Android Application development Introduction- Prepared for Software for Embedded Systems Course

1. Setting up The Tools and Development Environment

There are two choices for the development environment.

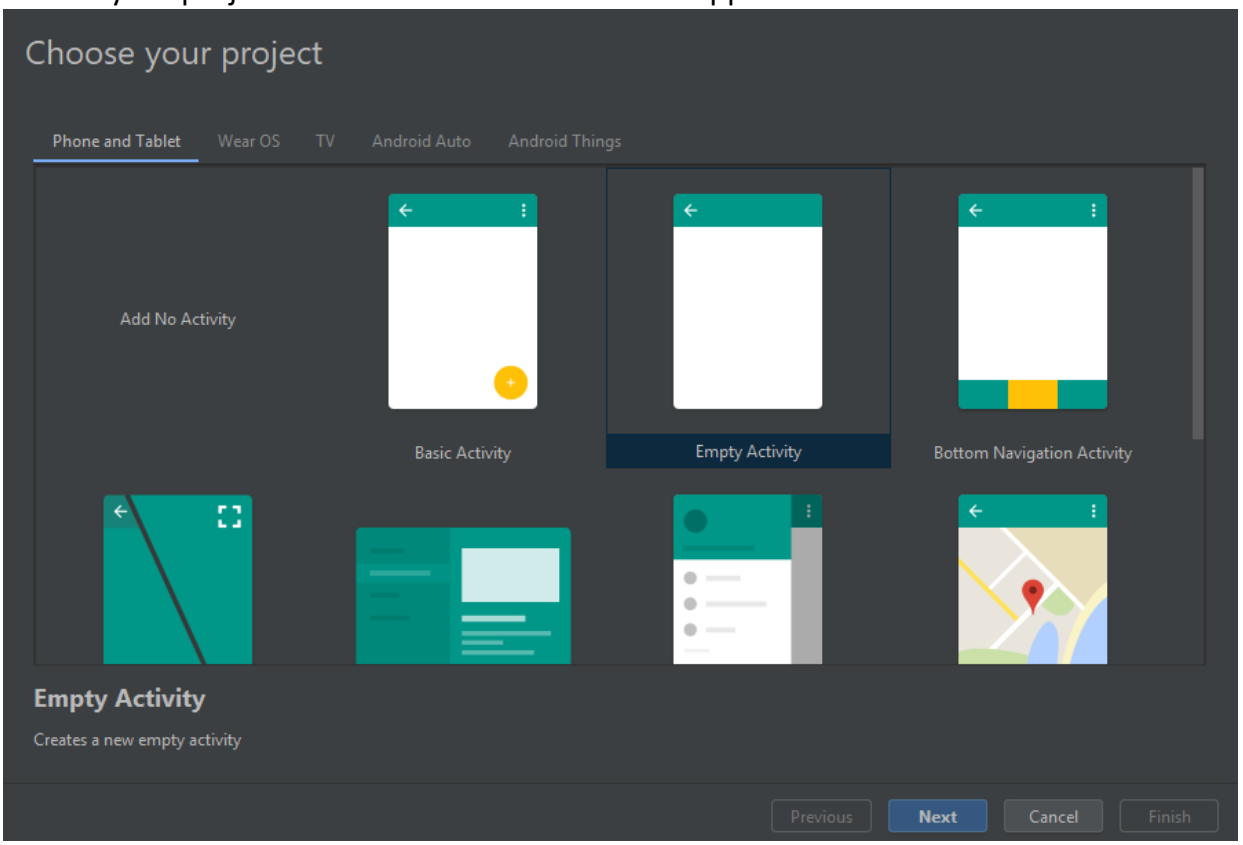
- Use 'Android Studio' – Recommended since Google has adopted it for future developments
- Use 'Eclipse' IDE

We will be using Android Studio for our Android App Development.
Download Android Studio from the following link.
https://developer.android.com/studio/index.html

Let's start with building a simple android app.
So Launch Android Studio

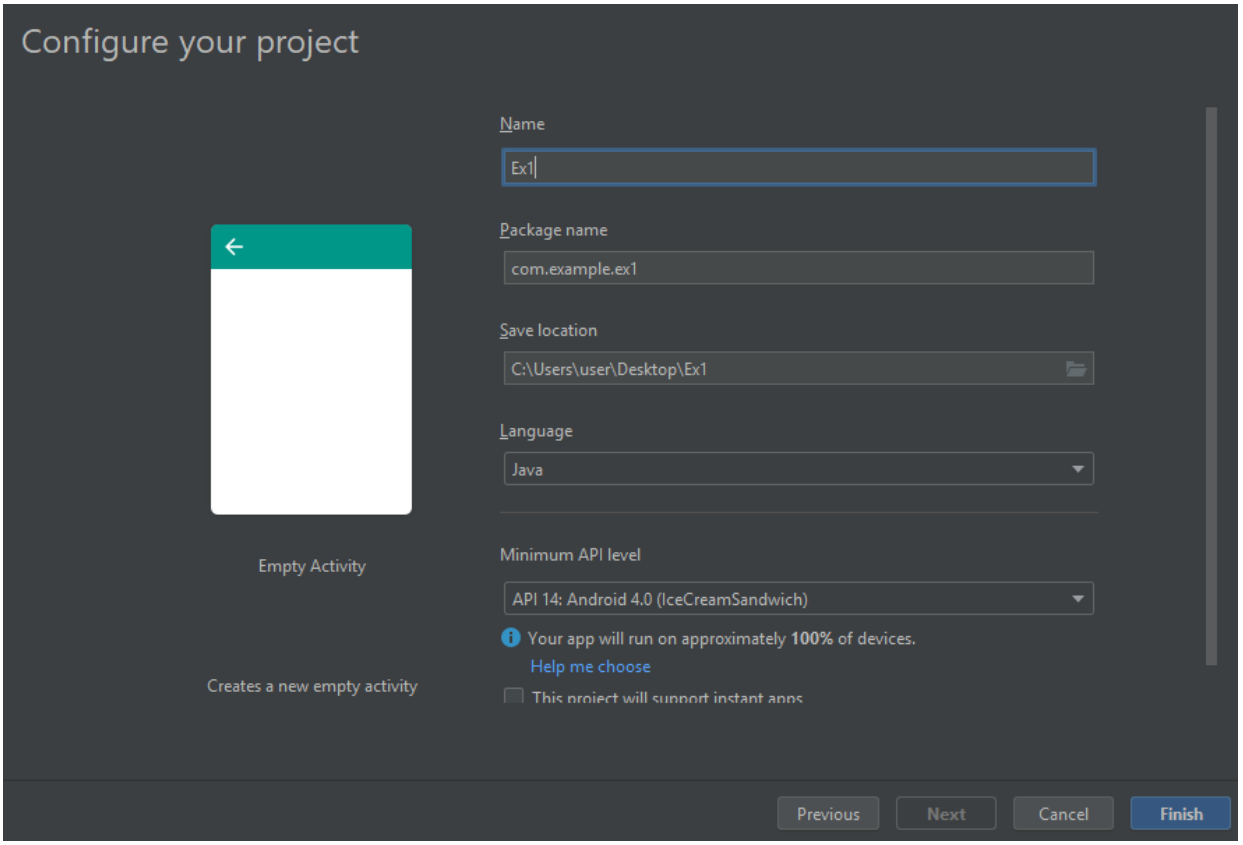
2. Create the Project

a. Click New->Project
Choose your project window as shown below will appear.



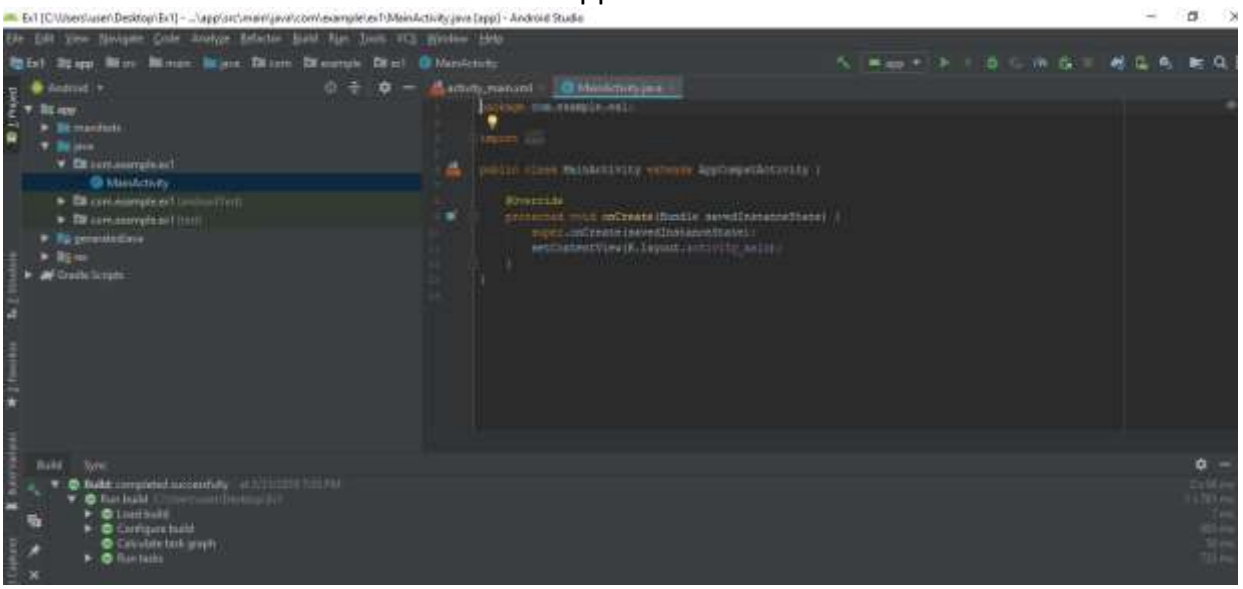
Choose Empty Activity and click on Next

b.When the configure your project dialog box appears, Enter the Name of application.



