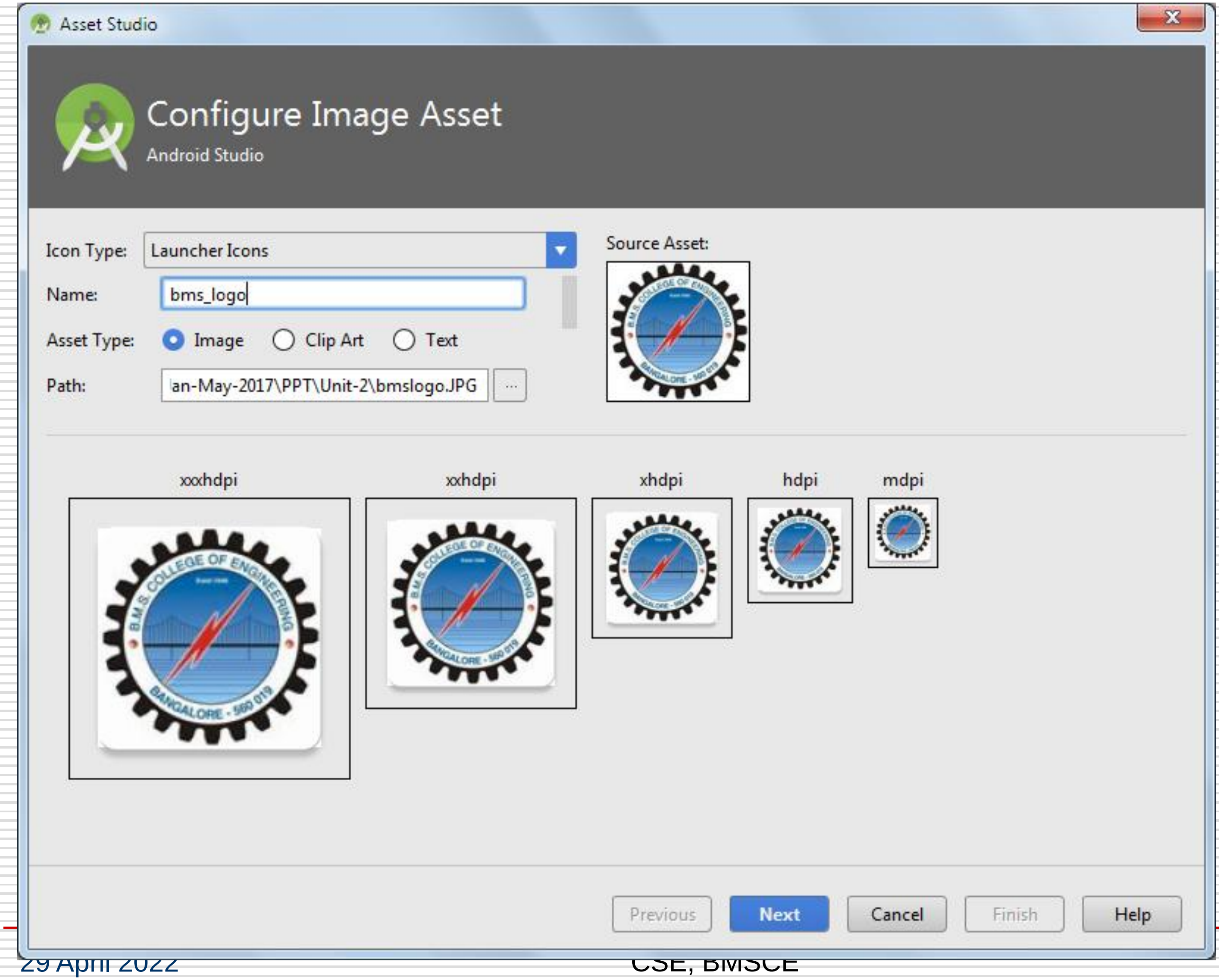
Hello World!

Add Image to Resources

- Put your images into folder "res/drawable-ldpi", "res/drawable-mdpi" or "res/drawable-hdpi".



Add Image to Resources



Asset Studio

Configure Image Asset

Android Studio

Icon Type: Launcher Icons

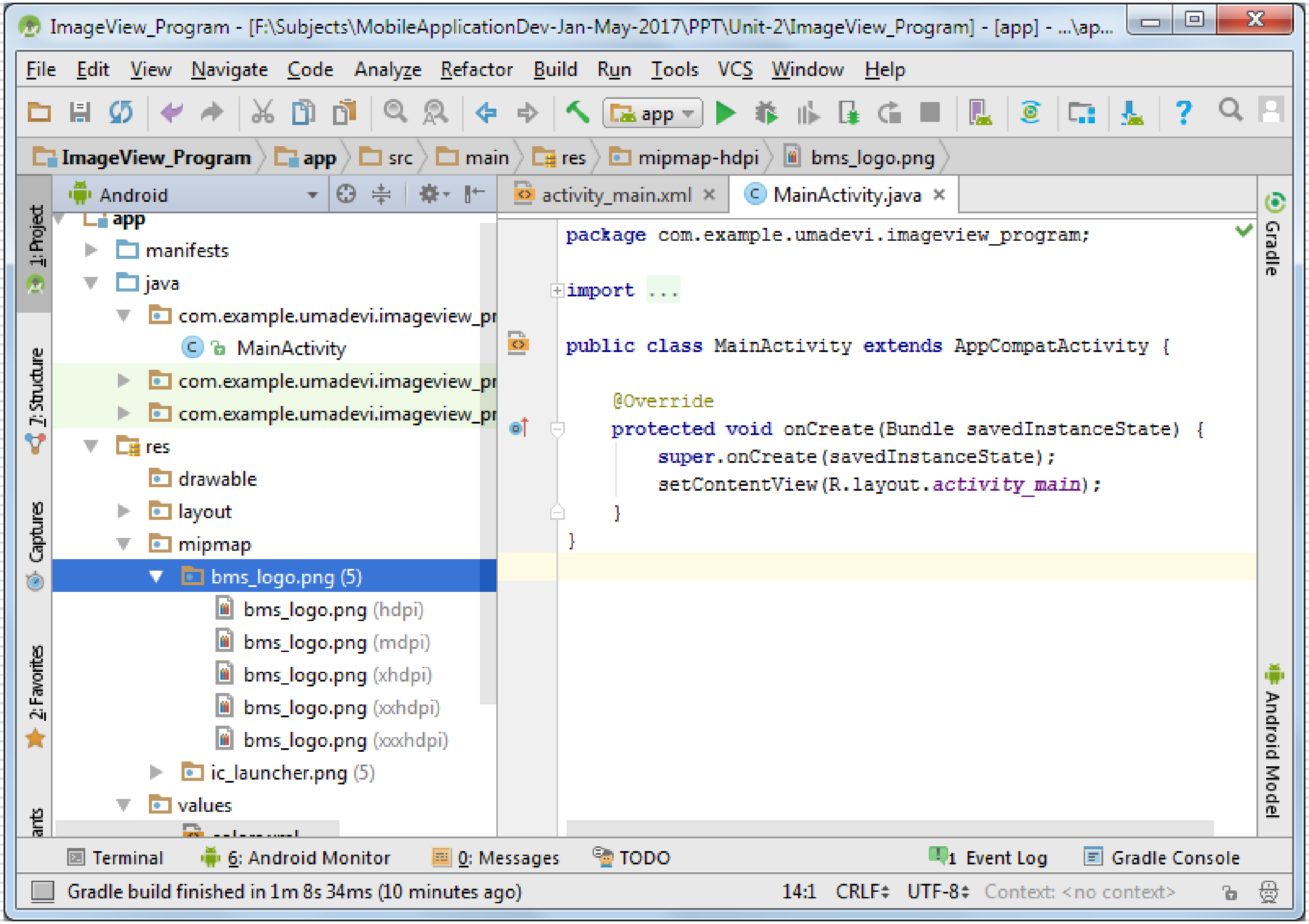
Name: bms_logo

Asset Type: ● Image ○ Clip Art ○ Text

Path: an-May-2017\PPT\Unit-2\bmslogo.JPG

Source Asset:

