Unit -2
Introduction to Android Programming   [4 Hrs]

Android Platform

Android is a mobile operating system currently developed by Google, based on the Linux kernel and designed primarily for touchscreen mobile devices such as smartphones and tablets. And as we said before, Android offers a unified approach to application development for mobile devices.

Android is an open-source operating system named Android. Google has made the code for all the low-level "stuff" as well as the needed middleware to power and use an electronic device, and gave Android freely to anyone who wants to write code and build the operating system from it. There is even a full application framework included, so third-party apps can be built and installed, then made available for the user to run as they like.

The "proper" name for this is the Android Open Source Project, and this is what people mean when they say things like Android is open and free. Android, in this iteration, is free for anyone to use as they like.

History of Android

The history and versions of android are interesting to know. The code names of android ranges from A to P currently, such as Aestro, Blender, Cupcake, Donut, Eclair, Froyo, Gingerbread, Honeycomb, Ice Cream Sandwich, Jelly Bean, KitKat, Lollipop, Marhmallow, Nougat, Oreo and Pie. Let's understand the android history in a sequence.

- Initially, **Andy Rubin** founded Android Incorporation in Palo Alto, California, United States in October, 2003.
- In 17th August 2005, Google acquired android Incorporation. Since then, it is in the subsidiary of Google Incorporation.
- The key employees of Android Incorporation are **Andy Rubin**, **Rich Miner**, **Chris White** and **Nick Sears**.
- Originally intended for camera but shifted to smart phones later because of low market for camera only.
- Android is the nick name of Andy Rubin given by co-workers because of his love to robots.
- In 2007, Google announces the development of android OS.
- In 2008, HTC launched the first android mobile.

Environment Setup

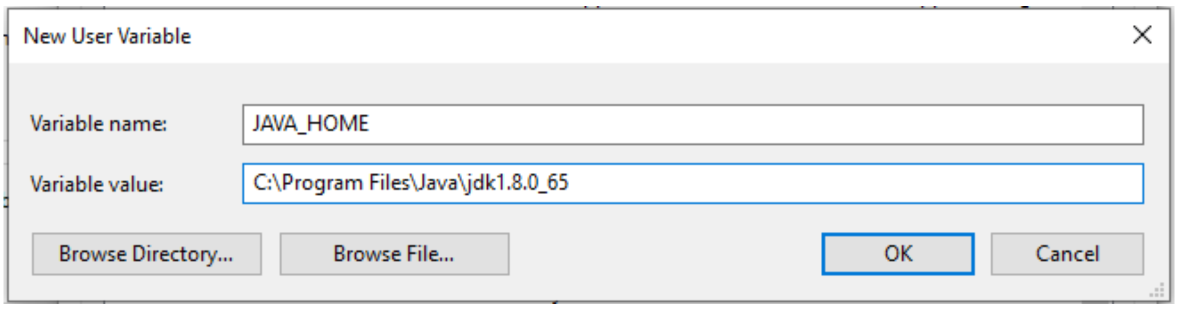
In order to write an Android application, we are going to need a development environment. **Google has made a very useful tool for all Android Developers, the Android Studio**. Android Studio is the official IDE for Android development, and with a single download includes everything you need to begin developing Android apps.

Included in the download kit, are the Software Development Kit (SDK), with all the Android libraries we may need, and the infrastructure to download the many Android emulator instances, so that we can initially run our application, without needing a real device. **So, we are going to download and install Android Studio.**

Step 1 - First we have to have installed the **Java Development Kit (JDK)** from Oracle. If you do not, please you should download the latest JDK from the https://www.oracle.com/. After downloading JDK, please install it in your system.

Step 2 – After installing JDK, now it's the time for you to setup environment variables. For this procedure, right click **This PC (My Computer) -> Properties -> Advanced System Settings -> Environment Variables -> New**. Now setup new environment variable as follows:

Figure 2-1. Setting up environment variable


New User Variable

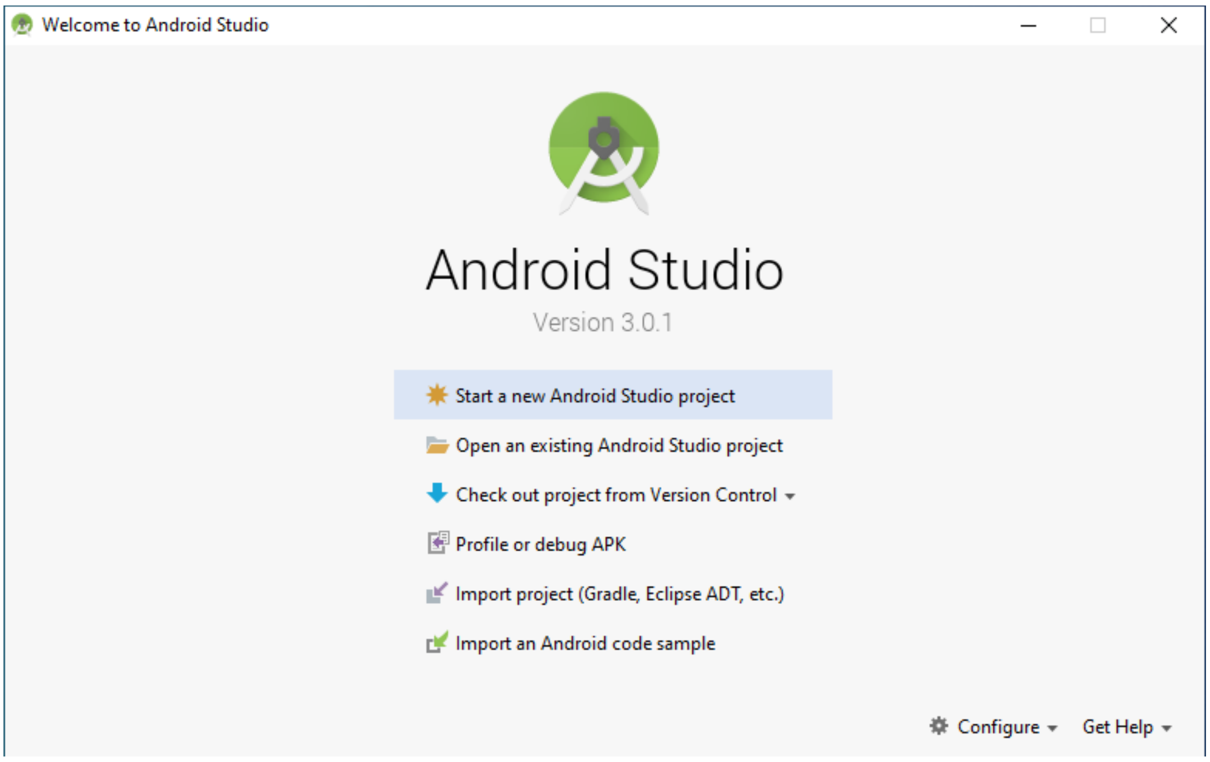
Variable name: JAVA_HOME

Variable value: C:\Program Files\Java\jdk1.8.0_65

Here, variable value must be the path where JDK is installed. Now click OK.

Step 3 – Now download the latest android studio and install in your system. After installation following window appears in your computer screen.