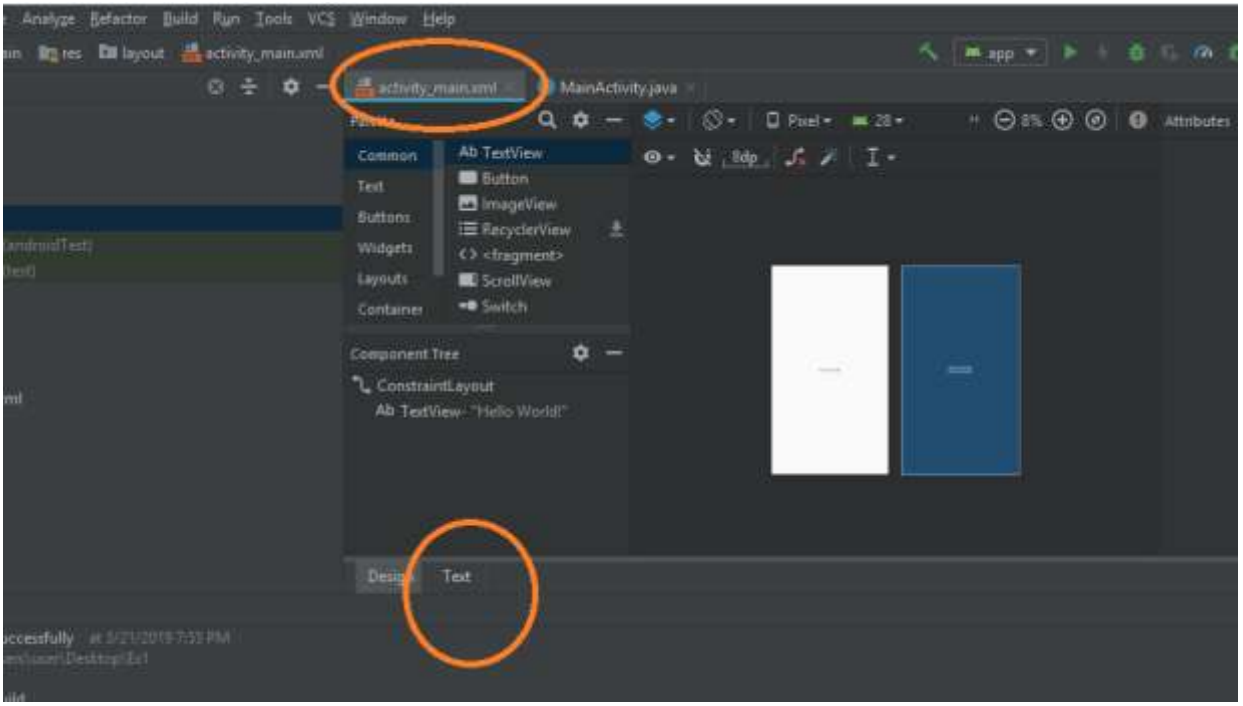
Note that when you select the Minimum SDK, the app developed will run on systems with atleast that version of Android.

c. Click on Finish to create the project.
Wait for some time till a similar window appears

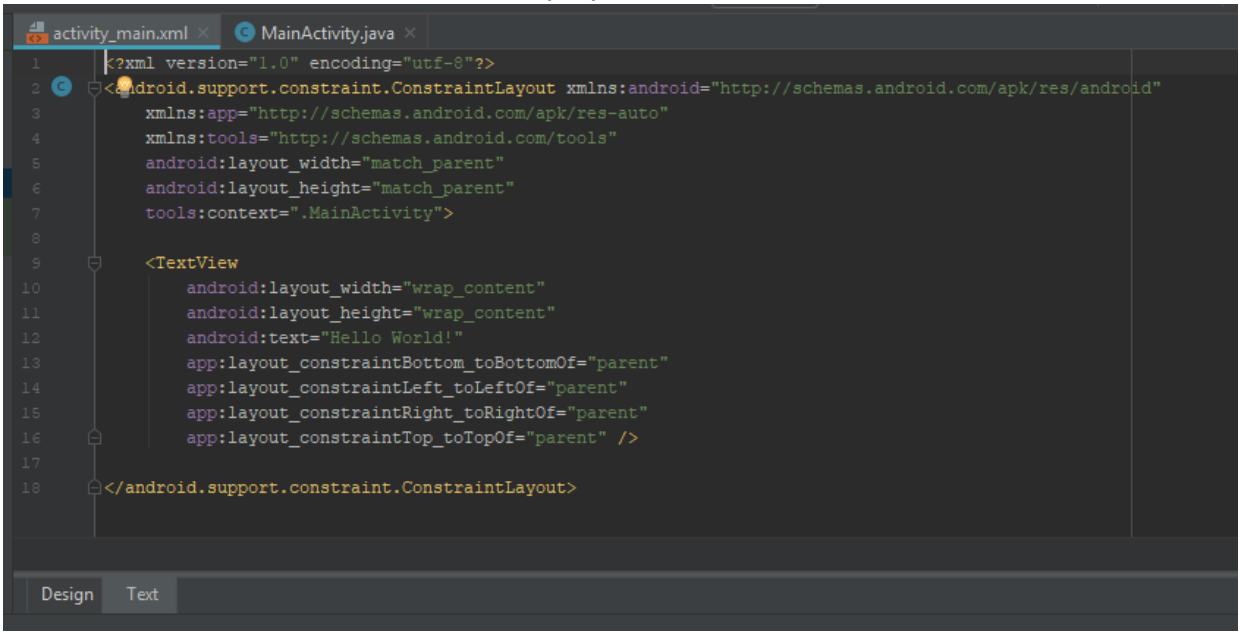


d. Write the Code

There are multiple places we have to add code, which are explained below. There are mainly two files MainActivity.java and activity_main.xml
The xml will have the static resources which the android application will use.
Select the activity_main.xml file and select the text option as highlighted in the figure.



The window as shown below will be displayed.

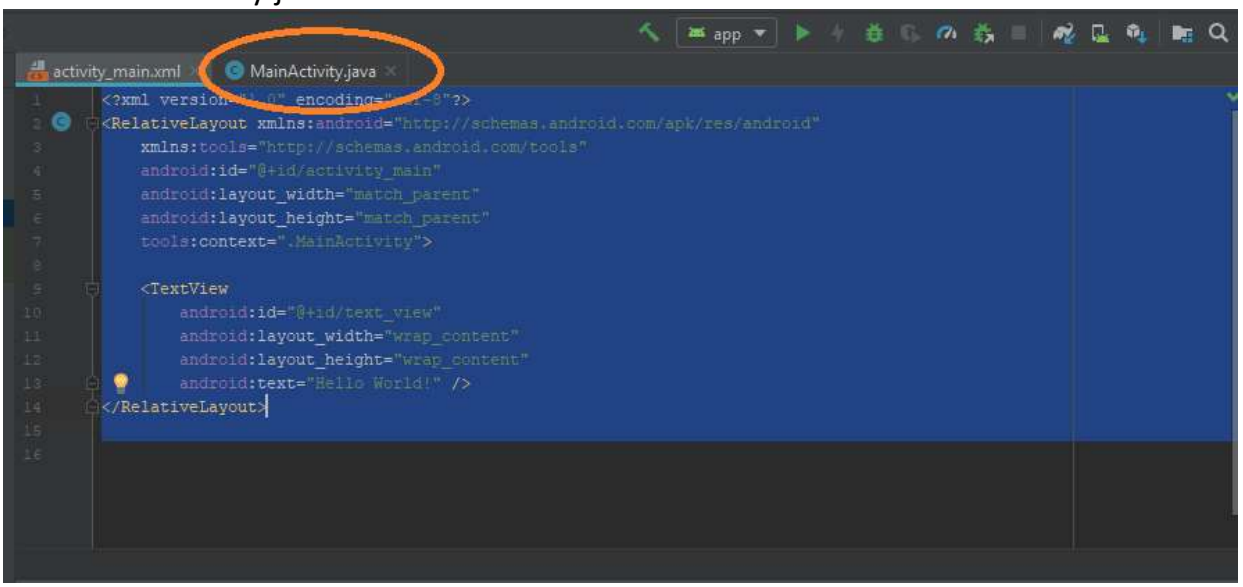


We will make the necessary changes in the code. (There will be three views for activity_main.xml code, split and design, select code option)