Add Image to Resources



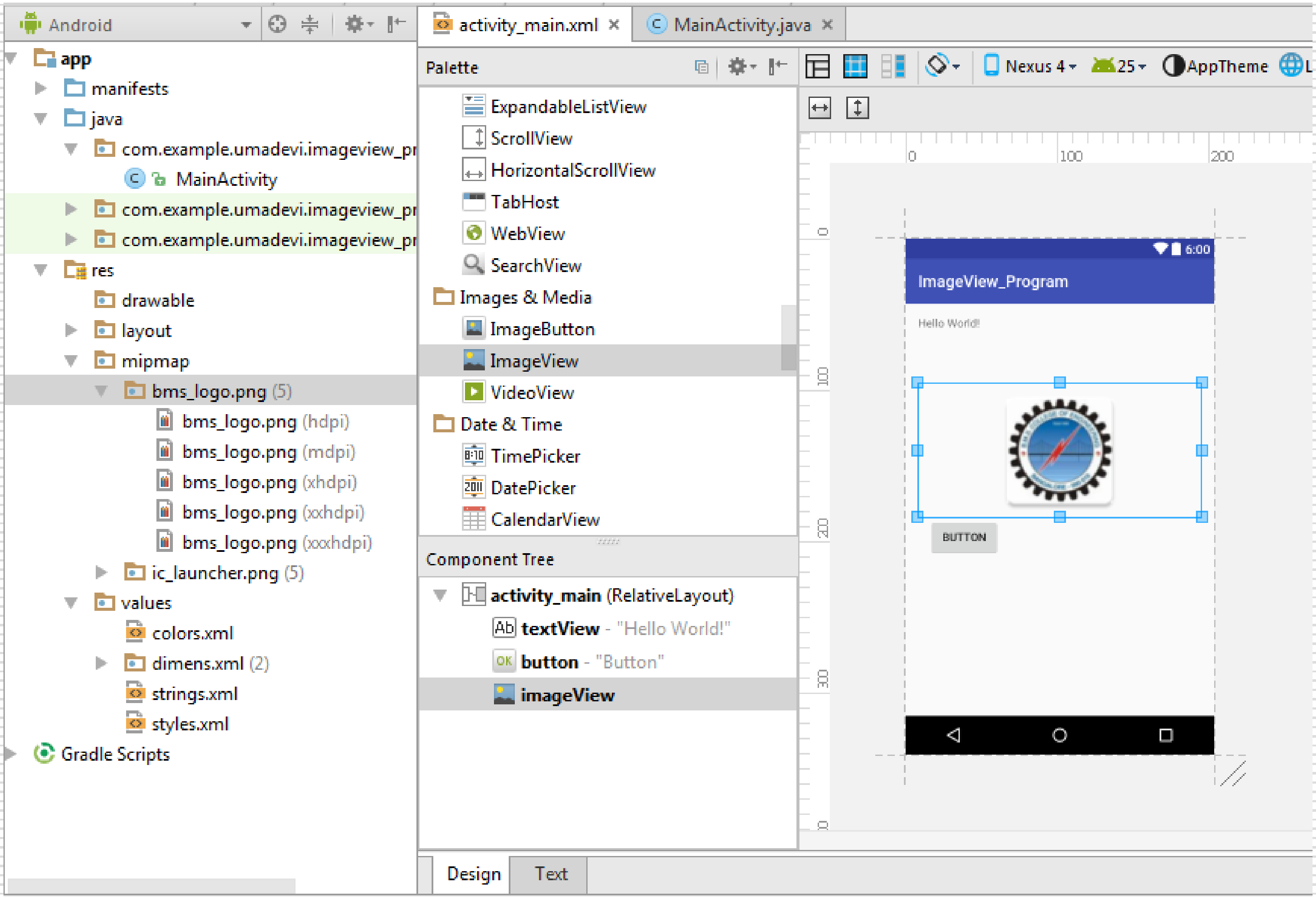
ImageView_Program - [F:\Subjects\MobileApplicationDev-Jan-May-2017\PPT\Unit-2\ImageView_Program] - [app] - ...\ap...

```
app
  manifests
  java
    com.example.umadevi.imageview_pr
      MainActivity
    com.example.umadevi.imageview_pr
    com.example.umadevi.imageview_pr
  res
    drawable
    layout
    mipmap
      bms_logo.png (5)
        bms_logo.png (hdpi)
        bms_logo.png (mdpi)
        bms_logo.png (xhdpi)
        bms_logo.png (xxhdpi)
        bms_logo.png (xxxhdpi)
      ic_launcher.png (5)
    values
```