Figure 2-2. First look of android studio after installation

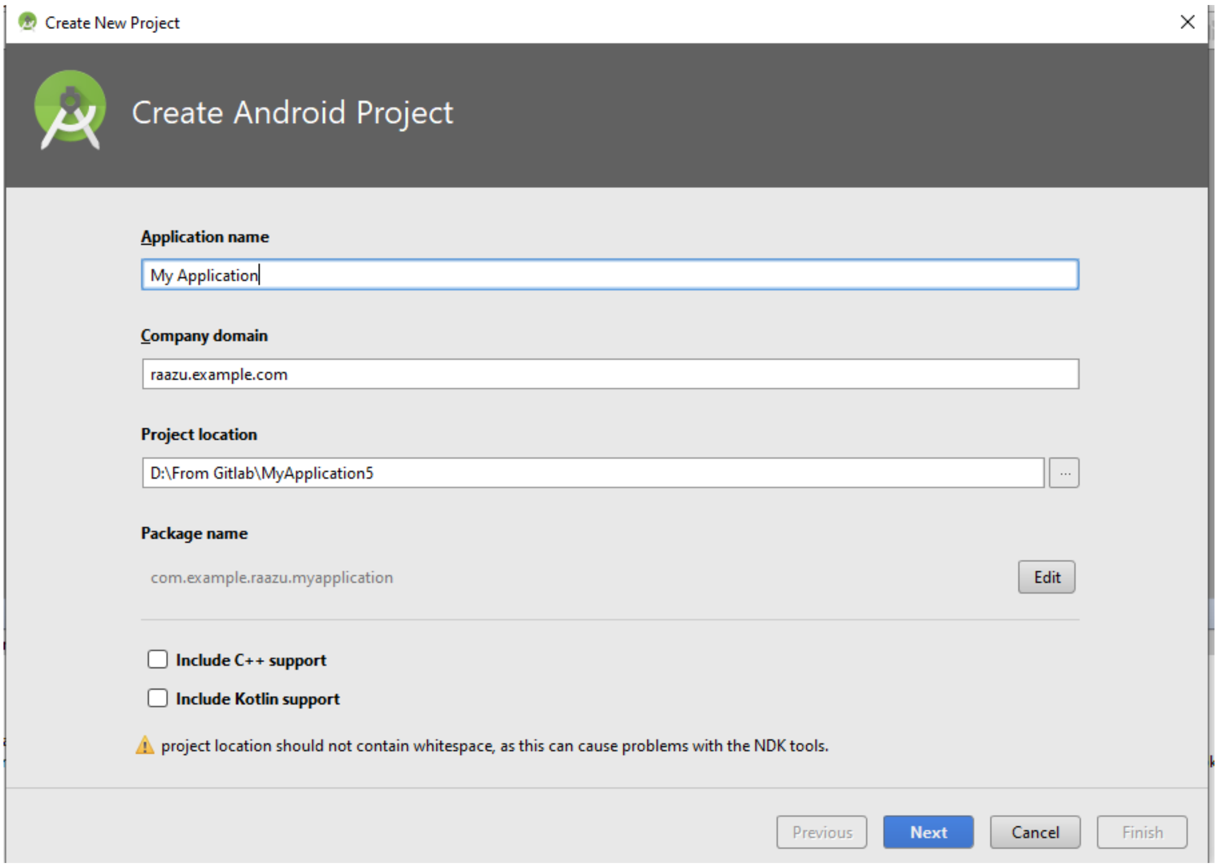

Welcome to Android Studio
Android Studio
Version 3.0.1
Start a new Android Studio project
Open an existing Android Studio project
Check out project from Version Control
Profile or debug APK
Import project (Gradle, Eclipse ADT, etc.)
Import an Android code sample
Configure
Get Help

Congratulations! Android Studio has been successfully installed in your computer system.

<u>Creating an Android Project</u>

The first step is to create an Android project. An Android project contains the files that make up an application. To create a new project, open Android Studio and choose **File -> New -> New Project** to open the new application wizard.

Figure 2-3. Creating a new Application



Create New Project

Create Android Project

Application name

My Application

Company domain

raazu.example.com

Project location

D:\From Gitlab\MyApplication5

Package name

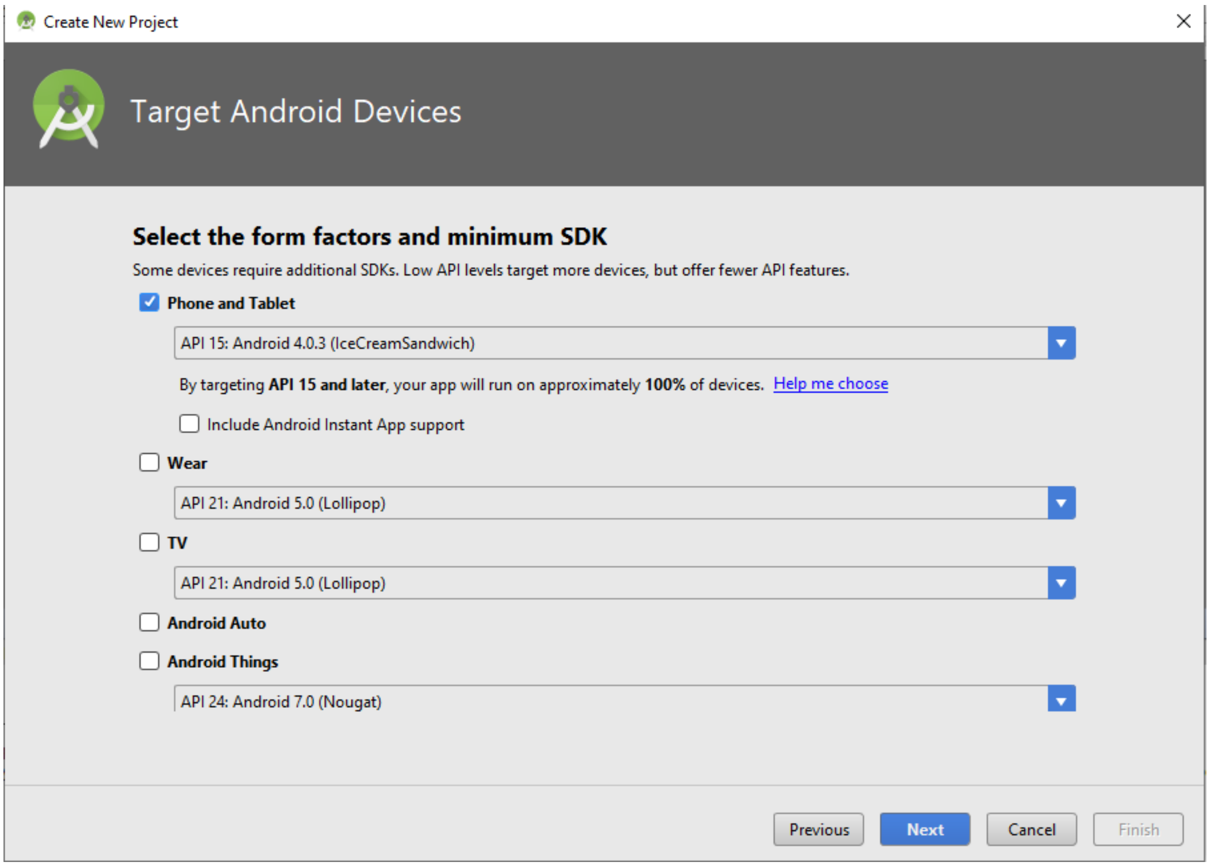
com.example.raazu.myapplication

- [ ] Include C++ support
- [ ] Include Kotlin support

⚠ project location should not contain whitespace, as this can cause problems with the NDK tools.

Notice that the package name you entered uses a "reverse DNS" convention in which the domain name of your organization is reversed and suffixed with further identifiers. This convention keeps package names unique and distinguishes applications from each other on a device and on Google Play.
Click Next.

Figure 2-4. Setup device and SDK


Create New Project

Target Android Devices

Select the form factors and minimum SDK

Some devices require additional SDKs. Low API levels target more devices, but offer fewer API features.

[✓] **Phone and Tablet**

API 15: Android 4.0.3 (IceCreamSandwich)

By targeting **API 15 and later**, your app will run on approximately **100%** of devices. Help me choose

[ ] Include Android Instant App support

[ ] **Wear**

API 21: Android 5.0 (Lollipop)

[ ] **TV**

API 21: Android 5.0 (Lollipop)

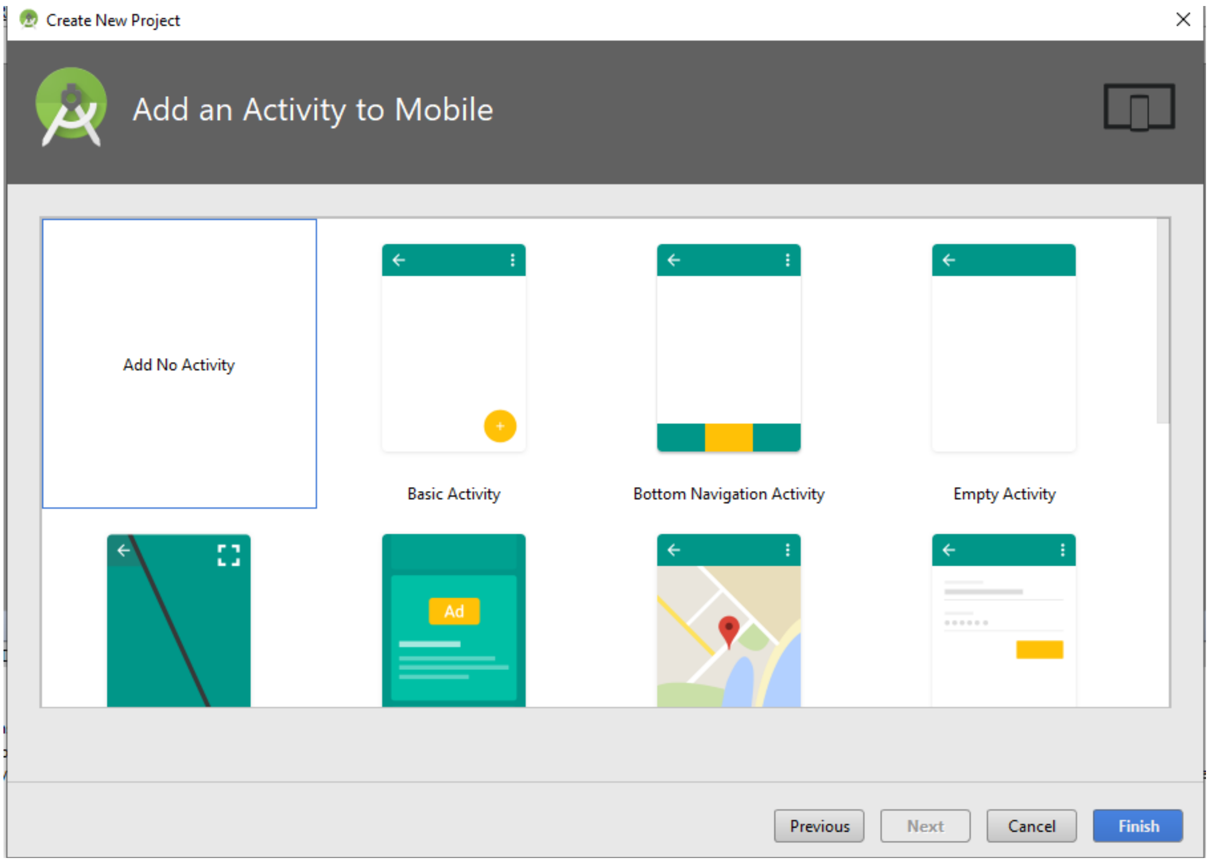
[ ] **Android Auto**

[ ] **Android Things**

API 24: Android 7.0 (Nougat)

Here we can select different devices and minimum SDK for or project. This setup configures your application to work with different versions of Android. We are using default settings for our project.
Android updates its tools several times a year, so your wizard may look slightly different from what we are showing you. This is usually not a problem; the choices to make should stay pretty much the same.
Click Next.

