



**Guru Nanak College of Arts, Science & Commerce**

**G.T.B.Nagar, Mumbai – 400 037.**

**Department of Information Technology**

**CERTIFICATE**

**This is to certify that Mr MOHAMMAD ARMAN TAHIR ALI of TY BSc [Information Technology] Semester VI, Seat No. \_\_\_\_\_ has successfully completed the practicals for the subject of ADVANCED MOBILE PROGRAMMING as partial fulfillment of the degree B.Sc.(IT) during the academic year 2021-22.**

**Mrs. Simran Kaur**

**Faculty-in-charge**

**Mrs. Harpreet Kaur**

**BSc IT-in-charge**

**Internal Examiner**

**External Examiner**

**Date: / /2022**

**College Seal**

Sr. No	Date	Title	Sign
1		Introduction to Android, Introduction to Android Studio IDE, Application Fundamentals: Creating a Project, Android Components, Activities, Services, Content Providers, Broadcast Receivers, Interface overview, Creating Android Virtual device, USB debugging mode, Android Application Overview. Simple "Hello World" program.	
2		Programming Resources Android Ressources: (Color, Theme, String, Drawable, Dimension, Image).	
3		Programming Activities and fragments Activity Life Cycle, Activity methods, Multiple Activities, Life Cycle of fragments and multiple fragments.	
4		Programs related to different Layouts Coordinate, Linear, Relative, Table, Absolute, Frame, ListView, Grid View.	
5		Programming UI elements AppBar, Fragments, UI Components	
6		Programming menus, dialog, dialog fragments	
7		Programs on Intents, Events, Listeners and Adapters The Android Intent Class, Using Events and Event Listeners	
8		Programming Media API and Telephone API	

## PRACTICAL 1

### **Introduction to Android, Introduction to Android Studio IDE, Application Fundamentals: Creating a Project, Android Components, Activities, Services, Content Providers, Broadcast Receivers, Interface overview, Creating Android Virtual device, USB debugging mode, Android Application Overview. Simple “Hello World” program.**

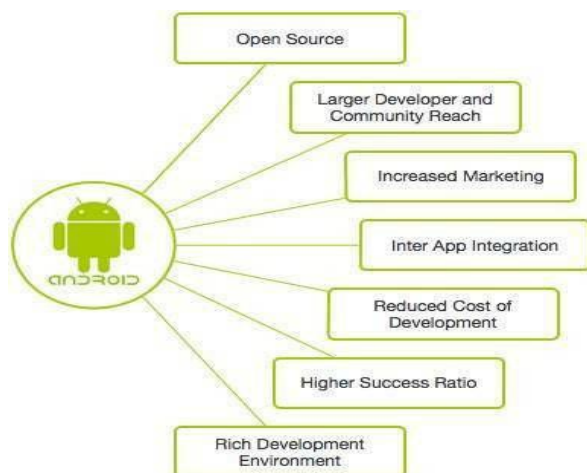
Android is an open-source and Linux-based Operating System for mobile devices such as smartphones and tablet computers. Android was developed by the Open Handset Alliance, led by Google, and other companies.

Android offers a unified approach to application development for mobile devices which means developers need only develop for Android, and their applications should be able to run on different devices powered by Android.

The first beta version of the Android Software Development Kit (SDK) was released by Google in 2007 whereas the first commercial version, Android 1.0, was released in September 2008.

On June 27, 2012, at the Google I/O conference, Google announced the next Android version, 4.1 Jelly Bean. Jelly Bean is an incremental update, with the primary aim of improving the user interface, both in terms of functionality and performance.

The source code for Android is available under free and open-source software licenses. Google publishes most of the code under the Apache License version 2.0 and the rest, Linux kernel changes, under the GNU General Public License version



Android is a powerful operating system competing with Apple 4GS and supports great features. Few of them are listed below –

Sr.No.	Feature & Description
1	<p>Beautiful UI</p> <p>Android OS basic screen provides a beautiful and intuitive user interface.</p>
2	<p>Connectivity</p> <p>GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, Wi-Fi, LTE, NFC, and WiMAX.</p>
3	<p>Storage</p> <p>SQLite, a lightweight relational database, is used for data storage purposes.</p>
4	<p>Media support</p> <p>H.263, H.264, MPEG-4 SP, AMR, AMR-WB, AAC, HE-AAC, AAC 5.1, MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF, and BMP.</p>
5	<p>Messaging</p> <p>SMS and MMS</p>

6	<p>Web browser</p> <p>Based on the open-source WebKit layout engine, coupled with Chrome's V8 JavaScript engine supporting HTML5 and CSS3.</p>
7	<p>Multi-touch</p> <p>Android has native support for multi-touch which was initially made available in handsets such as the HTC Hero.</p>
8	<p>Multi-tasking</p> <p>Users can jump from one task to another and at the same time various applications can run simultaneously.</p>
9	<p>Resizable widgets</p> <p>Widgets are resizable, so users can expand them to show more content or shrink them to save space.</p>
10	<p>Multi-Language</p> <p>Supports single direction and bi-directional text.</p>
11	<p>GCM</p>

	Google Cloud Messaging (GCM) is a service that lets developers send short message data to their users on Android devices, without needing a proprietary sync solution.
12	<p>Wi-Fi Direct</p> <p>A technology that lets apps discover and pair directly, over a high-bandwidth peer-to-peer connection.</p>
13	<p>Android Beam</p> <p>A popular NFC-based technology that lets users instantly share, just by touching two NFC-enabled phones together.</p>

Application components are the essential building blocks of an Android application. These components are loosely coupled by the application manifest file *AndroidManifest.xml* which describes each component of the application and how they interact.

There are following four main components that can be used within an Android application –

Sr.No	Components & Description
1	<p>Activities</p> <p>They dictate the UI and handle the user interaction to the smartphone screen.</p>
2	Services

	They handle background processing associated with an application.
3	<p>Broadcast Receivers</p> <p>They handle communication between Android OS and applications.</p>
4	<p>Content Providers</p> <p>They handle data and database management issues.</p>

## Activities

An activity represents a single screen with a user interface, in short Activity performs actions on the screen. 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. If an application has more than one activity, then one of them should be marked as the activity that is presented when the application is launched.

An activity is implemented as a subclass of Activity class as follows –

```
public class MainActivity extends Activity {
}
```