Make the necessary changes so that the following text appears in activity_main.xml file(After analysis you may copy paste the code as well)

```
<?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"
    tools:context=".MainActivity">

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

As we complete more examples you will understand the relevance of changes made.
Select MainActivity.java



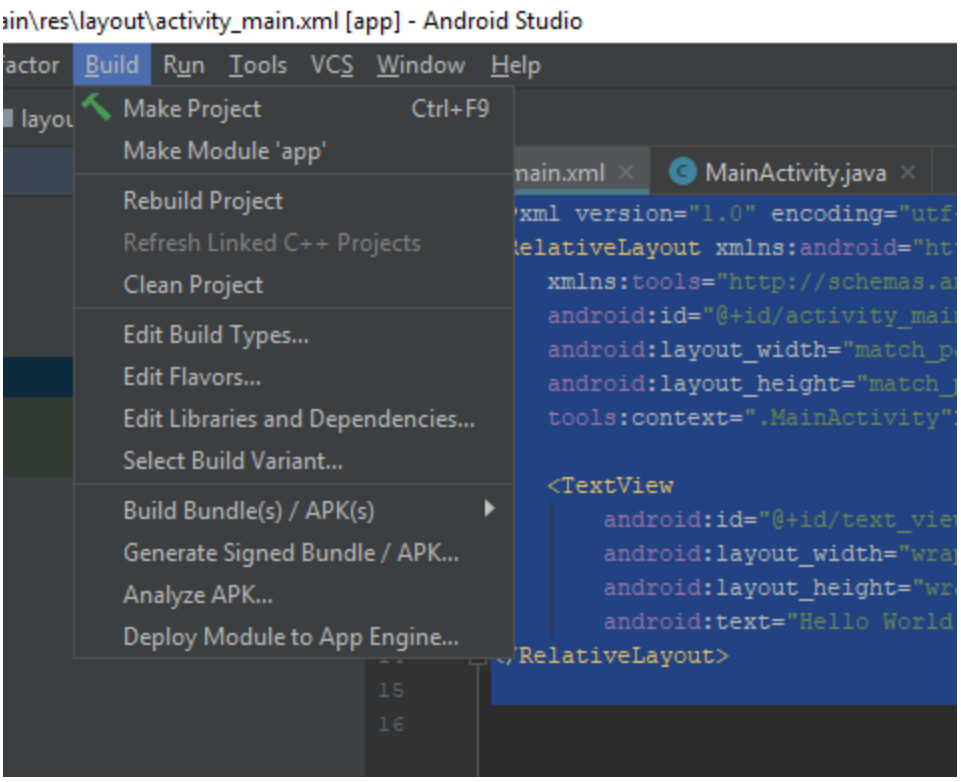
Make the changes in code as shown below (After analysis you may copy paste the code as well)

See Note-->

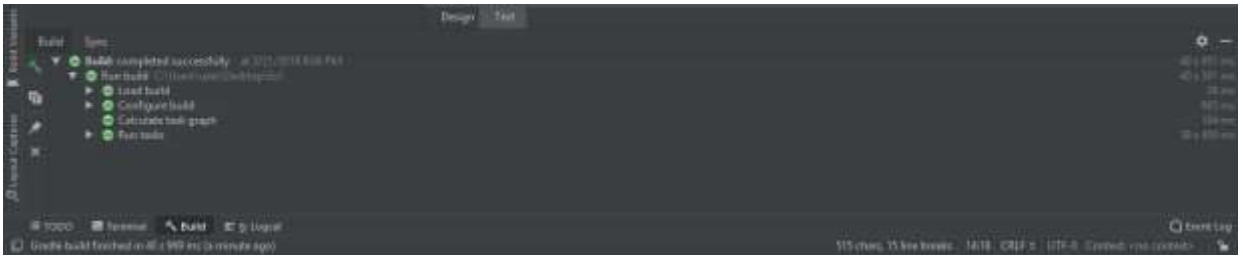
```
package com.example.ex1;

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

public class MainActivity extends AppCompatActivity {
    TextView textView;

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

        textView = (TextView) findViewById(R.id.text_view);
        textView.setText("Hello World ! I am learning Android.");
    }
}
```

Go to Build→ and select Make Project

Note . If you get build error for this line use
import androidx.appcompat.app.AppCompatActivity;
instead of the line which gave error



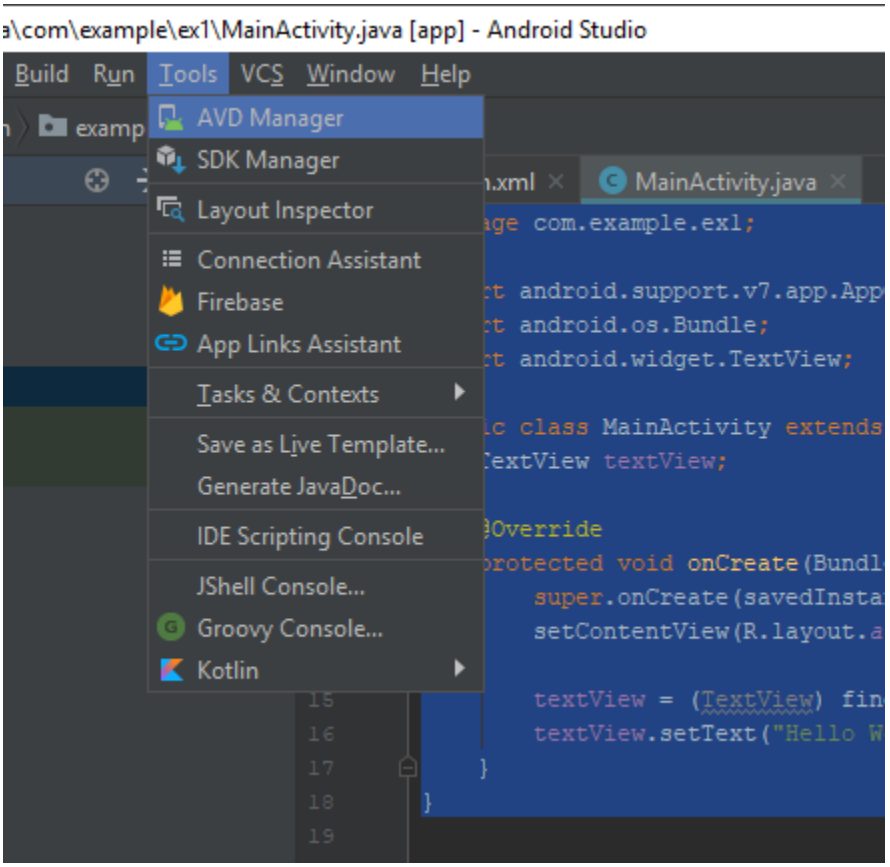
Check for errors.



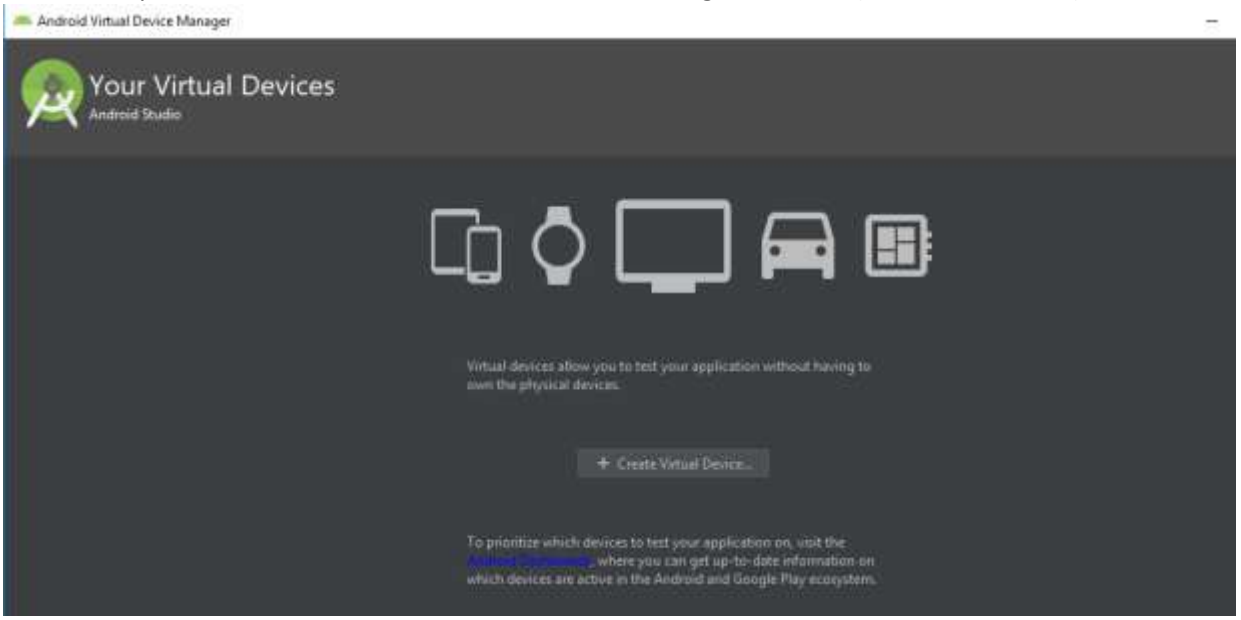
3. Launch the Simulator for Testing

We will use the simulator for testing. Android Simulator is called **AVD** (Android Virtual Device). For using it we need to create and AVD and launch it. It is done in the following way.

- Click Tools->Android->AVD Manager

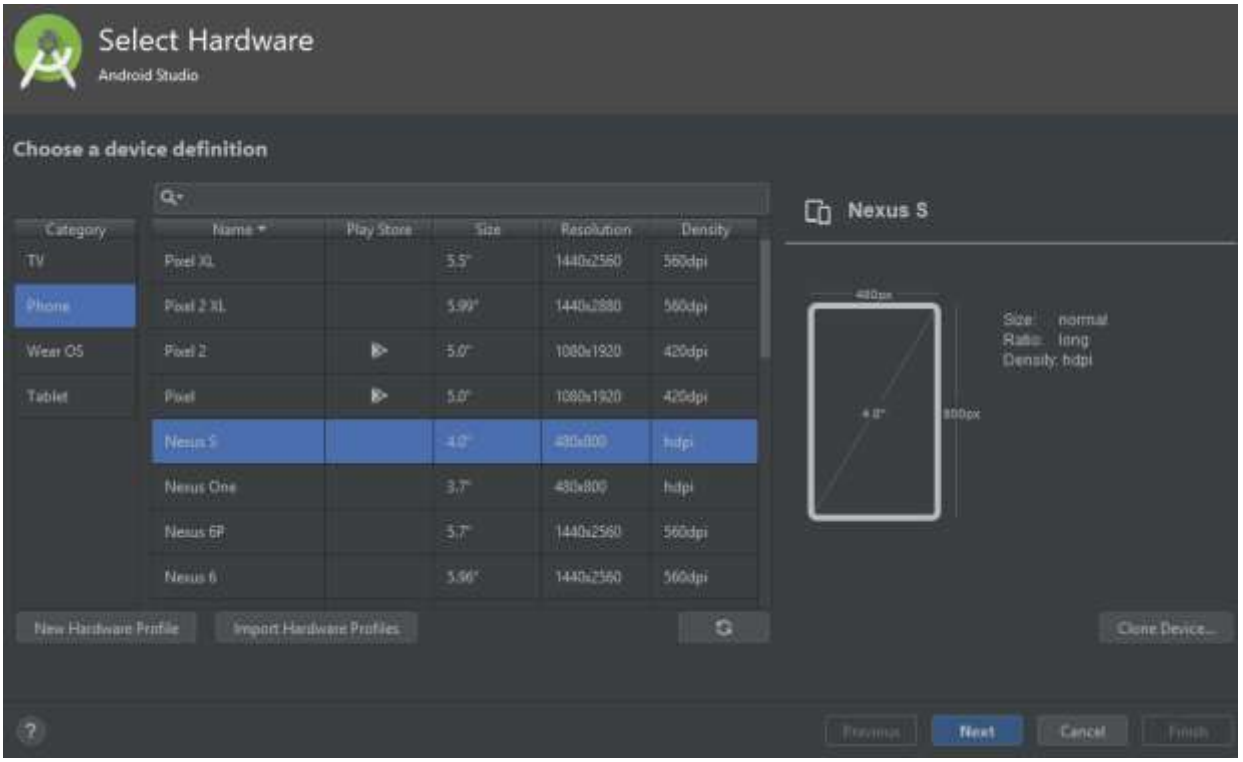


➢ It will open an window, which will list down the existing Simulators (or Virtual Devices)

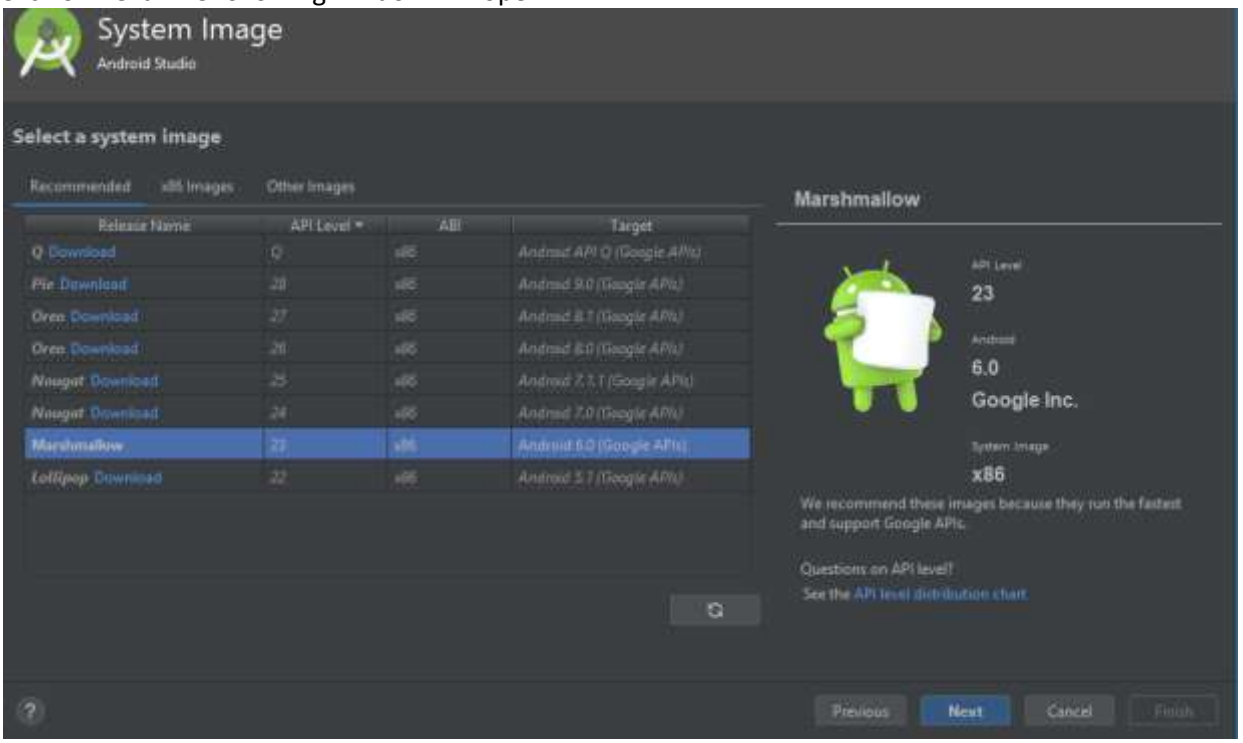


➢ In order to create a new one, do the following.

- Click ‘Create Virtual Device’ button & select any device as shown below



Click on next. The following window will open.

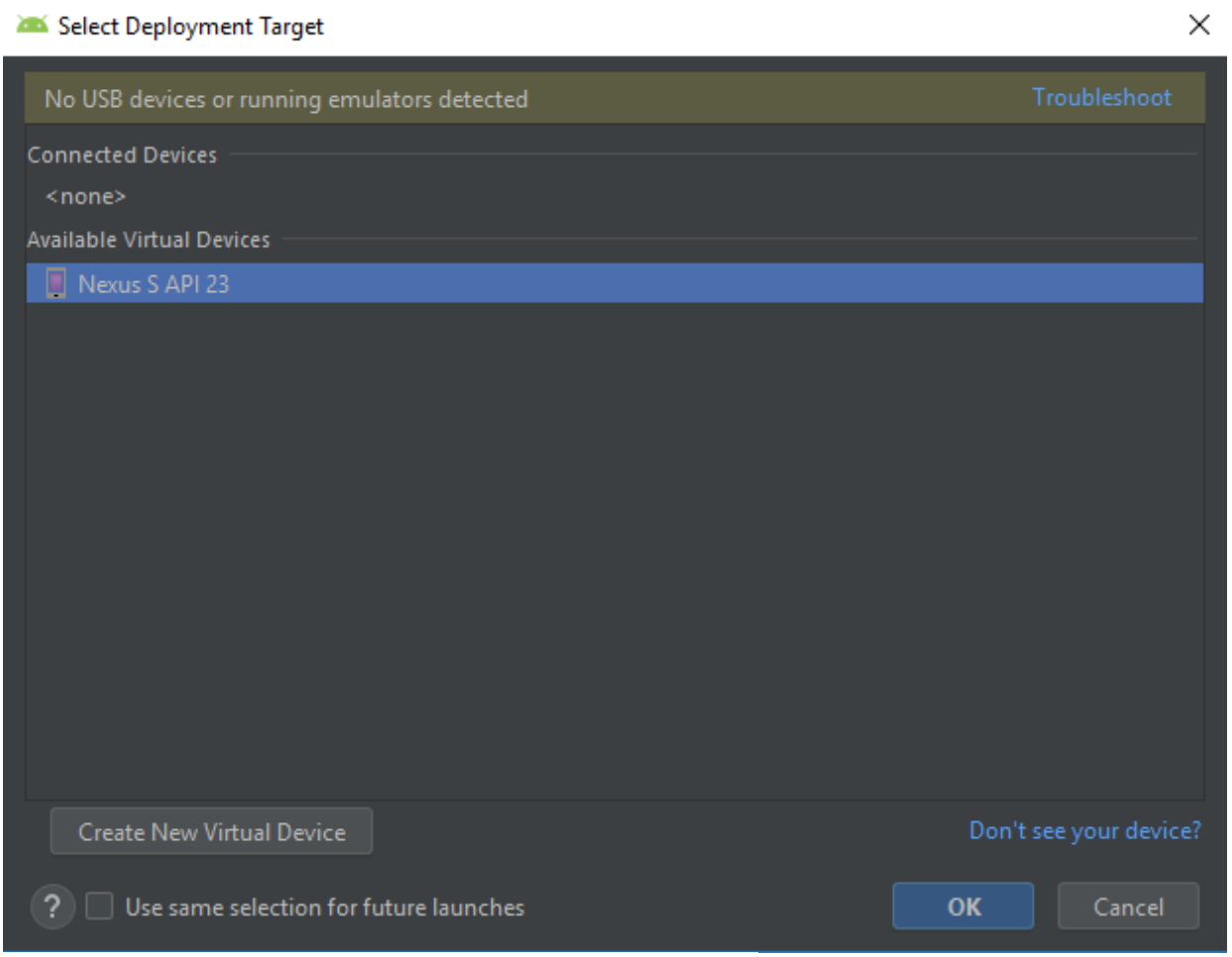


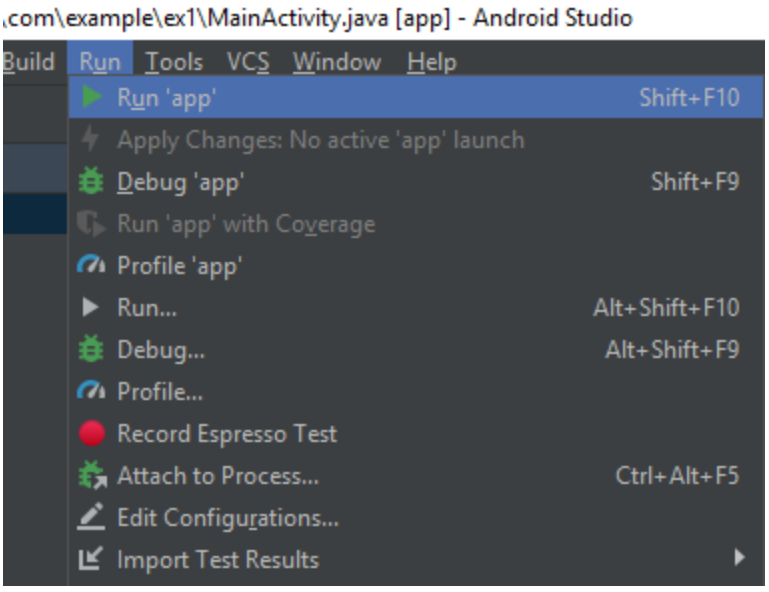
Download and install Mashmallow system image and click on Next

Click on Finish

Then click on Run→RunApp as shown below

Select the available virtual device

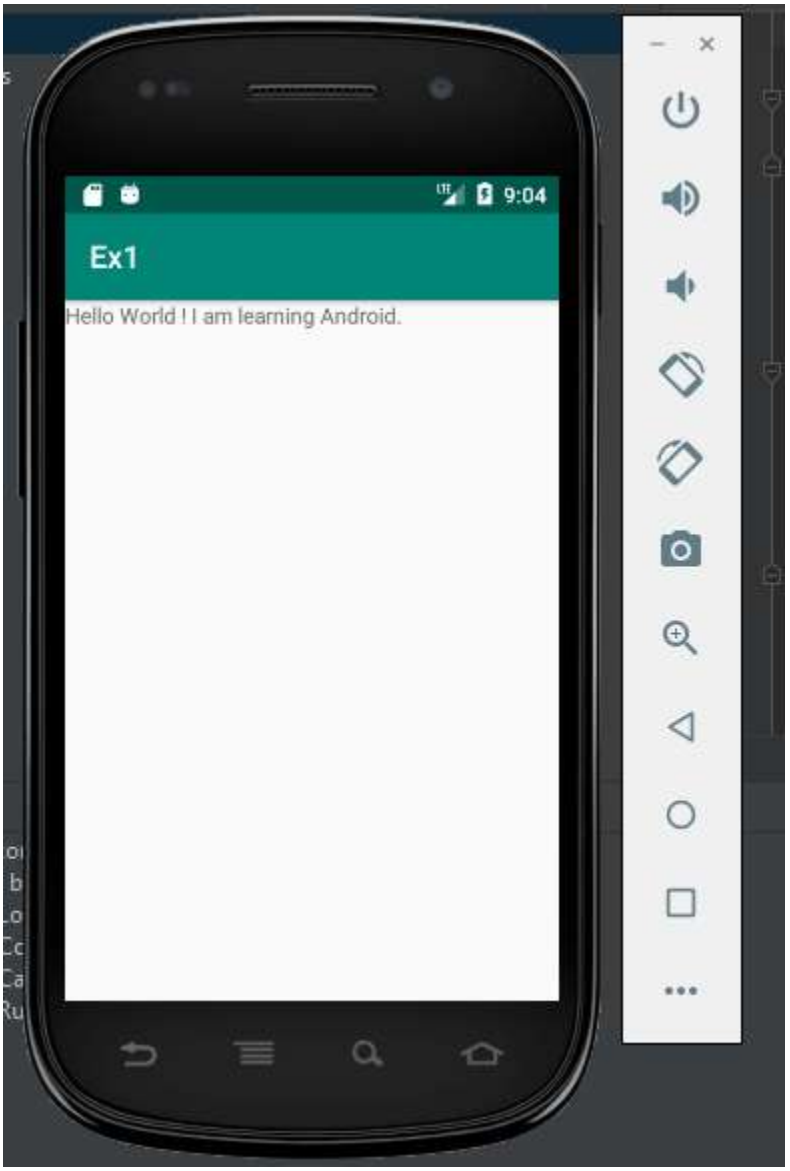




Android Emulator will open up as shown below

Note: If there is any error in launching the emulator, you may have to enable virtualization option in BIOS

Wait for some time till the device becomes online.
You will be able to see the text displayed on the android emulator


9:04
Ex1
Hello World ! I am learning Android.

Android Internals

Introduction

Android is based on a modified Linux 2.6 or of later version with a Java programming interface. It provides tools, e.g. a compiler, debugger and a device emulator.

Android uses a special virtual machine (java virtual machine) called the ART, in order to run the Java programs. Therefore standard Java bytecode can't be used on Android. Android applications are packed into an .apk (Android Package) file.

To simplify development, Google provides the Android Developer Tools (ADT) for Eclipse IDE and its own IDE Android Studio.

Followings are the different versions of Android released
(The new versions start with the next English alphabet).

Android Versions	Code Name	API Level
1.5	Cupcake	3
1.6	Donut	4
2.0 - 2.1	Eclair	5 - 7
2.2 – 2.2.3	Froyo	8
2.3 – 2.3.7	Gingerbread	9 - 10
3.0 – 3.2.6	Honeycomb	11 - 13
4.0 – 4.0.4	Ice-cream Sandwich	14 - 15
4.1 – 4.3.1	Jelly Bean	16 - 18
4.4 – 4.4.4	KitKat	19 - 20
5.0 – 5.1.1	Lollipop	21 - 22
6.0 – 6.0.1	Marshmallow	23
7.0 – 7.1.2	Nougat	24 - 25
8.0	Oreo	26

Fig. Android Versions

Android Platform Architecture

Source: https://developer.android.com/guide/platform/index.html

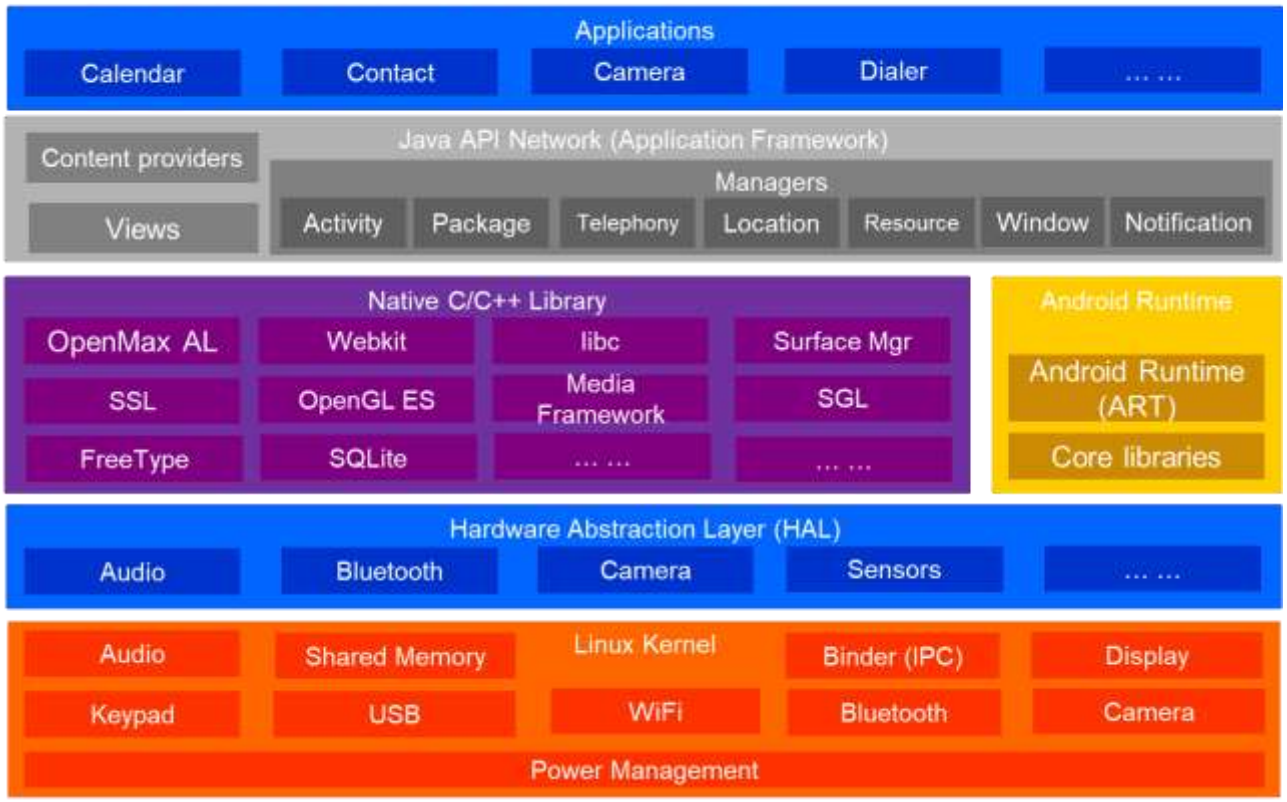


Fig. Android Platform Architecture Block Diagram

Fig above shows the top level block diagram of Android Platform. The entire platform can be divided into following major layers.

- Applications
- Application Framework
- Android Runtime
- Native C/C++ Library
- Hardware Abstraction Layer
- Linux Kernel

Each of these will be discussed in the following sections.

Linux Kernel Layer

The Linux Kernel is based on standard Linux kernel with version 2.6 and above. Latest versions of Android mandate recent versions of Linux kernel. Over the kernel, several enhancements have been done.

Binder is the IPC (Inter process communication) mechanism, which optimizes the context switch overhead.