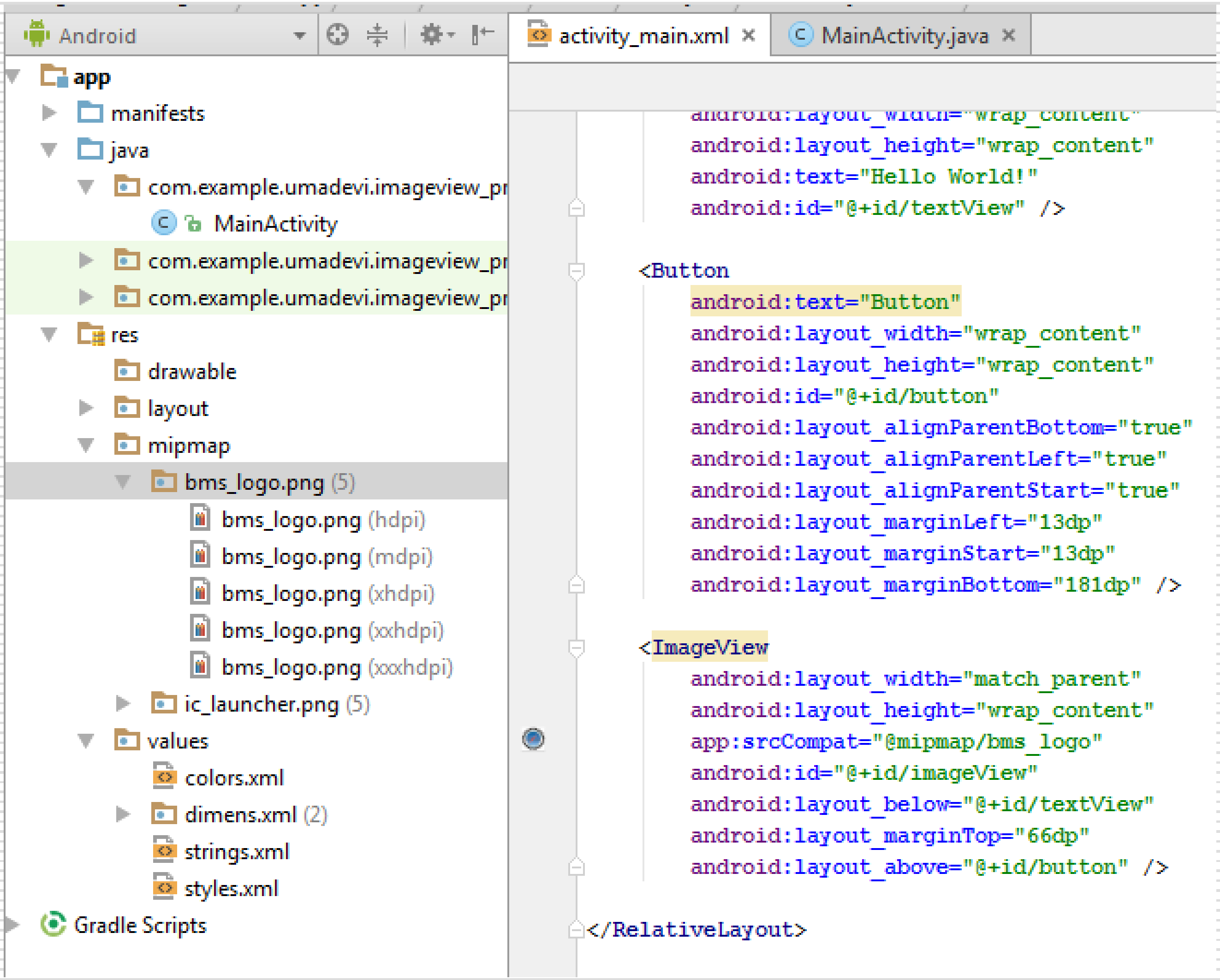
```java
package com.example.umadevi.imageview_program;

import ...

public class MainActivity extends AppCompatActivity {

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

Gradle build finished in 1m 8s 34ms (10 minutes ago)   14:1   CRLF   UTF-8   Context: <no context>

Add ImageView



Add ImageView



```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        android:id="@+id/textView" />

    <Button
        android:text="Button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/button"
        android:layout_alignParentBottom="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_marginLeft="13dp"
        android:layout_marginStart="13dp"
        android:layout_marginBottom="181dp" />

    <ImageView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:srcCompat="@mipmap/bms_logo"
        android:id="@+id/imageView"
        android:layout_below="@+id/textView"
        android:layout_marginTop="66dp"
        android:layout_above="@+id/button" />

</RelativeLayout>
```

MainActivity.java

- When button is clicked, change it to “bms_logo.png”.