## 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 the 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. 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 a 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 broadcasted 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(){}
}
```

We will go through these tags in detail while covering application components in individual chapters.

## Additional Components

There are additional components which will be used in the construction of above mentioned entities, their logic, and wiring between them. These components are –

S.No	Components & Description
1	<p>Fragments</p> <p>Represents a portion of the user interface in an Activity.</p>
2	<p>Views</p> <p>UI elements that are drawn on-screen including buttons, lists forms etc.</p>



3	<b>Layouts</b>  View hierarchies that control screen format and appearance of the views.
4	<b>Intents</b>  Messages wiring components together.
5	<b>Resources</b>  External elements, such as strings, constants and drawable pictures.
6	<b>Manifest</b>  Configuration file for the application.

## Android Applications

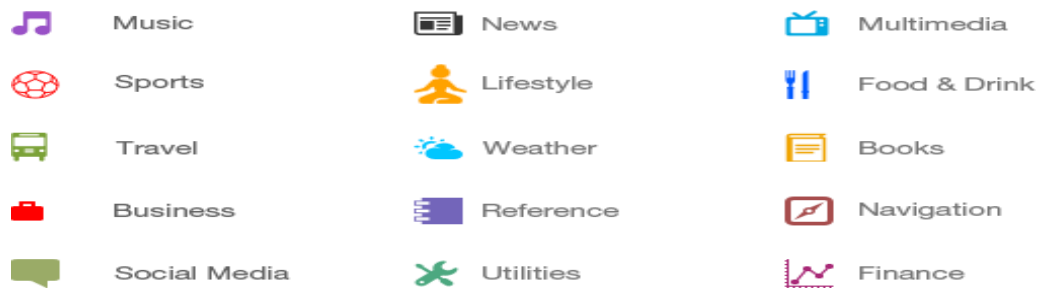
Android applications are usually developed in the Java language using the Android Software Development Kit.

Once developed, Android applications can be packaged easily and sold out either through a store such as Google Play, SlideME, Opera Mobile Store, Mobango, F-droid, and the Amazon Appstore.

Android powers hundreds of millions of mobile devices in more than 190 countries around the world. It's the largest installed base of any mobile platform and growing fast. Every day more than 1 million new Android devices are activated worldwide.

This tutorial has been written with the aim to teach you how to develop and package Android applications. We will start with an environment setup for Android application programming and then drill down to look into various aspects of Android applications.

## Categories of Android applications



## History of Android

The code names of android ranges from A to N currently, such as Aestro, Blender, Cupcake, Donut, Eclair, Froyo, Gingerbread, Honeycomb, Ice Cream Sandwich, Jelly Bean, KitKat, Lollipop, and Marshmallow. Let's understand the android history in a sequence.



## What is the API level?

API Level is an integer value that uniquely identifies the framework API revision offered by a version of the Android platform.

Platform Version	API Level	VERSION_CODE	

Android 6.0	23	MARSHMALLOW	
Android 5.1	22	LOLLIPOP_MR1	
Android 5.0	21	LOLLIPOP	
Android 4.4W	20	KITKAT_WATCH	KitKat for Wearables Only
Android 4.4	19	KITKAT	
Android 4.3	18	JELLY_BEAN_MR2	
Android 4.2, 4.2.2	17	JELLY_BEAN_MR1	
Android 4.1, 4.1.1	16	JELLY_BEAN	
Android 4.0.3, 4.0.4	15	ICE_CREAM_SANDWICH_MR1	
Android 4.0, 4.0.1, 4.0.2	14	ICE_CREAM_SANDWICH	

Android 3.2	13	HONEYCOMB_MR2	
Android 3.1.x	12	HONEYCOMB_MR1	
Android 3.0.x	11	HONEYCOMB	
Android 2.3.4 Android 2.3.3	10	GINGERBREAD_MR1	
Android 2.3.2 Android 2.3.1 Android 2.3	9	GINGERBREAD	
Android 2.2.x	8	FROYO	
Android 2.1.x	7	ECLAIR_MR1	
Android 2.0.1	6	ECLAIR_0_1	
Android 2.0	5	ECLAIR	

Android 1.6	4	DONUT	
Android 1.5	3	CUPCAKE	
Android 1.1	2	BASE_1_1	
Android 1.0	1	BASE	



## Step 1 - System Requirements

You will be delighted, to know that you can start your Android application development on either of the following operating systems –

- Microsoft® Windows® 10/8/7/Vista/2003 (32 or 64-bit)
- Mac® OS X® 10.8.5 or higher, up to 10.9 (Mavericks)
- GNOME or KDE desktop

Second point is that all the required tools to develop Android applications are open source and can be downloaded from the Web. Following is the list of software you will need before you start your Android application programming.

- Java JDK5 or later version
- Java Runtime Environment (JRE) 6
- Android Studio

## Step 2 - Setup Android Studio

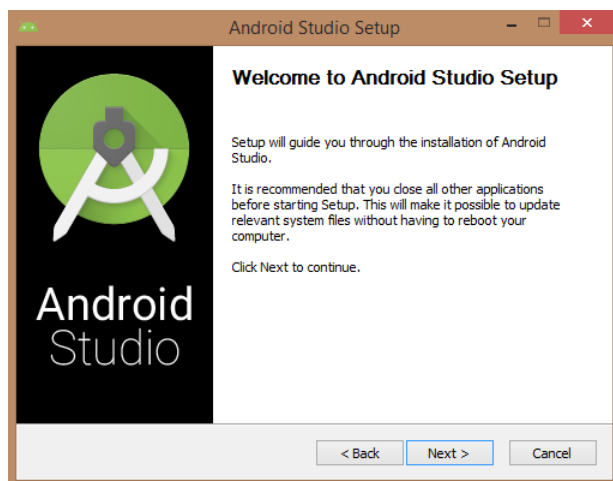
## Overview

Android Studio is the official IDE for android application development. It works based on IntelliJ IDEA, You can download the latest version of the android studio from [Android Studio 2.2 Download](#), If you are new to installing Android Studio on windows, you will find a file, which is named *android-studio-bundle-143.3101438-windows.exe*. So just download and run on a windows machine according to android studio wizard guidelines.

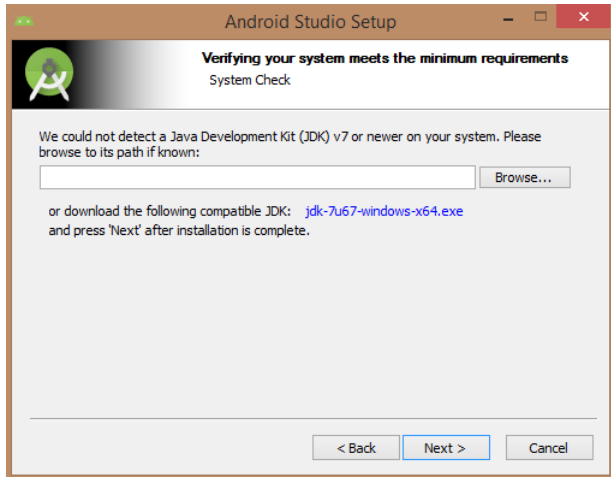
If you are installing Android Studio on Mac or Linux, You can download the latest version from [Android Studio Mac Download](#), or [Android Studio Linux Download](#), check the instructions provided along with the downloaded file for Mac OS and Linux. This tutorial will consider that you are going to set up your environment on a Windows machine having Windows 8.1 operating system.

## Installation

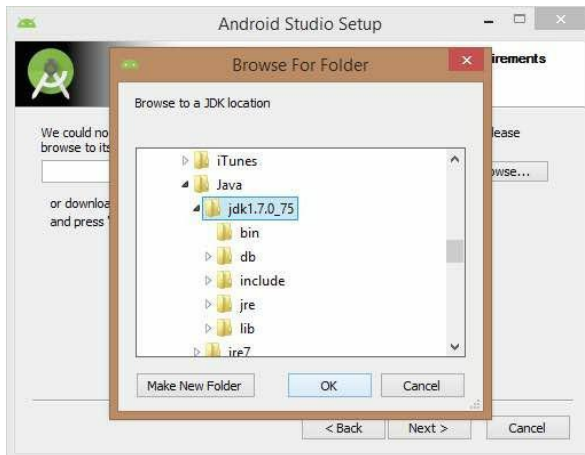
So let's launch *Android Studio.exe*, Make sure before launching Android Studio, Our Machine should be required to install Java JDK. To install Java JDK, take a reference of the [Android environment setup](#)



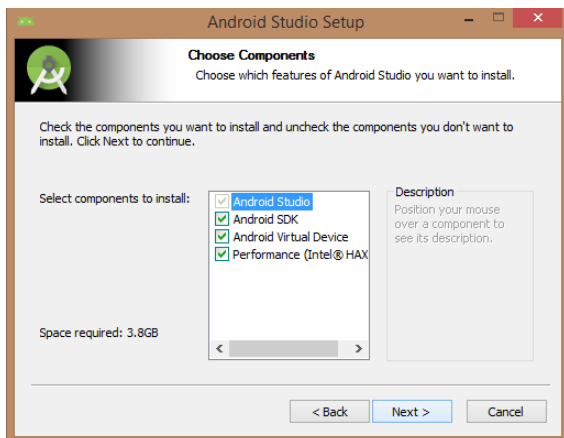
Once you launched Android Studio, it's time to mention the JDK7 path or later version in the android studio installer.



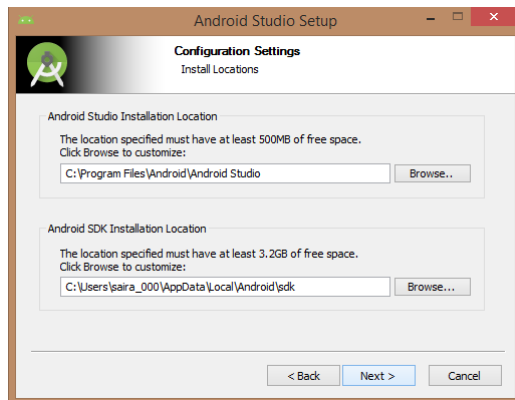
Below the image initiating JDK to android SDK



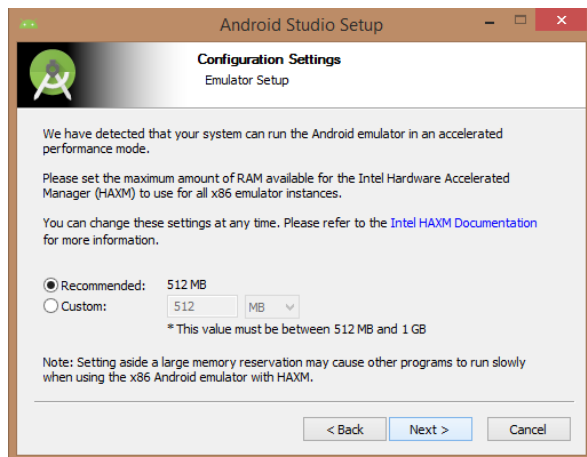
Need to check the components, which are required to create applications, below the image has selected Android Studio, Android SDK, Android Virtual Machine, and performance(Intel chip).



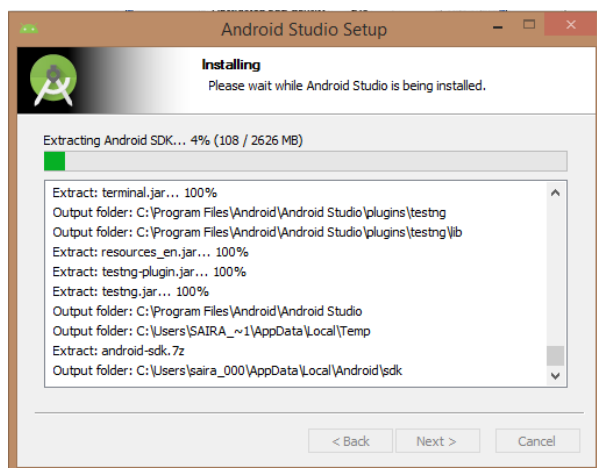
Need to specify the location of the local machine path for Android studio and Android SDK, below the image has taken default location of windows 8.1 x64 bit architecture.



Need to specify the ram space for the Android emulator by default it would take 512MB of local machine RAM.



At the final stage, it would extract SDK packages into our local machine, it would take a while time to finish the task, and would take 2626MB of Hard disk space.

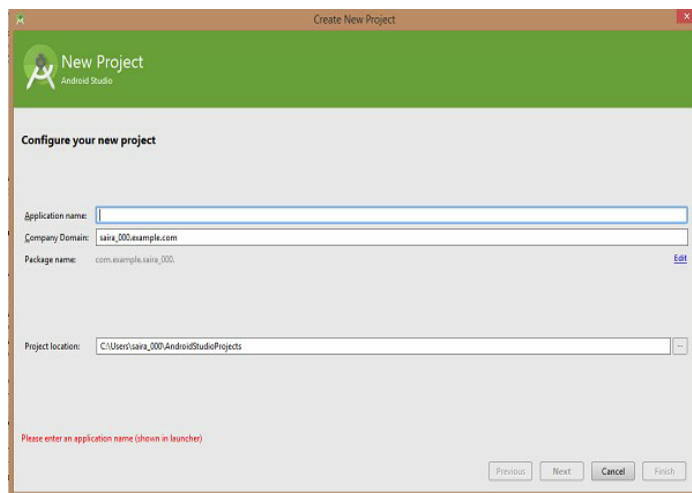




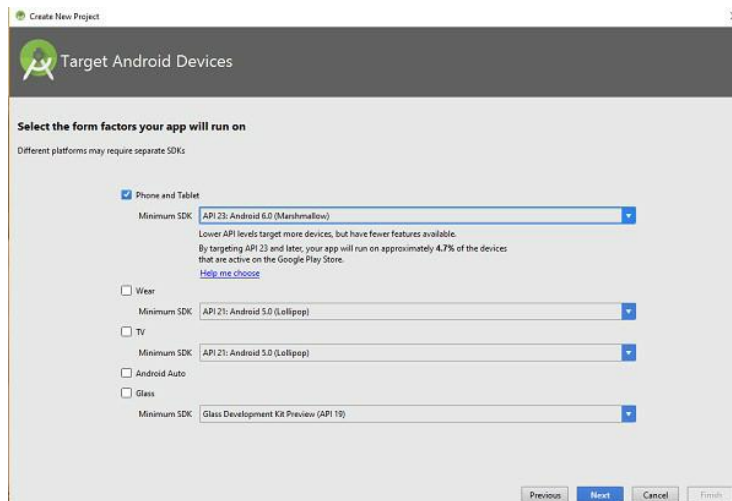
After doing all the above steps perfectly, you must get the finish button and it gonna be open the android studio project with the Welcome to android studio message as shown below



You can start your application development by calling start a new android studio project. a new installation frame should ask Application name, package information, and location of the project.



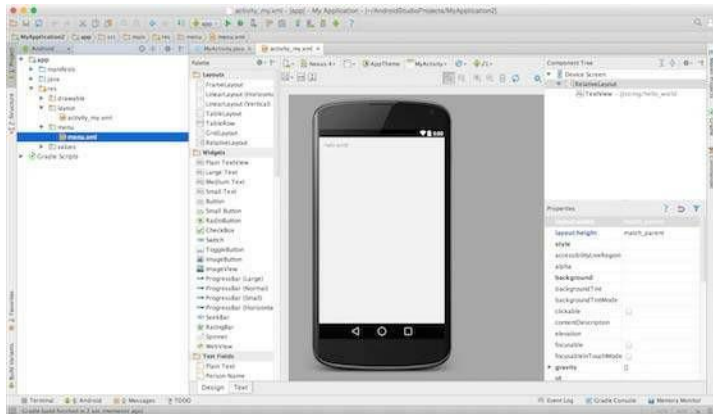
After entered application name, it going to be called select the form factors your application runs on, here need to specify Minimum SDK, in our tutorial, I have declared as API23: Android 6.0(Marshmallow)



The next level of installation should contain selecting the activity to mobile, it specifies the default layout for Applications

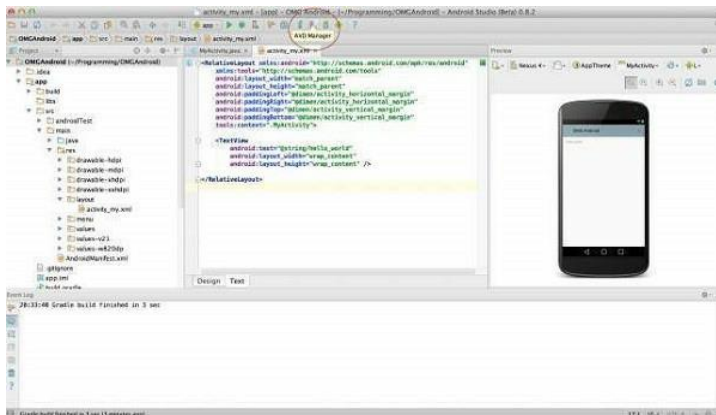


At the final stage it is going to be open development tool to write the application code.

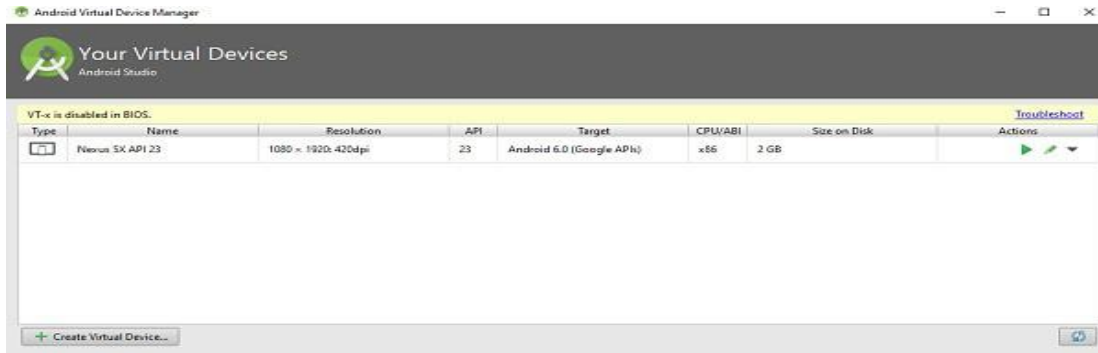


### Step 3 - Create Android Virtual Device

To test your Android applications, you will need a virtual Android device. So before we start writing our code, let us create an Android virtual device. Launch Android AVD Manager by Clicking the AVD\_Manager icon as shown below



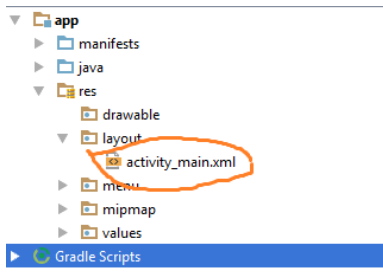
After Click on a virtual device icon, it going to be shown by default virtual devices which are present on your SDK, or else need to create a virtual device by clicking Create new Virtual device button



If your AVD is created successfully it means your environment is ready for Android application development. If you like, you can close this window using the top-right cross button. Better you restart your machine and once you are done with this last step, you are ready to proceed with your first Android example but before that, we will see a few more important concepts related to Android Application Development.

## Hello World Example

Before Writing a Hello world code, you must know about XML tags. To write hello world code, you should redirect to App>res>layout>Activity\_main.xml



To show hello word, we need to call a text view with layout ( about text view and layout, you must take references at Relative Layout and Text View ).