Binder is used for communication and resource management of different components of linux stack. All upper layer functionalities rely on underlying Linux functionalities.

Hardware Abstraction Layer (HAL)

This layer provides standard interface to provide hardware capabilities to upper layers. It consists of libraries for specific hardware modules like Bluetooth, Camera etc. Whenever upper layer try to call an API related to a hardware, the systems loads the library specific to that hardware.

Android Runtime

Android uses Java as the application development, but doesn't use JVM. It has its own virtual machine. 'Android Runtime' layer essentially include this virtual machine.

Android versions before **5.0 (API level 21),** were using **Dalvik VM (Dalvik Virtual Machine).** The aim was to come with a VM, which will be suitable for targeted devices, which will have requirements like **'limited processor speed', 'limited RAM', 'battery powere'** etc. Every Android application runs in its own process, with its own instance of the Dalvik virtual machine. The Dalvik VM executes files in the **Dalvik Executable (.dex)** format which is optimized for minimal memory footprint. The VM runs classes compiled by a Java language compiler (.class or .jar files) that have been transformed into the .dex format by the included "**dx**" tool. The Dalvik VM relies on the Linux kernel for underlying functionality such as threading and low-level memory management.

From **Android 5.0**, DVM was replaced by **ART (Android Runtime).** Like DVM, ART also relies on the Linux kernel for underlying functionalities. Some of the major features of ART include the following: (when compared to DVM)

- Better compile time performance
- Optimized garbage collection
- Better debugging support

ART is compatible with Dalvik Executable Format. So, if your code works with ART, it will work with DVM as well.

Native Library

Followings are different native libraries included in the Android middleware.

Bionic Libc: C Library, optimized for embedded use

Webkit: Opensource Engine for Web browser support (required to develop web browser based application)

FreeType: Bitmap- and vector- based font rendering subsystem

SQLite: A lightweight RDBMS subsystem for persistent data storage (contacts, calendar etc use this data base).This database resides inside android device.

Media Libraries (libraries which is required to play & record audio/video):

- Android media framework is based on **Stagefright**, which is based on **OpenMax**.

- Initial version of Android used to have PacketVideo's OpenCORE based media framework, but it has been replaced by Stagefright.

Surface Manager: Called **Surface Flinger**, meant for combining the display surfaces and rendering the final surface on the screen.

Audio Manager: Called Audio Flinger, meant for mixing all input audios and render the final audio.

For eg: if you are listening to a song and a phone call comes, then the song should be muted and the user should be able to receive phone call. While listening to a music, if a audio enabled pop up comes from web browser, the background volume (music) should be reduced and the pop up audio should be played. These kind of applications are managed by audio manager.

OpenGL: For 2D/3D graphics acceleration.

SGL: For 2D Graphics

SSL (Secured Socket Layer): It is based on Transport Layer Security (TLS) and Public-Key- Cryptography

Application Framework

It provides standard interface to application writer so that they can utilizing underlying functionalities of android stack so as to enable quick development.

Followings are different components of Application Framework.

Activity Manager

- Manages the lifecycle of applications.
- Applications cycle through various states: started, running, background, killed.(If you are running an application, but press the home button, so the application will not be killed but running in the background).
- Maintains a common stack allowing the user to navigate from one application to another.
- Interact with the overall activities running in the system.

Package Manager

- Applications are stored as packages & package manager Maintains information on the application packages that are currently installed on the device.

Window Manager

- The windows manager creates surfaces that applications can draw on.

Telephony Manager

- Provides APIs for the applications to use Telephony functionalities.

Location Manager

- This class provides access to the system location services (eg:GPS).

Resource Manager

- Manages the storing of resources such as layouts, strings, bitmaps.
- Provides ways to support multiple screen sizes and screen densities

Notification Manager

- Alerts the user about events like status bar updates, flashing lights, vibrations etc.

Content Provider

- Content providers manage access to a structured set of data. Responsible for data management. Provide APIs to access data inside device. The data can be even SQL data base.

View System

Provides API to help user develop UI of the application.

- Contains the building blocks of the user interface – buttons, text boxes, edit texts etc.
- Handles UI event dispatching, such as clicks, long clicks, gestures etc. Eg: If a user presses a button. How should that be handled- all of this comes under view system

3.2.1.8. Applications

Every Android device contains a set of core applications including an email client, SMS program, calendar, maps, browser, contacts. These are called '**Built-in'** apps. User can also download and install applications from various authorized play stores (such as Google Play Store, amazon Play Store etc). These are called '**User**' apps.

All applications are written using the **Java** programming language. The Android SDK tools compile your code—along with any data and resource files—into an APK: an *Android package*, which is an archive file with an **.apk** suffix.

Application Components

App components are the essential building blocks of an Android app. There are four types of application components.

Activities

- An activity represents a single screen with a user interface.
- Example: an email app might have one activity to show new emails, another activity to compose an email, and another activity for reading emails.

Services

- A service is a component which runs in the background
- It generally perform long-running operations, continuous operations or to perform work for remote processes.
- A service does not have a user interface.

Content providers

- A content provider manages data for the apps.
- This data can be the data in file system, SQLite database, on the web or on any other persistence storage

Broadcast receivers

- A broadcast receiver is a component which responds to system-wide broadcast announcements.(Eg:battery status, RSSI information of wireless communication)
- Example: a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured.

- Apps can also initiate broadcasts such as it has downloaded some data etc.

Ex2: Example for demonstrating Text &Image View

Repeat part a & b of Ex1. I am naming new project as Ex2

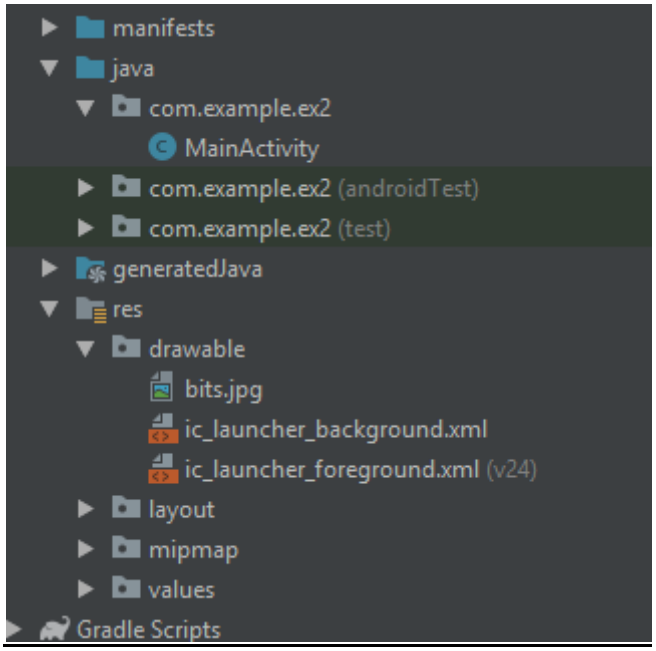
Copy the following code to Mainactivity.java