```
public class MyAndroidAppActivity extends Activity { Button button; ImageView
image;
@Override public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
addListenerOnButton();
}
 public void addListenerOnButton() {
image = (ImageView) findViewById(R.id.imageView1);
button = (Button) findViewById(R.id.btnChangeImage);
button.setOnClickListener(new OnClickListener() {
@Override
public void onClick(View arg0) {
//image.setImageResource(R.drawable.bms_logo);
image.setImageResource(R.mipmap.bms_logo);
}
 });
 } }
```

Unit 2: Android Application Life Cycle

Android aggressively manages its resources, doing whatever’s necessary to ensure a smooth and

stable user experience.

In practice that means that processes (and their hosted applications) will be killed, in some case without warning, to free resources for higher-priority applications.

Unit 2: UNDERSTANDING AN APPLICATION’S PRIORITY AND ITS PROCESS’ STATES

The order in which processes are killed to reclaim resources is determined by the priority of their hosted applications.

Application Priority tree /Application states

- □ A process on Android can be in one of five different states at any given time.

Who lives and who dies? Process priorities on Android

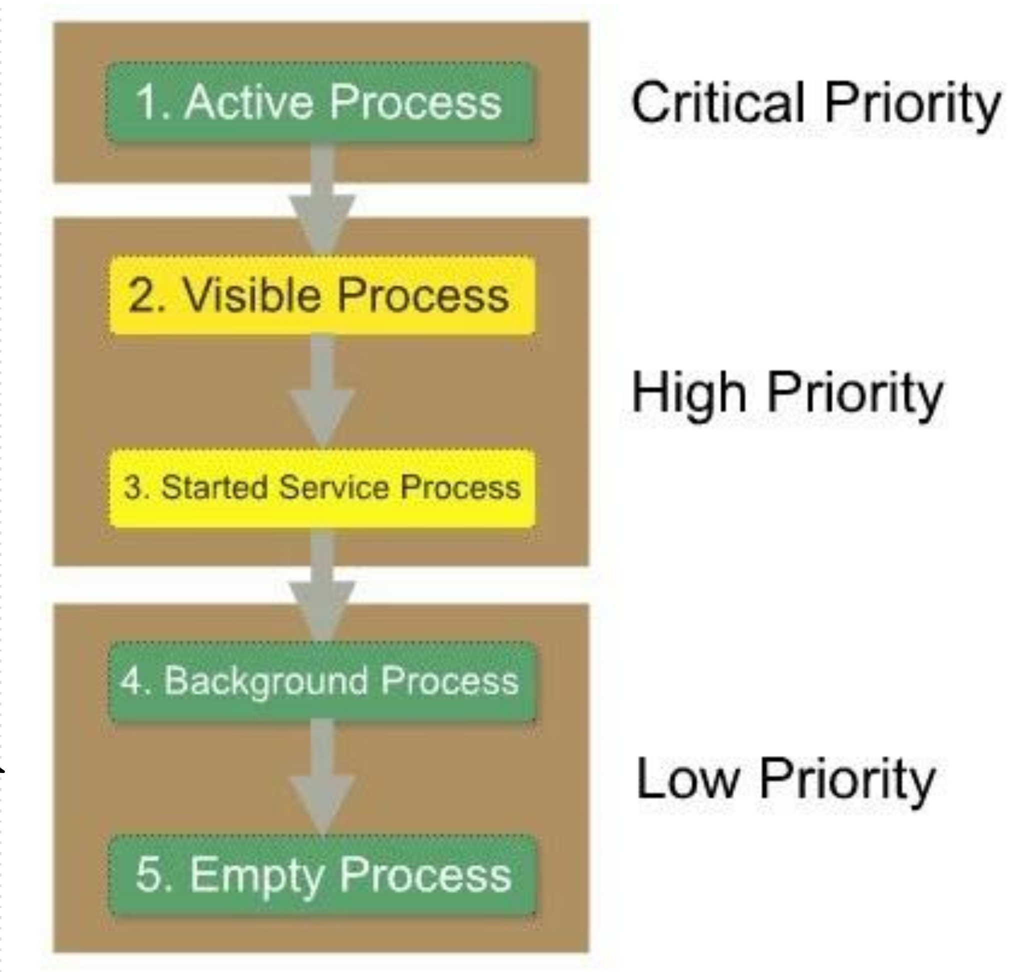


Figure :Priority tree used to determine the order of application termination.

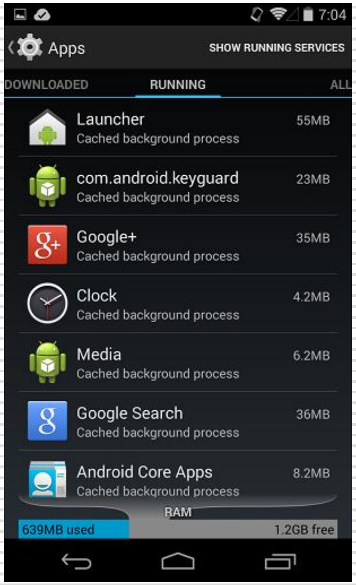
To Do

□ Open your phone Go to "Setting" -> "App" -> "Running‟

□ Check how applications are running

□ Click and open all the apps in your phone

□ Now Go to "Setting" -> "App" -> "Running‟



Launcher 55MB
Cached background process

com.android.keyguard 23MB
Cached background process

Google+ 35MB
Cached background process

Clock 4.2MB
Cached background process

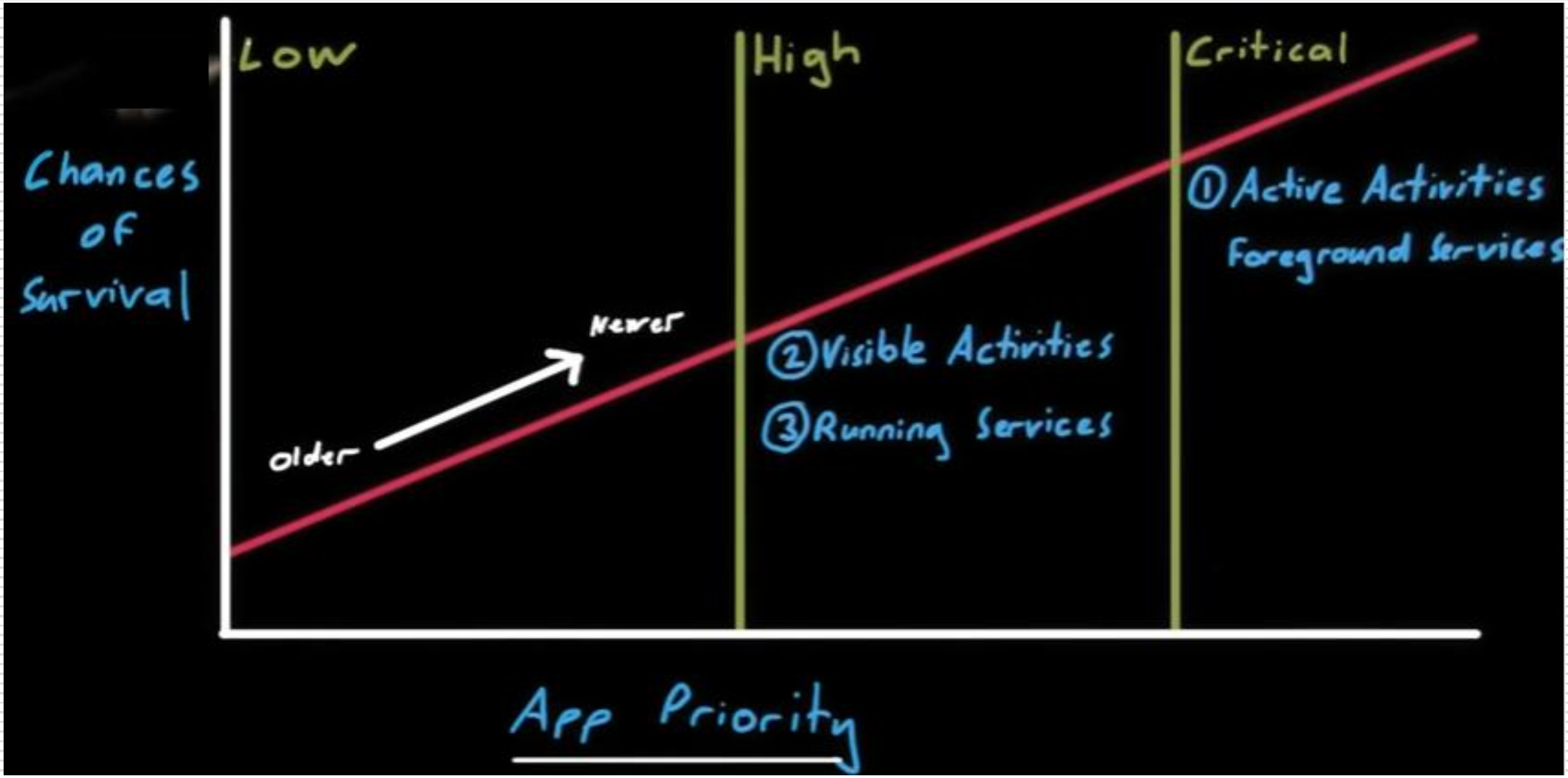
Media 6.2MB
Cached background process

Google Search 36MB
Cached background process

Android Core Apps 8.2MB
Cached background process

RAM
639MB used 1.2GB free

Application Priority /Application states



Later Watch this Video:
https://www.youtube.com/watch?v=XCmz0Mc4uzM

Application Priority tree /Application states

Process status	Description	Priority
Active/Foreground	An application in which the user is interacting with an activity, or which has an service which is bound to such an activity. Also if a service is executing one of its lifecycle methods or a broadcast receiver which runs its onReceive() method.	1
Visible	User is not interacting with the activity, but the activity is still (partially) visible or the application has a service which is used by a inactive but visible activity.	2
Service	Application with a running service which does not qualify for 1 or 2.	3
Background	Application with only stopped activities and without a service or executing receiver. Android keeps them in a least recent used (LRU) list and if requires terminates the one which was least used.	4
Empty	Application without any active components.	5

Application Priority tree /Application states

- **Active processes** — Active (foreground) processes have application components the user is interacting with. These are the processes Android tries to keep responsive by reclaiming resources from other applications. There are generally very few of these processes, and they will be killed only as a last resort.
- **Visible processes** — Visible but inactive processes are those hosting “visible” Activities. As the name suggests, visible Activities are visible, but they aren’t in the foreground or responding to user events. This happens when an Activity is only partially obscured (by a non-full-screen or transparent Activity). There are generally very few visible processes, and they’ll be killed only under extreme circumstances to allow active processes to continue.