Figure 2-5. Creating a new Activity

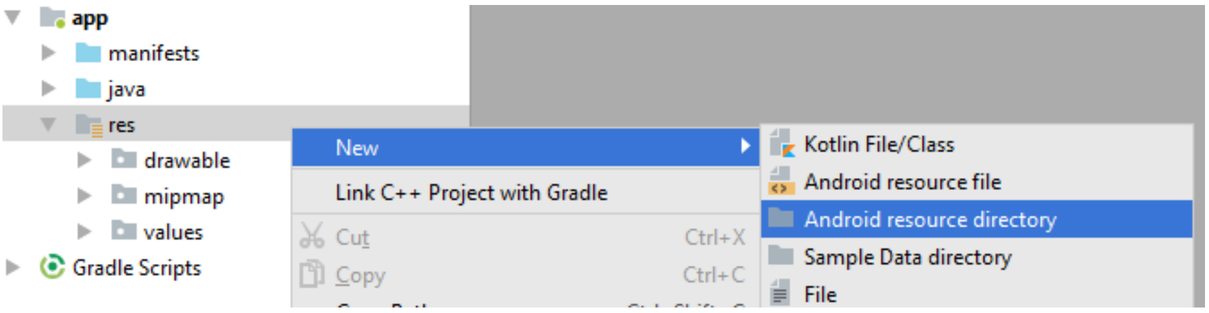


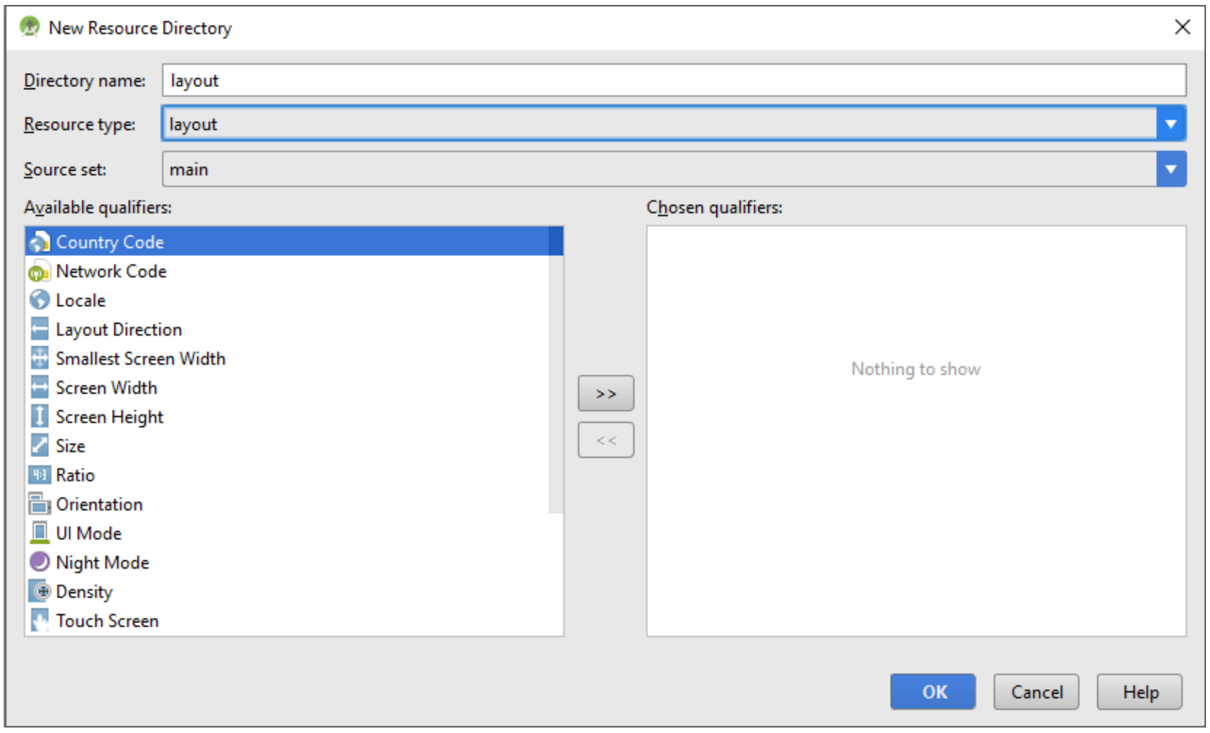
Here we can see different templates for our Application. We can use these templates as per the requirement. Since we are developing our first application, we start by **adding no activity in our project.**
Click Finish.

Now, our project has just been created!

Laying Out the User Interface

After creating a new project, firstly we design user interface for our project. In Android project we can design user interface using **xml**. In Android Studio, all xml files, images, audio, video, animations, colours etc. that are used for designing the interface are kept inside **res (Resources)** folder.
We need to create sub-folder named as **layout** to store different layout resource files. Layout resource files are the pages that we design for our application.
Figure 2-6. Creating layout directory





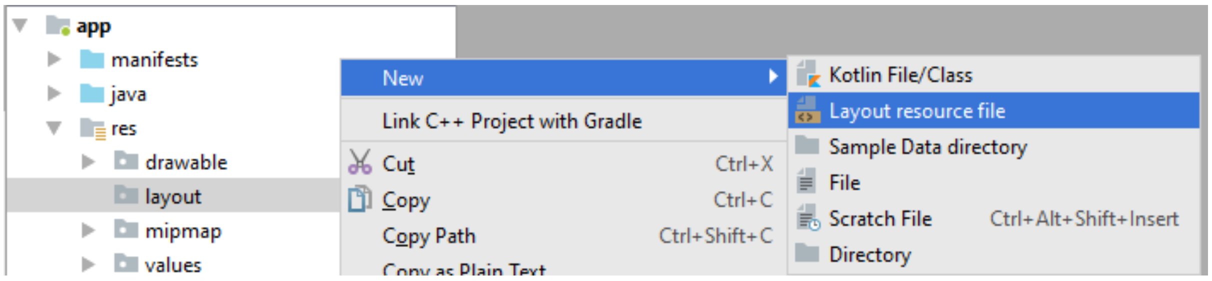
Directory name: layout
Resource type: layout
Source set: main

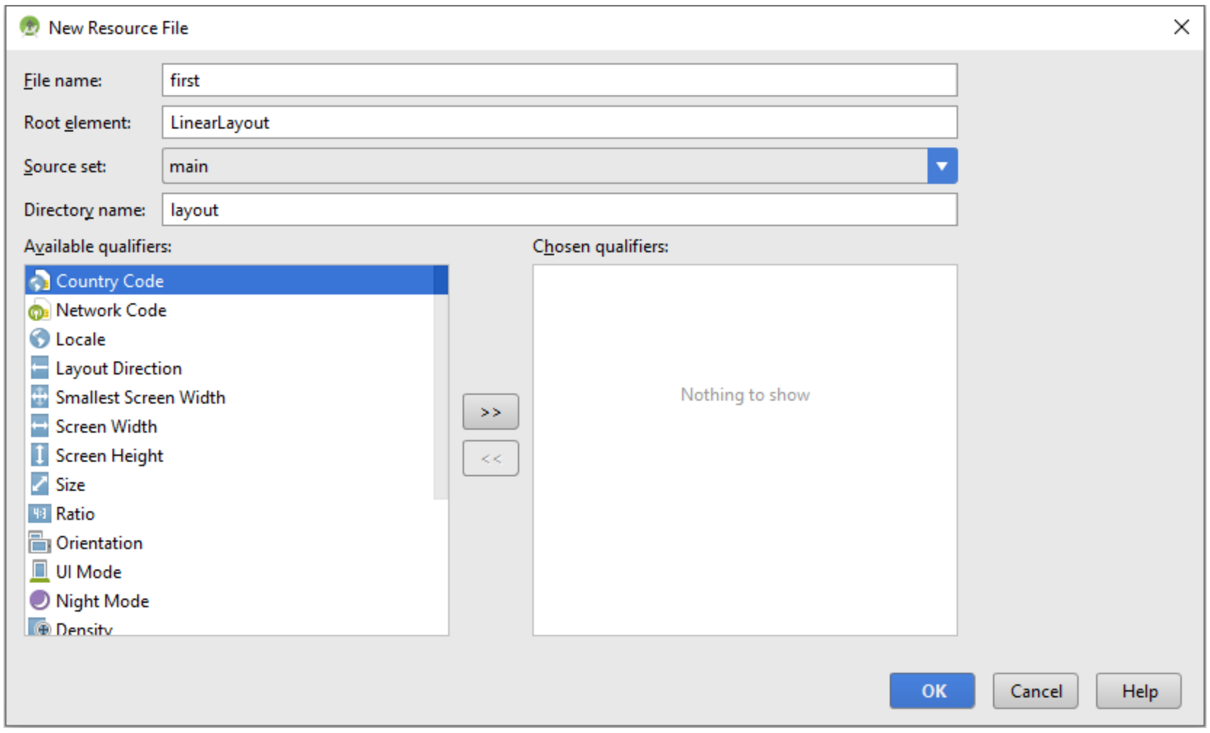
Now select Resource type **layout** and Click OK.

Creating a new Layout Resource File

Now we are going to create a file where we add different design components like image, button, label etc. **In this example we are creating a new layout file containing a label with a text "Hello World!".**

Figure 2-7. Adding new layout resource file





Now we need to write name of file. In this example I am giving name as first. For designing a page, we can use different types of layout like Linear, Relative, Constraint etc. In this case, I am using Linear Layout. We will discuss about different types of layout in coming chapters. Click OK.

Note: Name of all the files under resource folder must be in lower case.

Adding Label with text “Hello World!”

Following code will add Label under Linear Layout in our page.

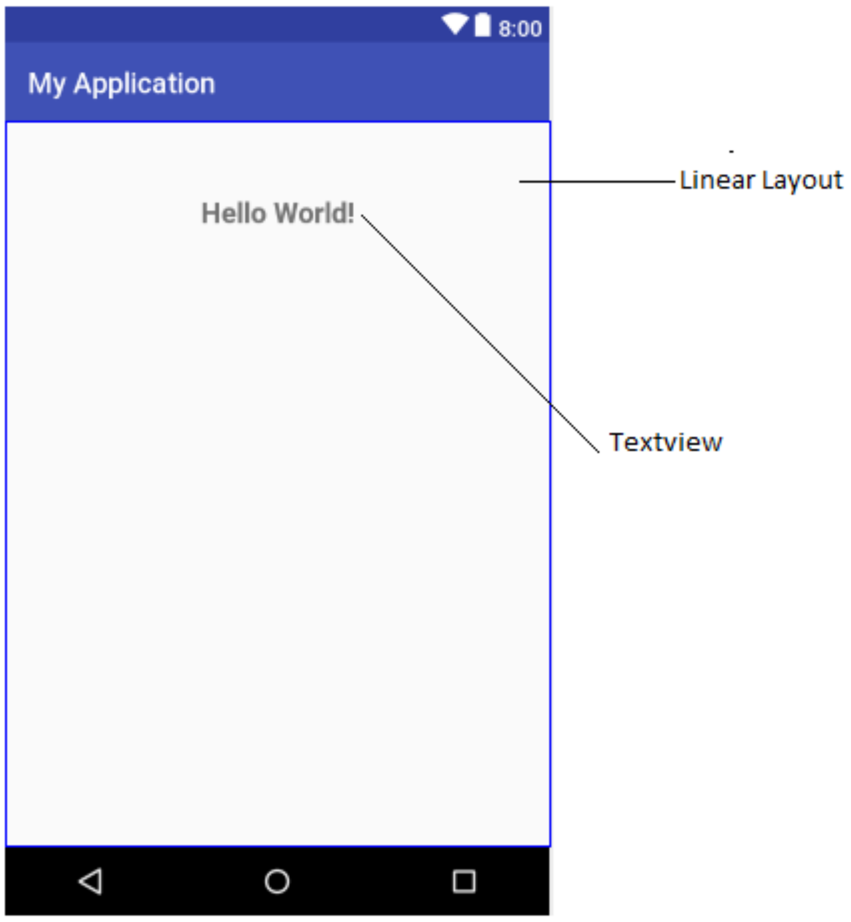
first.xml

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

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        android:gravity="center"
        android:textSize="20sp"
        android:textStyle="bold"
        android:layout_marginTop="50dp"
        />

</LinearLayout>
```

Our activity(page) layout defines two widgets: a **LinearLayout** and a **TextView**.

Widgets are the building blocks you use to compose a user interface. A widget can show text or graphics, interact with the user, or arrange other widgets on the screen. Buttons, text input controls, and check boxes are all types of widgets.

Figure 2-8. Widgets as seen on screen

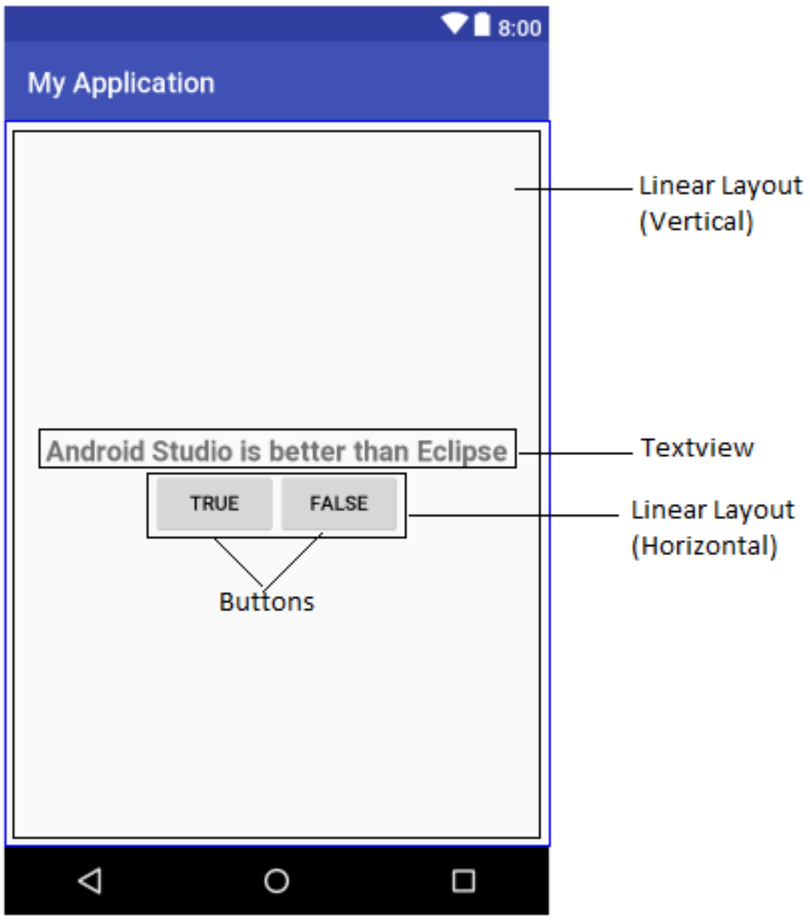


The View Hierarchy

Your widgets exist in a hierarchy of View objects called the view hierarchy. To demonstrate view hierarchy now I am going to create another layout file named as "second.xml" which contains the following widgets:

- a vertical **LinearLayout**
- a **TextView**
- a horizontal **LinearLayout**
- two **Buttons**

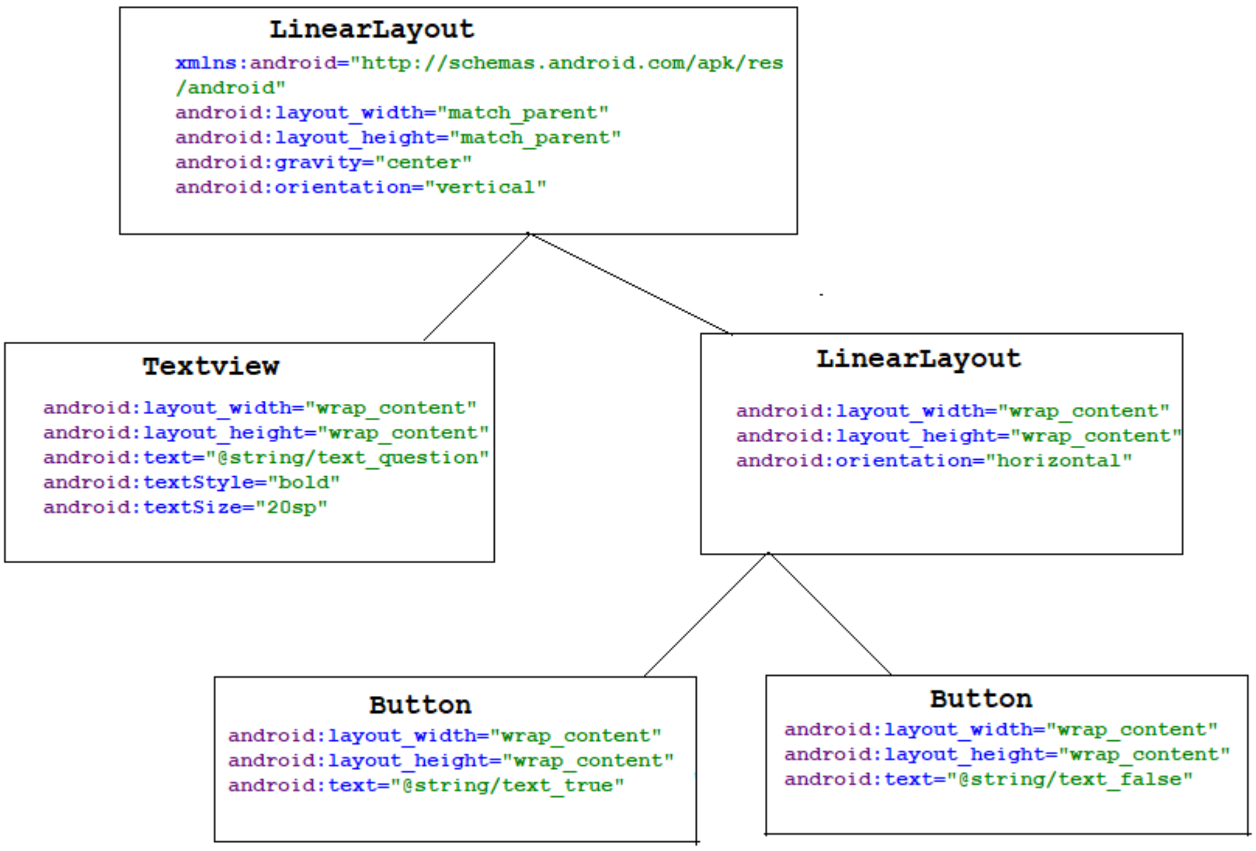
Figure 2-9. Arrangement of widgets to demonstrate view hierarchy



second.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/text_question"
        android:textStyle="bold"
        android:textSize="20sp"/>
    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal">
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/text_true"/>
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/text_false"/>
        </LinearLayout>
</LinearLayout>
```

Figure 2-10. Hierarchical layout of widgets and attributes


LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:gravity="center"
android:orientation="vertical"
Textview
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/text_question"
android:textStyle="bold"
android:textSize="20sp"
LinearLayout
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:orientation="horizontal"
Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/text_true"
Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/text_false"

The root element of this layout's view hierarchy is a **LinearLayout**. As the root element, the **LinearLayout** must specify the Android resource XML namespace at *http://schemas.android.com/ apk/res/android*.
LinearLayout inherits from a subclass of View named **ViewGroup**. A **ViewGroup** is a widget that contains and arranges other widgets. You use a **LinearLayout** when you want widgets arranged in a single column or row. Other **ViewGroup** subclasses are *FrameLayout, TableLayout, and RelativeLayout.*
When a widget is contained by a **ViewGroup**, that widget is said to be a child of the **ViewGroup**. The root **LinearLayout** has two children: **a TextView and another LinearLayout**. The child **LinearLayout** has two **Button** children of its own.

Widgets Attributes

Let's discuss some of the attributes that you have used to configure your widgets:

android:layout_width and android:layout_height

The **android:layout_width** and **android:layout_height** attributes are required for almost every type of widget. They are typically set to either ***match_parent*** or ***wrap_content***:

- match_parent   view will be as big as its parent

- wrap_content view will be as big as its contents require

(You may see fill_parent in some places. This deprecated value is equivalent to match_parent.)

android:orientation

The android:orientation attribute on the two LinearLayout widgets determines whether their children will appear vertically or horizontally. The root LinearLayout is vertical; its child LinearLayout is horizontal.

android:text

The **TextView** and **Button** widgets have *android:text* attributes. This attribute tells the widget what text to display. Notice that the values of these attributes are not literal strings. They are references to *string resources.*
A string resource is a string that lives in a separate XML file called a strings file. You can give a widget a hard-coded string, like android:text="True", but it is usually not a good idea. Placing strings into a separate file and then referencing them is better.

Creating String Resources

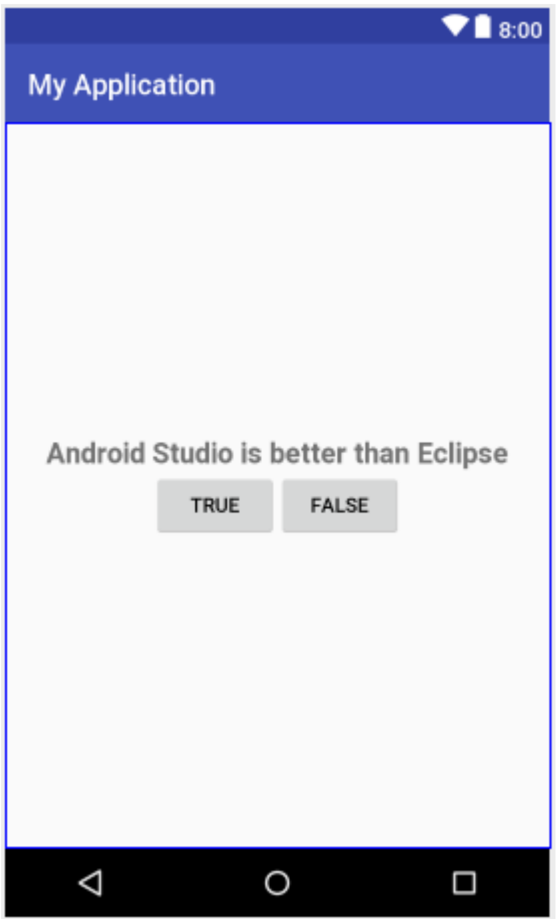
Since string resources we are referencing in second.xml do not exist yet. Now, we need to fix that by adding strings to **string.xml** file. Every project includes a default strings file named strings.xml. In the package explorer, find the res/values directory, reveal its contents, and open strings.xml.
Now add following lines of code in **strings.xml** file.

```
<resources>
    <string name="app_name">My Application</string>
    <string name="text_question">Android Studio is better than 
Eclipse</string>
    <string name="text_true">True</string>
    <string name="text_false">False</string>
</resources>
```

Previewing the Layout

Your layout is now complete, and you can preview the layout in the preview section of Android Studio. If your application is error free, following output will be displayed:

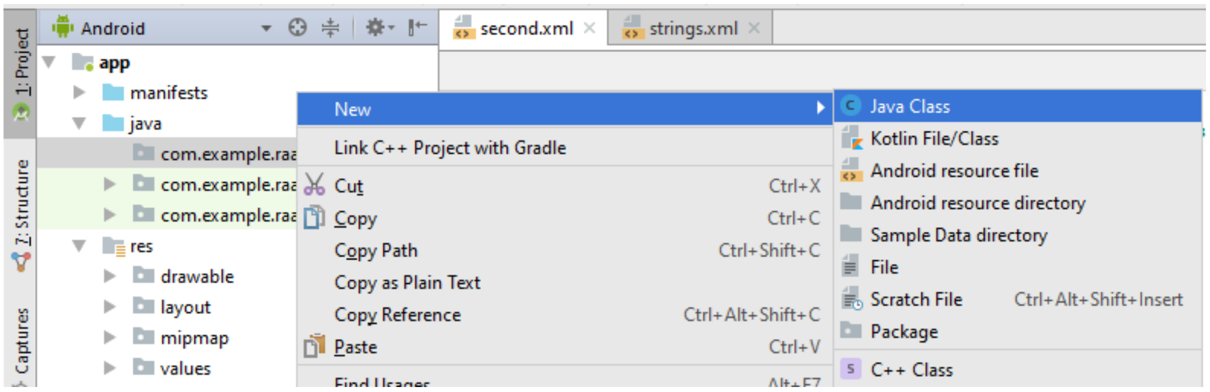
Figure 2-11. Layout Preview



Creating a New Class

Since for creating a page that can be displayed in an application we need layout resource file for designing the page and a class file written using object oriented programming language like Java, Kotlin, C# etc. that extends Activity class. In simple words, we can say that for creating pages we need class file that extends Activity class. In a class file we link layout file which we have designed and add events and other functionality to different widgets and views that we have created in layout resource file. For creating class file we are using **Java** programming language in our example.

Figure 2-12. Creating a new class file



Add following lines of code in newly created java file:

<u>Second.java</u>

```
package com.example.raazu.myapplication;

import android.app.Activity;
import android.os.Bundle;

public class Second extends Activity{
    @Override
    public void onCreate(Bundle b){
        super.onCreate(b);
        setContentView(R.layout.second);
    }
}
```

This file has one Activity methods: **onCreate(Bundle).**
The **onCreate(Bundle)** method is called when an instance of the activity subclass is created. When an activity is created, it needs a user interface to manage. To get the activity its user interface, you call the following Activity method:

public void setContentView(int layoutResID)

This method inflates a layout and puts it on screen. When a layout is inflated, each widget in the layout file is instantiated as defined by its attributes. You specify which layout to inflate by passing in the layout's resource ID.

<u>Resource and Resource ID's</u>

A layout is a resource. A resource is a piece of your application that is not code – things like image files, audio files, and XML files.
Resources for your project live in a subdirectory of the res directory. In the package explorer, you can see that second.xml lives in res/layout/. Your strings file, which contains string resources, lives in res/values/.
To access a resource in code, you use its resource ID. The resource ID for your layout is R.layout.second. Your strings also have resource IDs. You have not yet referred to a string in code, but if you did, it would look like this:

setTitle(R.string.app_name);

You can get ID's of your resources inside **R.java** file.

```
public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int ic_launcher=0x7f020000;
    }
    public static final class id {
        public static final int menu_settings=0x7f070003;
    }
    public static final class layout {
        public static final int activity_quiz=0x7f030000;
    }
    public static final class menu {
       public static final int activity_quiz=0x7f060000;
    }
    public static final class string {
        public static final int app_name=0x7f040000;
        public static final int false_button=0x7f040003;
        public static final int menu_settings=0x7f040006;
        public static final int question_text=0x7f040001;
        public static final int true_button=0x7f040002;
    }

    ...
}
```

Setting Up the Project

Now this is a final step to configure before running your application. Add your newly created activity in **AndroidMainfest.xml** file. Following lines of code will add and configure your activity to android manifest.

AndroidManifest.xml

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">

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

    </application>
</manifest>
```

The manifest file describes essential information about your app to the Android build tools, the Android operating system, and Google Play.

Among many other things, the manifest file is required to declare the following:

- The app's package name, which usually matches your code's namespace. The Android build tools use this to determine the location of code entities when building your project. When packaging the app, the build tools replace this value with the application ID from the Gradle build files, which is used as the unique app identifier on the system and on Google Play.
- The components of the app, which include all activities, services, broadcast receivers, and content providers. Each component must define basic properties such as the name of its Kotlin or Java class. It can also declare capabilities such as which device configurations it can handle, and intent filters that describe how the component can be started.
- The permissions that the app needs in order to access protected parts of the system or other apps. It also declares any permissions that other apps must have if they want to access content from this app.
- The hardware and software features the app requires, which affects which devices can install the app from Google Play.

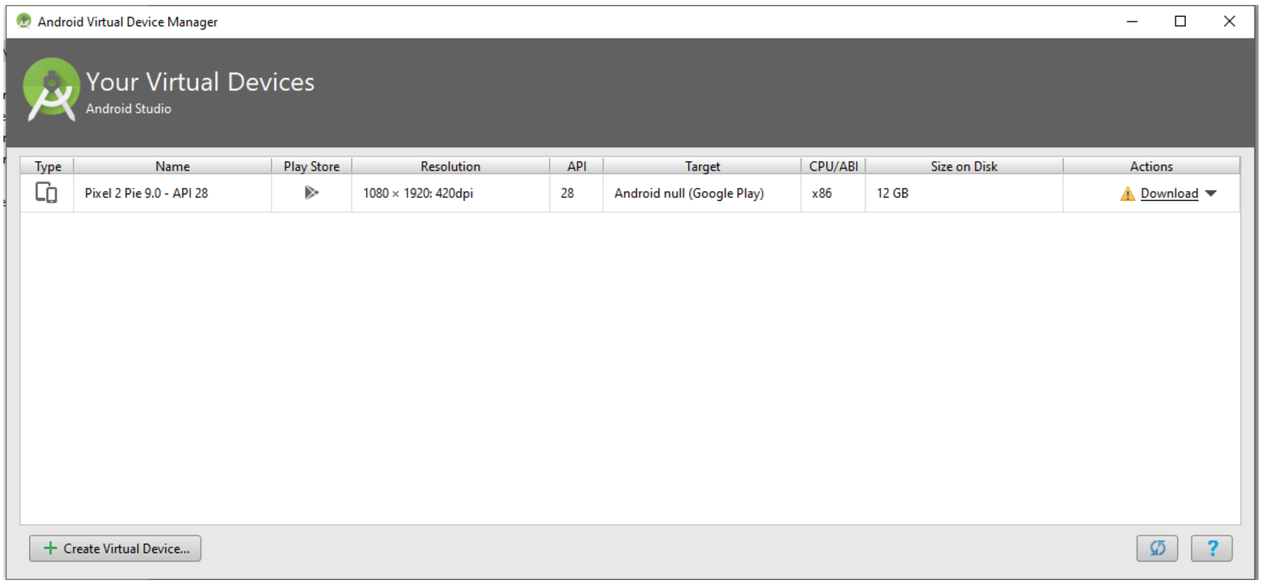
Intent filters

App activities, services, and broadcast receivers are activated by intents. *An **intent** is a message defined by an Intent object that describes an action to perform, including the data to be acted upon, the category of component that should perform the action, and other instructions*.
When an app issues an intent to the system, the system locates an app component that can handle the intent based on intent filter declarations in each app's manifest file. The system launches an instance of the matching component and passes the Intent object to that component. If more than one app can handle the intent, then the user can select which app to use.
An app component can have any number of intent filters (defined with the <intent-filter> element), each one describing a different capability of that component.
If activity is defined with <intent-filter> element in manifest file than that activity will be launched as the first activity of application.

Running on the Emulator

To run an Android application, you need a device – either a hardware device or a virtual device. Virtual devices are powered by the Android emulator, which ships with the developer tools.
An Android Virtual Device (AVD) is a configuration that defines the characteristics of an Android phone, tablet, Wear OS, Android TV, or Automotive OS device that you want to simulate in the Android Emulator. The AVD Manager is an interface you can launch from Android Studio that helps you create and manage AVDs.

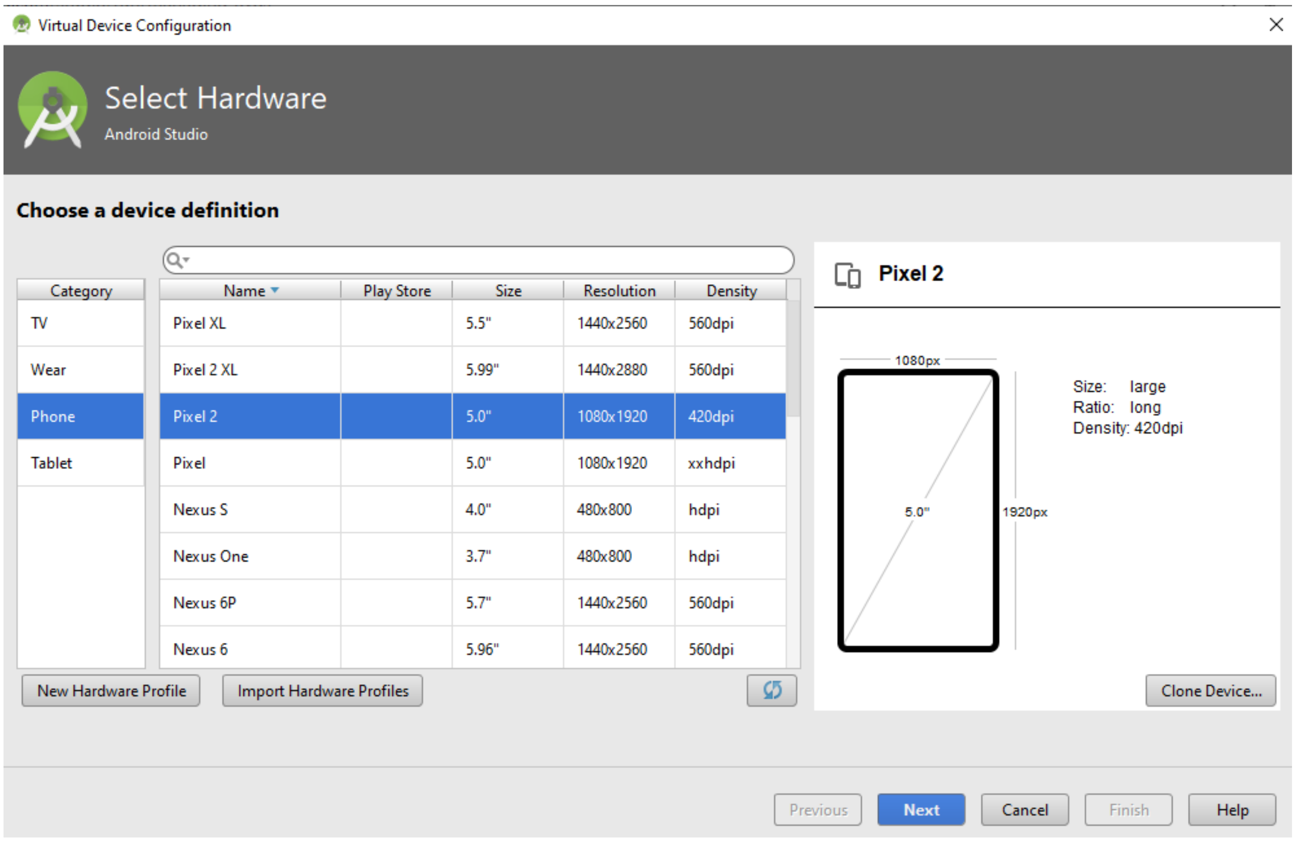
1. To open the AVD Manager, do one of the following:
    - Select **Tools > Android > AVD Manager**.
    - Click **AVD Manager** in the toolbar.



Type	Name	Play Store	Resolution	API	Target	CPU/ABI	Size on Disk	Actions
	Pixel 2 Pie 9.0 - API 28	▶	1080 × 1920: 420dpi	28	Android null (Google Play)	x86	12 GB	Download

Figure 2-13. Opening AVD Manager

2. Click **Create Virtual Device**, at the bottom of the AVD Manager dialog. The Select Hardware page appears.



Category
TV
Wear
Phone
Tablet

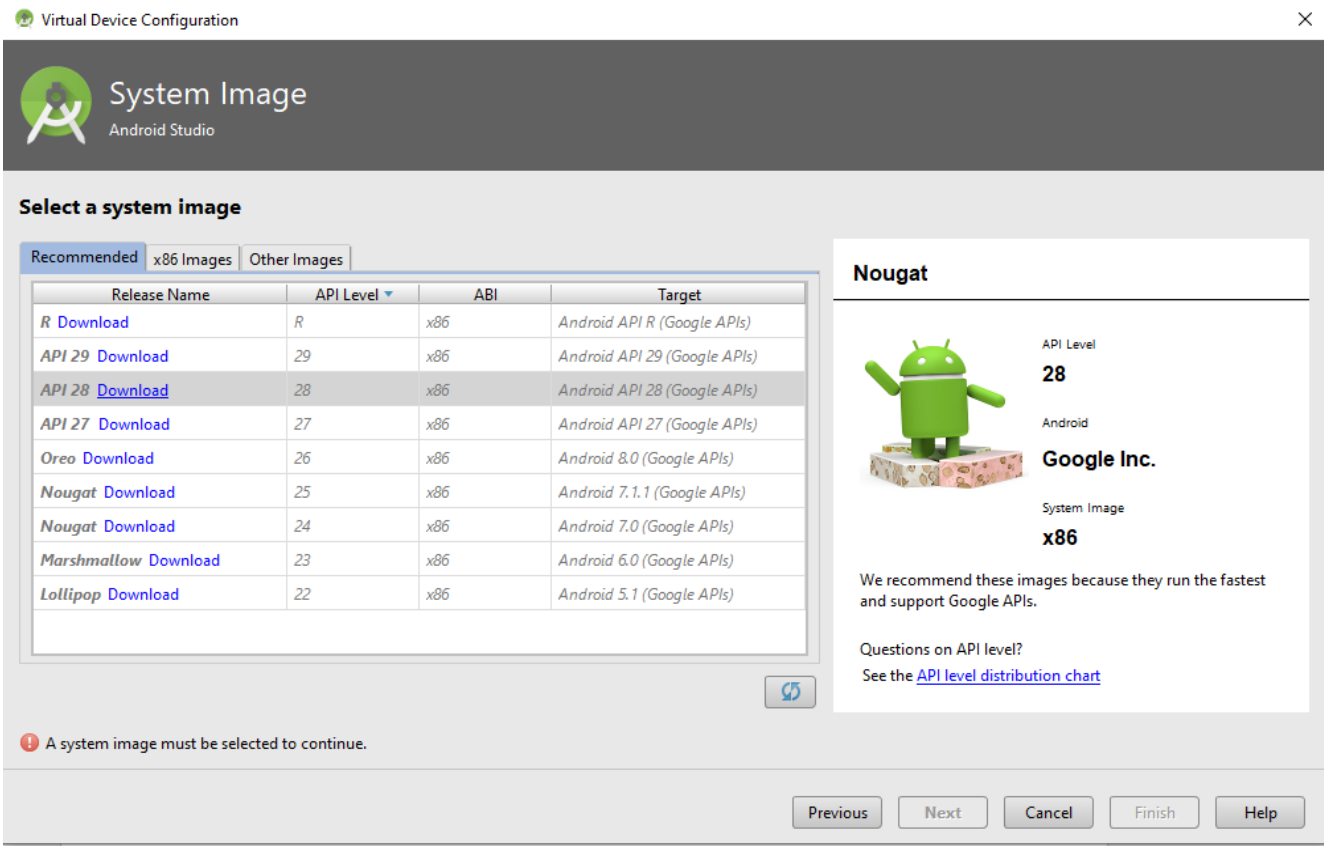
Name	Play Store	Size	Resolution	Density
Pixel XL		5.5"	1440x2560	560dpi
Pixel 2 XL		5.99"	1440x2880	560dpi
Pixel 2		5.0"	1080x1920	420dpi
Pixel		5.0"	1080x1920	xxhdpi
Nexus S		4.0"	480x800	hdpi
Nexus One		3.7"	480x800	hdpi
Nexus 6P		5.7"	1440x2560	560dpi
Nexus 6		5.96"	1440x2560	560dpi

Pixel 2

Size: large
Ratio: long
Density: 420dpi

Figure 2-14. Selecting the Hardware

3. Select a **hardware profile**, and then click Next.
   If you don't see the hardware profile you want, you can create or import a hardware profile.
   The System Image page appears.



Virtual Device Configuration

System Image
Android Studio

Select a system image

Release Name	API Level	ABI	Target
R Download	R	x86	Android API R (Google APIs)
API 29 Download	29	x86	Android API 29 (Google APIs)
API 28 Download	28	x86	Android API 28 (Google APIs)
API 27 Download	27	x86	Android API 27 (Google APIs)
Oreo Download	26	x86	Android 8.0 (Google APIs)
Nougat Download	25	x86	Android 7.1.1 (Google APIs)
Nougat Download	24	x86	Android 7.0 (Google APIs)
Marshmallow Download	23	x86	Android 6.0 (Google APIs)
Lollipop Download	22	x86	Android 5.1 (Google APIs)

Nougat

API Level
28

Android
Google Inc.

System Image
x86

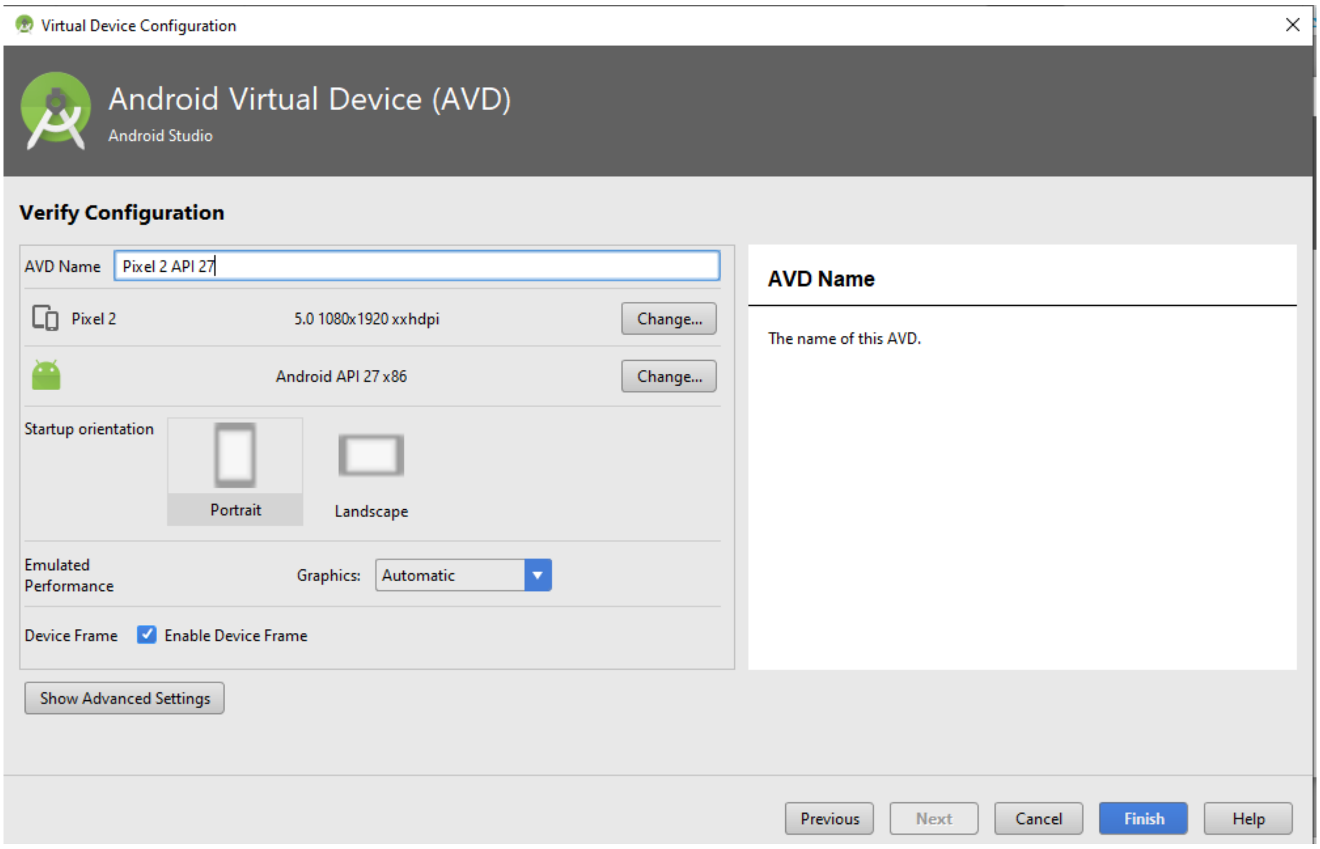
We recommend these images because they run the fastest and support Google APIs.

Questions on API level?
See the API level distribution chart

A system image must be selected to continue.

Figure 2-15. Configuring System Image

4. Select the system image for a particular API level, and then click **Next**.
   - The **Recommended** tab lists recommended system images. The other tabs include a more complete list. The right pane describes the selected system image. x86 images run the fastest in the emulator.
   - If you see **Download** next to the system image, you need to click it to download the system image. You must be connected to the internet to download it.


Virtual Device Configuration

Android Virtual Device (AVD)

Android Studio

Verify Configuration

AVD Name: Pixel 2 API 27

Pixel 2 5.0 1080x1920 xxhdpi Change...

Android API 27 x86 Change...

Startup orientation: Portrait / Landscape

Emulated Performance: Graphics: Automatic

Device Frame: ✓ Enable Device Frame

Show Advanced Settings

AVD Name

The name of this AVD.

Previous | Next | Cancel | Finish | Help

Figure 2-16. Verifying Configuration

5. The **Verify Configuration** page appears.
   Change **AVD properties** as needed, and then click **Finish.**

Now your AVD has been configured successfully.

Following are the steps used to run your application on emulator:

1. Click Run App from toolbar.

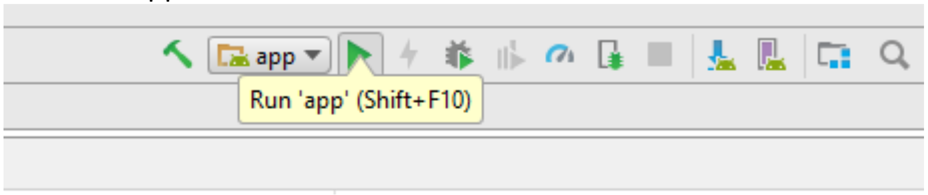
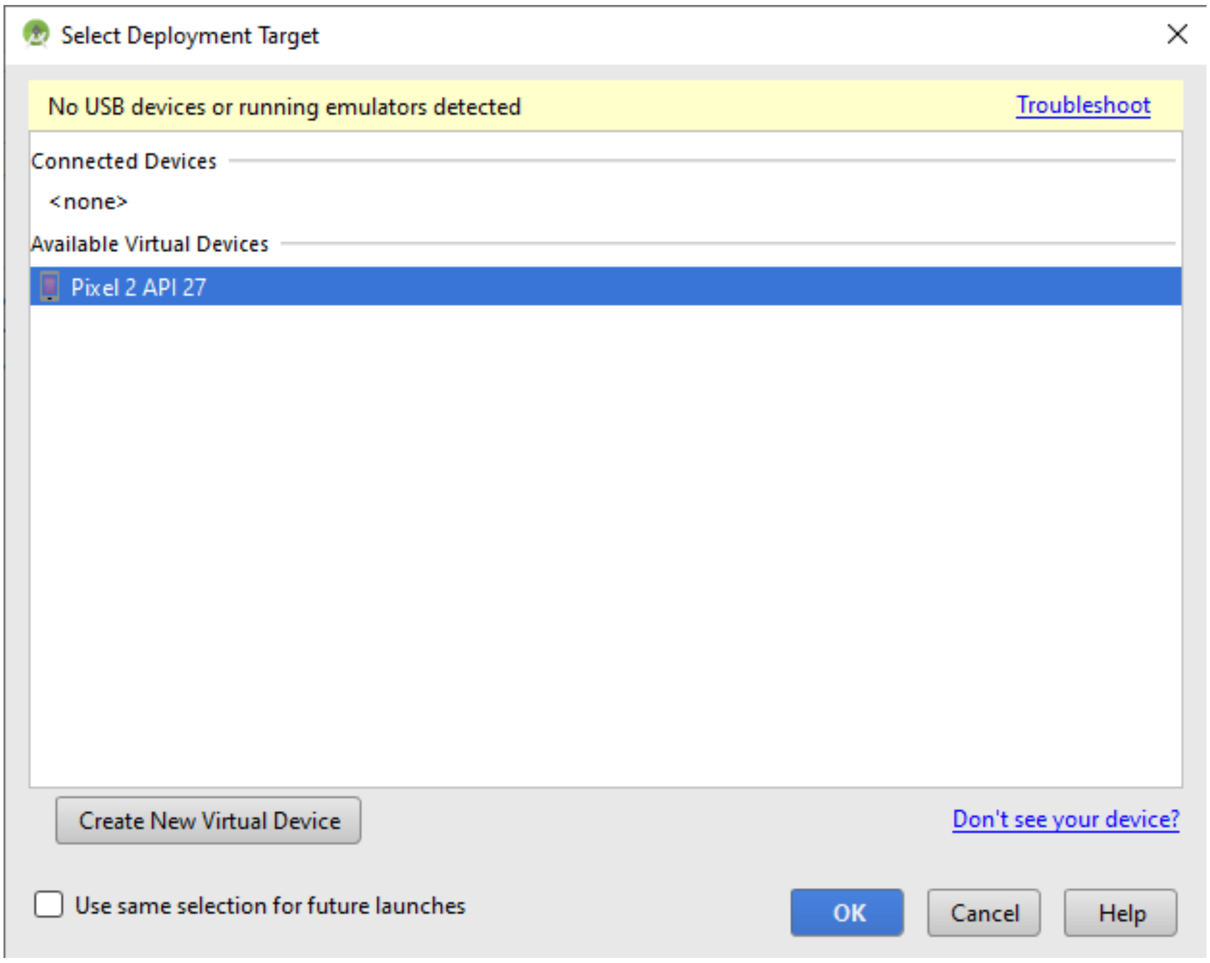


Figure 2-17. Running Application

2. Following screen will appear. Choose your virtual device from the list and press OK button.


Select Deployment Target

No USB devices or running emulators detected

Connected Devices

<none>

Available Virtual Devices

Pixel 2 API 27

Troubleshoot
Create New Virtual Device
Don't see your device?
Use same selection for future launches
OK
Cancel
Help

Figure 2-18. Selecting Deployment Device

3. Now your app will start running and after some time following output will be displayed in your emulator.

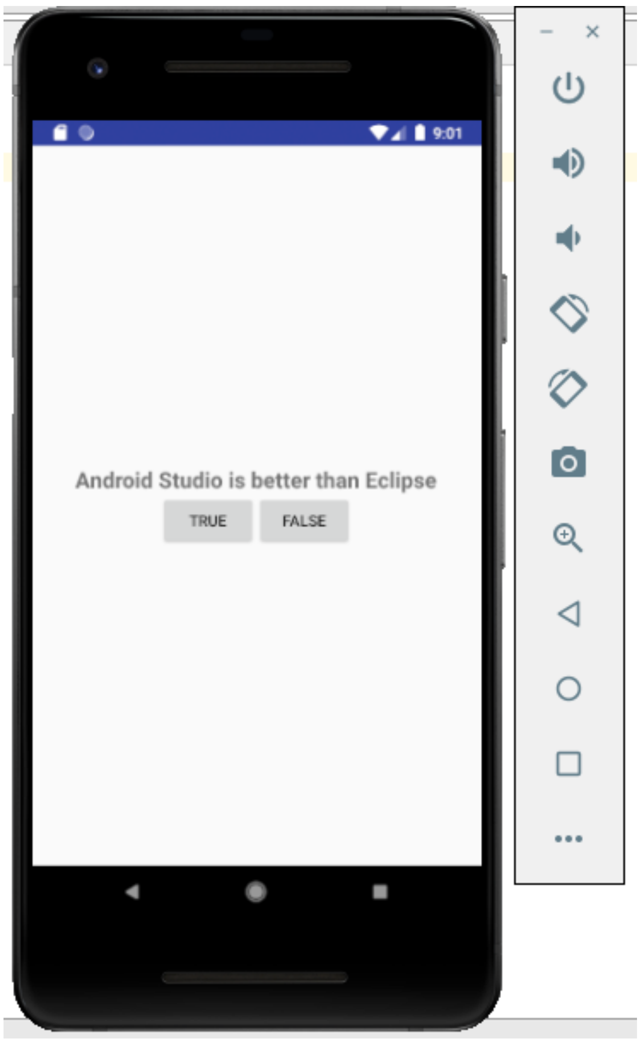


Figure 2-19. Displaying output in emulator

Exercise

1. What is android platform? Explain about history of android platform in detail.
2. List and explain different versions of android in detail.
3. How we can setup environment for android? Explain.
4. What do you mean by view hierarchy? Explain with the help of suitable example.
5. Define widget. Explain different widget attributes with example.
6. How can we create string resources? Explain with the help of suitable example.
7. Write a set of codes for creating activity. Also link xml file “activity_main.xml” to this activity.
8. Explain about manifest file. How can we add activity in manifest? Explain with example.
9. What is the role of intent-filter for any activity? Explain.
10. Explain the procedure for creating an AVD. Also write steps for running application on Emulator.