```
package com.example.ex2;

import android.graphics.BitmapFactory;
import android.net.Uri;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.widget.ImageView;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
    TextView textView;
    ImageView imageView;

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

        textView = (TextView) findViewById(R.id.text_view);
        textView.setText("Displaying Image from Drawable");

        // Display the image from Drawable
        imageView = (ImageView) findViewById(R.id.image_view);
        imageView.setImageResource(R.drawable.bits);
    }
}
```

Copy the following code to activity_main.xml

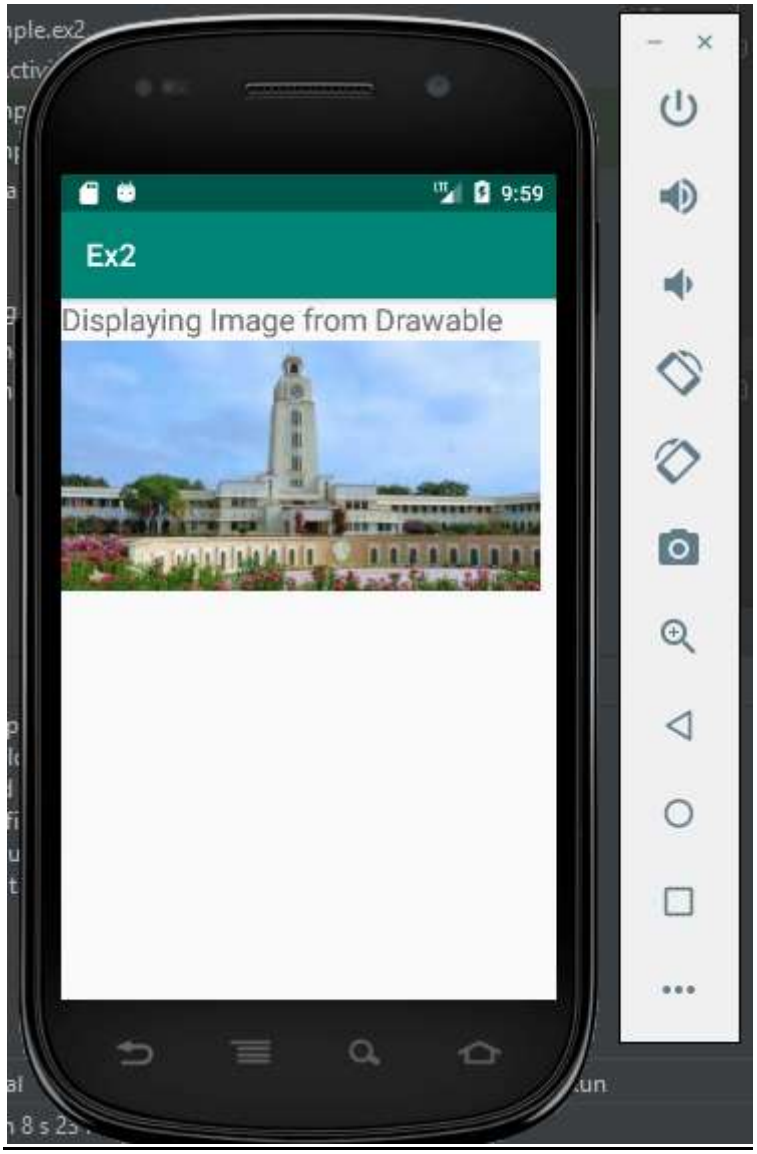
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">


    <TextView
        android:id="@+id/text_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="20sp"/>
    <ImageView
        android:id="@+id/image_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:adjustViewBounds="true"
        android:layout_centerHorizontal="true"
        android:scaleType="fitCenter" />
</LinearLayout>
```

Add the Emblem.jpg under res/drawable by dragging & droping the bits.jpg as shown below



Build the code. Check for errors

Run the app & observe the output



Ex3: Example for demonstrating Buttons

Repeat part a & b of Ex1. I am naming new project as Ex3

Copy the following code to Mainactivity.java

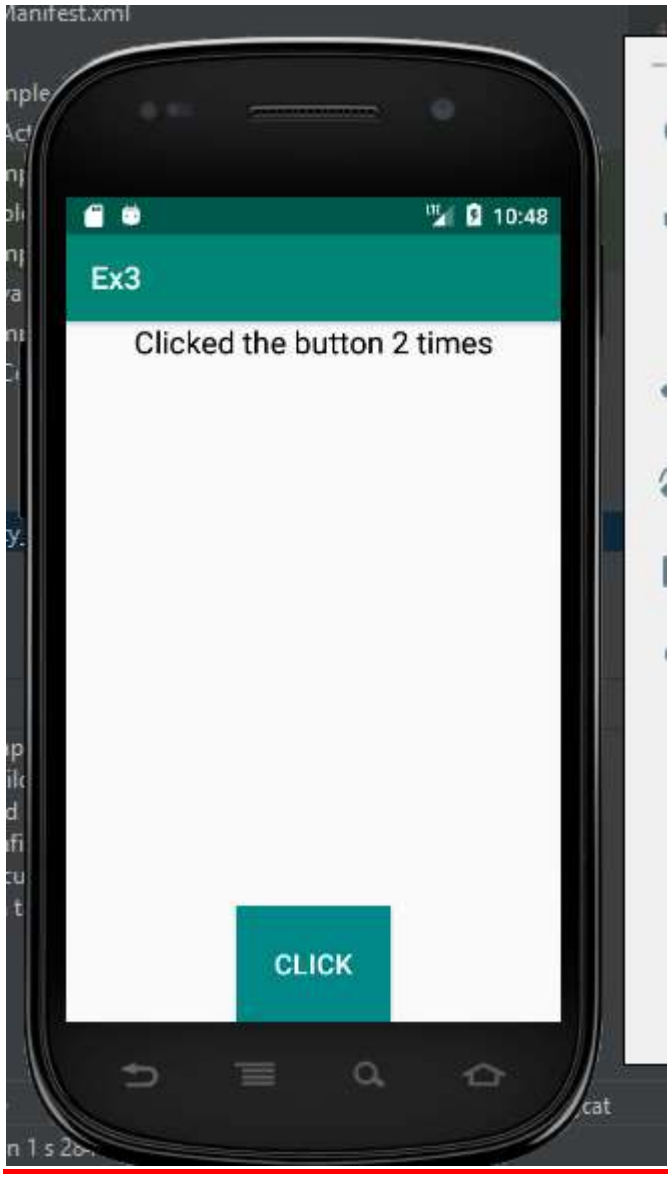
```
package com.example.ex3;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    TextView mTextView;
    Button mButton;
    int mButtonClickCount = 0;

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

        // Instantiate textView and button objects
        mTextView = (TextView) findViewById(R.id.text_view);
        mButton = (Button) findViewById(R.id.button);

        // Action taken when the button is clicked
        mButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                mButtonClickCount++;
                mTextView.setText("Clicked the button " + mButtonClickCount + " times");
            }
        });

    }
}
```