- **Service processes** — Processes hosting Services that have been started. Because these Services don’t interact directly with the user, they receive a slightly lower priority than visible Activities or foreground Services. Applications with running Services are still considered foreground processes and won’t be killed unless resources are needed for active or visible processes. When the system terminates a running Service it will attempt to restart them (unless you specify that it shouldn’t) when resources become available.
- **Background processes** — Processes hosting Activities that aren’t visible and that don’t have any running Services. There will generally be a large number of background processes that Android will kill using a last-seen-first-killed pattern in order to obtain resources for foreground processes.
- **Empty processes** — To improve overall system performance, Android will often retain an application in memory after it has reached the end of its lifetime. Android maintains this cache to improve the start-up time of applications when they’re relaunched. These processes are routinely killed, as required.

Question

- Consider these four apps. Which priority order would the system rank them in?

A) Gmail doing a background mail sync.
B) Google Music playing a song in the background.
C) The Camera app being used to take a photo.
D) Google Maps in the background

1. A,B,C,D
2. B,A,C,D
3. C,D,B,A
4. C,B,A,D
5. D,A,B,C
6. D,C,B,A

Answer

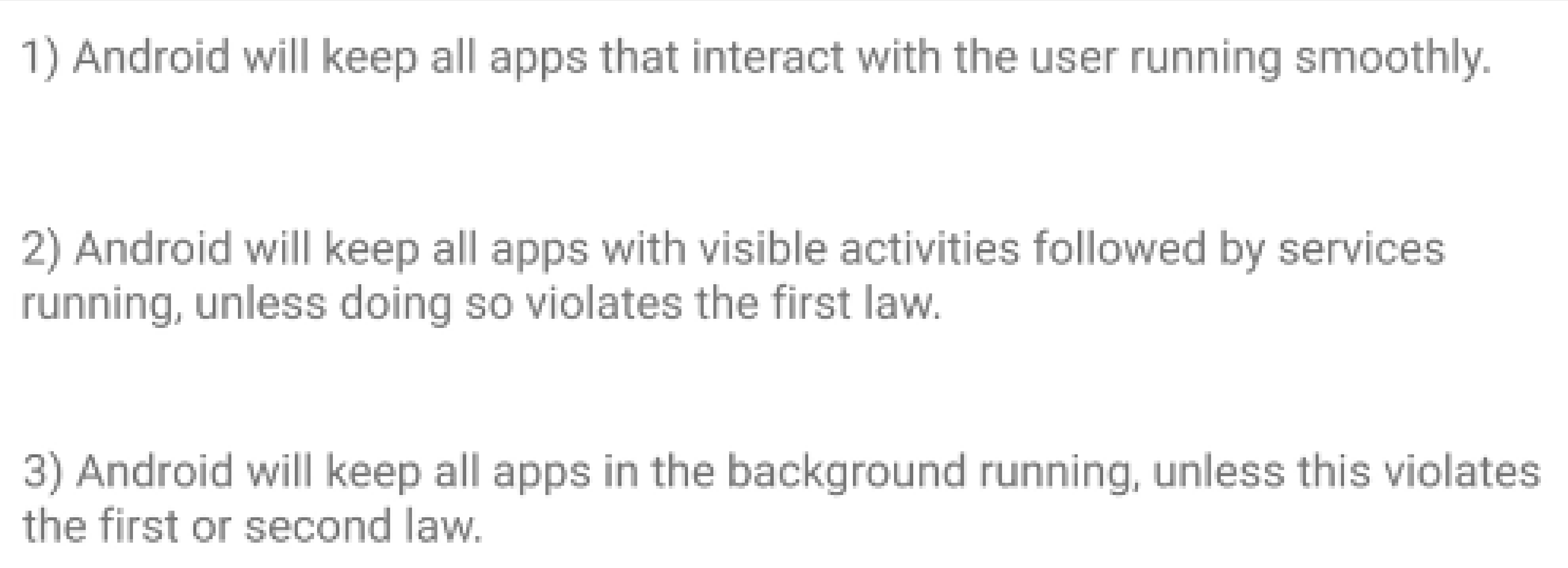
Consider these four apps. Which priority order would the system rank them in?

A) Gmail doing a background mail sync.
B) Google Music playing a song in the background.
C) The Camera app being used to take a photo.
D) Google Maps in the background

1. A,B,C,D
2. B,A,C,D
3. C,D,B,A
4. C,B,A,D
5. D,A,B,C
6. D,C,B,A

That's right -- Maps isn't visible or running any services, so it's the most likely to be killed --- if we were navigating, it would be completely different. Gmail is running a service of some sort but not directly interacting with the user, while Google Music and the Camera app are. Of those two, the music app is still only a foreground service, so it's got a slightly lower priority -- though neither will be killed unless memory pressure is exceptional. Which, since the camera app is pretty heavy-weight, it could be.

Three laws of Android Resource Management



1) Android will keep all apps that interact with the user running smoothly.

2) Android will keep all apps with visible activities followed by services running, unless doing so violates the first law.

3) Android will keep all apps in the background running, unless this violates the first or second law.

Question

Explain with neat diagram Android Application priorites /states

Unit 2: INTRODUCING THE ANDROID APPLICATION CLASS

INTRODUCING THE ANDROID APPLICATION CLASS

- Your application’s Application object remains instantiated whenever your application runs. Unlike
- Activities, the Application is not restarted as a result of configuration changes. Extending the

Application class with your own implementation enables you to do three things:

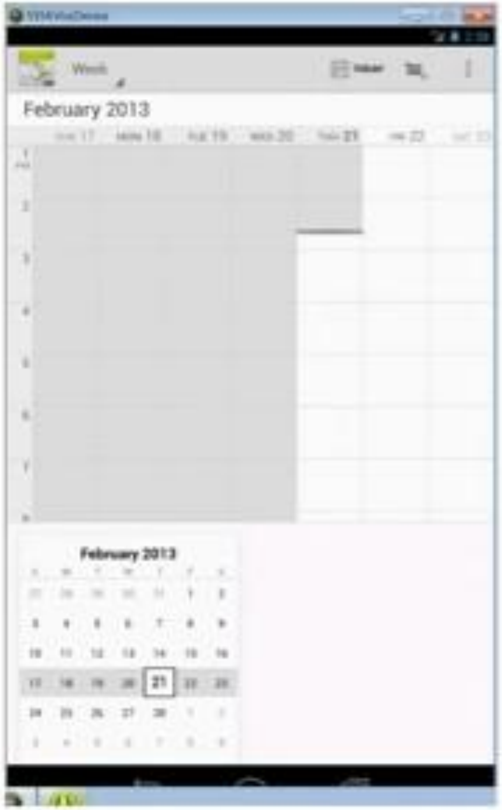
  - Respond to application level events broadcast by the Android run time such as low memory conditions.
  - Transfer objects between application components.
  - Manage and maintain resources used by several application components.

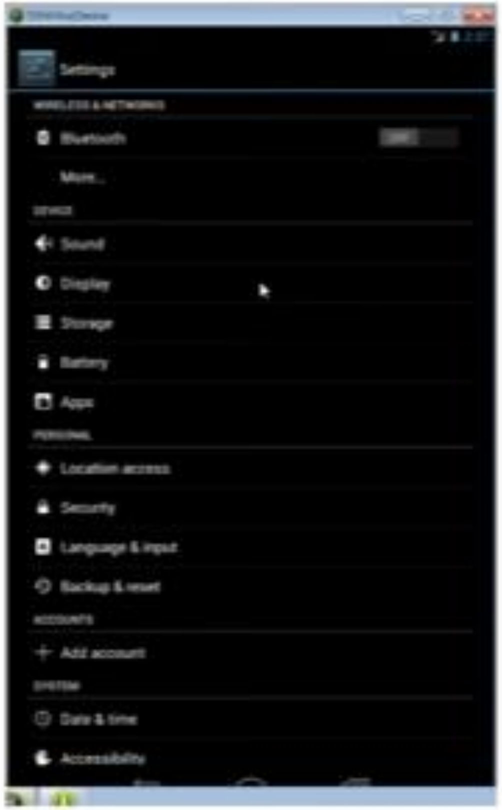
Skeleton code for extending the Application class and implementing it as a singleton.

```
import android.app.Application;
import android.content.res.Configuration;
public class MyApplication extends Application {
private static MyApplication singleton;
// Returns the application instance
public static MyApplication getInstance() {
return singleton;
}
@Override
public final void onCreate() {
super.onCreate();
singleton = this;
} }
```

Unit 2: A CLOSER LOOK AT ANDROID ACTIVITIES

What is an Activity ?





An application component that provides a screen

Draws its UI on its window

What is an Activity ?

Every app has 1 main activity and other activities

An app can start any Activity belonging to itself or other apps subject to certain conditions

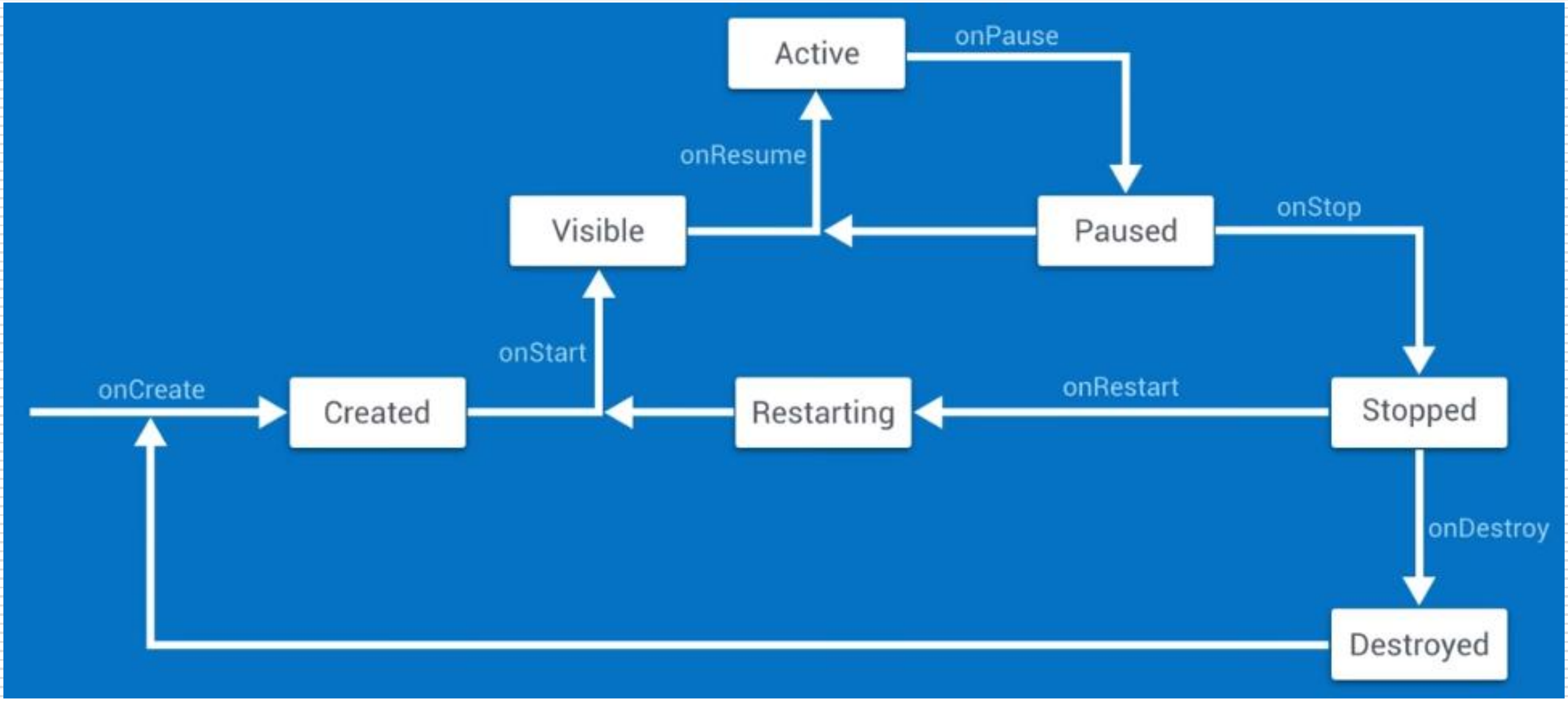
When a new Activity starts, the previous Activity is stopped and added to a stack known as BackStack

Seven lifecycle method of Activity describes how activity will behave at different states.

Activity Lifecycle Methods

Method	Description	Killable?	Next method
onCreate()	Called when activity first created	No	*onStart()*
onRestart()	Called after activity stopped, prior to restarting	No	*onStart()*
onStart()	Called when activity is becoming visible to user	No	*onResume()/onStop()*
onResume()	Called when activity starts interacting with user	No	*onPause()*
onPause()	Called when a previous activity is about to resume	Yes	*onResume()/onStop()*
onStop()	Called when activity no longer visible to user	Yes	*onRestart()/onDestroy()*
onDestroy()	Final call received before activity is destroyed	Yes	Nothing

Android Activity Lifecycle



Activity LifeCycle methods

Actions by mobile user	Methods Called

When user clicks on app	onCreate()
	onStart()
	onResume()

When user switches To another app	onPause()
	onStop()

When back button is presses	onRestart()
	onStart()
	onResume()

When user closes the app or there is no memory	onDestroy ()

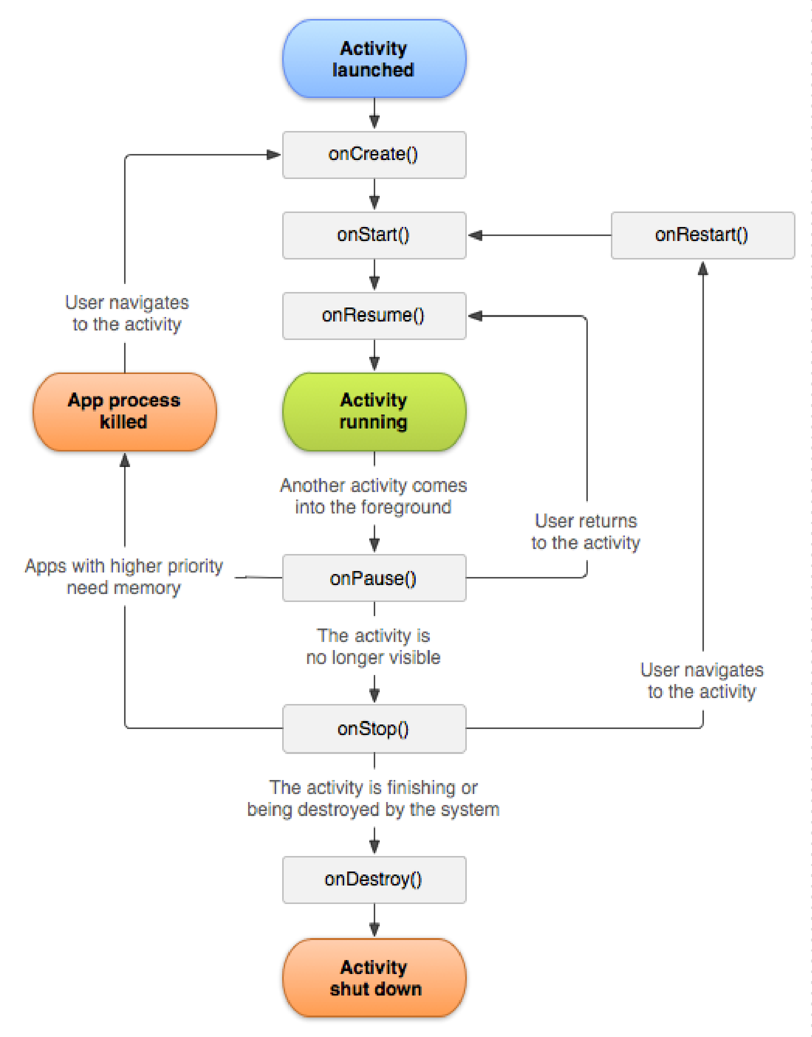
Android Activity Lifecycle methods

- □ As a user navigates through, out of, and back to your app, the Activity instances in your app transition through different states in their lifecycle.

To navigate transitions between stages of the activity lifecycle, the Activity class provides a core set of six callbacks:

onCreate(),
onStart(),
onResume(),
onPause(),
onStop(), and
onDestroy().

The system invokes each of these callback activity enters a new state.



Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
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

Question

- Explain with neat diagram Android Activity LifeCycle
- With Example program demonstrate Android activity LifeCycle Methods

Android Activity Lifecycle Example: MainActivity.java