```
<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="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin" tools:context=".MainActivity">

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

```
    android:layout_width="550dp"  
    android:layout_height="wrap_content" />  
</RelativeLayout>
```

Need to run the program by clicking Run>Run App or else need to call shift+f 10 key. Finally, result should be placed at Virtual devices as shown below



## PRACTICAL 2

### Programming Resources Android Ressources: (Color, Theme, String, Drawable, Dimension, Image).

There are many more items that you use to build a good Android application. Apart from coding for the application, you take care of various other resources like static content that your code uses, such as bitmaps, Colors, Themes, Drawable, layout definitions, user interface strings, animation instructions, and more. These resources are always maintained separately in various sub-directories under the res/ directory of the project.

```
MyProject/  
  app/  
    manifest/  
      AndroidManifest.xml  
  java/  
    MainActivity.java  
  
  res/  
    drawable/  
      icon.png  
    layout/  
      activity_main.xml  
      info.xml  
    values/  
      Strings.xml
```

Sr.No.	Directory & Resource Type
1	<p>anim/</p> <p>XML files that define property animations. They are saved in res/anim/ folder and accessed from the R.anim class.</p>
2	<p>color/</p> <p>XML files that define a state list of colors. They are saved in res/color/ and accessed from the R.color class.</p>
3	<p>drawable/</p> <p>Image files like .png, .jpg, .gif or XML files that are compiled into bitmaps, state lists, shapes, animation drawables. They are saved in res/drawable/ and accessed from the R.drawable class.</p>
4	<p>layout/</p>

	<p>XML files that define a user interface layout. They are saved in <code>res/layout/</code> and accessed from the <code>R.layout</code> class.</p>
5	<p><code>menu/</code></p> <p>XML files that define application menus, such as an Options Menu, Context Menu, or Sub Menu. They are saved in <code>res/menu/</code> and accessed from the <code>R.menu</code> class.</p>
6	<p><code>raw/</code></p> <p>Arbitrary files to save in their raw form. You need to call <code>Resources.openRawResource()</code> with the resource ID, which is <code>R.raw.filename</code> to open such raw files.</p>



7	<p>values/</p> <p>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 –</p> <ul style="list-style-type: none"> <li>• arrays.xml for resource arrays, and accessed from the R.array class.</li> <li>• integers.xml for resource integers, and accessed from the R.integer class.</li> <li>• bools.xml for resource boolean, and accessed from the R.bool class.</li> <li>• colors.xml for color values, and accessed from the R.color class.</li> <li>• dimens.xml for dimension values, and accessed from the R.dimen class.</li> <li>• strings.xml for string values, and accessed from the R.string class.</li> <li>• styles.xml for styles, and accessed from the R.style class.</li> </ul>
8	<p>xml/</p> <p>Arbitrary XML files that can be read at runtime by calling <i>Resources.getXML()</i>. You can save various configuration files here which will be used at run time.</p>

#### Alternative Resources

Your application should provide alternative resources to support specific device configurations. For example, you should include alternative drawable resources ( i.e.images ) for different screen resolution and alternative string resources for different languages. At runtime, Android detects the current device configuration and loads the appropriate resources for your application.

To specify configuration-specific alternatives for a set of resources, follow the following steps –

- Create a new directory in res/ named in the form <resources\_name>-<config\_qualifier>. Here resources\_name will be any of the resources

mentioned in the above table, like layout, drawable etc. The qualifier will specify an individual configuration for which these resources are to be used. You can check official documentation for a complete list of qualifiers for different types of resources.

- Save the respective alternative resources in this new directory. The resource files must be named exactly the same as the default resource files as shown in the below example, but these files will have content specific to the alternative. For example though image file name will be same but for high resolution screen, its resolution will be high.

Below is an example which specifies images for a default screen and alternative images for high resolution screens

.

```
MyProject/
  app/
    manifest/
      AndroidManifest.xml
    java/
      MainActivity.java
    res/
      drawable/
        icon.png
        Background.png
      drawable-hdpi/
        icon.png
        background.png
      layout/
        activity_main.xml
        Info.xml
      layout-ar/
        main.xml
      values/
        Strings.xml
```

Below is another example which specifies layout for a default language and alternative layout for Arabic language.

```
MyProject/
  app/
    manifest/
      AndroidManifest.xml
    java/
```

```
MyActivity.java
    res/
drawable/
    icon.png
    Background.png
drawable-hdpi/
    icon.png
    background.png
layout/
    activity_main.xml
    Info.xml
layout-ar/
    main.xml
values/
    Strings.xml
```

## Accessing Resources

During your application development you will need to access defined resources either in your code, or in your layout XML files. Following section explains how to access your resources in both the scenarios –

### Accessing Resources in Code

When your Android application is compiled, a R class gets generated, which contains resource IDs for all the resources available in your `res/` directory. You can use R class to access that resource using sub-directory and resource name or directly resource ID.

### Example

To access `res/drawable/myimage.png` and set an `ImageView` you will use following code –

```
ImageView imageView = (ImageView) findViewById(R.id.imageview);
imageView.setImageResource(R.drawable.image);
```

Here the first line of the code makes use of `R.id.imageview` to get `ImageView` defined with id `imageview` in a Layout file. Second line of code makes use of `R.drawable.myimage` to get an image with name `myimage` available in the `drawable` subdirectory under `/res`.

### Example

Consider next example where `res/values/strings.xml` has following definition –

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello, World!</string>
</resources>
```

Now you can set the text on a `TextView` object with ID `msg` using a resource ID as follows –

```
TextView msgTextView = (TextView) findViewById(R.id.msg);
msgTextView.setText(R.string.hello);
```

## Example

Consider a layout *res/layout/activity\_main.xml* with the following definition –

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

    <TextView android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a TextView"
    />

    <Button android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a Button"
    />

</LinearLayout>
```

This application code will load this layout for an Activity, in the onCreate() method as follows –

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

## Accessing Resources in XML

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

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

Now you can use these resources in the following layout file to set the text color and text string as follows –

```
<?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" />
```

## PRACTICAL 3

### Programming Activities and fragments-Activity Life Cycle, Activity methods, Multiple Activities, Life Cycle of fragments and multiple fragments.

#### Fragment Lifecycle

Each Fragment instance has its own lifecycle. When a user navigates and interacts with your app, your fragments transition through various states in their lifecycle as they are added, removed, and enter or exit the screen.

To manage lifecycle, Fragment implements LifecycleOwner, exposing a Lifecycle object that you can access through the `getLifecycle()` method.

Each possible Lifecycle state is represented in the `Lifecycle.State` enum.

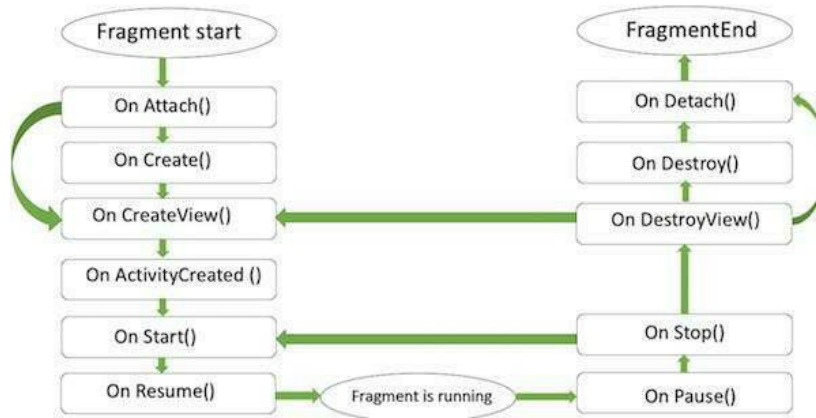
- **INITIALIZED**
- **CREATED**
- **STARTED**
- **RESUMED**
- **DESTROYED**

By building Fragment on top of Lifecycle, you can use the techniques and classes available for Handling Lifecycles with Lifecycle-Aware Components. For example, you might display the device's location on the screen using a lifecycle-aware component. This component could automatically start listening when the fragment becomes active and stop when the fragment moves to an inactive state.

As an alternative to using a LifecycleObserver, the Fragment class includes callback methods that correspond to each of the changes in a fragment's lifecycle. These include `onCreate()`, `onStart()`, `onResume()`, `onPause()`, `onStop()`, and `onDestroy()`.

A fragment's view has a separate Life Cycle that is managed independently from that of the fragment's Lifecycle. Fragments maintain a LifecycleOwner for their view, which can be accessed using `getViewLifecycleOwner()` or `getViewLifecycleOwnerLiveData()`. Having access to the view's Lifecycle is useful for situations where a Lifecycle-aware component should only perform work while a fragment's view exists, such as observing LiveData that is only meant to be displayed on the screen.