Copy the following code to 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:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"

    tools:context=".MainActivity">

    <TextView
        android:id="@+id/text_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:textSize="20sp"
        android:textColor="#FF000000"
        android:text="Clicked the button 0 times" />
    <Button
        android:id="@+id/button"
        android:layout_width="100dp"
        android:layout_height="75dp"
        android:text="Click"
        android:textSize="18sp"
        android:textColor="#FFFFFFFF"
        android:background="#FF008888"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"/>
</RelativeLayout>
```

Build the code. Check for errors

Run the app & observe the output



Layouts

Introduction

In Android, Layout defines how the structure of the screen will look. It usually contains various UI components such as buttons, menus, text areas etc. These UI components in a layout are called '**layout elements'**.

Usual practice is following. First, define a static layout in XML. Then implement the dynamic behaviours programmatically.

Of course one can implement the layouts entirely in a programmable way.

Defining Layouts Using XML

In XML, a layout should have only ONE Root element. Under that you can add multiple UI components. Then save the file with *.xml* extension and keep it in app/src/main/res/layout directory. (You may store in a different directory as well). Usually while creating a project, Android Studio usually creates some default layouts for the developer. Following is a typical XML code for a layout.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout                                          ←Root element
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical" >                    ←Orientation of the layout

   <Button android:id="@+id/button“                    ←UI Component (button)
        android:layout_width="wrap_content“            ←Dimension
        android:layout_height="wrap_content“           ←Dimension
        android:text=“This is a Button" />

   <TextView android:id="@+id/text_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text=“This is a text view" />

</LinearLayout>
```

Inflating the XML Layout

We need to Load the XML file after the Activity is created. This process is called 'inflating the layout'. It is done by using the method '**setContentView'**, passing the layout id. Following code shows inflating of the layout XML file **activity_main.xml** located in the **'layout/'** directory.

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

      ……
}
```

IDs of the Layout Elements in an XML Layout

Every UI component in an XML layout can be assigned an ID, to identify that component.

Example:

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

     ….                    />
```

Here 'text_view' is the id of the 'TextView' element in the layout.

Following is the way to create the instance of a TextView component in the source code, as following.

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

Other than ID, there are other parameters in the XML Layout for a view. Followings code shows those parameters.

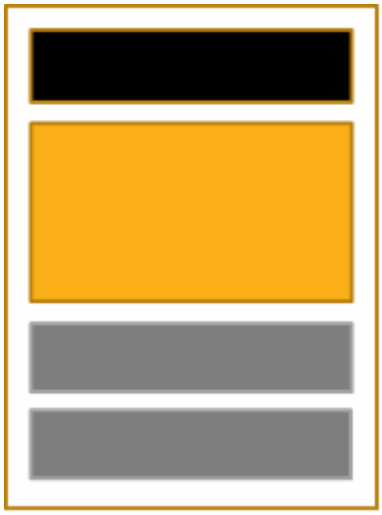
```
<TextView
  android:id="@+id/text_view"                  → ID
  android:layout_width="match_parent"          → Width
  android:layout_height="wrap_content"         → Height
  android:layout_margin="1dp"                  → Margin
  android:layout_alignParentLeft="true"        → Alignment related parameter
  android:layout_below="@+id/button"/>         → Parameter related to placement
```

Commonly Used Layouts

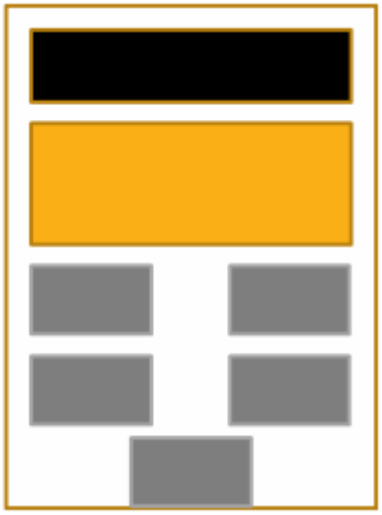
Followings are the commonly used layouts in Android.

- **Linear Layout**: Here the layout elements are stacked vertically or horizontally.
- **Relative Layout**: Here the layout elements are placed relative to each other. It can be used if the layout elements should be placed in custom fashion.
- **Web view**: It is used to display HTML pages.

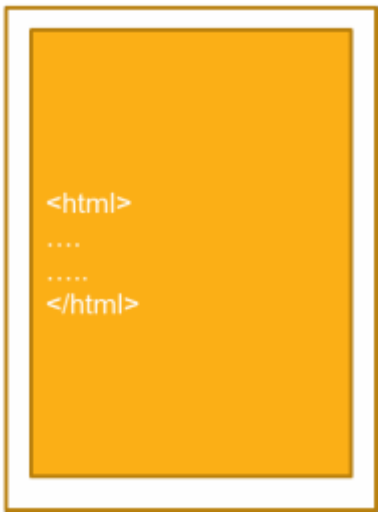
Next figure shows various layout skeletons.



Linear Layout



Relative Layout



Web view

Fig Commonly used Android Layouts

Views

Introduction

View is a basic building block of the user interface in Android. View is the Android UI element which are visible and also has mechanisms to handle user interactions.

Followings are the examples of Views:

- Button
- TextView
- EditText
- ImageView
- KeyboardView
- ….

All these views are derived from the base class 'View'. We will go over TextView and ImageView in the subsequent sections.

TextView

'TextView' is used to display plain text on the screen. You can refer to the 'Hello World' application, where we were displaying the text `"Hello World ! I am learning Android."` in a TextView. Following is the XML code for a TextView.

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

You can see the width and height of the TextView are marked as 'wrap_content'. It indicates that the width and height of the TextView will be adjusted depending upon the width and hight required to display the text. If any of these dimension were marked as 'match_parent' then that dimension would have been equal to corresponding dimension of the parent window minus the margin. The text to be displayed initially can be mentioned through the tag 'android:text'. It can be changed later programmatically using the method 'setText' as shown in the following code segment.

```
… … …

import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
    TextView textView;
```

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

        textView = (TextView) findViewById(R.id.text_view);
        textView.setText("Hello World ! I am learning Android.");
    }
}
```

You can also set other attributes like font size, margins, paddings, text colour etc in the XML as programmatically.

'**EditText'** is a special type of 'TextView' (it is derived from 'TextView' class, which can take text entered by the user.

ImageView

ImageView is used for displaying an Image. Following is the XML code snippet for an ImageView.

```
<ImageView
    android:id="@+id/image_view"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:scaleType="fitCenter" />
```

Apart from width and height, you can see here another XML tag 'android:scaleType'. It is used for scalling and fitting the image with the ImageView. 'fitCenter' is used for fitting the image in the ImageView maintaining the aspect ratio of the image.

Following is the Java code snippet for displaying an image from the 'drawable' folder. It displays 'tajmahal.jpg' from the *app/src/main/res/drawable* folder. Please note that while referring to images in 'drawable' folder, you don't have to mention their extension (e.g. we are referring to the image resource here as `R.drawable.bits,` omitting the .jpg extension).

```
… … …
import android.graphics.BitmapFactory;
import android.net.Uri;
import android.widget.ImageView;


public class MainActivity extends AppCompatActivity
    { ImageView imageView;

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

```
        // Display the image from Drawable
        imageView = (ImageView) findViewById(R.id.image_view);

        //(assuming tajmahal.jpg is in app/src/main/res/drawable folder
        imageView.setImageResource(R.drawable.bits);

    }
}
```

Buttons

Button is a view, when clicked, an action is taken.

Following is the XML code for a Button.

```
<Button android:id="@+id/button"
    android:layout_width="100dp"
    android:layout_height="75dp"
    android:text="Click"
    android:textSize="18sp"
    android:textColor="#FFFFFFFF"
    android:background="#FF008888"
    android:layout_alignParentBottom="true"
    android:layout_centerHorizontal="true"/>
```

Let us try to understand the above XML code for the button. Like any other UI element, we should give an id to it. We have to provide the height and width as well. The text to be displayed on the button can be provided using XML tag 'android:text'. We can provide the color of the text also using XML tag
'android:textColor'. Please note that the color can be provided as AlphaRGB value, which is a string starting with # followed by Alpha, R, G, B values, each of one byte. You can also define the colors in colors.xml and use them here. Similarly background color can also be specified using the XML tag 'android:background'. In fact a drawable resource can also be specified as the button background. The positioning of the buttons can be provided as well. Here we have placed the button at the bottom of the screen and at the center horizontally.

Following is the Java code snippet for the Button.

```
… … ….

Button mButton;
… … …

@Override
protected void onCreate(Bundle savedInstanceState) {
    … … …
    mButton = (Button) findViewById(R.id.button);

    // Action taken when the button is clicked
    mButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            // TODO : Code the Action to be taken

        }
    });
}
```

As any other UI component, we can instantiate an object of the Button by using '**findViewById'**.

Then we have to define the action to be taken, when the button is clicked. It is done by defining a '**View.OnClickListener'** object and passing it to '**setOnClickListener'** method of the button object.
Inside the 'View.OnClickListener' object, we have to implement the method '**onClick'**, where we have to define the action to be taken when the button is clicked.

Please refer to 'ButtonTest' project for getting hands-on experience on buttons in Android.