```
package com.example.activitylifecycle;
import android.os.Bundle;
import android.app.Activity;
import android.util.Log;
import android.view.Menu;
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Log.d("lifecycle","onCreate invoked");
    }
    @Override
    protected void onStart() {
        super.onStart();
         Log.d("lifecycle","onStart invoked");
    }
    @Override
    protected void onResume() {
        super.onResume();
         Log.d("lifecycle","onResume invoked");
    }
    @Override
    protected void onPause() {
        super.onPause();
         Log.d("lifecycle","onPause invoked");
    }
    @Override
    protected void onStop() {
        super.onStop();
         Log.d("lifecycle","onStop invoked");
    }
       @Override
    protected void onRestart() {
        super.onRestart();
         Log.d("lifecycle","onRestart invoked");
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
         Log.d("lifecycle","onDestroy invoked");
    }
}
```

Question

- □ You can shut down an activity by calling its __________ method

  a)onDestory()

  b)onPause()

  c)onRestart()

  d)None of the above

Question

You can shut down an activity by calling its __________ method

a)onDestory()
b)onPause()
c)onRestart()
d)None of the above

Question

When the Activity reaches the top of the activity stack and becomes the foreground process, the _______________ method is called

- □ onCreate()
- □ onRestart()
- □ onPause()
- □ onResume()

Question

When another Activity moves to the top of the activity stack, the current Activity is informed that it is being pushed down the activity stack by way of the _______________ method

□ onResume()

□ onStart()

□ onPause()

□ None of the above

Extending and Using the Application Class

Skeleton of Application class