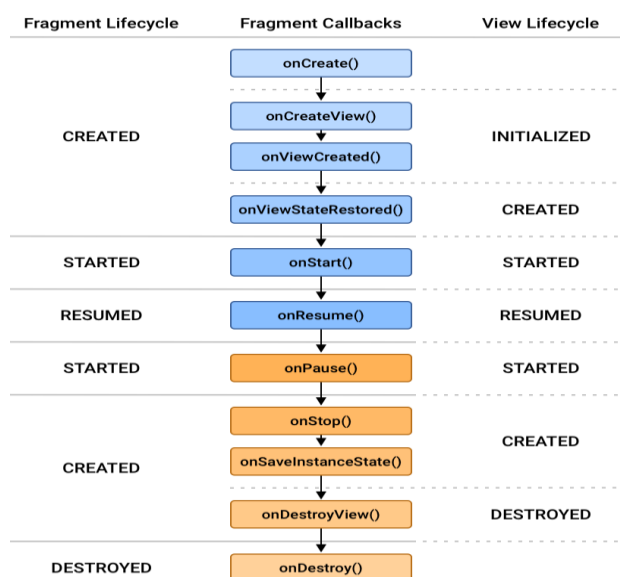
This topic discusses the Fragment lifecycle in detail, explaining some of the rules that determine a fragment's lifecycle state and showing the relationship between the Lifecycle states and the fragment lifecycle callbacks.



## Fragment lifecycle states and callbacks

When determining a fragment's lifecycle state, FragmentManager considers the following:

- A fragment's maximum state is determined by its FragmentManager. A fragment cannot progress beyond the state of its FragmentManager.
- As part of a FragmentTransaction, you can set a maximum lifecycle state on a fragment using `setMaxLifecycle()`.
- A fragment's lifecycle state can never be greater than its parent. For example, a parent fragment or activity must be started before its child fragments. Likewise, child fragments must be stopped before their parent fragment or activity.



**Figure 1.** Fragment Lifecycle states and their relation both the fragment's lifecycle callbacks and the fragment's view Lifecycle.

Figure 1 shows each of the fragment's Lifecycle states and how they relate to both the fragment's lifecycle callbacks and the fragment's view Lifecycle.

As a fragment progresses through its lifecycle, it moves upward and downward through its states. For example, a fragment that is added to the top of the back stack moves upward from CREATED to STARTED to RESUMED. Conversely, when a fragment is popped off of the back stack, it moves downward through those states, going from RESUMED to STARTED to CREATED and finally DESTROYED.

#### Upward state transitions

When moving upward through its lifecycle states, a fragment first calls the associated lifecycle callback for its new state. Once this callback is finished, the relevant Lifecycle.Event is emitted to observers by the fragment's Lifecycle, followed by the fragment's view Lifecycle, if it has been instantiated.

#### Fragment CREATED

When your fragment reaches the CREATED state, it has been added to a FragmentManager and the `onAttach()` method has already been called.

This would be the appropriate place to restore any saved state associated with the fragment itself through the fragment's `SavedStateRegistry`. Note that the fragment's view has not been created at this time, and any state associated with the fragment's view should be restored only after the view has been created.

This transition invokes the `onCreate()` callback. The callback also receives a `savedInstanceState` Bundle argument containing any state previously saved by `onSaveInstanceState()`. Note that `savedInstanceState` has a null value the first time the fragment is created, but it is always non-null for subsequent recreations, even if you do not override `onSaveInstanceState()`. See [Saving state with fragments](#) for more details.

#### Fragment CREATED and View INITIALIZED

The fragment's view Lifecycle is created only when your Fragment provides a valid View instance. In most cases, you can use the fragment constructors that take a `@LayoutId`, which automatically inflates the view at the appropriate time. You can also override `onCreateView()` to programmatically inflate or create your fragment's view.

If and only if your fragment's view is instantiated with a non-null View, that View is set on the fragment and can be retrieved using `getView()`. The `getViewLifecycleOwnerLiveData()` is then updated with the newly INITIALIZED LifecycleOwner corresponding with the fragment's view. The `onViewCreated()` lifecycle callback is also called at this time.

This is the appropriate place to set up the initial state of your view, to start observing LiveData instances whose callbacks update the fragment's view, and to set up adapters on any RecyclerView or ViewPager2 instances in your fragment's view.

## Fragment and View CREATED

After the fragment's view has been created, the previous view state, if any, is restored, and the view's Lifecycle is then moved into the CREATED state. The view lifecycle owner also emits the ON\_CREATE event to its observers. Here you should restore any additional state associated with the fragment's view.

This transition also invokes the `onViewStateRestored()` callback.

## Fragment and View STARTED

It is strongly recommended to tie Lifecycle-aware components to the STARTED state of a fragment, as this state guarantees that the fragment's view is available, if one was created, and that it is safe to perform a `FragmentManager` transaction on the child `FragmentManager` of the fragment. If the fragment's view is non-null, the fragment's view Lifecycle is moved to STARTED immediately after the fragment's Lifecycle is moved to STARTED.

When the fragment becomes STARTED, the `onStart()` callback is invoked.

## The Activity Lifecycle

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. The Activity class provides a number of callbacks that allow the activity to know that a state has changed: that the system is creating, stopping, or resuming an activity, or destroying the process in which the activity resides.

Within the lifecycle callback methods, you can declare how your activity behaves when the user leaves and re-enters the activity. For example, if you're building a streaming video player, you might pause the video and terminate the network connection when the user switches to another app. When the user returns, you can reconnect to the network and allow the user to resume the video from the same spot. In other words, each callback allows you to perform specific work that's appropriate to a given change of state. Doing the right work at the right time and handling transitions properly make your app more robust and performant. For example, good implementation of the lifecycle callbacks can help ensure that your app avoids:

- Crashing if the user receives a phone call or switches to another app while using your app.
- Consuming valuable system resources when the user is not actively using it.
- Losing the user's progress if they leave your app and return to it at a later time.
- Crashing or losing the user's progress when the screen rotates between landscape and portrait orientation.

This document explains the activity lifecycle in detail. The document begins by describing the life cycle paradigm. Next, it explains each of the callbacks: what happens internally while they execute, and what you should implement during them. It then briefly introduces the relationship between the activity state and a process's vulnerability to being killed by the system. Last, it discusses several topics related to transitions between activity states.



For information about handling life cycles, including guidance about best practices, see [Handling Lifecycles with Lifecycle-Aware Components and Saving UI States](#). To learn how to architect a robust, production-quality app using activities in combination with architecture components, see [Guide to App Architecture](#).

Sr.No	Callback & Description
1	<code>onCreate()</code>  This is the first callback and called when the activity is first created.
2	<code>onStart()</code>  This callback is called when the activity becomes visible to the user.
3	<code>onResume()</code>  This is called when the user starts interacting with the application.
4	<code>onPause()</code>  The paused activity does not receive user input and cannot execute any code and is called when the current activity is being paused and the previous activity is being resumed.
5	<code>onStop()</code>

	<p>This callback is called when the activity is no longer visible.</p>
6	<p><code>onDestroy()</code></p> <p>This callback is called before the activity is destroyed by the system.</p>
7	<p><code>onRestart()</code></p> <p>This callback is called when the activity restarts after stopping it.</p>

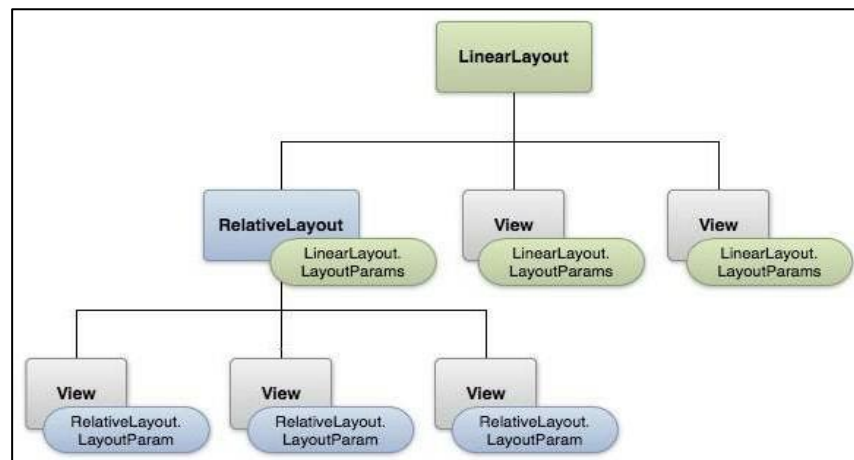
## PRACTICAL 4

### Programs related to different Layouts Coordinate, Linear, Relative, Table, Absolute, Frame, ListView, Grid View.

The basic building block for the user interface is a View object which is created from the View class and occupies a rectangular area on the screen and is responsible for drawing and event handling. The view is the base class for widgets, which are used to create interactive UI components like buttons, text fields, etc.

The ViewGroup is a subclass of View and provides an invisible container that holds other Views or other ViewGroups and defines their layout properties.

At the third level, we have different layouts which are subclasses of ViewGroup class and a typical layout defines the visual structure for an Android user interface and can be created either at run time using View/ViewGroup objects or you can declare your layout using simple XML file main\_layout.xml which is located in the res/layout folder of your project.



## Layout params

This tutorial is more about creating your GUI based on layouts defined in XML files. A layout may contain any type of widgets such as buttons, labels, textboxes, and so on. Following is a simple example of XML file having `LinearLayout` –

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

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

    <Button android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        " android:text="This is a Button"
    />

    <!-- More GUI components go here -->

</LinearLayout>
```

Once your layout has created, you can load the layout resource from your application code, in your *Activity.onCreate()* callback implementation as shown below –

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

## Android Layout Types

There are a number of Layouts provided by Android which you will use in almost all the Android applications to provide a different view, look and feel.

Sr.No	Layout & Description
1	<u>Linear Layout</u>  LinearLayout is a view group that aligns all children in a single direction, vertically or horizontally.
2	<u>Relative Layout</u>  RelativeLayout is a view group that displays child views in relative positions.
3	<u>Table Layout</u>  TableLayout is a view that groups views into rows and columns.
4	<u>Absolute Layout</u>  AbsoluteLayout enables you to specify the exact location of its children.
5	<u>Frame Layout</u>  The FrameLayout is a placeholder on screen that you can use to display a single view.
6	<u>List View</u>  ListView is a view group that displays a list of scrollable items.

7	<u>Grid View</u>  GridView is a ViewGroup that displays items in a two-dimensional, scrollable grid.
---	--

### Layout Attributes

Each layout has a set of attributes that define the visual properties of that layout. There are few common attributes among all the layouts and there are other attributes that are specific to that layout. The following are common attributes and will be applied to all the layouts:

Sr.No	Attribute & Description
1	<b>android:id</b>  This is the ID which uniquely identifies the view.
2	<b>android:layout_width</b>  This is the width of the layout.
3	<b>android:layout_height</b>  This is the height of the layout
4	<b>android:layout_marginTop</b>  This is the extra space on the top side of the layout.

5	<b>android:layout_marginBottom</b>  This is the extra space on the bottom side of the layout.
6	<b>android:layout_marginLeft</b>  This is the extra space on the left side of the layout.
7	<b>android:layout_marginRight</b>  This is the extra space on the right side of the layout.
8	<b>android:layout_gravity</b>  This specifies how child Views are positioned.
9	<b>android:layout_weight</b>  This specifies how much of the extra space in the layout should be allocated to the View.
10	<b>android:layout_x</b>  This specifies the x-coordinate of the layout.

11	<p><b>android:layout_y</b></p> <p>This specifies the y-coordinate of the layout.</p>
12	<p><b>android:layout_width</b></p> <p>This is the width of the layout.</p>
13	<p><b>android:paddingLeft</b></p> <p>This is the left padding filled for the layout.</p>
14	<p><b>android:paddingRight</b></p> <p>This is the right padding for the layout.</p>
15	<p><b>android:paddingTop</b></p> <p>This is the top padding filled for the layout.</p>
16	<p><b>android:paddingBottom</b></p> <p>This is the bottom padding filled for the layout.</p>

Here width and height are the dimension of the layout/view which can be specified in terms of dp (Density-independent Pixels), sp ( Scale-independent Pixels), pt ( Points which is 1/72 of an inch), px( Pixels), mm ( Millimeters) and finally in (inches).



You can specify width and height with exact measurements but more often, you will use one of these constants to set the width or height –

- `android:layout_width=wrap_content` tells your view to size itself to the dimensions required by its content.
- `android:layout_width=fill_parent` tells your view to become as big as its parent view.

Gravity attribute plays important role in positioning the view object and it can take one or more (separated by '|') of the following constant values.

Constant	Value	Description
top	0x30	Push the object to the top of its container, not changing its size.
bottom	0x50	Push the object to the bottom of its container, not changing its size.
left	0x03	Push the object to the left of its container, not changing its size.
right	0x05	Push the object to the right of its container, not changing its size.
center_vertical	0x10	Place the object in the vertical center of its container, not changing its size.
fill_vertical	0x70	Grow the vertical size of the object if needed so it completely fills its container.
center_horizontal	0x01	Place the object in the horizontal center of its container, not changing its size.

fill_horizontal	0x07	Grow the horizontal size of the object if needed so it completely fills its container.
center	0x11	Place the object in the center of its container in both the vertical and horizontal axis, not changing its size.
fill	0x77	Grow the horizontal and vertical size of the object if needed so it completely fills its container.
clip_vertical	0x80	Additional option that can be set to have the top and/or bottom edges of the child clipped to its container's bounds. The clip will be based on the vertical gravity: a top gravity will clip the bottom edge, a bottom gravity will clip the top edge, and neither will clip both edges.
clip_horizontal	0x08	Additional option that can be set to have the left and/or right edges of the child clipped to its container's bounds. The clip will be based on the horizontal gravity: a left gravity will clip the right edge, a right gravity will clip the left edge, and neither will clip both edges.
start	0x00800003	Push the object to the beginning of its container, not changing its size.
end	0x00800005	Push the object to the end of its container, not changing its size.

#### View Identification

A view object may have a unique ID assigned to it which will identify the View uniquely within the tree. The syntax for an ID, inside an XML tag is –

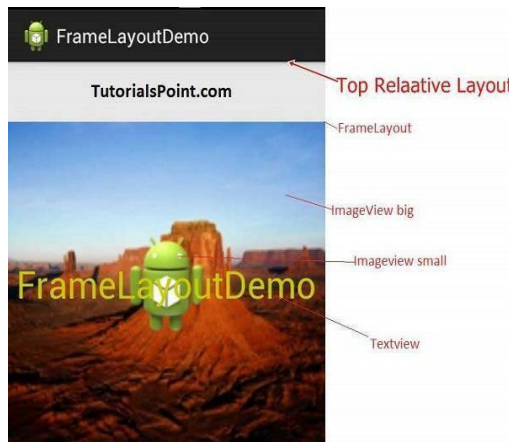
```
android:id="@+id/my_button"
```

Following is a brief description of @ and + signs –

- The at-symbol (@) at the beginning of the string indicates that the XML parser should parse and expand the rest of the ID string and identify it as an ID resource.
- The plus-symbol (+) means that this is a new resource name that must be created and added to our resources. To create an instance of the view object and capture it from the layout, use the following –

```
Button myButton = (Button) findViewById(R.id.my_button);
```

You can, however, add multiple children to a FrameLayout and control their position within the FrameLayout by assigning gravity to each child, using the android:layout\_gravity attribute.



Frame Layout

### FrameLayout Attributes

Following are the important attributes specific to FrameLayout –

Sr.No	Attribute & Description
1	<p>android:id</p> <p>This is the ID which uniquely identifies the layout.</p>

2	<p>android:foreground</p> <p>This defines the drawable to draw over the content and possible values may be a color value, in the form of "#rgb", "#argb", "#rrggbb", or "#aarrggbb".</p>
3	<p>android:foregroundGravity</p> <p>Defines the gravity to apply to the foreground drawable. The gravity defaults to fill. Possible values are top, bottom, left, right, center, center_vertical, center_horizontal etc.</p>
4	<p>android:measureAllChildren</p> <p>Determines whether to measure all children or just those in the VISIBLE or INVISIBLE state when measuring. Defaults to false.</p>

### Example

This example will take you through simple steps to show how to create your own Android application using frame layout. Follow the following steps to modify the Android application we created in *Hello World Example* chapter –

Step	Description
1	You will use Android studio IDE to create an Android application and name it as a demo under a package <i>com.example.demo</i> as explained in the <i>Hello World Example</i> chapter.
2	Modify the default content of <i>res/layout/activity_main.xml</i> file to include few widgets in frame layout.

3	No need to change string.xml, android takes care default constants
4	Run the application to launch the Android emulator and verify the result of the changes done in the application.

Following is the content of the modified main activity file src/com.example.demo/MainActivity.java. This file can include each of the fundamental lifecycle methods.

```
package com.example.demo;

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

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

Following will be the content of res/layout/activity\_main.xml file –

```
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">


    <ImageView
        android:src="@drawable/ic_launcher
        " android:scaleType="fitCenter"
        android:layout_height="250px"
        android:layout_width="250px"/>

    <TextView
        android:text="Frame Demo"
        android:textSize="30px"
        android:textStyle="bold"
        android:layout_height="fill_parent
        "
        android:layout_width="fill_parent"
        android:gravity="center"/>
</FrameLayout>
```

Following will be the content of res/values/strings.xml to define two new constants –

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">demo</string>
    <string name="action_settings">Settings</string>
```

```
</resources>
```

Let's try to run our modified Hello World! application we just modified. I assume you had created your AVD while doing environment setup. To run the app from Android Studio, open one of your project's activity files and click Run  icon from the toolbar. Android Studio installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display following Emulator window

—



## PRACTICAL 5

### Programming UI elements AppBar, Fragments, UI Components

The top app bar provides a consistent place along the top of your app window for displaying information and actions from the current screen.



an example top app bar

When using fragments, the app bar can be implemented as an ActionBar that is owned by the host activity or a toolbar within your fragment's layout. Ownership of the app bar varies depending on the needs of your app.

If all your screens use the same app bar that's always at the top and spans the width of the screen, then you should use a theme-provided action bar hosted by the activity. Using theme app bars helps to maintain a consistent look and provides a place to host option menus and an up button.

Use a toolbar hosted by the fragment if you want more control over the size, placement, and animation of the app bar across multiple screens. For example, you might need a collapsing app bar or one that spans only half the width of the screen and is vertically centered.

Understanding the different approaches and employing a correct one saves you time and helps to ensure your app functions properly. Different situations require different approaches for things like inflating menus and responding to user interaction.

The examples in this topic reference an `ExampleFragment` that contains an editable profile. The fragment inflates the following XML-defined menu in its app bar:

```
<!-- sample_menu.xml -->
<menu
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto">

  <item
    android:id="@+id/action_settings"
    android:icon="@drawable/ic_settings"
    android:title="@string/settings"
    app:showAsAction="ifRoom"/>
  <item
    android:id="@+id/action_done"
    android:icon="@drawable/ic_done"
    android:title="@string/done"
    app:showAsAction="ifRoom|withText"/>

</menu>
```

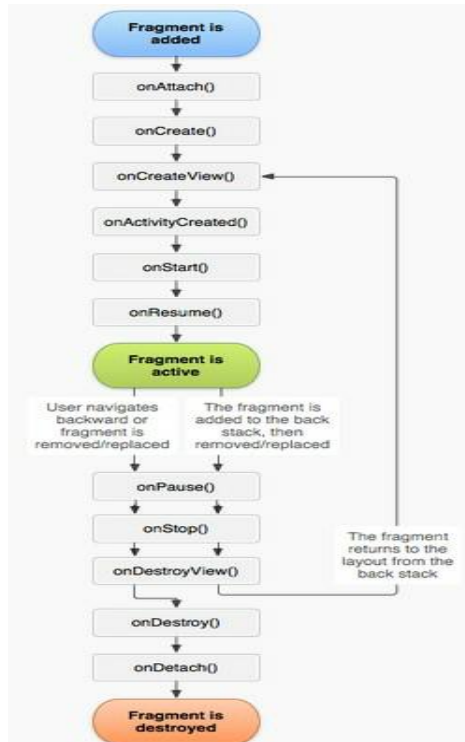
The menu contains two options: one for navigating to a profile screen, and one to save any profile changes made.

#### Fragment-owned app bar

If most screens in your app don't need an app bar, or if perhaps one screen needs a dramatically-different app bar, you can add a `Toolbar` to your fragment layout. Though you can add a `Toolbar` anywhere within your fragment's view hierarchy, you should generally keep it at the top of the screen. To use the `Toolbar` in your fragment, provide an ID and obtain a reference to it in your fragment, as you would with any other view.