```
import android.app.Application;
import android.content.res.Configuration;
public class MyApplication extends Application {
private static MyApplication singleton;
// Returns the application instance
public static MyApplication getInstance() {
return singleton;
}
@Override
public final void onCreate() {
super.onCreate();
singleton = this;
}
}
```

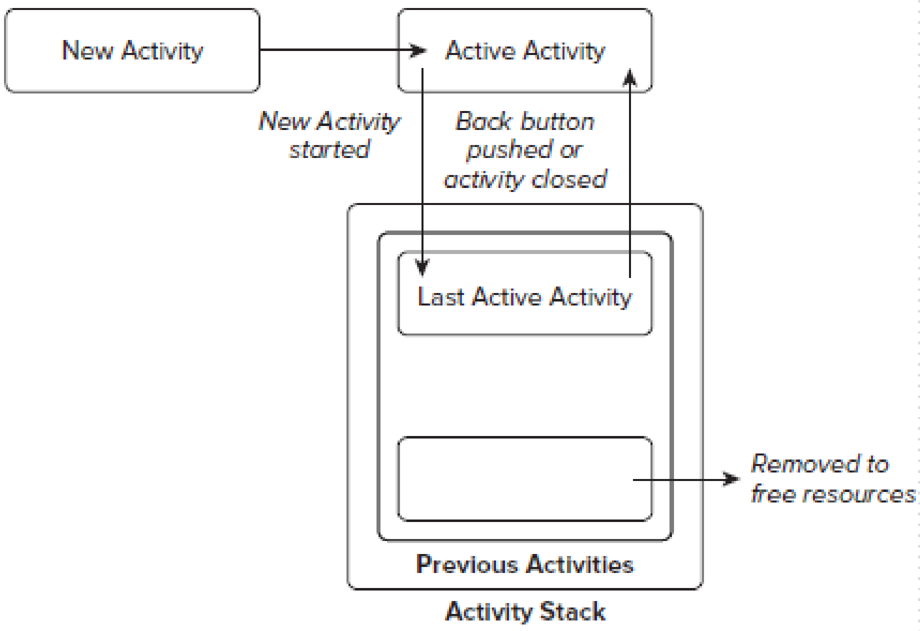
Overriding the Application Lifecycle Events

The Application class provides event handlers for application creation and termination, low memory conditions, and configuration changes

- □ onCreate — Called when the application is created. Override this method to initialize your application singleton and create and initialize any application state variables or shared resources.
- □ onLowMemory — Provides an opportunity for well-behaved applications to free additional memory when the system is running low on resources. This will generally only be called when background processes have already been terminated and the current foreground applications are still low on memory. Override this handler to clear caches or release unnecessary resources.
- □ onTrimMemory — An application specific alternative to the onLowMemory handler introduced in Android 4.0 (API level 13). Called when the run time determines that the current application should attempt to trim its memory overhead – typically when it moves to the background. It includes a level parameter that provides the context around the request.
- □ onConfigurationChanged — Unlike Activities Application objects are not restarted due to configuration changes. If your application uses values dependent on specific configurations, override this handler to reload those values and otherwise handle configuration changes at an application level

Activity Stacks

- The state of each Activity is determined by its position on the Activity stack, a last-in–first-out collection of all the currently running Activities. When a new Activity starts, it becomes active and is moved to the top of the stack. If the user navigates back using the Back button, or the foreground Activity is otherwise closed, the next Activity down on the stack moves up and becomes active.

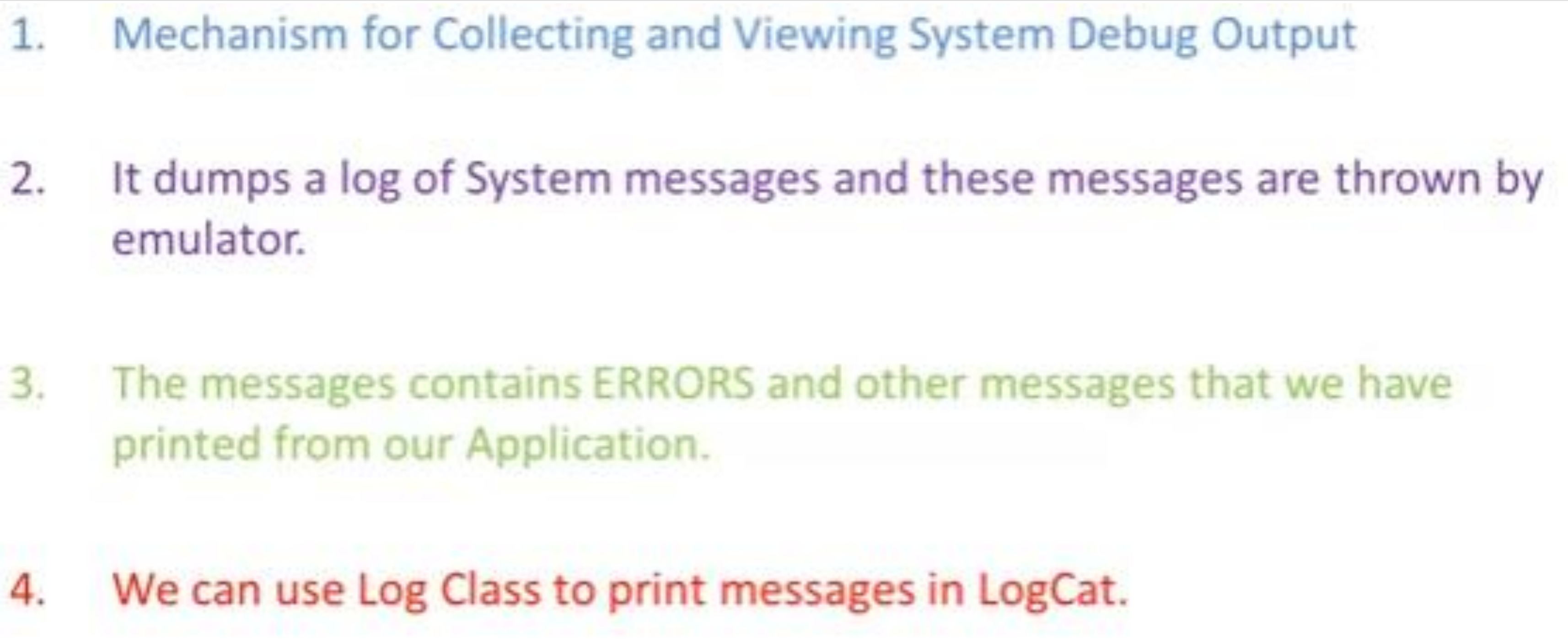


Activity States

- As Activities are created and destroyed, they move in and out of the stack. As they do so, they transition through four possible states:
- **Active** — When an Activity is at the top of the stack it is the visible, focused, foreground Activity that is receiving user input. Android will attempt to keep it alive at all costs, killing Activities further down the stack as needed, to ensure that it has the resources it needs. When another Activity becomes active, this one will be paused.
- **Paused** — In some cases your Activity will be visible but will not have focus; at this point it's paused. This state is reached if a transparent or non-full-screen Activity is active in front of it. When paused, an Activity is treated as if it were active; however, it doesn't receive user input events. In extreme cases Android will kill a paused Activity to recover resources for the active Activity. When an Activity becomes totally obscured, it is stopped.
- **Stopped** — When an Activity isn't visible, it "stops." The Activity will remain in memory, retaining all state information; however, it is now a candidate for termination when the system requires memory elsewhere. When an Activity is in a stopped state, it's important to save data and the current UI state, and to stop any non-critical operations. Once an Activity has exited or closed, it becomes inactive.
- **Inactive** — After an Activity has been killed, and before it's been launched, it's inactive. Inactive Activities have been removed from the Activity stack and need to be restarted before they can be displayed and used.

Thanks for Listening

About Log Cat



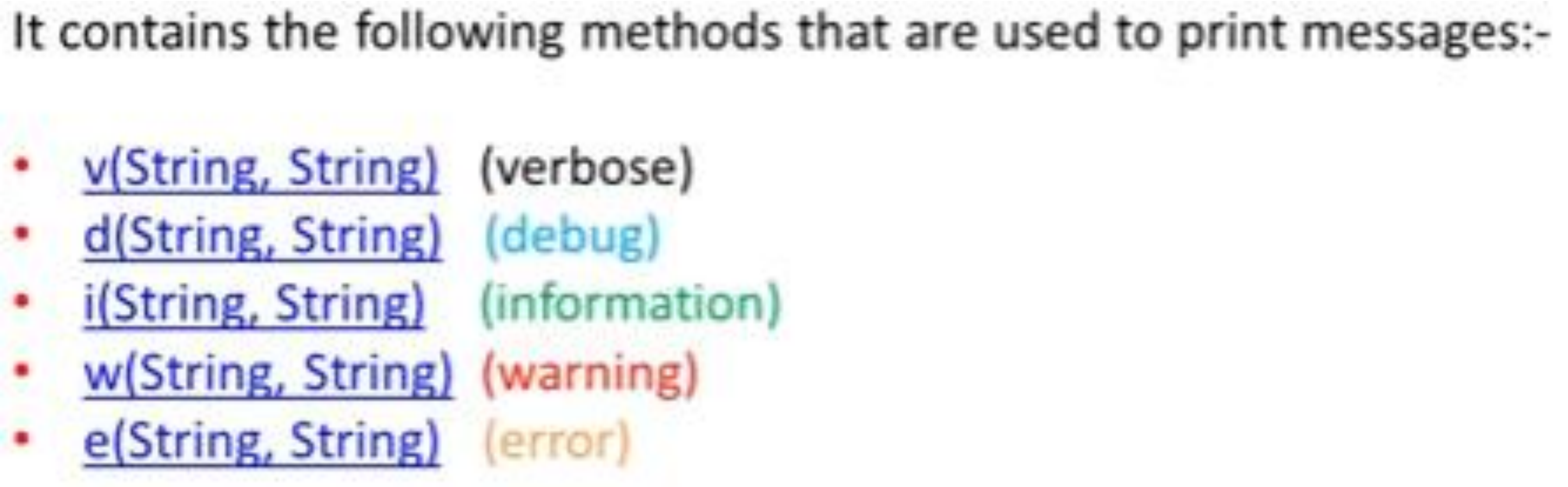
1. Mechanism for Collecting and Viewing System Debug Output

2. It dumps a log of System messages and these messages are thrown by emulator.

3. The messages contains ERRORS and other messages that we have printed from our Application.

4. We can use Log Class to print messages in LogCat.

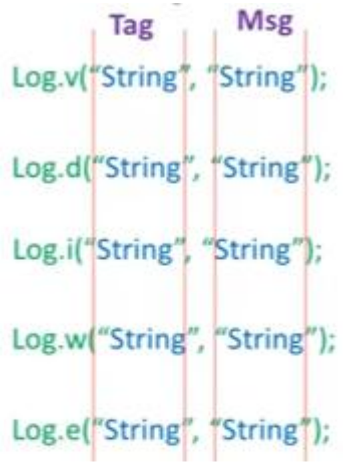
The Log Class



It contains the following methods that are used to print messages:-

- v(String, String) (verbose)
- d(String, String) (debug)
- i(String, String) (information)
- w(String, String) (warning)
- e(String, String) (error)

Syntax



```
Log.v("String", "String");

Log.d("String", "String");

Log.i("String", "String");

Log.w("String", "String");

Log.e("String", "String");
```

Verbose - Show all log messages (the default).
Debug - Show debug log messages that are useful during development only, as well as the message levels lower in this list.
Info - Show expected log messages for regular usage, as well as the message levels lower in this list.
Warn - Show possible issues that are not yet errors, as well as the message levels lower in this list.
Error - Show issues that have caused errors, as well as the message level lower in this list.