```
<android.support.design.widget.Toolbar
  android:id="@+id/myToolbar"
  ... />
```





When using a fragment-owned app bar, we strongly recommended using the Toolbar APIs directly. Do not use `setSupportActionBar()` and the Fragment menu APIs, which are appropriate only for activity-owned app bars.

## Fragments

A Fragment is a piece of an activity which enables more modular activity design. It will not be wrong if we say, a fragment is a kind of sub-activity.

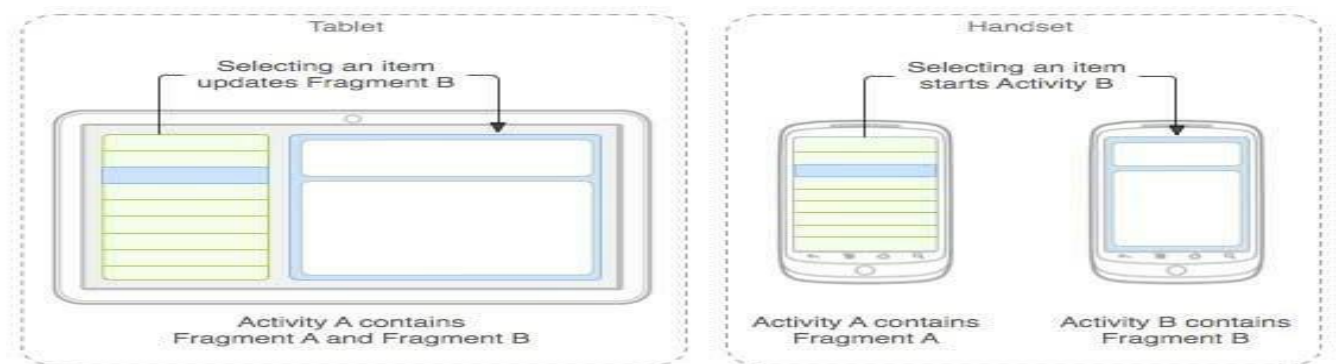
Following are important points about fragment –

- A fragment has its own layout and its own behavior with its own life cycle callbacks.
- You can add or remove fragments in an activity while the activity is running.
- You can combine multiple fragments in a single activity to build a multi-pane UI.
- A fragment can be used in multiple activities.
- Fragment life cycle is closely related to the life cycle of its host activity which means when the activity is paused, all the fragments available in the activity will also be stopped.
- A fragment can implement a behavior that has no user interface component.
- Fragments were added to the Android API in Honeycomb version of Android which API version 11.

You create fragments by extending Fragment class and You can insert a fragment into your activity layout by declaring the fragment in the activity's layout file, as a <fragment> element.

Prior to fragment introduction, we had a limitation because we can show only a single activity on the screen at one given point in time. So we were not able to divide the device screen and control different parts separately. But with the introduction of fragment we got more flexibility and removed the limitation of having a single activity on the screen at a time. Now we can have a single activity but each activity can consist of multiple fragments which will have their own layout, events and complete life cycle.

Following is a typical example of how two UI modules defined by fragments can be combined into one activity for a tablet design, but separated for a handset design.



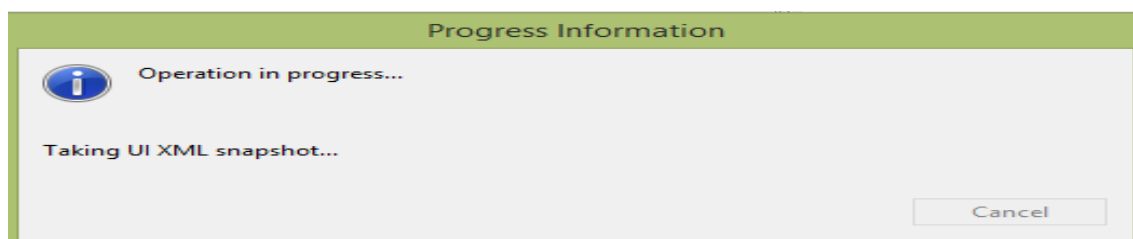
The application can embed two fragments in Activity A, when running on a tablet-sized device. However, on a handset-sized screen, there's not enough room for both fragments, so Activity A includes only the fragment for the list of articles, and when the user selects an article, it starts Activity B, which includes the second fragment to read the article.

## UI components

A typical user interface of an android application consists of an action bar and the application content area.

- Main Action Bar
- View Control
- Content Area
- Split Action Bar

These components have also been shown in the image below –



## Understanding Screen Components

The basic unit of android application is the activity. A UI is defined in an xml file. During compilation, each element in the XML is compiled into an equivalent Android GUI class with attributes represented by methods.

#### View and ViewGroups

An activity consists of views. A view is just a widget that appears on the screen. It could be a button e.t.c. One or more views can be grouped together into one GroupView. Example of ViewGroup includes layouts.

#### Types of layout

There are many types of layout. Some of which are listed below –

- Linear Layout
- Absolute Layout
- Table Layout
- Frame Layout
- Relative Layout

# PRACTICAL 6

## Programming menus, dialog, dialog fragments

### Android Option Menu Example

Android Option Menus are the primary menus of android. They can be used for settings, search, delete items etc. Here, we are going to see two examples of option menus. First, the simple option menus and second, options menus with images. Here, we are inflating the menu by calling the `inflate()` method of the `MenuInflater` class. To perform event handling on menu items, you need to override `onOptionsItemSelected()` method of Activity class.

Let's see how to create a menu in android. Let's see the simple option menu example that contains three menu items.

### **activity\_main.xml**

**We have only one textview in this file.**

*File: activity\_main.xml*

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

<android.support.design.widget.CoordinatorLayout
xmlns:android="http://schemas.android.com/apk/res/android"
```

```

xmlns:app="http://schemas.android.com/apk/res-auto"

xmlns:tools="http://schemas.android.com/tools"

android:layout_width="match_parent"

android:layout_height="match_parent"

tools:context="example.javatpoint.com.optionmenu.MainActivity"

">

<android.support.design.widget.AppBarLayout

android:layout_width="match_parent"

android:layout_height="wrap_content"

android:theme="@style/AppTheme.AppBarOverlay"

>

<android.support.v7.widget.Toolbar

android:id="@+id/toolbar"

android:layout_width="match_parent"

android:layout_height="?attr/actionBarSize"

android:background="?attr/colorPrimary"

app:popupTheme="@style/AppTheme.PopupOverlay" />

</android.support.design.widget.AppBarLayout>

<include layout="@layout/content_main" />

</android.support.design.widget.CoordinatorLayout>

```

*File: context\_main.xml*

```

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

<android.support.constraint.ConstraintLayout

xmlns:android="http://schemas.android.com/apk/res/androi
d"

```

```
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"

android:layout_width="match_parent"

android:layout_height="match_parent"

app:layout_behavior="@string/appbar_scrolling_view_behavior"

tools:context="example.javatpoint.com.optionmenu.MainActivity"

y"

tools:showIn="@layout/activity_main">

<TextView

android:layout_width="wrap_content"

android:layout_height="wrap_content

" android:text="Hello World!"

app:layout_constraintBottom_toBottomOf="paren

t" app:layout_constraintLeft_toLeftOf="parent"

app:layout_constraintRight_toRightOf="parent"

app:layout_constraintTop_toTopOf="parent" />

</android.support.constraint.ConstraintLayout>
```

## **menu\_main.xml**

It contains three items as shown below. It is created automatically inside the res/menu directory.

*File: menu\_main.xml*

```
<menu
```

```
xmlns:android="http://schemas.android.com/apk/res/android"
```

```
xmlns:app="http://schemas.android.com/apk/res-auto"
```

```
xmlns:tools="http://schemas.android.com/tools"
```

```

tools:context="example.javatpoint.com.optionmenu.MainActivity">

<item android:id="@+id/item1"

android:title="Item 1"/>

<item android:id="@+id/item2"

android:title="Item 2"/>

<item

android:id="@+id/item3"

android:title="Item 3"

app:showAsAction="withText"/

>

</menu>

```

### **Activity class**

This class displays the content of the menu.xml file and performs event handling on clicking the menu items.

*File: MainActivity.java*

```

package example.javatpoint.com.optionmenu;

import android.os.Bundle;

import android.support.v7.app.AppCompatActivity;

import android.support.v7.widget.Toolbar;

import android.view.Menu;

import android.view.MenuItem;

import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

```



```

@Override

protected void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);

    setContentView(R.layout.activity_main);

    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);

    setSupportActionBar(toolbar);

}

@Override

public boolean onCreateOptionsMenu(Menu menu) {

    // Inflate the menu; this adds items to the action bar if it is present.

    getMenuInflater().inflate(R.menu.menu_main, menu);

    return true;

}

@Override

public boolean onOptionsItemSelected(MenuItem item)

{ int id = item.getItemId();

    switch (id){

    case

    R.id.item1:

    Toast.makeText(getApplicationContext(),"Item 1
    Selected",Toast.LENGTH_LONG).show();

    return true;

    case

    R.id.item2:

```

```

Toast.makeText(getApplicationContext(),"Item 2
Selected",Toast.LENGTH_LONG).show();

return true;

case

R.id.item3:

Toast.makeText(getApplicationContext(),"Item 3
Selected",Toast.LENGTH_LONG).show();

return true;

default:

return super.onOptionsItemSelected(item);

}

}

}

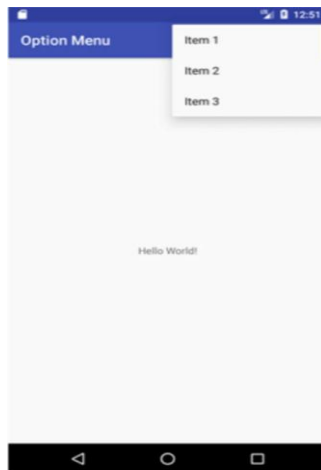
```

**Output:**

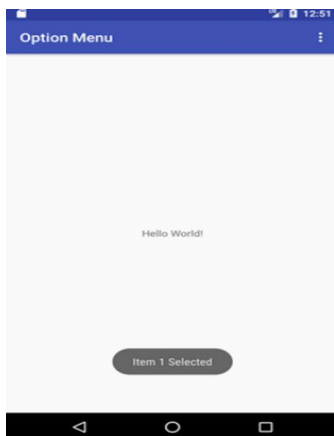
Output without clicking on the menu button.



Output after clicking on the menu button.



Output after clicking on the second menu item .



### Option Menu with Icon

You need to have icon images inside the res/drawable directory. The android:icon element is used to display the icon on the option menu. You can write the string information in the strings.xml file. But we have written it inside the menu\_main.xml file.

*File: menu\_main.xml*

**<menu**

xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:app="http://schemas.android.com/apk/res-auto"

xmlns:tools="http://schemas.android.com/tools"

tools:context="example.javatpoint.com.optionmenu.MainActivity"

**>**

```

<item android:id="@+id/item1"

android:title="Item 1"

app:showAsAction="always"

android:icon="@android:drawable/btn_star"/

>

<item android:id="@+id/item2"

android:title="Item 2"

app:showAsAction="ifRoom"

android:icon="@android:drawable/btn_plus"/

>

<item android:id="@+id/item3"

android:title="Item 3"

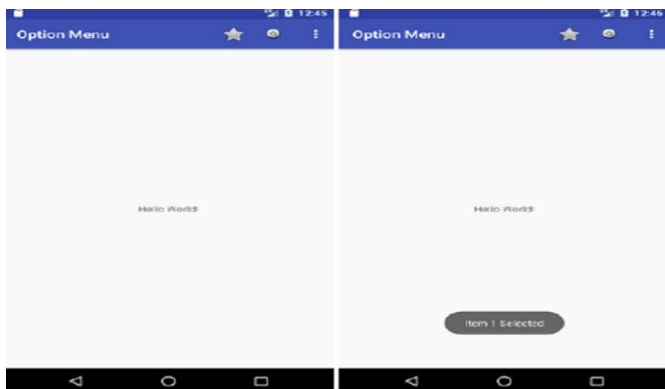
app:showAsAction="withText"

android:icon="@android:drawable/btn_plus"/

>

</menu>

```



A Dialog is a small window that prompts the user to make a decision or enter additional information.

In order to make an alert dialog, you need to make an object of `AlertDialogBuilder` which is an inner class of `AlertDialog`. Its syntax is given below

```
AlertDialog.Builder alertDialogBuilder = new AlertDialog.Builder(this);
```

AlertDialogBuilder.setPositiveButton(CharSequence text,

DialogInterface.OnClickListener listener)

AlertDialogBuilder.setNegativeButton(CharSequence text,

DialogInterface.OnClickListener listener)

Sr.No	Method type & description
1	<p>setIcon(Drawable icon)</p> <p>This method sets the icon of the alert dialog box.</p>
2	<p>setCancelable(boolean cancel able)</p> <p>This method sets the property that the dialog can be canceled or not</p>
3	<p>setMessage(CharSequence message)</p> <p>This method sets the message to be displayed in the alert dialog</p>
4	<p>setMultiChoiceItems(CharSequence[] items, boolean[] checkedItems, DialogInterface.OnMultiChoiceClickListener listener)</p> <p>This method sets a list of items to be displayed in the dialog as the content. The selected option will be notified by the listener</p>

5	<p><code>setOnCancelListener(DialogInterface.OnCancelListener onCancelListener)</code></p> <p>This method Sets the callback that will be called if the dialog is canceled.</p>
6	<p><code>setTitle(CharSequence title)</code></p> <p>This method set the title to be appear in the dialog</p>

### Dialog fragment

Before enter into an example we should need to know dialog fragment. Dialog fragment is a fragment which can show fragment in dialog box

```
public class DialogFragment extends DialogFragment

{ @Override

    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // Use the Builder class for convenient dialog construction
        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
        builder.setPositiveButton(R.string.fire, new
DialogInterface.OnClickListener() {

            public void onClick(DialogInterface dialog, int id) {

                toast.makeText(this, "enter a text here", Toast.LENGTH_SHORT).show();

            }

        })
    }
}
```

```

        .setNegativeButton(R.string.cancel, new DialogInterface.OnClickListener()
        {

            public void onClick(DialogInterface dialog, int id) {

                finish();

            });

            // Create the AlertDialog object and return it

            return builder.create();

        }

    }
}

```

### List dialog

It is used to show a list of items in a dialog box. For suppose, users need to select a list of items or else need to click an item from multiple lists of items. At this situation we can use list dialog.

```

public Dialog onCreateDialog(Bundle savedInstanceState) {

    AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());

    builder.setTitle(Pick a Color)

        .setItems(R.array.colors_array, new DialogInterface.OnClickListener()

        { public void onClick(DialogInterface dialog, int which) {

            // The 'which' argument contains the index position

            // of the selected item

        }

    });
}

```



```
        return builder.create();  
    }  
}
```

### Single-choice list dialog

It is used to add a single choice list to the Dialog box. We can check or uncheck as per user choice.

```
public Dialog onCreateDialog(Bundle savedInstanceState) {  
  
    mSelectedItems = new ArrayList();  
  
    AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());  
  
    builder.setTitle("This is list choice dialog box");  
  
    .setMultiChoiceItems(R.array.toppings, null,  
  
        new DialogInterface.OnMultiChoiceClickListener()  
  
        { @Override  
  
        public void onClick(DialogInterface dialog, int which, boolean  
  
            isChecked) if (isChecked) {  
  
                // If the user checked the item, add it to the selected  
  
                items mSelectedItems.add(which);  
  
            }  
  
            else if (mSelectedItems.contains(which)) {  
  
                // Else, if the item is already in the array, remove  
  
                it mSelectedItems.remove(Integer.valueOf(which));  
  
            }  
  
        }  
  
    })  
}
```

```

        // Set the action buttons

        .setPositiveButton(R.string.ok, new DialogInterface.OnClickListener()

        { @Override

        public void onClick(DialogInterface dialog, int id) {

            // User clicked OK, so save the mSelectedItems results somewhere

            // or return them to the component that opened the dialog

            ...

        }

    })

    .setNegativeButton(R.string.cancel, new DialogInterface.OnClickListener()

    { @Override

    public void onClick(DialogInterface dialog, int id) {

        ...

    }

    });

    return builder.create();

}

```

**Here is the modified code of src/MainActivity.java**

```

package com.example.sairamkrishna.myapplication;

import android.app.AlertDialog;

import android.content.DialogInterface;

import android.support.v7.app.ActionBarActivity;

```

```

import android.os.Bundle;

import android.view.View;

import android.widget.Toast;

public class MainActivity extends ActionBarActivity

{ @Override

protected void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);

    setContentView(R.layout.activity_main);

}

public void open(View view){

    AlertDialog.Builder alertDialogBuilder = new

    AlertDialog.Builder(this); alertDialogBuilder.setMessage("Are you

    sure,

        You wanted to make decision");

    alertDialogBuilder.setPositiveButton("yes"

    ,

        new DialogInterface.OnClickListener()

        { @Override

        public void onClick(DialogInterface arg0, int arg1) {

            Toast.makeText(MainActivity.this,"You clicked yes

                button",Toast.LENGTH_LONG).show();

            }

        });

    alertDialogBuilder.setNegativeButton("No",ne

w DialogInterface.OnClickListener() {

```

```

        Override

        public void onClick(DialogInterface dialog, int which)

            { finish();

            }

        });

        AlertDialog alertDialog = alertDialogBuilder.create();

        alertDialog.show();

    }

}

```

Here is the modified code of res/layout/activity\_main.xml

In the below code abc indicates the logo of [tutorialspoint.com](http://tutorialspoint.com)

```

<?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:paddingLeft="@dimen/activity_horizontal_margin"

    "

    android:paddingRight="@dimen/activity_horizontal_margi

    n"

    android:paddingTop="@dimen/activity_vertical_margin"

    android:paddingBottom="@dimen/activity_vertical_margin"

    " tools:context=".MainActivity"

    <TextView

```

```
        android:layout_width="wrap_content"

        android:layout_height="wrap_content"

        android:text="Alert Dialog"

        android:id="@+id/textView"

        android:textSize="35dp"

        android:layout_alignParentTop="true"

        android:layout_centerHorizontal="true" />
```

```
<TextView
```

```
        android:layout_width="wrap_content"

        android:layout_height="wrap_content"

        android:text="Tutorialspoint"

        android:id="@+id/textView2"

        android:textColor="#ff3eff0f"

        android:textSize="35dp"

        android:layout_below="@+id/textView"

        android:layout_centerHorizontal="true" />
```

```
<ImageView
```

```
        android:layout_width="wrap_content"

        android:layout_height="wrap_content"

        android:id="@+id/imageView"

        android:src="@drawable/abc"

        android:layout_below="@+id/textView2
```

```
"
```

```

        android:layout_alignRight="@+id/textView2"

        android:layout_alignEnd="@+id/textView2"

        android:layout_alignLeft="@+id/textView"

        android:layout_alignStart="@+id/textView" />
<Button

        android:layout_width="wrap_content"

        android:layout_height="wrap_content"

        android:text="Alert dialog"

        android:id="@+id/button"

        android:layout_below="@+id/imageView"

        android:layout_alignRight="@+id/textView2"

        android:layout_alignEnd="@+id/textView2"

        android:layout_marginTop="42dp"

        android:onClick="open"

        android:layout_alignLeft="@+id/imageView"

        android:layout_alignStart="@+id/imageView" />

```

```
</RelativeLayout>
```

Here is

ofStrings.xml

```
<resources>
```

```
    <string name="app_name">My Application</string>
```

```
</resources>
```

Here is the default code of AndroidManifest.xml

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

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

    package="com.example.sairamkrishna.myapplication" >

    <application

        android:allowBackup="true"

        android:icon="@drawable/ic_launcher

        " android:label="@string/app_name"

        android:theme="@style/AppTheme" >

        <activity

            android:name="com.example.sairamkrishna.myapplication.MainActivity"

            android:label="@string/app_name" >

                <intent-filter>

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

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

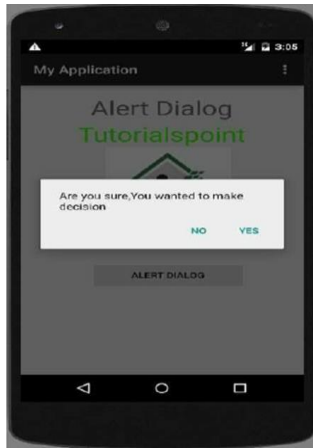
                </intent-filter>

            </activity>

        </application>

    </manifest>
```

Let's try to run your application. I assume you have connected your actual Android Mobile device with your computer. To run the app from Android studio, open one of your project's activity files and click Run Eclipse Run Icon icon from the toolbar. Before starting your application, ]Android studio will display the following window to select an option where you want to run your Android application.



Select your options and then click on it. For suppose, if you have clicked on yes button, then result would as follows



if you click on no button it will call finish() and it will close your application.



## PRACTICAL 7

### Programs on Intents, Events, Listeners, and Adapters

#### **The Android Intent Class, Using Events and Event Listeners**

Android Intent to launch various Android built-in

applications

Step	Description
------	-------------

- |   |   |
|---|---|
| 1 | You will use Android studio IDE to create an Android application and name it as My Application under a package com.example.saira_000.myapplication. |
| 2 | Modify src/main/java/MainActivity.java file and add the code to define two listeners corresponding two buttons ie. Start Browser and Start Phone.   |
| 3 | Modify layout XML file res/layout/activity_main.xml to add three buttons in linear layout.  |
| 4 | Run the application to launch the Android emulator and verify the result of the changes done in the application.                                    |

Following is the content of the modified main activity file

**src/com.example.MyApplication/MainActivity.java.**

```

package com.example.saira_000.myapplication;

import android.content.Intent;
import android.net.Uri;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import
android.widget.Button;

public class MainActivity extends AppCompatActivity
{ Button b1,b2;

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

    b1=(Button)findViewById(R.id.button);
    b1.setOnClickListener(new View.OnClickListener() {

        @Override
        public void onClick(View v) {
            Intent i = new Intent(android.content.Intent.ACTION_VIEW,
                Uri.parse("http://www.example.com"));
            startActivity(i);
        }
    });

    b2=(Button)findViewById(R.id.button2);
    b2.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Intent i = new Intent(android.content.Intent.ACTION_VIEW,
                Uri.parse("tel:9510300000"));
            startActivity(i);
        }
    });
}
}

```

**Following will be the content of res/layout/activity\_main.xml file -**

```

<?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:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"

```

```
android:paddingTop="@dimen/activity_vertical_margin"
android:paddingBottom="@dimen/activity_vertical_margi
n" tools:context=".MainActivity">
```

```
<TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Intent Example"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:textSize="30dp" />
```

```
<TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Tutorials point"
    android:textColor="#ff87ff09"
    android:textSize="30dp"
    android:layout_below="@+id/textView1"
    android:layout_centerHorizontal="true" />
```

```
<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/imageButton"
    android:src="@drawable/abc"
    android:layout_below="@+id/textView2"
    android:layout_centerHorizontal="true" />
```

```
<EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/editText"
    android:layout_below="@+id/imagutton"
    android:layout_alignRight="@+id/imageButton"
    android:layout_alignEnd="@+id/imageButton" />
```

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Start Browser"
    android:id="@+id/button"
    android:layout_alignTop="@+id/editText"
    android:layout_alignRight="@+id/textView1"
    android:layout_alignEnd="@+id/textView1"
    android:layout_alignLeft="@+id/imageButton"
    android:layout_alignStart="@+id/imageButton"
    />
```

```

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Start Phone"
    android:id="@+id/button2"
    android:layout_below="@+id/button"
    android:layout_alignLeft="@+id/button"
    android:layout_alignStart="@+id/button"
    android:layout_alignRight="@+id/textView2"
    android:layout_alignEnd="@+id/textView2" />
</RelativeLayout>

```

Following will be the content of res/values/strings.xml to define two new constants

```

-
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">My Application</string>
</resources>

```

Following is the default content of AndroidManifest.xml -

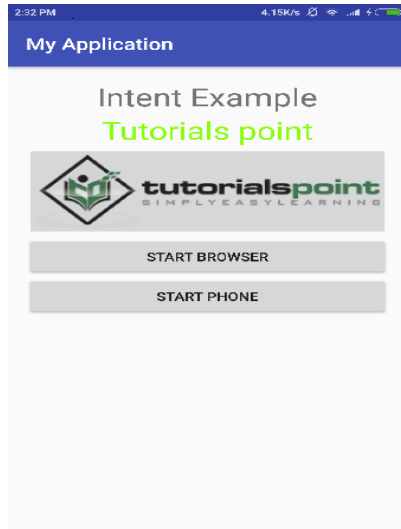
```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.saira_000.myapplication">

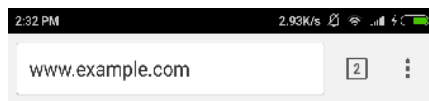
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        "
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
      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

```

run your My Application application. I assume you had created your AVD while doing environment setup. To run the app from Android Studio, open one of your project's activity files and click Run Eclipse Run Icon icon from the toolbar. Android Studio installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display following Emulator window:



Now click on Start Browser button, which will start a browser configured and display <http://www.example.com> as shown below –



## Example Domain

This domain is established to be used for illustrative examples in documents. You may use this domain in examples without prior coordination or asking for permission.

[More information...](#)

## Program on events and event listener:

Following is the content of the modified main activity file `src/com.example.myapplication/MainActivity.java`. This file can include each of the fundamental lifecycle methods.

```
package com.example.myapplication;

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

public class MainActivity extends AppCompatActivity
{
    private ProgressDialog progress;
    Button b1,b2;
```

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    progress = new ProgressDialog(this);

    b1=(Button)findViewById(R.id.button);
    b2=(Button)findViewById(R.id.button2);
    b1.setOnClickListener(new View.OnClickListener() {

        @Override
        public void onClick(View v) {
            TextView txtView = (TextView) findViewById(R.id.textView);
            txtView.setTextSize(25);
        }
    });
    b2.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            TextView txtView = (TextView) findViewById(R.id.textView);
            txtView.setTextSize(55);
        }
    });
}
}

```

Following will be the content of res/layout/activity\_main.xml file –

Here abc indicates about tutorialspoint logo

```

<?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="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Event Handling "
    >

```

```

        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        " android:textSize="30dp"/>

<TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Tutorials point "
    android:textColor="#ff87ff09"
    android:textSize="30dp"
    android:layout_above="@+id/imageButton"
    "

    android:layout_centerHorizontal="true"
    android:layout_marginBottom="40dp" />

<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/imageButton"
    android:src="@drawable/abc"
    android:layout_centerVertical="true"
    android:layout_centerHorizontal="true" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Small font"
    android:id="@+id/button"
    android:layout_below="@+id/imageButton"
    android:layout_centerHorizontal="true" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Large Font"
    android:id="@+id/button2"
    android:layout_below="@+id/button"
    android:layout_alignRight="@+id/button"
    android:layout_alignEnd="@+id/button" />

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!"
    android:id="@+id/textView"
    android:layout_below="@+id/button2"
    android:layout_centerHorizontal="true"
    " android:textSize="25dp" />

```

```
</RelativeLayout>
```

Following will be the content of res/values/strings.xml to define two new constants –

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">myapplication</string>
</resources>
```

Following is the default content of AndroidManifest.xml –

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

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >

        <activity
            android:name="com.example.myapplication.MainActivity"
            android:label="@string/app_name" >

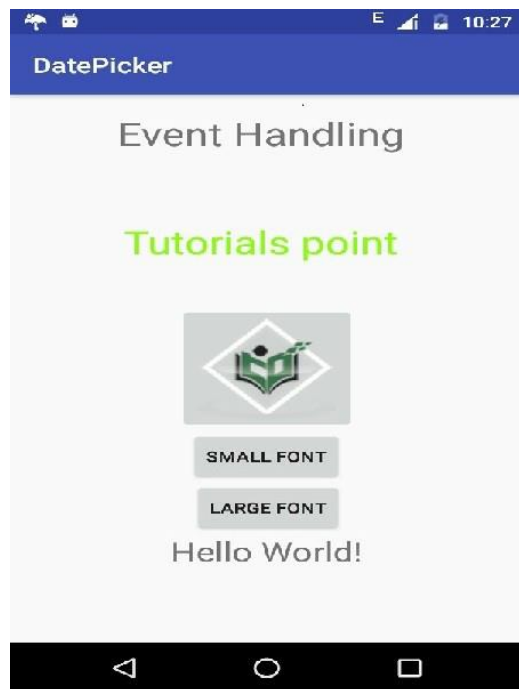
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>

        </activity>

    </application>
</manifest>
```

**Output:**





# PRACTICAL 8

## Programming Media API and Telephone API.

### AUDIO

#### AudioDemo.java

Code:

```
package org.example.audio;

import android.app.Activity;
import android.media.MediaPlayer;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.Toast;

public class AudioDemo extends Activity implements OnClickListener {
    private static final String TAG = "AudioDemo";
    private static final String isPlaying = "Media is Playing";
    private static final String notPlaying = "Media has stopped Playing";

    MediaPlayer player;
    Button playerButton;

    public void onClick(View v) {
```

```

        Log.d(TAG, "onClick: " + v);
        if (v.getId() == R.id.play)
            { playPause();
            }
    }

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        player = MediaPlayer.create(this, R.raw.robotrock);
        player.setLooping(false); // Set looping

        // Get the button from the view
        playerButton = (Button) this.findViewById(R.id.play);
        playerButton.setText(R.string.stop_label);
        playerButton.setOnClickListener(this);

        // Begin playing selected
        media demoPlay();

        // Release media instance to
        system player.release();
    }

    @Override
    public void onPause() {
        super.onPause();
        player.pause();
    }

    // Initiate media player
    private void
    demoPause() {
        player.pause();
        playerButton.setText(R.string.play_label)
        ;
        Toast.makeText(this, notPlaying, Toast.LENGTH_LONG).show();
        Log.d(TAG, notPlaying);
    }

    // Initiate playing the media
    private void demoPlay() {
        player.start();
        playerButton.setText(R.string.stop_label);
        Toast.makeText(this, isPlaying,
        Toast.LENGTH_LONG).show(); Log.d(TAG, isPlaying);
    }

```

```

        // Toggle between the play and pause
        private void playPause() {
            if(player.isPlaying()) {
                demoPause();
            } else {
                demoPlay();
            }
        }
    }
}

```

### **main.xml**

Layout file

Code:

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="https://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="@string/hello"
    />

    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:id="@+id/play"
        android:text="@string/play_label"></Button>
</LinearLayout>

```

strings.xml

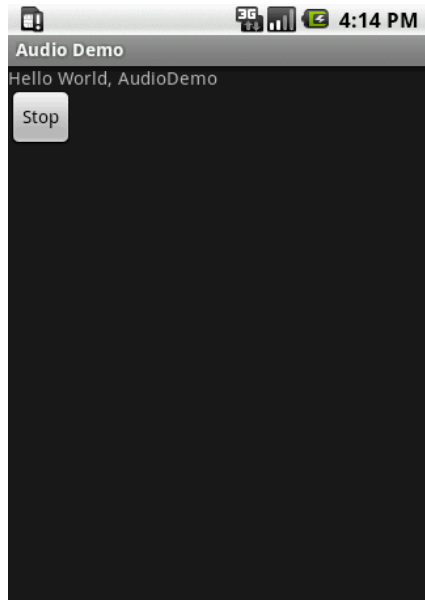
Code:

```

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, AudioDemo</string>
    <string name="app_name">Audio Demo</string>
<string name="play_label">Play</string>
<string name="stop_label">Stop</string>
</resources>

```

**Output:**



## VIDEO

activity\_main.xml

Drag the **VideoView** from the palette, now the **activity\_main.xml** file will like this:

**File: 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"
    " tools:context=".MainActivity" >

    <VideoView
        android:id="@+id/videoView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        "
```

```
        android:layout_alignParentLeft="true"

        android:layout_centerVertical="true" />

</RelativeLayout>
```

## **Activity class**

**File: MainActivity.java**

```
package com.example.video1;

import android.net.Uri;

import android.os.Bundle;

import android.app.Activity;

import android.view.Menu;

import android.widget.MediaController;

import android.widget.VideoView;

public class MainActivity extends Activity {

    @Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_main);

        VideoView videoView = (VideoView) findViewById(R.id.videoView1);
```

```

        //Creating MediaController

MediaController mediaController= new

        MediaController(this);

        mediaController.setAnchorView(videoView);


        //specify the location of media
        file Uri

uri=Uri.parse(Environment.getExternalStorageDirectory().getPath()+"/media/1.m
p4 ");


        //Setting MediaController and URI, then starting the videoView

videoView.setMediaController(mediaController);

videoView.setVideoURI(uri);

videoView.requestFocus();

videoView.start();

    }


    @Override

    public boolean onCreateOptionsMenu(Menu menu) {

        // Inflate the menu; this adds items to the action bar if it is
present.

        getMenuInflater().inflate(R.menu.activity_main, menu);

        return true;

    }

```

```
}
```

## Telephone API

*File: 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:paddingBottom="@dimen/activity_vertical_margin"

"

android:paddingLeft="@dimen/activity_horizontal_margin"

"

android:paddingRight="@dimen/activity_horizontal_margi
n"

android:paddingTop="@dimen/activity_vertical_margin"

tools:context=".MainActivity" >

<TextView

    android:id="@+id/textView1"

    android:layout_width="wrap_content"

    android:layout_height="wrap_content"

    android:layout_alignParentLeft="true"

    "

    android:layout_alignParentTop="true"

    android:layout_marginLeft="38dp"

    android:layout_marginTop="30dp"
```



```
android:text="Phone Details:" />
```

```
</RelativeLayout>
```

### **Activity class**

Now, write the code to display the information about the telephony services.

#### ***File: MainActivity.java***

```
package com.javatpoint.telephonymanager;

import android.os.Bundle;

import android.app.Activity;

import
android.content.Context;

import android.telephony.TelephonyManager;

import android.view.Menu;

import android.widget.TextView;

public class MainActivity extends Activity {

    TextView textView1;

    @Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_main);

        textView1=(TextView)findViewById(R.id.textView1);

        //Get the instance of
        TelephonyManager TelephonyManager

        tm=(TelephonyManager) getSystemService(Context.TELEPHONY_SERVICE);
```

```
//Calling the methods of TelephonyManager the returns the  
information
```

```
String IMEINumber=tm.getDeviceId();  
  
String subscriberID=tm.getDeviceId();  
  
String SIMSerialNumber=tm.getSimSerialNumber();  
  
String  
  
networkCountryISO=tm.getNetworkCountryIso();  
  
String SIMCountryISO=tm.getSimCountryIso();  
  
String  
  
softwareVersion=tm.getDeviceSoftwareVersion();  
  
String voiceMailNumber=tm.getVoiceMailNumber();
```

```
//Get the phone type
```

```
String strphoneType="";
```

```
int phoneType=tm.getPhoneType();
```

```
switch (phoneType)
```

```
{
```

```
    case
```

```
        (TelephonyManager.PHONE_TYPE  
_CDMA): strphoneType="CDMA";  
  
        break;
```

```
    case
```

```
        (TelephonyManager.PHONE_TYP  
E_GSM): strphoneType="GSM";
```

```
break;
```

```
case (TelephonyManager.PHONE_TYPE_NONE):
```

```

        strphoneType="NONE";

        break;

    }

    //getting information if phone is in roaming

    boolean isRoaming=tm.isNetworkRoaming();

    String info="Phone Details:\n";

    info+="\n IMEI Number:"+IMEINumber;

    info+="\n

    SubscriberID:"+subscriberID;

    info+="\n Sim Serial Number:"+SIMSerialNumber;

    info+="\n Network Country

    ISO:"+networkCountryISO; info+="\n SIM Country

    ISO:"+SIMCountryISO; info+="\n Software

    Version:"+softwareVersion; info+="\n Voice Mail

    Number:"+voiceMailNumber; info+="\n Phone Network

    Type:"+strphoneType; info+="\n In Roaming?

    :"+isRoaming;

    textView1.setText(info);//displaying the information in the
textView

    }

    }

```

### AndroidManifest.xml

You need to provide **READ\_PHONE\_STATE** permission in the AndroidManifest.xml file.

*File: AndroidManifest.xml*

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

<manifest

    xmlns:android="http://schemas.android.com/apk/res/android"

    package="com.javatpoint.telephonymanager"

    android:versionCode="1"

    android:versionName="1.0" >

    <uses-sdk

        android:minSdkVersion="8"

        android:targetSdkVersion="17" />

    <uses-permission android:name="android.permission.READ_PHONE_STATE"/>

    <application

        android:allowBackup="true"

        android:icon="@drawable/ic_launcher"

        android:label="@string/app_name"

        android:theme="@style/AppTheme" >

        <activity

            android:name="com.javatpoint.telephonymanager.MainActivity"

            android:label="@string/app_name" >

            <intent-filter>

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

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

    </intent-filter>

</activity>

</application>

</manifest>
```

Output:

