

Version 2.1

ANDROID APP DEVELOPMENT

By AbhiAndroid



Awesome

Thank you for downloading this guide...

Hi,

My name is Abhishek. I'm 26 year old guy from India.

I'm on mission and my mission is to help people learn and grow as an Android Developer.

So, that's why I created AbhiAndroid to reach more and more people like you.

I have created this eBook with love for those developer who want to take a deeper dive in Android App Development and want to drive their skills to the next level.

With love & respect,

Abhishek Saini

You can [connect with me on Facebook](#)

Live Android App Project Source Code

Before you start reading this guide, I would like to share premium live Android App project source code build specially for you. These source code are available at a very affordable price giving chance for developer and student to understand how complex Android App are built.

With each source code you will get step by step documentation and 6 months of FREE email support.

You can grab the source code here: <http://abhiandroid.com/sourcecode/>

Service

I also offer Android App development service and you can consider hiring me. Some of my services include developing App from scratch, adding new feature in App, fixing bugs, enhancing Android App or customization.

You can know more about my services here: <http://abhiandroid.com/service/> To reach us please email: info@abhiandroid.com

Important Message

This copy is purely dedicated to you. You can use it in several ways. Save it in your laptop, mobile, take a printout, and please, no need to say thanks. But you can't sell it or you can't make a change in it because all rights of this copy is with AbhiAndroid.com. If want some changes in it or some addition to it, you can mail me at info@abhiandroid.com. And, if you like this guide, don't forget to share it with your buddies. I'm sure they will appreciate it.

Tables Of Content

1.	History of Android.....	Page 4
2.	XML in Android.....	Page 7
3.	Activity Lifecycle in Android.....	Page 17
4.	LinearLayout.....	Page 28
5.	Relative Layout.....	Page 36
6.	Table Layout.....	Page 54
7.	Frame Layout.....	Page 64
8.	Adapter.....	Page 73
9.	ListView.....	Page 82
10.	GridView.....	Page 94
11.	TextView.....	Page 110
12.	ScrollView.....	Page 120
13.	Spinner.....	Page 129
14.	EditText.....	Page 134
15.	Button.....	Page 148
16.	ImageView.....	Page 160
17.	ImageButton.....	Page 169
18.	CheckBox.....	Page 177
19.	Switch	Page 191
20.	RadioButton & RadioGroup.....	Page 205
21.	RatingBar.....	Page 220
22.	WebView.....	Page 232
23.	AutoCompleteTextView.....	Page 242
24.	ProgressBar.....	Page 252
25.	TimePicker.....	Page 266
26.	CalendarView.....	Page 280
27.	ExpandableListView.....	Page 248
28.	Chronometer.....	Page 317
29.	Zoom Controls.....	Page 333
30.	VideoView.....	Page 344
31.	SearchView.....	Page 357
32.	Toast.....	Page 371
33.	Intent in Android.....	Page 379
34.	Internal Storage.....	Page 390
35.	External Storage.....	Page 400
36.	Shared Preference.....	Page 410
37.	Sqlite.....	Page 424
38.	Json Parsing.....	Page 435
39.	Asynctask.....	Page 451
40.	Splash Screen.....	Page 461
41.	HTML in Android.....	Page 473
42.	Fragment	Page 485
43.	Basic Calculator App.....	Page 501
44.	Youtube Android App.....	Page 512
45.	Countdown Timer Android App.....	Page 525

History of Android

The Android is a Linux Based Operating System by GOOGLE which provide a rich application Framework and help in developing interactive applications. The OS first OS version was introduced in 2007 with many of its versions named in Alphabetical order ranging from A-N and upcoming is O.

HERE IS DETAILED ABOUT ANDROID VERSIONS :

Alpha – In this(Android 1.0) was the first versions of Android operating System by Google. It has basic functionality with a simple browser and other Google apps like Gmail, Maps and YouTube.

Beta – Later on with Android 1.1 few more functionality added, the API changes from Level 1 in Android 1.0 to Level 2. It supports attachment with MMS.

Cupcake – Cupcake was Android second version with new features as well as the Android framework API updated. It was Android 1.5 with on Screen Keyboard , Bluetooth and Updated UI for applications.

Donut – It was Android 1.6 nicknamed as DONUT. It added support for CDMA , additional screen sizes, talk to speech engine and battery indicator.

Eclair – Android 2.0-2.1 as like other versions this also come up with a nickname as ECLAIR and lot more functions & features. It come up with Bluetooth 2.1 , live wallpaper, HTML 5 support, ability to search sms & mms, flash support, digital zoom and more camera features

Froyo – Android version 2.2-2.2.3 introduced with USB tethering & WiFi hotspot functionality and apps can now be installed on memory card. Support Adobe flash, increased speed and performance of applications with new features.

Gingerbread – Gingerbread (Android 2.3-2.3.7) introduced with updated User Interface which provide more ease to use. Features are like sensors, multiple cameras(Front & back), virtual keyboard, better text suggestion, voice input capability and press hold copy paste capability.

Honeycomb – This Android platform Honeycomb was designed for large screens like tablets so interface elements like virtual keyboard optimized for bigger screen. Home screen is optimized, tabs are introduced in browser with additional incognito mode and video chat & Gtalk is supported.

Ice Cream Sandwich – Ice Cream sandwich come in 2011 bringing all new look. It gives more ease to user like user can quickly swipe to close the apps, new gallery layout and built in photo editor.

Jelly Bean – Google made Operating System more responsive with Jelly Bean and introduces file sharing with Android Beam. Restricted profile, Dial Pad complete, supported other languages like Hindi, changed camera UI.

KitKat – Kitkat come up in 2013 with API Level 19. It has wireless printing capability, new dialer id, chrome webview and screen recording.

Lollipop – Android version 5.0-5.1.1 come up with improved RAM and battery management. Further restyling through Material design, no interrupts feature, unlock phone through Bluetooth trusted devices, print previews and smart lock feature.

Marshmallow – Marshmallow was released in year 2015, come up with smarter battery and doze mode (it prevents certain task from running if the phone being setting idle), Now On Tap, better privacy settings, easier to upgrade phone, Fingerprint sensor and built in visual voice mail.

Nougat – Android Nougat was made official in 2016 with updated emoji, 72 newly added, multi window view (switch between apps with double tap), smarter battery with data saver mode, more secured and high quality virtual reality with new dimensions.

ANDROID PLATFORM VERSIONS:

Version	Codename	API	Distribution
2.2	Froyo	8	0.1%
2.3.3 – 2.3.7	Gingerbread	10	1.2%

4.0.3 – 4.0.4	Ice Cream Sandwich	15	1.2%
4.1.X	Jelly Bean	16	4.5%
4.2.X	Jelly Bean	17	6.4%
4.3	Jelly Bean	18	1.9%
4.4	Kitkat	19	24.0%
5.0	Lollipop	21	10.8%
5.1	Lollipop	22	23.2%
6.0	Marshmallow	23	26.3%
7.0	Nougat	24	0.4%

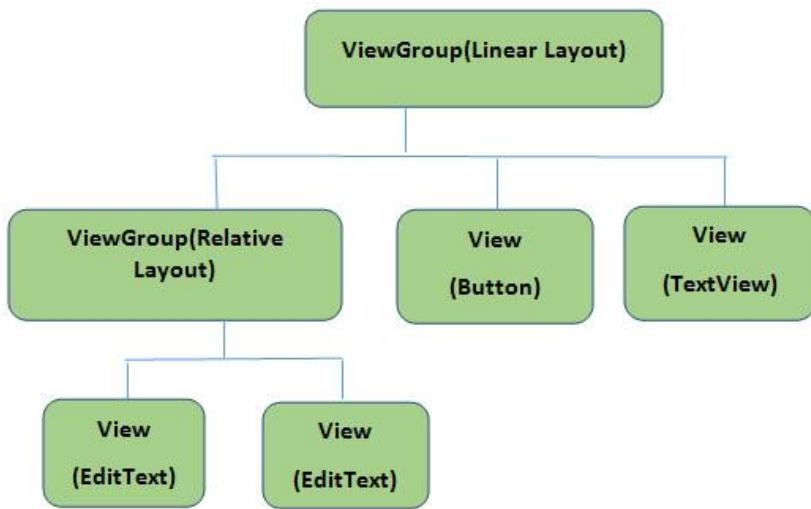
XML in Android

XML stands for Extensible Markup Language. XML is a markup language much like HTML used to describe data. XML tags are not predefined in XML. We must define our own Tags. XML as itself is well readable both by human and machine. Also, it is scalable and simple to develop. In Android we use XML for designing our layouts because XML is a lightweight language so it doesn't make our layout heavy.

In this article we will go through the basic concepts of XML in Android and different XML files used for different purposes in Android. This will help you in writing UI code to design your desired user interface.

Basics Of User Interface:

The whole concept of Android User Interface is defined using the hierarchy of View and ViewGroup objects. A ViewGroup is an invisible container that organizes child views. These child views are other widgets which are used to make the different parts of UI. One ViewGroup can have another ViewGroup as a child element as shown in the figure given below:



Here in above Diagram ViewGroup (Linear Layout) contains one ViewGroup (i.e. Relative Layout) and two View(Button and TextView). Further two more View (i.e. 2 EditText) are nested inside Relative Layout ViewGroup. It is important to note that one layout can be nested in another layout.

The below code snippet will explain the above image in better way. Paste it in activity_main.xml:

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

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button"/>

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="sample Text"
        android:layout_marginTop="15dp"
        android:textSize="30dp"/>

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content">

        <EditText
            android:id="@+id/editText1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"/>
    
```

```
    android:layout_height="match_parent">>

    <EditText
        android:id="@+id/editTextName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:hint="First Name"
    />

    <EditText
        android:id="@+id/editTextLastName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:hint="Last Name"/>

</RelativeLayout>
</LinearLayout>
```

Every Android application screen has some components like button, Text or images. These are contained inside the ViewGroup. Layouts are the best examples for ViewGroups. The different types of layout in android are Linear Layout, Relative Layout, Absolute Layout, Table Layout and Frame Layout.

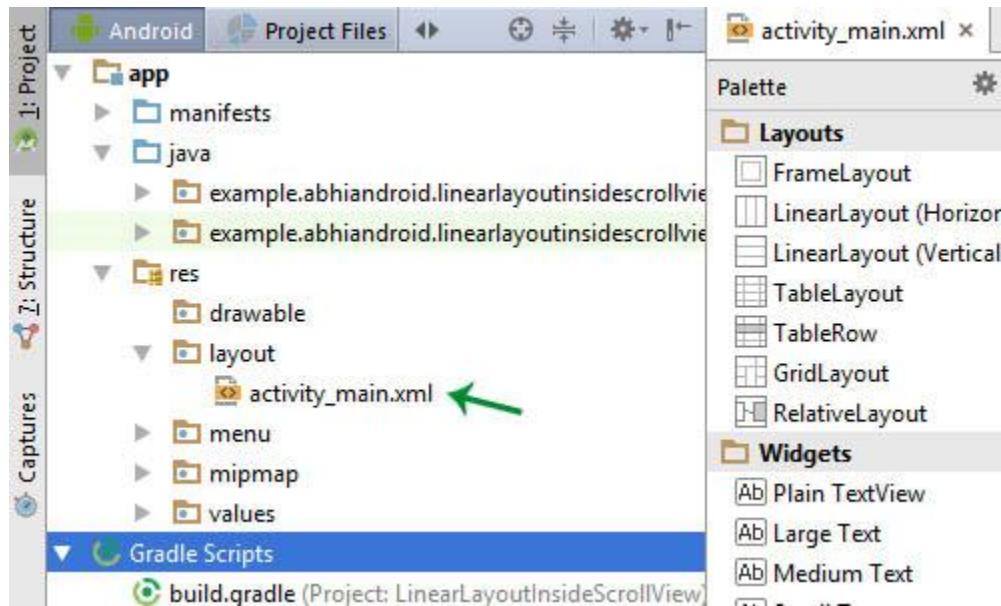
Different XML Files Used in Android:

In Android there are several xml files used for several different purposes. Below we define each and every one.

1. Layout XML Files: Layout xml files are used to define the actual UI(User interface) of our application. It holds all the elements(views) or the tools that we want to use in our application. Like the TextView's, Button's and other UI elements.

Location in Android Studio:

You will find out this file inside the res folder and inside it there is another folder named layout where you will get all the layout files for their respective activities or fragments.



Basic Layout XML Code:

Below we show activity_main.xml file in which we have two TextView's.

```
<!-- RelativeLayout in which we set green color for the background -->
<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:background="@color/greenColor"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/firstTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_margin="20dp"
        android:padding="10dp"
        android:text="First Text View"
        android:textColor="@color/white"
        android:textSize="20sp"
        android:textStyle="bold" />
```

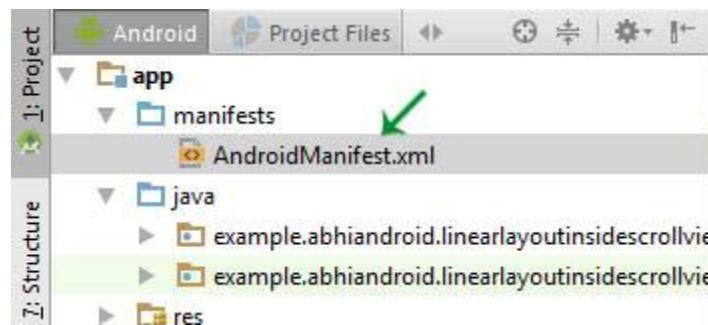
```
<!-- second TextView -->
<TextView
    android:id="@+id/secondTextView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/firstTextView"
    android:layout_centerHorizontal="true"
    android:layout_margin="20dp"
    android:padding="10dp"
    android:text="Second Text View"
    android:textColor="@color/white"
    android:textSize="20sp"
    android:textStyle="bold" />

</RelativeLayout>
```

2. Manifest xml File(Mainfest.xml**):** This xml is used to define all the components of our application. It includes the names of our application packages, our Activities, receivers, services and the permissions that our application needs. For Example – Suppose we need to use internet in our app then we need to define Internet permission in this file.

Location in Android Studio:

It is located inside app > manifests folder



Defining Internet Permission in **AndroidManifest.xml**

Below we show the **AndroidManifest.xml** file and define the Internet Permission in that file.

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

    <uses-permission android:name="ANDROID.PERMISSION.INTERNET" />
    <!-- define Internet Permission -->
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme">

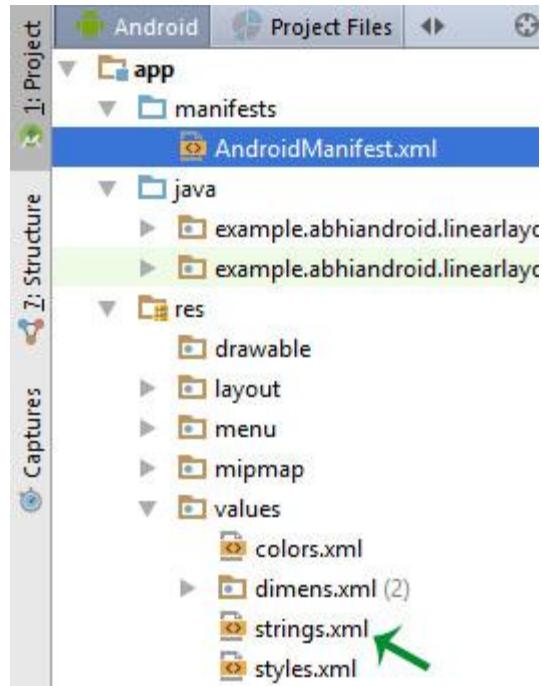
        <!-- add your Activities, Receivers, Services Names Here -->
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

3. Strings xml File(strings.xml): This xml file is used to replace the Hard-coded strings with a single string. We define all the strings in this xml file and then access them in our app(Activity or in Layout XML files) from this file. This file enhance the reusability of the code.

Location in Android Studio:



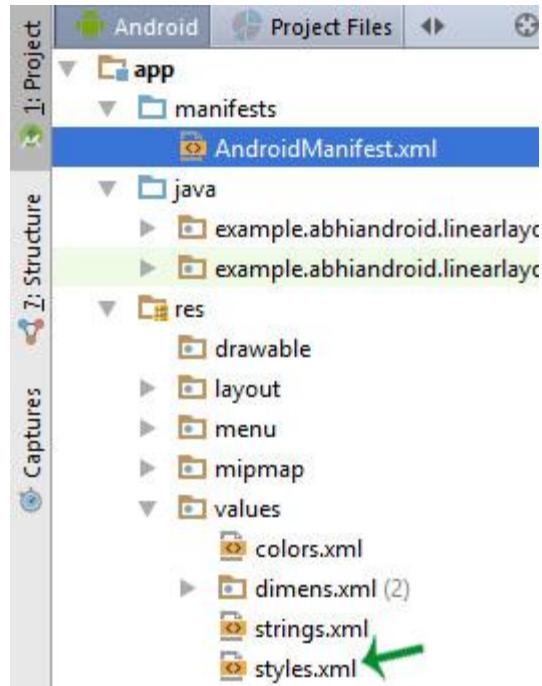
Below we show **strings.xml** file and define a string in the file.

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

<string name="hello_world">Hello world!</string>
<string name="action_settings">Settings</string>
<string name="login">User Login</string>
<!-- define your strings here -->
</resources>
```

4. Styles xml File(styles.xml): This xml is used to define different styles and looks for the UI(User Interface) of application. We define our custom themes and styles in this file.

Location in Android Studio:



Below we show the **style.xml** file.

```
<resources>

    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- Customize your theme here. -->
    </style>

</resources>
```

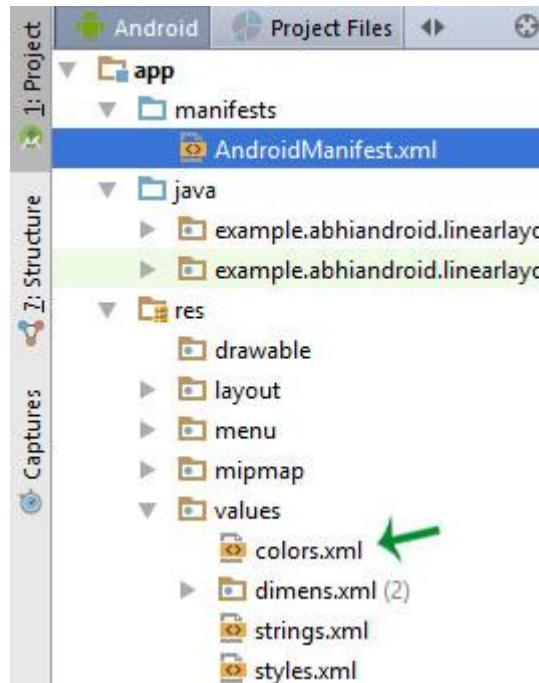
5. Drawable xml Files: These are those xml files that are used to provide various graphics to the elements or views of application. When we need to create a custom UI we use drawable xml files. Suppose if we need to define a gradient color in the background of Button or any custom shape for a view then we create a Drawable xml file and set it in the background of View.

Below we show custom_drawable.xml file and create a gradient background color using style attribute.

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android">
<!-- define start, center and end color for gradient -->
<gradient
    android:centerColor="#0f0"
    android:endColor="#00f"
    android:startColor="#f00" />
</shape>
```

6. Color xml File (colors.xml): This file is used to define the color codes that we used in our app. We simply define the color's in this file and used them in our app from this file.

Location in Android Studio

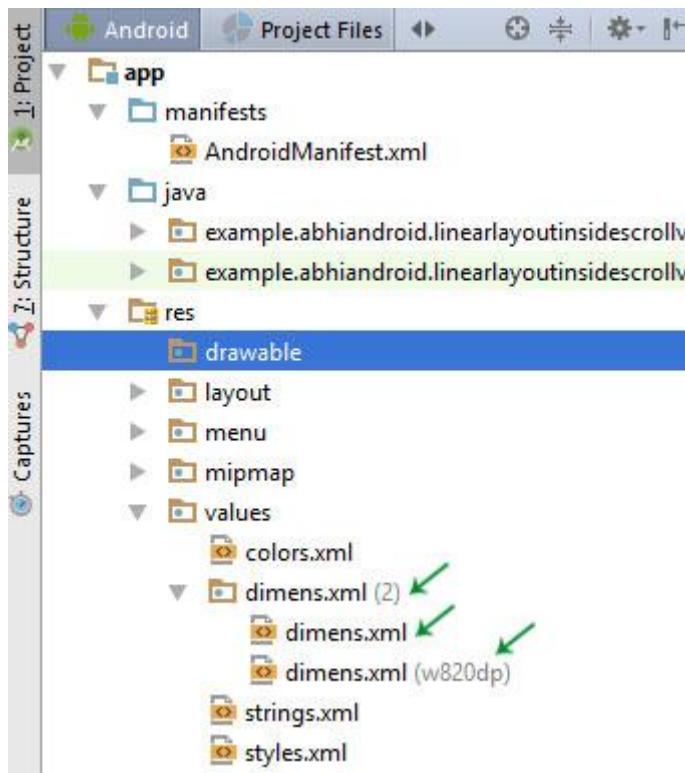


Below we show the **colors.xml** file in which we define green and white color.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!-- define your colors Here -->
    <color name="greenColor">#0f0</color>
    <color name="white">#fff</color>
</resources>
```

7. Dimension xml File(dimens.xml): This xml file is used to define the dimensions of the View's. Suppose we need a Button with 50dp(density pixel) height then we define the value 50dp in dimens.xml file and then use it in our app from this file.

Location in Android Studio:



Below we show the dimens.xml file in which we define 50dp dimension for Button height.

```
<resources>
<!-- Default screen margins, per the Android Design guidelines. -->
<dimen name="activity_horizontal_margin">16dp</dimen>
<dimen name="activity_vertical_margin">16dp</dimen><dimen name="btnheight">50dp</dimen>
</resources>
```

Activity Lifecycle in Android

Activity Lifecycle: Activity is one of the building blocks of Android OS. In simple words Activity is a screen that user interact with. Every Activity in android has lifecycle like created, started, resumed, paused, stopped or destroyed. These different states are known as Activity Lifecycle. In other words we can say Activity is a class pre-written in Java Programming.

Below is Activity Lifecycle Table:

Short description of Activity Lifecycle example:

onCreate() – Called when the activity is first created

onStart() – Called just after it's creation or by restart method after onStop(). Here Activity start becoming visible to user

onResume() – Called when Activity is visible to user and user can interact with it

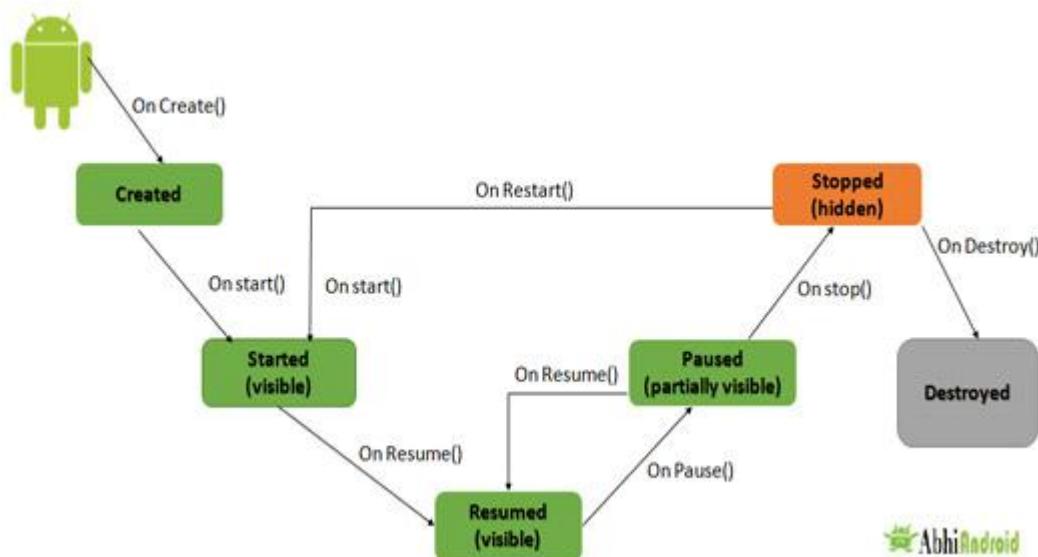
onPause() – Called when Activity content is not visible because user resume previous activity

onStop() – Called when activity is not visible to user because some other activity takes place of it

onRestart() – Called when user comes on screen or resume the activity which was stopped

onDestroy() – Called when Activity is not in background

Below Activity Lifecycle Diagram Shows Different States:



Different Types of Activity Lifecycle States:

Activity have different states or it's known as Activity life cycle. All life cycle methods aren't required to override but it's quite important to understand them. Lifecycles methods can be overridden according to requirements.

LIST OF ACTIVITY LIFECYCLE METHODS OR STATES:

Activity Created: `onCreate(Bundle savedInstanceState)`:

`onCreate()` method is called when activity gets memory in the OS. To use create state we need to override `onCreate(Bundle savedInstanceState)` method. Now there will be question in mind

what is Bundle here, so Bundle is a data repository object that can store any kind of primitive data and this object will be null until some data isn't saved in that.

When an Activity first call or launched then onCreate(Bundle savedInstanceState) method is responsible to create the activity.

When ever orientation(i.e. from horizontal to vertical or vertical to horizontal) of activity gets changed or when an Activity gets forcefully terminated by any Operating System then savedInstanceState i.e. object of Bundle Class will save the state of an Activity.

It is best place to put initialization code.

Activity Started: onStart():

onStart() method is called just after it's creation. In other case Activity can also be started by calling restart method i.e after activity stop. So this means onStart() gets called by Android OS when user switch between applications. For example, if a user was using Application A and then a notification comes and user clicked on notification and moved to Application B, in this case Application A will be paused. And again if a user again click on app icon of Application A then Application A which was stopped will again gets started.

Activity Resumed:.onResume():

Activity resumed is that situation when it is actually visible to user means the data displayed in the activity is visible to user. In lifecycle it always gets called after activity start and in most use case after activity paused (onPause).

Activity Paused: onPause():

Activity is called paused when it's content is not visible to user, in most case onPause() method called by Android OS when user press Home button (Center Button on Device) to make hide.

Activity also gets paused before stop called in case user press the back navigation button. The activity will go in paused state for these reasons also if a notification or some other dialog is overlaying any part (top or bottom) of the activity (screen). Similarly, if the other screen or dialog is transparent then user can see the screen but cannot interact with it. For example, if a call or notification comes in, the user will get the opportunity to take the call or ignore it.

Activity Stopped: onStop():

Activity is called stopped when it's not visible to user. Any activity gets stopped in case some other activity takes place of it. For example, if a user was on screen 1 and click on some button and moves to screen 2. In this case Activity displaying content for screen 1 will be stopped.

Every activity gets stopped before destroy in case of when user press back navigation button. So Activity will be in stopped state when hidden or replaced by other activities that have been launched or switched by user. In this case application will not present anything useful to the user directly as it's going to stop.

Activity Restarted: onRestart():

Activity is called in restart state after stop state. So activity's onRestart() function gets called when user comes on screen or resume the activity which was stopped. In other words, when Operating System starts the activity for the first time onRestart() never gets called. It gets called only in case when activity is resumes after stopped state.

Activity Destroyed: onDestroy():

Any activity is known as in destroyed state when it's not in background. There can different cases at what time activity get destroyed.

First is if user pressed the back navigation button then activity will be destroyed after completing the lifecycle of pause and stop.

In case if user press the home button and app moves to background. User is not using it no more and it's being shown in recent apps list. So in this case if system required resources need to use somewhere else then OS can destroy the Activity.

After the Activity is destroyed if user again click the app icon, in this case activity will be recreated and follow the same lifecycle again. Another use case is with Splash Screens if there is call to finish() method from onCreate() of an activity then OS can directly call onDestroy() with calling onPause() and onStop().

Activity Lifecycle Example:

In the below example we have used the below JAVA and Android topics:

JAVA Topics Used: Method Overriding, static variable, package, Inheritance, method and class.

Android Topic Used: We have used Log class which is used to printout message in Logcat. One of the important use of Log is in debugging.

First we will create a new Android Project and name the activity as HomeActivity. In our case we have named our App project as Activity Lifecycle Example.

We will initialize a static String variable with the name of the underlying class using getSimpleName() method. In our case HOME_ACTIVITY_TAG is the name of the String variable which store class name HomeActivity.

```
private static final String HOME_ACTIVITY_TAG = HomeActivity.class.getSimpleName();
```

Now we will create a new method which will print message in Logcat.

```
private void showLog(String text){  
    Log.d(HOME_ACTIVITY_TAG, text);  
}
```

Now we will override all activity lifecycle method in Android and use showLog() method which we creating for printing message in Logcat.

```
@Override  
  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        showLog("Activity Created");  
    }  
  
@Override  
  
protected void onRestart(){  
    super.onRestart(); //call to restart after onStop  
    showLog("Activity restarted");  
}  
  
@Override  
  
protected void onStart() {  
    super.onStart(); //soon be visible  
    showLog("Activity started");  
}  
  
@Override  
  
protected void onResume() {  
    super.onResume(); //visible  
    showLog("Activity resumed");  
}
```

```
@Override  
  
protected void onPause() {  
    super.onPause(); //invisible  
    showLog("Activity paused");  
}  
  
@Override  
  
protected void onStop() {  
    super.onStop();  
    showLog("Activity stopped");  
}  
  
@Override  
  
protected void onDestroy() {  
    super.onDestroy();  
    showLog("Activity is being destroyed");  
}
```

Complete JAVA code of HomeActivity.java:

```
package com.abhiandroid.activitylifecycleexample;  
  
import android.os.Bundle;  
import android.support.design.widget.FloatingActionButton;  
import android.support.design.widget.Snackbar;  
import android.support.v7.app.AppCompatActivity;  
import android.support.v7.widget.Toolbar;  
import android.util.Log;  
import android.view.View;  
import android.view.Menu;  
import android.view.MenuItem;  
  
  
public class HomeActivity extends AppCompatActivity {  
    private static final String HOME_ACTIVITY_TAG = HomeActivity.class.getSimpleName();  
  
    private void showLog(String text){  
        Log.d(HOME_ACTIVITY_TAG, text);  
    }
```

```
}

@Override

public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    showLog("Activity Created");
}

@Override

protected void onRestart(){
    super.onRestart();//call to restart after onStop
    showLog("Activity restarted");
}

@Override

protected void onStart() {
    super.onStart();//soon be visible
    showLog("Activity started");
}

@Override

protected void onResume() {
    super.onResume();//visible
    showLog("Activity resumed");
}

@Override

protected void onPause() {
    super.onPause();//invisible
    showLog("Activity paused");
}

@Override

protected void onStop() {
    super.onStop();
}
```

```
        showLog("Activity stopped");

    }

    @Override

    protected void onDestroy() {

        super.onDestroy();

        showLog("Activity is being destroyed");

    }

}
```

When creating an Activity we need to register this in AndroidManifest.xml file. Now question is why need to register? **It's actually because manifest file has the information which Android OS read very first.** When registering an activity other information can also be defined within manifest like Launcher Activity (An activity that should start when user click on app icon).

Here is declaration example in AndroidManifest.xml file

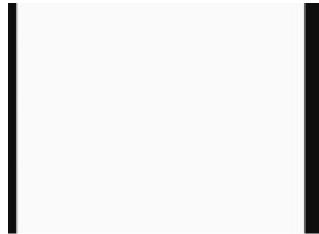
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.abhiandroid.homeactivity"

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

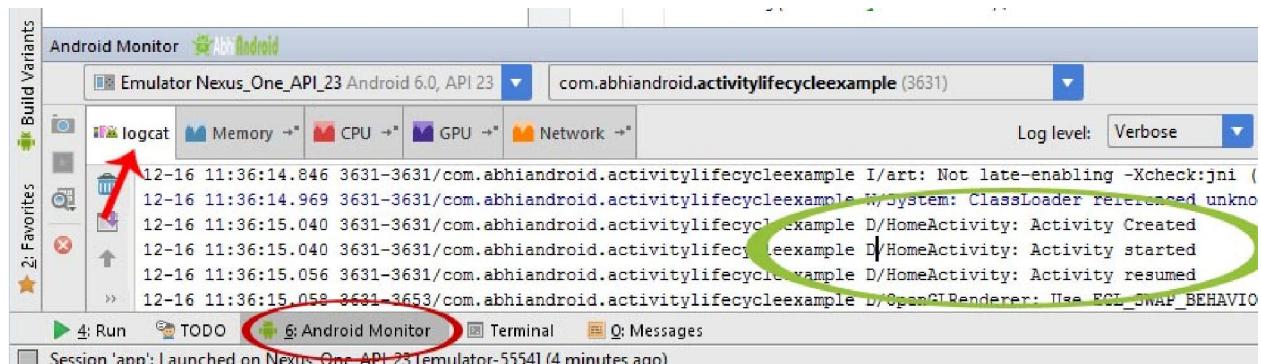
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Output Of Activity Lifecycle:

When you will run the above program you will notice a blank white screen will open up in Emulator. You might be wondering where is default Hello world screen. Actually we have removed it by overriding `onCreate()` method. Below is the blank white screen that will pop up.



Now go to Logcat present inside Android Monitor: Scroll up and you will notice three methods which were called: Activity Created, Activity started and Activity resumed.



So this clears:

first `onCreate()` method was called when activity was created

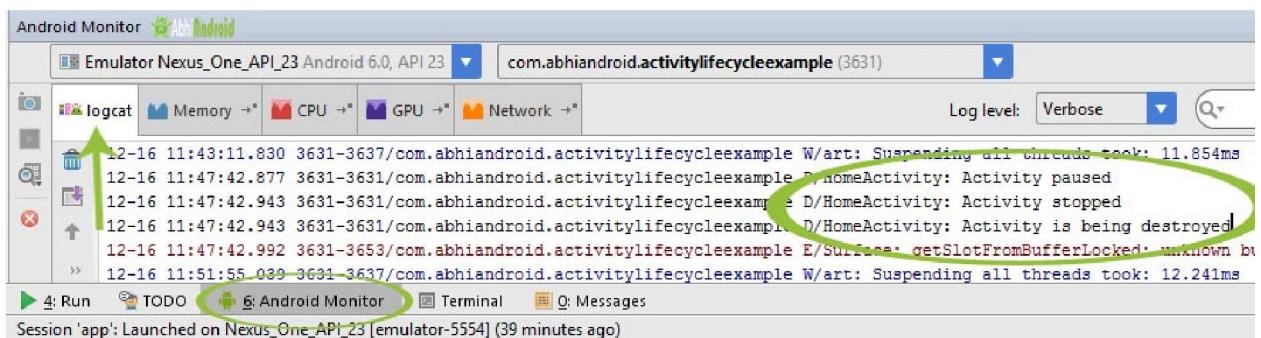
second `onStart()` method was called when activity start becoming visible to user

Finally `onResume()` method was called when activity is visible to user and user can interact with it

Now press the back button on the Emulator and exit the App:



Go to Logcat again and scroll down to bottom. You will see 3 more methods were called:
Activity paused, Activity stopped and Activity is being destroyed.



So this clears:

- ❖ onPause() method was called when user resume previous activity
- ❖ onStop() method was called when activity is not visible to user
- ❖ Last onDestroy() method was called when Activity is not in background

Important Note: In the above example onRestart() won't be called because there was no situation when we can resume the onStart() method again. In future example we will show you onRestart() in action as well.

Importance Of Activity Life Cycle:

Activity is the main component of Android Application, as every screen is an activity so to create any simple app first we have to start with Activities. Every lifecycle method is quite important to implement according to requirements, However onCreate(Bundle state) is always needed to implement to show or display some content on screen.

Linear Layout

Linear layout is a simple layout used in android for layout designing. In the Linear layout all the elements are displayed in linear fashion means all the childs/elements of a linear layout are displayed according to its orientation. The value for orientation property can be either horizontal or vertical.



Types Of Linear Layout Orientation

There are two types of linear layout orientation:

1. Vertical
2. Horizontal

As the name specified these two orientations are used to arrange there child one after the other, in a line, either vertically or horizontally. Let's we describe these in detail.

1.Vertical:

In this all the child are arranged vertically in a line one after the other. In below code snippets we have specified orientation “vertical” so the childs/views of this layout are displayed vertically.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical" > <!-- Vertical Orientation set -->

    <!-- Child Views(In this case 2 Button) are here -->

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button1"
        android:id="@+id/button"
        android:background="#358a32" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button2"
        android:id="@+id/button2"
        android:background="#0058b6" />

</LinearLayout>
```



2. Horizontal:

In this all the child are arranged horizontally in a line one after the other. In below code snippets we have specified orientation “horizontal” so the childs/views of this layout are displayed horizontally.

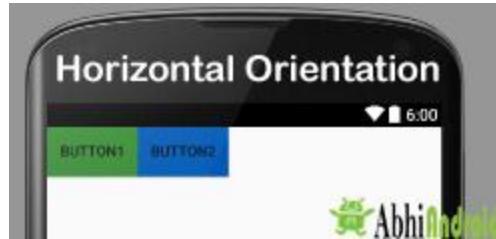
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"> <!-- Horizontal Orientation set -->

    <!-- Child Views(In this case 2 Button) are here -->

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button1"
        android:id="@+id/button"
        android:background="#358a32" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button2"
        android:id="@+id/button2"
        android:background="#0058b6" />

</LinearLayout>
```



Important Note: All of the layout managers can be nested. This means that you can put a Relative Layout or Frame Layout as a child to Linear Layout.

Main Attributes In Linear Layout:

Now let's we discuss about the attributes that helps us to configure a linear layout and its child controls. Some of the most important attributes you will use with linear layout include:

1. orientation: The orientation attribute used to set the childs/views horizontally or vertically. In Linear layout default orientation is vertical.

Example: Orientation vertical:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"> <!-- Vertical Orientation set -->
    <!-- Put Child Views like Button here -->
</LinearLayout>
```

Example: Orientation Horizontal:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"> <!-- Horizontal Orientation set -->
    <!-- Child Views are here -->
```

```
</LinearLayout>
```



2. gravity: The gravity attribute is an optional attribute which is used to control the alignment of the layout like left, right, center, top, bottom etc.

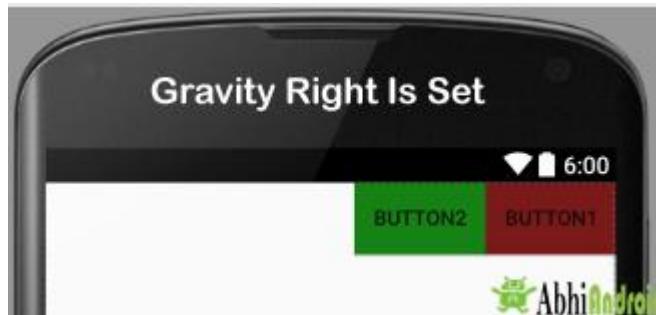
Example: We have set gravity right for linear layout. So the buttons gets align from right side in Horizontal orientation.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="right"
    android:orientation="horizontal">

    <!--Button Child View Here-->

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button2"
        android:id="@+id/button2"
        android:background="#0e7d0d" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button1"
        android:id="@+id/button"
        android:background="#761212" />
</LinearLayout>
```



3. layout_weight: The layout weight attribute specifies each child control's relative importance within the parent linear layout.

Example: weight property for button in linear layout. In the below example one button is of weight 2 and other is of weight 1.

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

    <!--Add Child View Here-->

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Weight 2"
        android:background="#761212"
        android:layout_margin="5dp"
        android:id="@+id/button"
        android:layout_weight="2" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="#761212"
        android:layout_margin="5dp"
        android:layout_weight="1"
        android:text="Weight 1" />
</LinearLayout>
```

In the layout image you can notice Button with weight 2 gets more size related to the other.



4. weightSum: weightSum is the sum up of all the child attributes weight. This attribute is required if we define weight property of the childs.

Example: In the same above example of weight, we can define weightSum value 3.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:weightSum="3"
    android:orientation="horizontal">
    <!--Add Child View Here-->

</LinearLayout>
```

Example of Linear Layout:

Now lets design 2 linear layout UI. First we have designed using weight attribute and second without using it. So below layout output will clear the difference between them:



Example 1: First we will design Android Linear Layout without using weight property

In this example we have used one TextView and 4 Button. The orientation is set to vertical.

Below is the code of activity_main.xml

```
<!-- Vertical Orientation is set -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <!-- Text Displayed At Top -->

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="Linear Layout (Without Weight)"
        android:id="@+id/textView"
        android:layout_gravity="center_horizontal" />

    <!-- Button Used -->
```

```
<Button  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:text="Button 1"  
    android:background="#009300" />  
  
<Button  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:text="Button 2"  
    android:background="#e6cf00" />  
  
<Button  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:text="Button 3"  
    android:background="#0472f9" />  
  
<Button  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:text="Button 4"  
    android:background="#e100d5" />  
</LinearLayout>
```

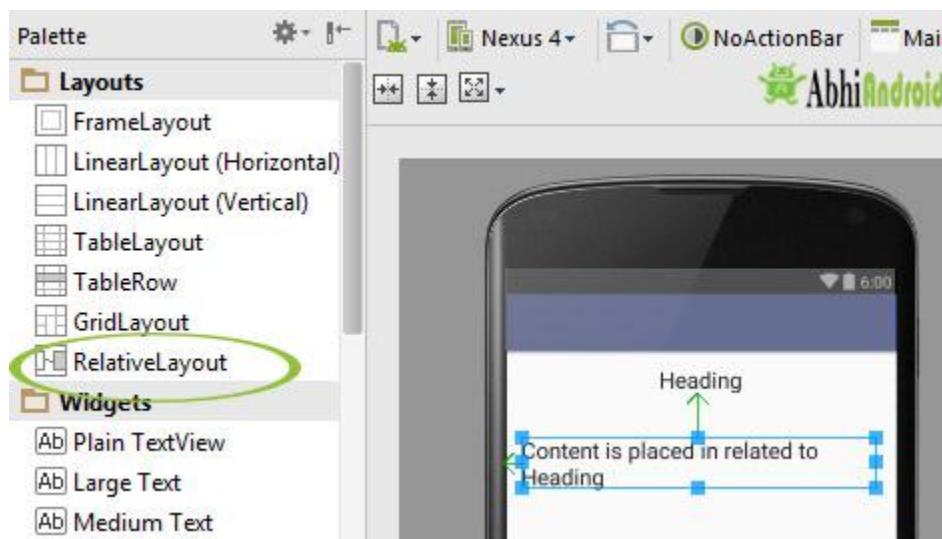
Output Screen:



Relative Layout

The Relative Layout is very flexible layout used in android for custom layout designing. It gives us the flexibility to position our component/view based on the relative or sibling component's position. Just because it allows us to position the component anywhere we want so it is considered as most flexible layout. For the same reason Relative layout is the most used layout after the Linear Layout in Android. It allow its child view to position relative to each other or relative to the container or another container.

In Relative Layout, you can use “above, below, left and right” to arrange the component's position in relation to other component. **For example, in the below image you can see content is placed in related to Heading.**



Even though Android has drag and drop system to put one component in related to other inside relative layout. But actually in the background lots of XML properties are working which does this task. So Android developer can design UI either using drag & drop or using XML code. Professional developer uses both for designing UI.

Attributes of Relative layout:

Lets see different properties of Relative Layout which will be used while designing Android App UI:

1.above: Position the bottom edge of the view above the given anchor view ID and must be a reference of the another resource in the form of id. Example,
android:layout_above="@+id/textView".

For example, suppose a view with id textView2 is what we want to place above another view with id textView. Below is the code and layout image.



2. alignBottom: alignBottom is used to makes the bottom edge of the view match the bottom edge of the given anchor view ID and it must be a reference to another resource, in the form of id. Example: android:layout_alignBottom ="@+id/button1"

In the below example we have aligned a view with id textView2 Bottom of another view with id textView. Below is the coded and layout image.

```
<!-- textView2 alignBottom of textView -->
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:layout_centerHorizontal="true"
    android:id="@+id/textView2"
    android:layout_alignBottom="@+id/textView"
    android:text="Text2 alignBottom of Text1"
    android:layout_marginBottom="90dp"
/>
```



3. alignLeft: alignLeft is used to make the left edge of the view match the left edge of the given anchor view ID and must be a reference to another resource, in the form of Example:
android:layout_alignLeft =”@+id/button1”.

Below is the code and layout image in which we have aligned a view with id textView2 left of another view with id textView.

```
<!-- textView2 alignLeft of textView -->
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:id="@+id/textView2"
    android:layout_alignLeft="@+id/textView"
    android:text="Text2 alignLeft of Text1"
    android:layout_below="@+id/textView"
    android:layout_marginTop="20dp"/>
```



4. alignRight: alignRight property is used to make the right edge of this view match the right edge of the given anchor view ID and must be a reference to another resource, in the form like this example: android:layout_alignRight="@+id/button1"

Below is the code and layout image in which we have aligned a view with id textView2 right of another view with id textView.

```
<!-- textView2 alignRight of textView-->
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:id="@+id/textView2"
    android:layout_alignRight="@+id/textView"
    android:text="Text2 alignRight of Text1"
    android:layout_below="@+id/textView"
    android:layout_marginTop="20dp"/>
```



5. alignStart: alignStart property is used to makes the start edge of this view match the start edge of the given anchor view ID and must be a reference to another resource, in the form of like this example: android:layout_alignStart="@+id/button1"

Below is the alignStart code and layout image in which we have aligned a view with id textView2 start of another view with id textView.

```
<!-- Text2 alignStart-->
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:id="@+id/textView2"
    android:text="Text2 align start of Text1"
    android:layout_alignStart="@+id/textView"
    />
```



6. alignTop: alignTop property is used to makes the top edge of this view match the top edge of the given anchor view ID and must be a reference to another resource, in the form like this example: android:layout_alignTop="@+id/button1".

Below is the alignTop code and layout image in which we have aligned a view with id textView Top of another image with id imageView.

```
<!--text is align top on Image-->
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:id="@+id/textView"
    android:layout_alignTop="@+id/imageView"
    />
```

```
    android:text="Text Here is AlignTop on Image"  
    />
```



7.alignParentBottom: If alignParentBottom property is true, makes the bottom edge of this view match the bottom edge of the parent. The value of align parent bottom is either true or false. Example: android:layout_alignParentBottom="true"

Important Note: alignParentBottom and alignBottom are two different properties. In alignBottom we give the reference of another view in the form of id that the view is aligned at the bottom of referenced view but in alignParentBottom the bottom edge of the view matches the bottom edge of the parent.

Below is the alignParentBottom code and layout image in which textView is simply displayed using the alignParentBottom.

```
<!-- textView is alignParentBottom -->  
<TextView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:textAppearance="?android:attr/textAppearanceLarge"  
    android:id="@+id/textView"  
    android:text="Text Here is AlignParentBottom with bottom margin of 120dp"  
    android:layout_alignParentBottom="true"  
    android:layout_alignParentLeft="true"  
    android:layout_alignParentStart="true"  
    android:layout_marginBottom="120dp" />
```



8. alignParentEnd: If alignParentEnd property is true, then it makes the end edge of this view match the end edge of the parent. The value of align parent End is either true or false. Example: android:layout_alignParentEnd="true".

Important Note: In alignParentEnd the bottom edge of the view matches the bottom edge of the parent.

Below is the alignParentEnd code and layout image in which textView is simply displayed on Image in the end.

```
<!-- Text displayed in the end of parent image-->
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:id="@+id/textView"
    android:text="Text in Parent End"
    android:layout_alignBottom="@+id/imageView"
    android:layout_alignParentEnd="true"
/>
```



9. alignParentLeft: If alignParentLeft property is true, makes the left edge of this view match the left edge of the parent. The value of align parent left is either true or false. Example: android:layout_alignParentLeft="true".

Important Note: alignParentLeft and alignLeft are two different properties. In alignLeft we give the reference of another view in the form of id that the view is aligned to the left of the referenced view but in alignParentLeft the left edge of the view matches the left edge of the parent.

Below is the alignParentLeft example code and layout image in which textView is simply displayed on parent Image in the left side.

```
<!-- align parent left in Android -->
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:id="@+id/textView"
    android:text="Text in Parent Left"
    android:layout_alignBottom="@+id/imageView"
    android:layout_alignParentLeft="true"
    />
```



10. alignParentLeft: If alignParentLeft property is true, then it makes the left edge of this view match the left edge of the parent. The value of align parent left is either true or false. Example: android:layout_alignParentLeft="true".

Important Note: alignParentRight and alignRight are two different properties. In alignRight we give the reference of another view in the form of id that the view is aligned to the right of the referenced view but in alignParentRight the right edge of the view matches the right edge of the parent.

Below is the alignParentRight example code and layout image in which textView is simply displayed on parent Image in the right side.

```
<!-- alignRightParent Example -->
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:id="@+id/textView"
    android:text="Text in Parent Right"
    android:layout_alignBottom="@+id/imageView"
    android:layout_alignParentRight="true"
/>
```



11.alignParentStart: If alignParentStart is true, then it makes the start edge of this view match the start edge of the parent. The value of align parent start is either true or false. Example: android:layout_alignParentStart="true".

Important Note: alignParentStart and alignStart are two different properties, In alignStart we give the reference of another view in the form of id that the view is aligned at the start of referenced view but in alignParentStart the start edge of the view matches the start edge of the parent(RelativeLayout).

Below is the alignParentStart example code and layout image in which textView is simply displayed on parent Image in the right side.

```
<!-- alignParentStart Example -->
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:id="@+id/textView"
    android:text="Text in Parent Start"
    android:layout_alignTop="@+id/imageView"
    android:layout_alignParentStart="true"
    />
```



12.alignParentTop: If alignParentTop is true, then it makes the top edge of this view match the top edge of the parent. The value of align parent Top is either true or false. Example: android:layout_alignParentTop="true".

Important Note: alignParentTop and alignTop are two different properties, In alignTop we give the reference of another view in the form of id that the view is aligned to the top of the referenced view but in alignParentTop the top edge of the view matches the top edge of the parent(RelativeLayout).

Below is the example code of alignParentTop property and also layout image.

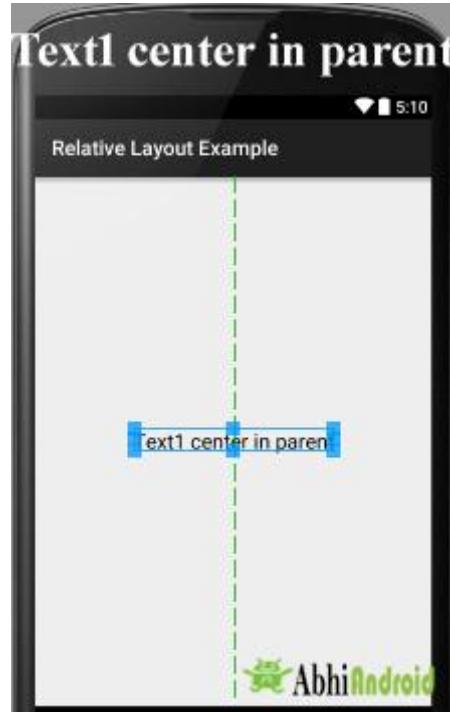
```
<!-- alignParentTop example -->
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Text1 align parent top"
    android:layout_alignParentTop="true"
    android:layout_margin="20dp"
    android:textSize="20sp"
    android:textColor="#000"/>
```



13. centerInParent: If center in parent is true, makes the view in the center of the screen vertically and horizontally. The value of center in parent is either true or false. Example: android:layout_centerInParent="true".

Below is the example code of centerInParent property and also layout image.

```
<!-- centerInParent example -->
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Text1 center in parent"
    android:layout_centerInParent="true"
    android:textSize="20sp"
    android:textColor="#000"
/>
```



14. centerHorizontal: If centerHorizontal property is true, makes the view horizontally center. The value of centerHorizontal is either true or false.Example:
android:layout_centerHorizontal="true".

Below is the example code of centerHorizontal property and also layout image.

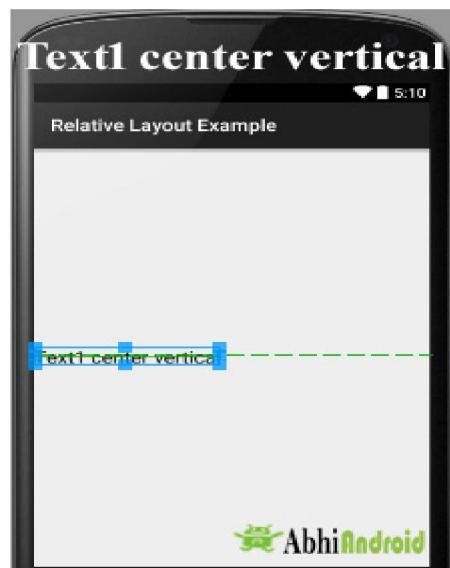
```
<!-- centerHorizontal example -->
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Text1 center Horizontal"
    android:layout_centerHorizontal="true"
    android:textSize="20sp"
    android:textColor="#000"
/>
```



15. centerVertical: If centerVertical property is true, make the view vertically center. The value of align parent bottom is either true or false. Example:
android:layout_centerVertical="true".

Below is the example code of centerVertical property and also layout image.

```
<!-- centerVertical example -->
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Text1 center vertical"
    android:layout_centerVertical="true"
    android:textSize="20sp"
    android:textColor="#000"
/>
```



Relative Layout Examples With Code And Explanation:

Example 1: Here we are designing a simple log in screen in Android Studio using Relative Layout. Below is the final output:



Below is the code of activity_main.xml for designing UI with explanation included in it:

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

<!--Relative Layout Is Used-->

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <!--Text View for Displaying SIGN IN Text At Top of UI-->

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="SIGN IN"
        android:id="@+id/textView3"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true" />

    <!--Text View for Displaying Username-->

    <TextView
        android:id="@+id/userName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="@dimen/activity_horizontal_margin"
        android:layout_marginTop="110dp"
        android:text="UserName:"
        android:textColor="#000000"
        android:textSize="20sp" />

    <!--Text View for Displaying Password-->

    <TextView
        android:id="@+id/password"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/userName"
        android:layout_margin="@dimen/activity_horizontal_margin"
        android:text="Password:"
        android:textColor="#000000"
        android:textSize="20sp" />

    <!--Edit Text for Filling Username-->

    <EditText
        android:id="@+id/edt_userName"
        android:layout_width="fill_parent"
        android:layout_height="40dp"
        android:layout_marginLeft="@dimen/activity_horizontal_margin"
        android:layout_marginTop="100dp"
        android:layout_toRightOf="@+id/userName"
        android:hint="User Name" />

    <!--Edit Text for Filling Password-->

    <EditText
        android:layout_width="fill_parent"
        android:layout_height="40dp"
        android:layout_below="@+id/edt_userName"
        android:layout_centerVertical="true"
        android:layout_toRightOf="@+id/password"
        android:hint="Password" />

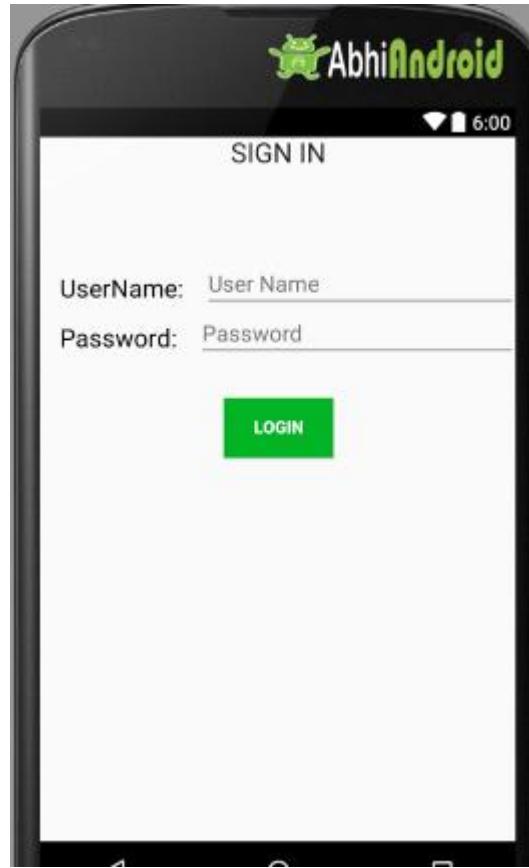
    <!--Button for Clicking after filling details-->

    <Button
        android:id="@+id/btnLogin"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
    android:layout_below="@+id/password"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="20dp"
    android:background="#03B424"
    android:text="Login"
    android:textColor="#ffffff"
    android:textStyle="bold" />

</RelativeLayout>
```

Output:



Difference between Linear And Relative Layout:

RELATIVE LAYOUT:

- Every element of relative layout arranges itself to the other element or a parent element.
- It is helpful while adding views one next to other etc

- In a relative layout you can give each child a Layout Property that specifies exactly where it should go in relative to the parent or relative to other children.
- Views can be layered on top of each other.

LINEAR LAYOUT:

- In a linear layout, like the name suggests, all the elements are displayed in a linear fashion either vertically or horizontally.
- Either Horizontally or Vertically this behavior is set in android:orientation which is a property of the node Linear Layout.

```
android:orientation="horizontal"  or  android:orientation="vertical"
```

- Linear layouts put every child, one after the other, in a line, either horizontally or vertically.

Table Layout

In Android, Table Layout is used to arrange the group of views into rows and columns. Table Layout containers do not display a border line for their columns, rows or cells. A Table will have as many columns as the row with the most cells.

Row 1 Column 1	Row 1 Column 2	Row 1 Column 3
Row 2 Column 1		Row 2 Column 2
Row 3 Column 1		

A table can also leave the cells empty but cells can't span the columns as they can in HTML(Hypertext markup language).

Important Points About Table Layout In Android:

For building a row in a table we will use the `<TableRow>` element. Table row objects are the child views of a table layout.

Each row of the table has zero or more cells and each cell can hold only one view object like ImageView, TextView or any other view.

Total width of a table is defined by its parent container

Column can be both stretchable and shrinkable. If shrinkable then the width of column can be shrunk to fit the table into its parent object and if stretchable then it can expand in width to fit any extra space available.

Important Note: We cannot specify the width of the children's of the Table layout. Here, width always match parent width. However, the height attribute can be defined by a child; default value of height attribute is wrap content.

Basic Table Layout code in XML:

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:collapseColumns="0"> <!-- collapse the first column of the table row-->

    <!-- first row of the table layout-->
    <TableRow
        android:id="@+id/row1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content">

        <!-- Add elements/columns in the first row-->

    </TableRow>
</TableLayout>
```

Attributes of TableLayout in Android:

Now let's we discuss some important attributes that help us to configure a table layout in XML file (layout).

1. id: id attribute is used to uniquely identify a Table Layout.

```
<TableLayout
    android:id="@+id/simpleTableLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

2. stretchColumns: Stretch column attribute is used in Table Layout to change the default width of a column which is set equal to the width of the widest column but we can also stretch the columns to take up available free space by using this attribute. The value that assigned to this attribute can be a single column number or a comma delimited list of column numbers (1, 2, 3...n).

If the value is 1 then the second column is stretched to take up any available space in the row, because of the column numbers are started from 0.

If the value is 0,1 then both the first and second columns of table are stretched to take up the available space in the row.

If the value is '*' then all the columns are stretched to take up the available space.

Below is the example code of stretch column attribute of table layout with explanation included in which we stretch the first column of layout.

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

<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/simpleTableLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:stretchColumns="1" > <!-- stretch the second column of the layout-->

    <!-- first row of the table layout-->
    <TableRow

        android:id="@+id/firstRow"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content">

        <!-- first element of the row-->
        <TextView

            android:id="@+id/simpleTextView"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:background="#b0b0b0"
            android:padding="18dip"
            android:text="Text 1"
            android:textColor="#000"
            android:textSize="12dp" />
```

```

<TextView
    android:id="@+id/simpleTextView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="#FF0000"
    android:padding="18dip"
    android:text="Text 2"
    android:textColor="#000"
    android:textSize="14dp" />

</TableRow>
</TableLayout>

```



3. shrinkColumns: Shrink column attribute is used to shrink or reduce the width of the column's. We can specify either a single column or a comma delimited list of column numbers for this attribute. The content in the specified columns word-wraps to reduce their width.

If the value is 0 then the first column's width shrinks or reduces by word wrapping its content.

If the value is 0,1 then both first and second columns are shrinks or reduced by word wrapping its content.

If the value is '*' then the content of all columns is word wrapped to shrink their widths.

Below is the example code of shrink column attribute of table layout with explanation included in which we shrink the first column of layout.

```

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

```

```
        android:layout_height="match_parent"
        android:shrinkColumns="0"> <!-- shrink the first column of the layout-->

        <!-- first row of the table layout-->
<TableRow
    android:id="@+id/firstRow"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content">

    <!-- first element of the first row-->
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="#b0b0b0"
    android:padding="18dip"
    android:text="Shrink Column Example"
    android:textColor="#000"
    android:textSize="18dp" />

</TableRow>
</TableLayout>
```



4. collapseColumns: collapse columns attribute is used to collapse or invisible the column's of a table layout. These columns are the part of the table information but are invisible.

If the values is 0 then the first column appears collapsed, i.e it is the part of table but it is invisible.

Below is the example code of collapse columns with explanation included in which we collapse the first column of table means first column is an part of table but it is invisible so you can see only the second column in screenshot.

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

```
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:collapseColumns="0">> <!-- collapse the first column of the table row-->

        <!-- first row of the table layout-->
<TableRow
    android:id="@+id/simpleTableLayout"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content">

    <!-- first element of the row that is the part of table but it is invisible-->
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="#b0b0b0"
    android:padding="18dip"
    android:text="Columns 1"
    android:textColor="#000"
    android:textSize="18dp" />

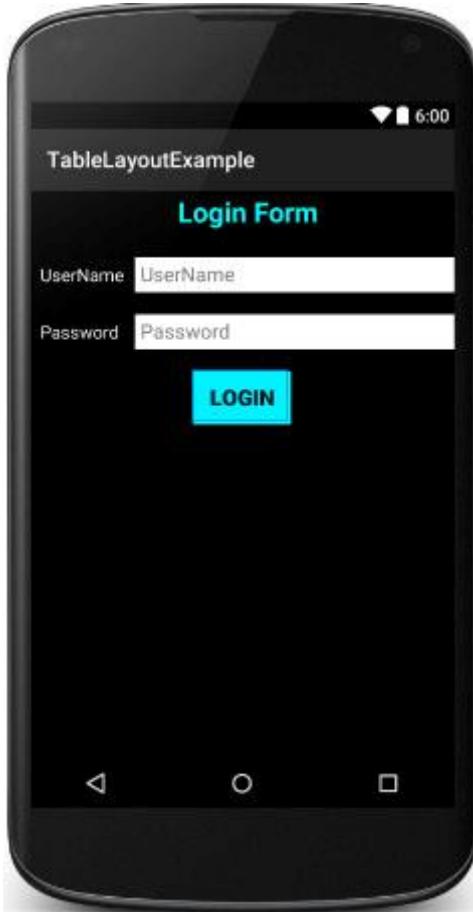
    <!-- second element of the row that is shown in the screenshot-->
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="#b0b0b0"
    android:padding="18dip"
    android:text="Columns 2"
    android:textColor="#000"
    android:textSize="18dp" />
</TableRow>
</TableLayout>
```



TableLayout Example In Android Studio:

Below is an example of Table layout in Android where we display a login form with two fields user name and password and one login button and whenever a user click on that button a message will be displayed by using a Toast.

Below you can download project code, see final output and step by step explanation of the example:



Step 1: Create a new project and name it TableLayoutExample

Step 2: Open res -> layout ->activity_main.xml (or) main.xml and add following code:

In this step we open an xml file (activity_main.xml) and add the code for displaying username and password fields by using textView and editText with one login button.

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#000"
    android:orientation="vertical"
    android:stretchColumns="1">

    <TableRow android:padding="5dip">

        <TextView
            android:layout_height="wrap_content"
            android:layout_marginBottom="20dp"
            android:layout_span="2"
            android:gravity="center_horizontal"
            android:text="@string/loginForm"
            android:textColor="#0ff"
            android:textSize="25sp"
            android:textStyle="bold" />
    </TableRow>

    <TableRow>

        <TextView
            android:layout_height="wrap_content"
            android:layout_column="0"
            android:layout_marginLeft="10dp"
            android:text="@string/userName"
            android:textColor="#fff"
            android:textSize="16sp" />

        <EditText
            android:id="@+id/userName"
            android:layout_height="wrap_content"
            android:layout_column="1"
            android:layout_marginLeft="10dp"
            android:background="#fff"
            android:hint="@string/userName"
            android:padding="5dp"
            android:textColor="#000" />
    </TableRow>

    <TableRow>

        <TextView
            android:layout_height="wrap_content"
            android:layout_column="0"
            android:layout_marginLeft="10dp"
            android:layout_marginTop="20dp"
            android:text="@string/password"
            android:textColor="#fff"
            android:textSize="16sp" />

        <EditText
            android:id="@+id/password"
            android:layout_height="wrap_content"
            android:layout_column="1"
            android:layout_marginLeft="10dp"
```

```
        android:layout_marginTop="20dp"
        android:background="#fff"
        android:hint="@string/password"
        android:padding="5dp"
        android:textColor="#000" />
    </TableRow>

    <TableRow android:layout_marginTop="20dp">

        <Button
            android:id="@+id/loginBtn"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:layout_span="2"
            android:background="#0ff"
            android:text="@string/login"
            android:textColor="#000"
            android:textSize="20sp"
            android:textStyle="bold" />
    </TableRow>
</TableLayout>
```

Step 3: Open src -> package -> MainActivity.java

In this step we open MainActivity and add the code to initiate the edittext and button and then perform click event on button and display the message by using a Toast.

```
package example.abhiandriod.tablelayoutexample;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // initiate a button
        Button loginButton = (Button) findViewById(R.id.loginBtn);
        // perform click event on the button
        loginButton.setOnClickListener(new View.OnClickListener() {
            @Override
```

```
        public void onClick(View v) {  
            Toast.makeText(getApplicationContext(), "Hello AbhiAndroid..!!!!",  
            Toast.LENGTH_LONG).show(); // display a toast message  
        }  
    }  
}
```

Step 4: Open res -> values -> strings.xml

In this step we open string file which is used to store string data of the app.

```
<resources>  
    <string name="app_name">TableLayoutExample</string>  
    <string name="hello_world">Hello world!</string>  
    <string name="action_settings">Settings</string>  
    <string name="loginForm">Login Form</string>  
    <string name="userName">UserName</string>  
    <string name="password">Password</string>  
    <string name="login">LogIn</string>  
</resources>
```

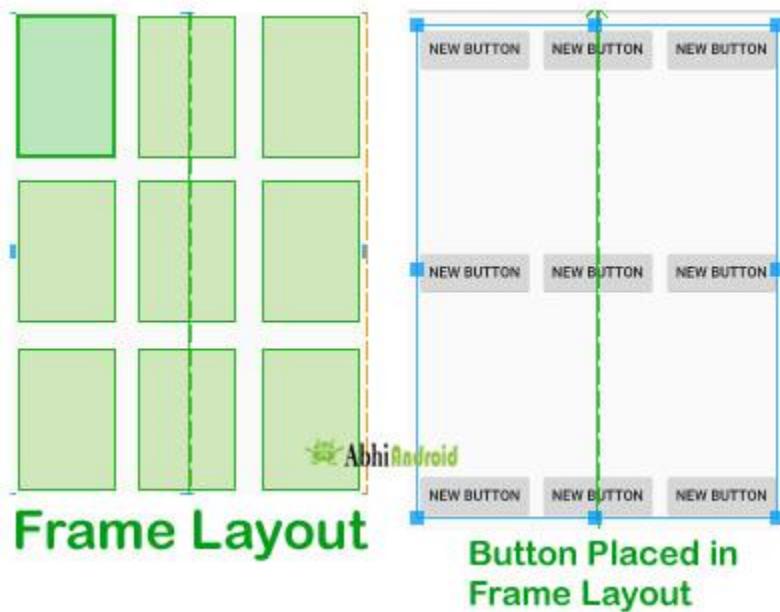
Output:

Now run the App and you will see the Login form UI which we designed in Table Layout.

Frame Layout

Frame Layout is one of the simplest layout to organize view controls. They are designed to block an area on the screen. Frame Layout should be used to hold child view, because it can be difficult to display single views at a specific area on the screen without overlapping each other.

We can add multiple children to a FrameLayout and control their position by assigning gravity to each child, using the android:layout_gravity attribute.



Attributes of Frame Layout:

Lets see different properties of Frame Layout which will be used while designing Android App UI:

1. android:id

This is the unique id which identifies the layout in the R.java file.

Below is the id attribute's example with explanation included in which we define the id for Frame Layout.

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

2. android:foreground

Foreground defines the drawable to draw over the content and this may be a color value.

Possible color values can be in the form of “#rgb”, “#argb”, “#rrggbb”, or “#aarrggbb”. This all are different color code model used.

Example: In Below example of foreground we set the green color for foreground of frameLayout so the ImageView and other child views of this layout will not be shown.

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

    android:id="@+id/framelayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_gravity="center"
    android:foregroundGravity="fill"
    android:foreground="#0f0"><!--foreground color for a FrameLayout-->

    <LinearLayout
        android:orientation="vertical"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
    >

        <!-- ImageView will not be shown because of foreground color which is drawn over it-->

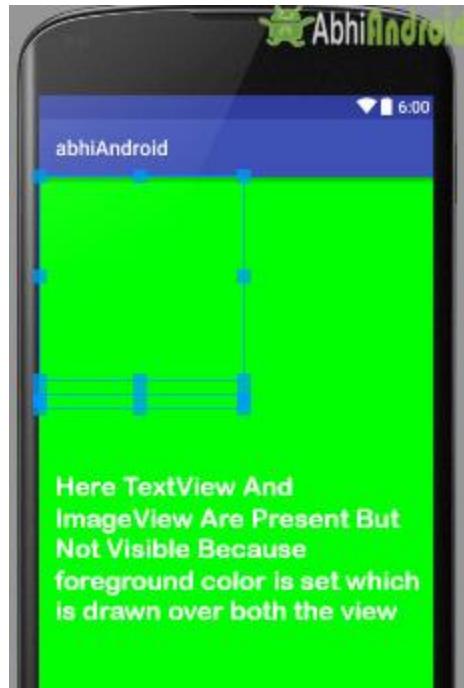
        <ImageView
            android:layout_width="200dp"
            android:layout_height="200dp"
            android:layout_marginBottom="10dp"
            android:src="@mipmap/ic_launcher"
            android:scaleType="centerCrop"
        />

        <!-- TextView will not be shown because of foreground color is drawn over it-->

        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:gravity="center_horizontal"
            android:text="abhiAndroid"/>

    </LinearLayout>

</FrameLayout>
```



Important Note: On applying android:foreground feature, all the views taken in the framelayou will goes to the background and the framelayou comes in the foreground i.e. over the views. We can set the @drawable/image_name or the color in the foreground.

3. android:foregroundGravity

This defines the gravity to apply to the foreground drawable. Default value of gravity is fill. We can set values in the form of “top”, “center_vertical”, “fill_vertical”, “center_horizontal”, “fill_horizontal”, “center”, “fill”, “clip_vertical”, “clip_horizontal”, “bottom”, “left” or “right” .

It is used to set the gravity of foreground. We can also set multiple values by using “|”. Ex: fill_horizontal|top .Both the fill_horizontal and top gravity are set to framelayou.

In the above same example of foreground we also set the foregroundGravity of FrameLayout to fill.

4. android:visibility

This determine whether to make the view visible, invisible or gone.

visible – the view is present and also visible

invisible – The view is present but not visible

gone – The view is neither present nor visible



5. android:measureAllChildren

This determines whether to measure all children including gone state visibility or just those which are in the visible or invisible state of measuring visibility. The default value of measureallchildren is false. We can set values in the form of Boolean i.e. “true” OR “false”.

This may also be a reference to a resource (in the form “[@][package:]type:name”) or theme attribute (in the form “[?][package:][type:]name”) containing a value of this type.

Important Note: If measureallchildren is set true then it will show actual width and height of frame layout even if the views visibility is in gone state.

The above property has been explained below:

Example of Frame layout having measureAllChildren attribute:

In the below code the ImageView visibility is set gone and measureAllChildren is set true. The output will show actual height and width the Frame Layout despite visibility is set gone. We have used Toast to display the height and width:

Below is the Code of activity_main.xml

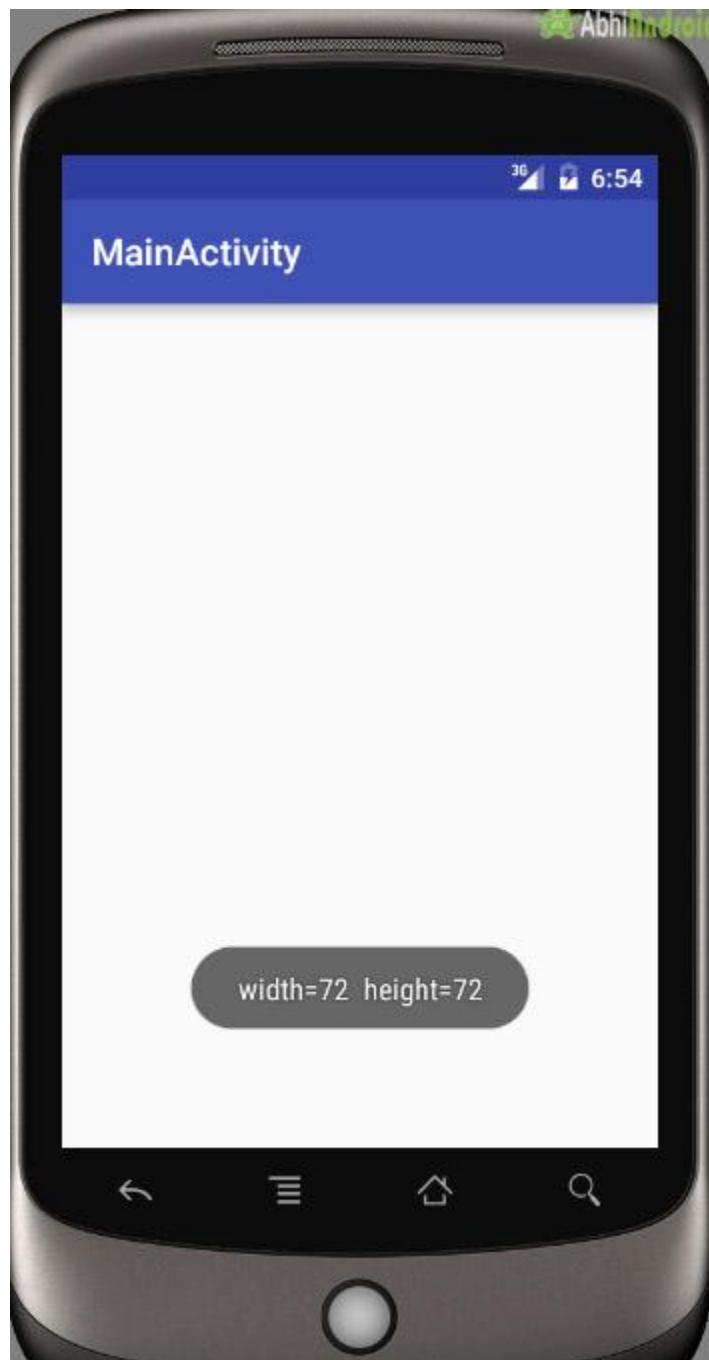
Important Note: Make sure you have image in Drawable folder. In our case we have used ic_launcher image which we added in Drawable.

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/frame"
    android:orientation="vertical" android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:measureAllChildren="true"
    >
    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:visibility="gone"
        android:src="@drawable/ic_launcher"/>
</FrameLayout>
```

Below is the code of MainActivity.java . Here we have used Toast to display height and width on screen.

```
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.demo);
        FrameLayout frame=(FrameLayout)findViewById(R.id.frame);
        frame.measure(View.MeasureSpec.UNSPECIFIED, View.MeasureSpec.UNSPECIFIED);
        int width = frame.getMeasuredWidth();
        int height = frame.getMeasuredHeight();
        Toast.makeText(getApplicationContext(),"width="+width+
height+"+height,Toast.LENGTH_SHORT).show();
    }
}
```

When you run the App in Emulator you will see the below output:

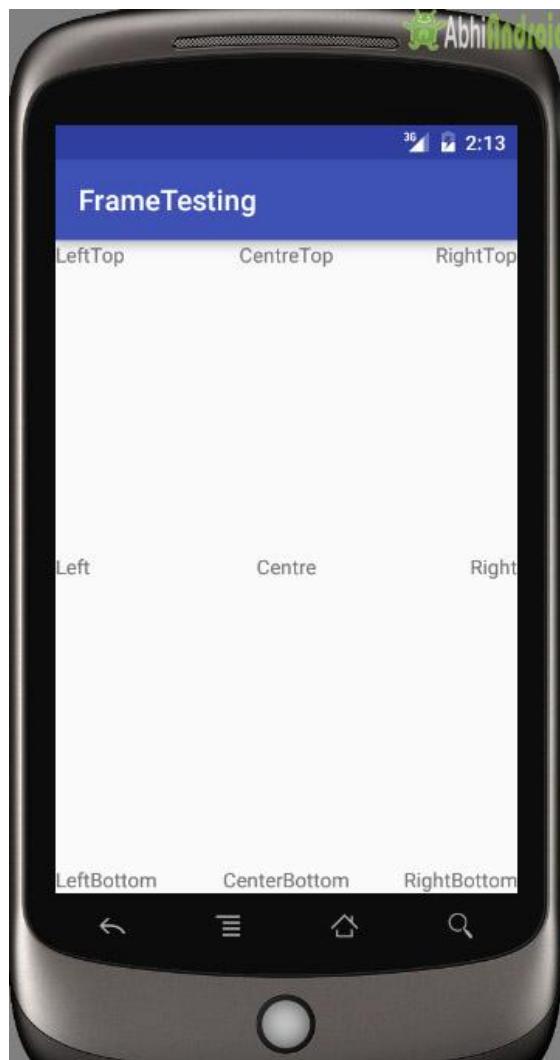


Explanation of Example: It measures all the children in the layout. For ex: If we setVisibility of an view be gone and set measuresAllChildren property to be true, then also it will also count to that view which is not visible, but if we set the measuresAllChildren property to be false, then it will not count to that view which is gone.

Things To Do Yourself: Try changing measuresAllChildren value to false and run the App in Emulator. You will see the output shows 0 width and height of Frame Layout.

Example Of Frame Layout in Android Studio:

Example 1: Frame Layout using layout gravity. Here we will put textView at different position in Frame Layout. Below is the code and final output:



Step 1: Create a new project in Android Studio and name it FrameTesting. (Select File -> New -> New Project. Fill the forms and click “Finish” button)

Step 2: Now Open res -> layout -> activity_main.xml and add the following code. Here we are putting different TextView in Frame Layout.

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

<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_height="match_parent"
    android:layout_width="match_parent"
    >
    <TextView android:text="LeftTop"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <TextView android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:text="RightTop"
        android:layout_gravity="top|right" />
    <TextView android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:text="CentreTop"
        android:layout_gravity="top|center_horizontal" />
    <TextView android:text="Left"
        android:layout_gravity="left|center_vertical"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <TextView android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:text="Right"
        android:layout_gravity="right|center_vertical" />
    <TextView android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:text="Centre"
        android:layout_gravity="center" />
    <TextView android:text="LeftBottom"
        android:layout_gravity="left|bottom"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <TextView android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:text="RightBottom"
        android:layout_gravity="right|bottom" />
    <TextView android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:text="CenterBottom"
        android:layout_gravity="center|bottom" />
</FrameLayout>
```

Step 3: Let the MainActivity.java has default Android code or add the below code:

```
package abhiandroid.com.frametesting;
```

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

public class MainActivity extends AppCompatActivity {

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

Output:

Run the App in Emulator, you will see TextView positioned at various position in FrameLayout

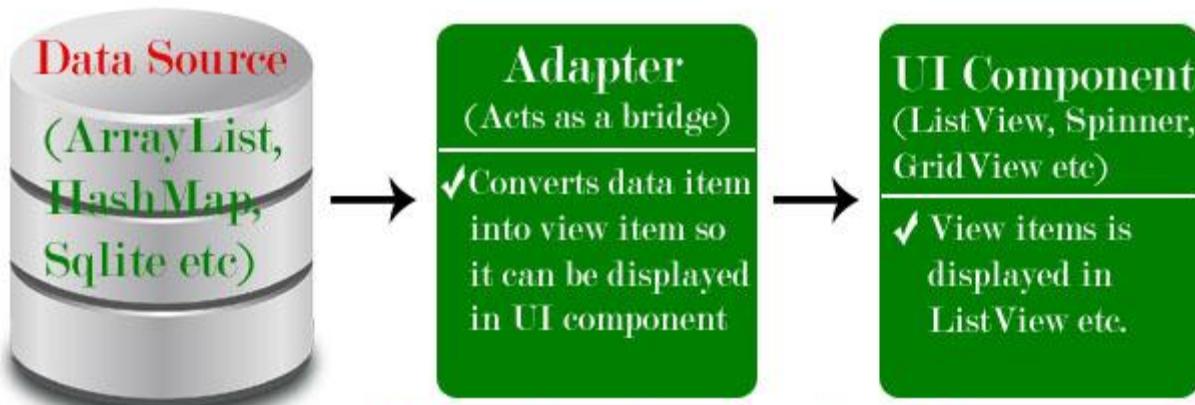
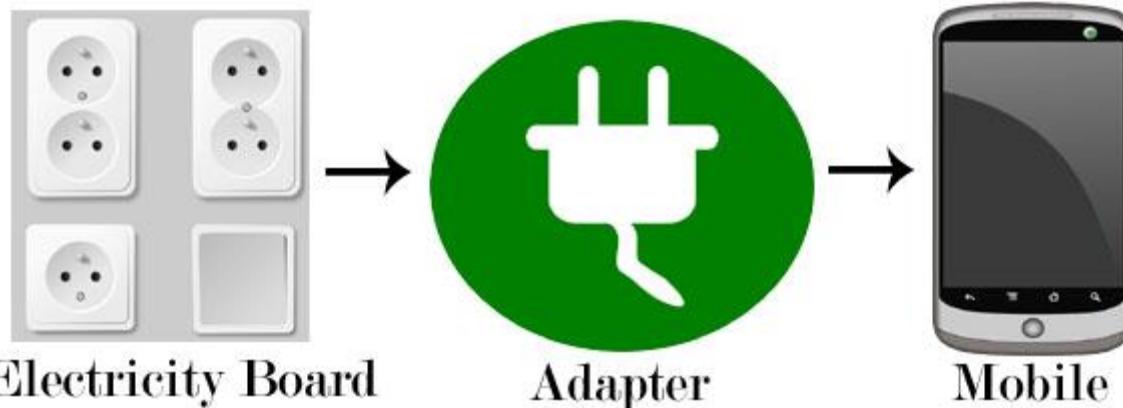


Adapter

In Android, Adapter is a bridge between UI component and data source that helps us to fill data in UI component. It holds the data and send the data to an Adapter view then view can takes the data from the adapter view and shows the data on different views like as ListView, GridView, Spinner etc. For more customization in Views we uses the base adapter or custom adapters.

To fill data in a list or a grid we need to implement Adapter. Adapters acts like a bridge between UI component and data source. Here data source is the source from where we get the data and UI components are list or grid items in which we want to display that data.

Below is a conceptual diagram of Adapter:



Adapters In Android:

There are some commonly used Adapter in Android used to fill the data in the UI components.

BaseAdapter – It is parent adapter for all other adapters

ArrayAdapter – It is used whenever we have a list of single items which is backed by an array

Custom ArrayAdapter – It is used whenever we need to display a custom list

SimpleAdapter – It is an easy adapter to map static data to views defined in your XML file

Custom SimpleAdapter – It is used whenever we need to display a customized list and needed to access the child items of the list or grid

Now we describe each Adapters one by one in detail:

1. BaseAdapter In Android:

BaseAdapter is a common base class of a general implementation of an Adapter that can be used in ListView, GridView, Spinner etc. Whenever we need a customized list in a ListView or customized grids in a GridView we create our own adapter and extend base adapter in that. Base Adapter can be extended to create a custom Adapter for displaying a custom list item. ArrayAdapter is also an implementation of BaseAdapter.

Custom Adapter code which extends the BaseAdapter in that:

```
public class CustomAdapter extends BaseAdapter {  
  
    @Override  
    public int getCount() {  
        return 0;  
    }  
  
    @Override  
    public Object getItem(int i) {  
        return null;  
    }  
  
    @Override  
    public long getItemId(int i) {  
        return 0;  
    }  
  
    @Override  
    public View getView(int i, View view, ViewGroup viewGroup) {  
  
        return null;  
    }  
}
```

In above code snippet we see the overridden functions of BaseAdapter which are used to set the data in a list, grid or a spinner. These functions are described in BaseAdapter tutorial with example.

2. ArrayAdapter In Android:

Whenever we have a list of single items which is backed by an Array, we can use ArrayAdapter. For instance, list of phone contacts, countries or names.

Here is how android ArrayAdapter looks :

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

The above function are described in ArrayAdapter tutorial with example.

3. Custom ArrayAdapter In Android:

ArrayAdapter is also an implementation of BaseAdapter, so if we want more customization then we can create a custom adapter and extend ArrayAdapter in that. Since array adapter is an implementation of BaseAdapter, so we can override all the function's of BaseAdapter in our custom adapter.

Below Custom adapter class MyAdapter extends ArrayAdapter in that:

```
public class MyAdapter extends ArrayAdapter {  
  
    public MyAdapter(Context context, int resource, int textViewResourceId, List objects) {  
        super(context, resource, textViewResourceId, objects);  
    }  
    @Override  
    public int getCount() {  
        return super.getCount();  
    }  
  
    @Override  
    public View getView(int position, View convertView, ViewGroup parent) {  
        return super.getView(position, convertView, parent);  
    }  
}
```

These functions are described in Custom ArrayAdapter tutorial with example.

4. SimpleAdapter In Android:

In Android SimpleAdapter is an easy Adapter to map static data to views defined in an XML file(layout). In Android we can specify the data backing to a list as an ArrayList of Maps(i.e. hashmap or other). Each entry in a ArrayList is corresponding to one row of a list.

The Map contains the data for each row. Here we also specify an XML file(custom list items file) that defines the views which is used to display the row, and a mapping from keys in the Map to specific views.

Whenever we have to create a custom list we need to implement custom adapter. As we discuss earlier ArrayAdapter is used when we have a list of single item's backed by an Array. So if we need more customization in a ListView or a GridView we need to implement simple adapter.

SimpleAdapter code in Android:

```
SimpleAdapter (Context context, List<? extends Map<String, ?>> data, int resource, String[] from, int[] to)
```

The above parameters of a simple Adapter is described in SimpleAdapter tutorial with example.

5. Custom SimpleAdapter In Android:

Whenever we have to create a custom list we need to implement custom adapter. As we discuss earlier ArrayAdapter is used when we have a list of single item's backed by an Array. So if we need customization in a ListView or a GridView we need to implement simple Adapter but when we need more customization in list or grid items where we have many view's in a list item and then we have to perform any event like click or any other event to a particular view then we need to implement a custom adapter who fulfills our requirement's and quite easy to be implemented.

BaseAdapter is the parent adapter for all other adapters so if we extends a SimpleAdapter then we can also override the base adapter's function in that class.

Important Note: We can't perform events like click and other event on child item of a list or grid but if we have some requirements to do that then we can create our own custom adapter and extends the simple adapter in that.

Custom Adapter extends SimpleAdapter in that:

```
public class CustomAdapter extends SimpleAdapter {  
    public CustomAdapter(Context context, List<? extends Map<String, ?>> data, int resource,  
    String[] from, int[] to) {  
        super(context, data, resource, from, to);  
    }  
  
    @Override  
    public View getView(int position, View convertView, ViewGroup parent) {  
        return super.getView(position, convertView, parent);  
    }  
  
    @Override  
    public int getCount() {  
        return super.getCount();  
    }  
}
```

The above overrided functions of simple adapter are already described in Custom SimpleAdapter article.

Adapter Example In Android Studio:

Below is the Android Studio example which show the use of the Adapter in Android. In this example we display a list of fruits names with images by using SimpleAdapter and whenever user click on a list item the fruit's name displayed in a Toast.

Below you can download complete Android Studio code, see final output and read step by step explanation:



Step 1: Create a new project and name it SimpleAdapterExample.

Step 2: Open res -> layout -> xml (or) main.xml and add following code :

In this step we open an xml file and add the code for displaying a ListView by using its different attributes.

```
<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">

    <ListView
        android:id="@+id/simpleListView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:divider="#000"
        android:dividerHeight="2dp"
        android:listSelector="#600"/>

</RelativeLayout>
```

Step 3: Save fruit images in drawable folder with name apple, banana, litchi, mango and pineapple.

Step 4: Open src -> package -> MainActivity.java

In this step we add the code for initiate ListView and set the data in the list. In this firstly we create two arrays first for fruit names and second for fruits images and then set the data in the ListView using SimpleAdapter.

```
package example.abhiandriod.simpleadapterexample;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ListView;
import android.widget.SimpleAdapter;
import android.widget.Toast;

import java.util.ArrayList;
import java.util.HashMap;

public class MainActivity extends AppCompatActivity {

    //initialize view's
    ListView simpleListView;
    String[] fruitsNames = {"Apple", "Banana", "Litchi", "Mango", "PineApple"};//fruit names
array
    int[] fruitsImages = {R.drawable.apple, R.drawable.banana, R.drawable.litchi,
R.drawable.mango, R.drawable.pineapple};//fruits images
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        simpleListView=(ListView)findViewById(R.id.simpleListView);

        ArrayList<HashMap<String, String>> arrayList=new ArrayList<>();
        for (int i=0;i<fruitsNames.length;i++)
        {
            HashMap<String, String> hashMap=new HashMap<>();//create a hashmap to store the
data in key value pair
            hashMap.put("name",fruitsNames[i]);
            hashMap.put("image",fruitsImages[i]+"");
            arrayList.add(hashMap);//add the hashmap into arrayList
        }
    }
}
```

```
String[] from={"name","image"};//string array
int[] to={R.id.textView,R.id.imageView};//int array of views id's
SimpleAdapter simpleAdapter=new
SimpleAdapter(this,arrayList,R.layout.list_view_items,from,to);//Create object and set the
parameters for simpleAdapter
simpleListView.setAdapter(simpleAdapter);//sets the adapter for listView

//perform listView item click event
simpleListView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> adapterView, View view, int i, long l) {

Toast.makeText(getApplicationContext(),fruitsNames[i],Toast.LENGTH_LONG).show();//show the
selected image in toast according to position
    }
});
}

}
```

Step 5:

Create new layout-> rec-> layout-> list_view_items.xml and add following code:

In this step we create a xml file for displaying ListView items. In this xml we add the code for displaying a ImageView and a TextView.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="#ffff">

    <ImageView
        android:id="@+id/imageView"
        android:layout_width="50dp"
        android:layout_height="50dp"
        android:padding="5dp"
        android:layout_alignParentRight="true"
        android:layout_marginRight="10dp"
        android:src="@drawable/ic_launcher" />

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="@dimen/activity_horizontal_margin"
        android:text="Demo"
        android:textColor="#0000" />
</RelativeLayout>
```

Output:

Now run the App and you will see different fruit names listed in ListView. Here we used Simple Adapter to fill data in ListView.

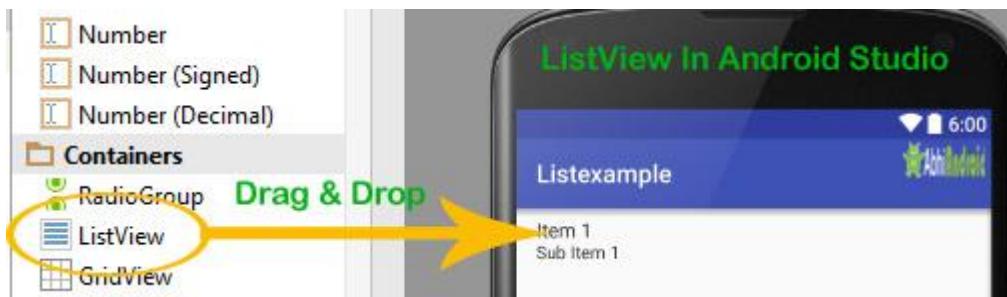
ListView

ListView is a list of scrollable items can be displayed in Android using ListView. It helps you to displaying the data in the form of a scrollable list. Users can then select any list item by clicking on it. ListView is default scrollable so we do not need to use scroll View or anything else with ListView.

ListView is widely used in android applications. A very common example of ListView is your phone contact book, where you have a list of your contacts displayed in a ListView and if you click on it then user information is displayed.

Adapter: To fill the data in a ListView we simply use adapters. List items are automatically inserted to a list using an Adapter that pulls the content from a source such as an arraylist, array or database.

ListView in Android Studio: Listview is present inside Containers. From there you can drag and drop on virtual mobile screen to create it. Alternatively you can also XML code to create it.

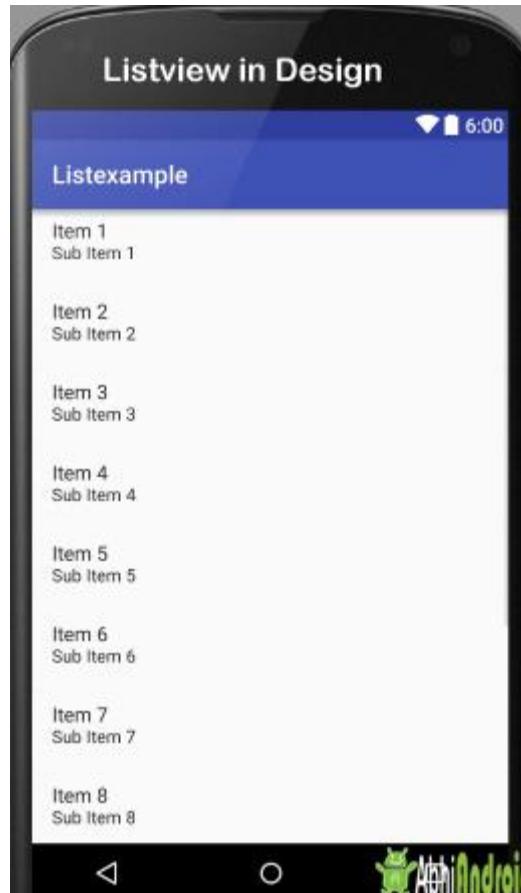


Here is Android ListView XML Code:

```
<ListView xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:tools="http://schemas.android.com/tools"  
    android:id="@+id/simpleListView"  
    android:layout_width="match_parent"
```

```
    android:layout_height="wrap_content"
    tools:context="abhiandroid.com.listexample.MainActivity">
</ListView>
```

Listview look in Design:



Attributes of ListView:

Lets see some different attributes of ListView which will be used while designing a custom list:

1. id: id is used to uniquely identify a ListView.

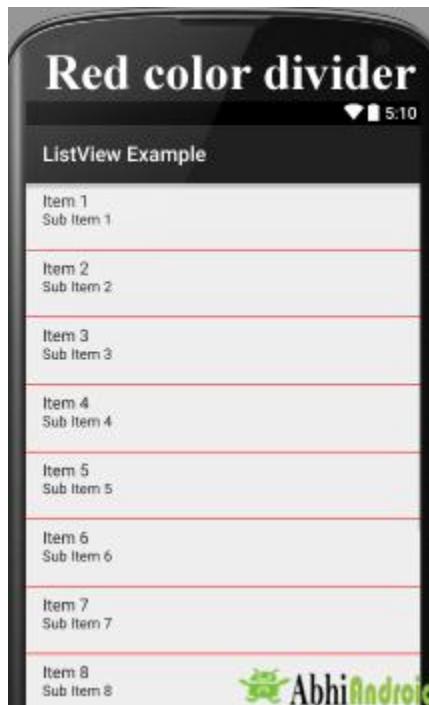
Below is the id attribute's example code with explanation included.

```
<!-- Id of a list view uniquely identify it-->
<ListView
    android:id="@+id/simpleListView"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
/>
```

2. divider: This is a drawable or color to draw between different list items.

Below is the divider example code with explanation included, where we draw red color divider between different views.

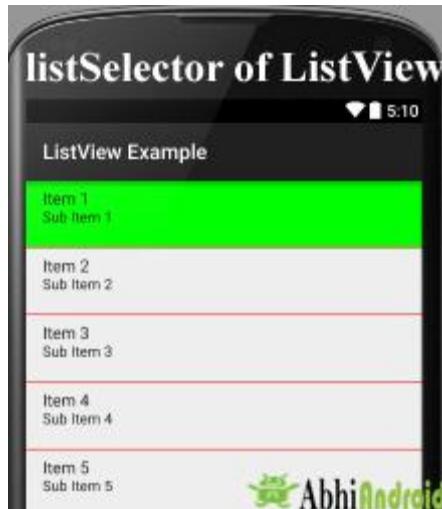
```
<!--Divider code in ListView-->
<ListView
    android:id="@+id/simpleListView"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:divider="#f00"
    android:dividerHeight="1dp"
/>
```



3. dividerHeight: This specify the height of the divider between list items. This could be in dp(density pixel),sp(scale independent pixel) or px(pixel).

In above example of divider we also set the divider height 1dp between the list items. The height should be in dp,sp or px.

4. listSelector: listSelector property is used to set the selector of the listView. It is generally orange or Sky blue color mostly but you can also define your custom color or an image as a list selector as per your design.



Below is listSelector example code with explanation includes, where list selector color is green, when you select any list item then that item's background color is green .

```
<!-- List Selector Code in ListView -->
<ListView
    android:id="@+id/simpleListView"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:divider="#f00"
    android:dividerHeight="1dp"
    android:listSelector="#0f0"/> <!--list selector in green color-->
```

Adapters Use in ListView:

An adapter is a bridge between UI component and data source that helps us to fill data in UI component. It holds the data and send the data to adapter view then view can takes the data

from the adapter view and shows the data on different views like as list view, grid view, spinner etc.

ListView is a subclass of AdapterView and it can be populated by binding to an Adapter, which retrieves the data from an external source and creates a View that represents each data entry.

In android commonly used adapters are:

Array Adapter

Base Adapter

Now we explain these two adapter in detail:

1.Array Adapter:

Whenever you have a list of single items which is backed by an array, you can use ArrayAdapter. For instance, list of phone contacts, countries or names.

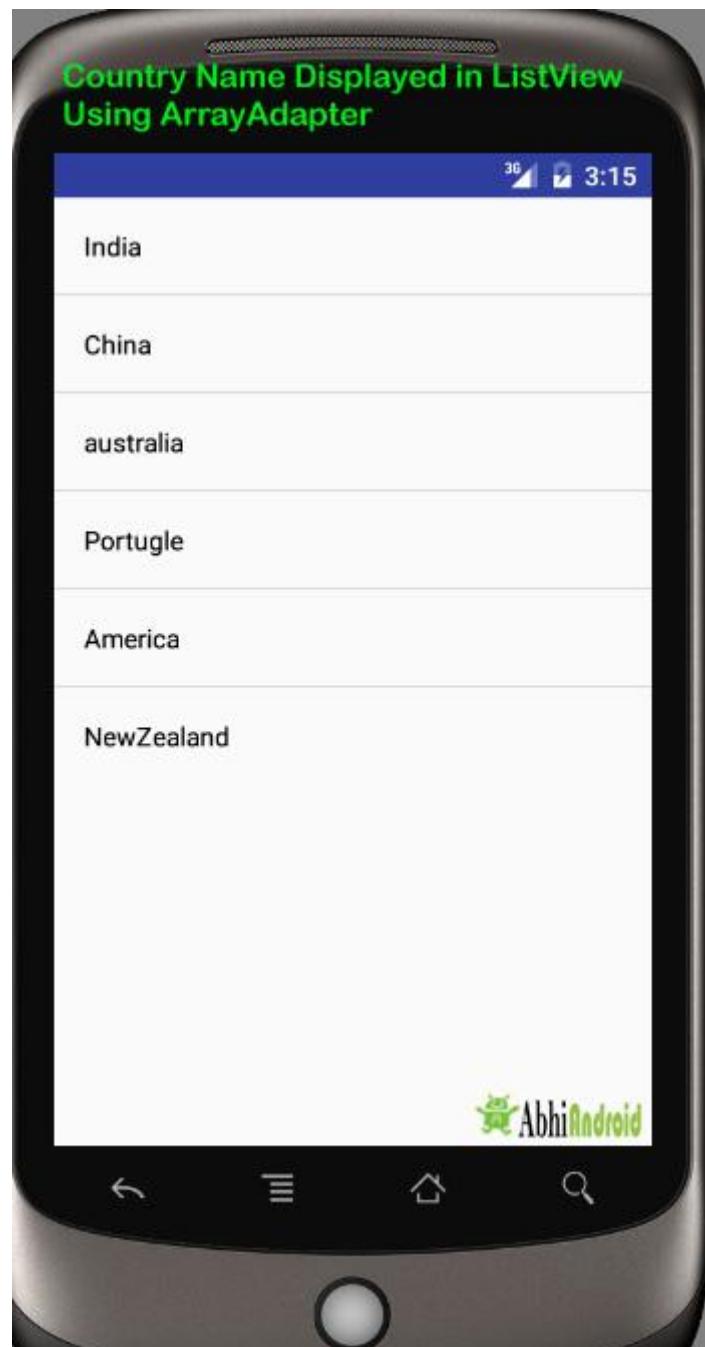
Important Note: By default, ArrayAdapter expects a Layout with a single TextView, If you want to use more complex views means more customization in list items, please avoid ArrayAdapter and use custom adapters.

Below is Array Adapter code:

```
ArrayAdapter adapter = new  
ArrayAdapter<String>(this,R.layout.ListView,R.id.textView,ArrayList);
```

Example of list view using Array Adapter:

In this example, we display a list of countries by using simple array adapter. Below is the final output we will create:



Step 1: Create a new project Listexample and activity Main Activity. Here we will create a ListView in LinearLayout. Below is the code of activity_main.xml or content_main.xml:

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

```

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <ListView
        android:id="@+id/simpleListView"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:divider="@color/material_blue_grey_800"
        android:dividerHeight="1dp" />
</LinearLayout>

```

Step 2: Create a new activity name Listview and below is the code of activity_listview.xml

```

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

    <TextView
        android:id="@+id/textView"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:padding="@dimen/activity_horizontal_margin"
        android:textColor="@color/black" />
</LinearLayout>

```

Step 3: Now in this final step we will use ArrayAdapter to display the country names in UI.

Below is the code of MainActivity.java

```

package abhiandroid.com.listexample;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.widget.ArrayAdapter; import android.widget.ListView;

public class MainActivity extends Activity
{
    // Array of strings...
    ListView simpleList;
    String countryList[] = {"India", "China", "australia", "Portugle", "America",
    "NewZealand"};

    @Override    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);      setContentView(R.layout.activity_main);
        simpleList = (ListView)findViewById(R.id.simpleListView);
        ArrayAdapter<String> arrayAdapter = new ArrayAdapter<String>(this,
R.layout.activity_listview, R.id.textView, countryList);
        simpleList.setAdapter(arrayAdapter);
    }
}

```

{}

Output Screen:

Now run the App in Emulator. You will see the below output screen where list of country names will be printed:



2. Base Adapter:

BaseAdapter is a common base class of a general implementation of an Adapter that can be used in ListView. Whenever you need a customized list you create your own adapter and extend base adapter in that. Base Adapter can be extended to create a custom Adapter for displaying a custom list item. ArrayAdapter is also an implementation of BaseAdapter.

Example of list view using Custom adapter(Base adapter):

In this example we display a list of countries with flags. For this, we have to use custom adapter as shown in example:



Step 1: Create a new project Listebasexample and activity Main Activity. Here we will create a ListView in LinearLayout. Below is the code of activity_main.xml or content_main.xml:

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

    <ListView
        android:id="@+id/simpleListView"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:divider="@color/material_blue_grey_800"
```

```
    android:dividerHeight="1dp"
    android:footerDividersEnabled="false" />
</LinearLayout>
```

Step 2: Create a new activity name Listview and below is the code of activity_listview.xml

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

    <ImageView
        android:id="@+id/icon"
        android:layout_width="50dp"
        android:layout_height="50dp"
        android:src="@drawable/ic_launcher" />

    <TextView
        android:id="@+id/textView"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:padding="@dimen/activity_horizontal_margin"
        android:textColor="@color/black" />
</LinearLayout>
```

Step 3: In third step we will use custom adapter to display the country names in UI by coding MainActivity.java. Below is the code of MainActivity.java

Important Note: Make sure flag images are stored in drawable folder present inside res folder with correct naming.

```
package com.abhiandroid.listbaseexample;

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

public class MainActivity extends Activity {
```

```
ListView simpleList;
String countryList[] = {"India", "China", "australia", "Portugle", "America", "NewZealand"};
int flags[] = {R.drawable.india, R.drawable.china, R.drawable.australia,
R.drawable.portugle, R.drawable.america, R.drawable.new_zealand};

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
simpleList = (ListView) findViewById(R.id.simpleListView);
CustomAdapter customAdapter = new CustomAdapter(getApplicationContext(), countryList,
flags);
simpleList.setAdapter(customAdapter);
}
}
```

Step 4: Now create another class Custom Adapter which will extend BaseAdapter. Below is the code of CustomAdapter.java

```
package com.abhiandroid.listbaseexample;

import android.content.Context;
import android.media.Image;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.ImageView;
import android.widget.TextView;

import java.util.zip.Inflater;

public class CustomAdapter extends BaseAdapter {
Context context;
String countryList[];
int flags[];
LayoutInflater inflter;

public CustomAdapter(Context applicationContext, String[] countryList, int[] flags) {
this.context = context;
this.countryList = countryList;
this.flags = flags;
inflter = (LayoutInflater.from(applicationContext));
}

@Override
public int getCount() {
return countryList.length;
}
```

```
@Override
public Object getItem(int i) {
    return null;
}

@Override
public long getItemId(int i) {
    return 0;
}

@Override
public View getView(int i, View view, ViewGroup viewGroup) {
    view = inflter.inflate(R.layout.activity_listview, null);
    TextView country = (TextView) view.findViewById(R.id.textView);
    ImageView icon = (ImageView) view.findViewById(R.id.icon);
    country.setText(countryList[i]);
    icon.setImageResource(flags[i]);
    return view;
}
```

Output:

Now run the App in Emulator and it will show you name of countries along with flags. Below is the output screen:



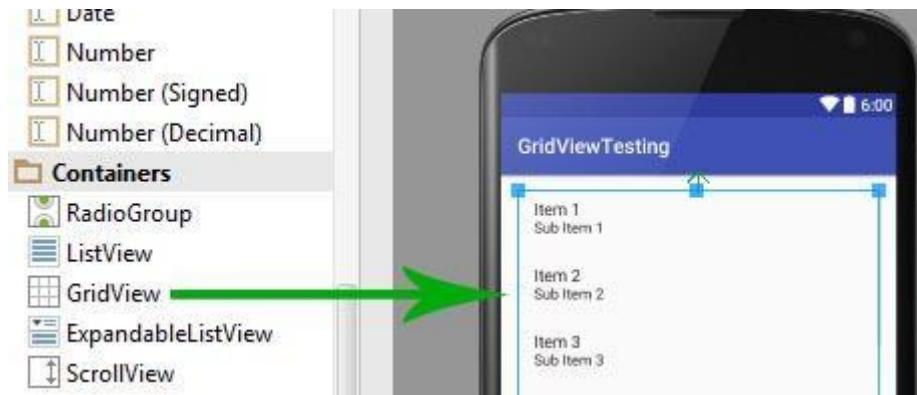
GridView

In android GridView is a view group that display items in two dimensional scrolling grid (rows and columns), the grid items are not necessarily predetermined but they are automatically inserted to the layout using a ListAdapter. Users can then select any grid item by clicking on it. GridView is default scrollable so we don't need to use ScrollView or anything else with GridView.

GridView is widely used in android applications. An example of GridView is your default Gallery, where you have number of images displayed using grid.

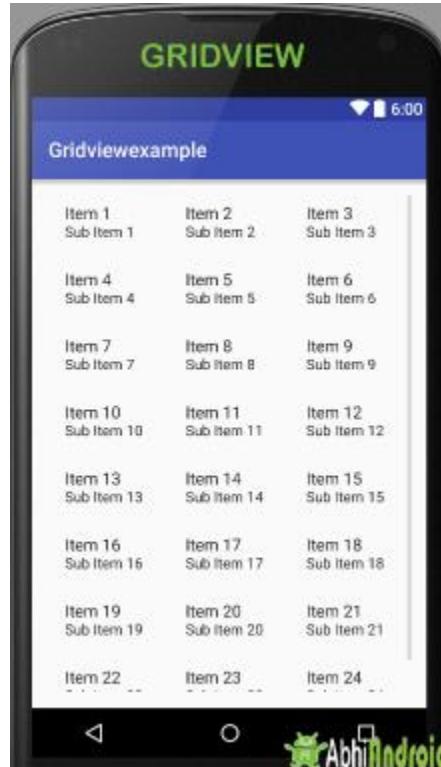
Adapter Is Used To Fill Data In Gridview: To fill the data in a GridView we simply use adapter and grid items are automatically inserted to a GridView using an Adapter which pulls the content from a source such as an arraylist, array or database.

GridView in Android Studio: Gridview is present inside Containers. From there you can drag and drop on virtual mobile screen to create it. Alternatively you can also XML code to create it.



Basic GridView code in XML:

```
<GridView  
    android:id="@+id/simpleGridView"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:numColumns="3"/>
```



Important Note: numColumns property has to be specified otherwise GridView behaves like a ListView with just singleChoice. In the above image numColumns property specified that there is 3 columns to show, if we set it to auto_fit then it automatically display as many column as possible to fill the available space of the screen. Even if the phone is in portrait mode or landscape mode it automatically fill the whole space.

Attributes of GridView:

Lets see different attributes of GridView which will be used while designing a custom grid view:

1.id: id is used to uniquely identify a GridView.

Below is the id attribute's example code with explanation included in which we don't specify the number of columns in a row that's why the GridView behaves like a ListView.

Below is the id attribute example code for Gridview:

```
<GridView
```

```
    android:id="@+id/simpleGridView"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
  />
```

2.numColumns: numColumn define how many columns to show. It may be a integer value, such as “5” or auto_fit.

auto_fit is used to display as many columns as possible to fill the available space on the screen.

Important Note: If we don’t specify numColumn property in GridView it behaves like a ListView with singleChoice.

Below is the numColumns example code where we define 4 columns to show in the screen.

```
<!-- numColumns example code -->
<GridView
  android:id="@+id/simpleGridView"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:numColumns="4"/> <!-- numColumns set to 4-->
```



3. horizontalSpacing: horizontalSpacing property is used to define the default horizontal spacing between columns. This could be in pixel(px), density pixel(dp) or scale independent pixel(sp).

Below is the horizontalSpacing example code with explanation included where horizontal spacing between grid items is 50 dp.

```
<!--Horizontal space example code in grid view-->
<GridView
    android:id="@+id/simpleGridView"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:numColumns="3"
    android:horizontalSpacing="50dp"/><!--50dp horizontal space between grid items-->
```



4. verticalSpacing: verticalSpacing property used to define the default vertical spacing between rows. This should be in px, dp or sp.

Below is the verticalSpacing example code with explanation included, where vertical spacing between grid items is 50dp.

```
<!-- Vertical space between grid items code -->
<GridView
    android:id="@+id/simpleGridView"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:numColumns="3"
    android:verticalSpacing="50dp"/><!--50dp vertical space set between grid items-->
```

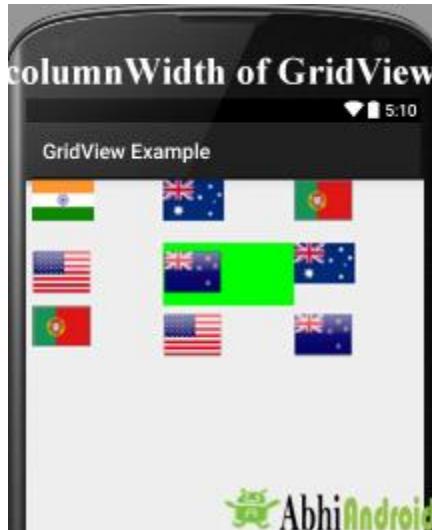


5.columnWidth: columnWidth property specifies the fixed width of each column. This could be in px, dp or sp.

Below is the columnWidth example code. Here column width is 80dp and selected item's background color is green which shows the actual width of a grid item.

Important Note: In the below code we also used listSelector property which define color for selected item. Also to see the output of columnWidth using listSelector we need to use Adapter which is our next topic. The below code is not sufficient to show you the output.

```
<!--columnWidth in Grid view code-->
<GridView
    android:id="@+id/simpleGridView"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:numColumns="3"
    android:columnWidth="80dp"
    android:listSelector="#0f0"/><!--define green color for selected item-->
```



GridView Example Using Different Adapters In Android Studio:

An adapter is a bridge between UI component and data source that helps us to fill data in UI component. It holds the data and sends the data to adapter view, then view can takes the data from the adapter view and shows the data on different views like as list view, grid view, spinner etc.

GridView and ListView both are subclasses of AdapterView and it can be populated by binding to an Adapter, which retrieves the data from an external source and creates a View that represents each data entry. In android commonly used adapters which fill data in GridView are:

1. Array Adapter
2. Base Adapter
3. Custom Array Adapter

Now we explain these adapters in detail:

1. Avoid Array Adapter To Fill Data In GridView:

Whenever you have a list of single items which is backed by an array, you can use ArrayAdapter. For instance, list of phone contacts, countries or names.

By default, ArrayAdapter expects a Layout with a single TextView, If you want to use more complex views means more customization in grid items, please avoid ArrayAdapter and use custom adapters.

```
ArrayAdapter adapter = new  
ArrayAdapter<String>(this,R.layout.ListView,R.id.textView,ArrayList);
```

2. GridView Using Base Adapter In Android:

Base Adapter is a common base class of a general implementation of an Adapter that can be used in GridView. Whenever you need a customized grid view you create your own adapter and extend base adapter in that. Base Adapter can be extended to create a custom Adapter for displaying custom grid items. ArrayAdapter is also an implementation of BaseAdapter. You can read BaseAdapter tutorial here.

Example of GridView using Base Adapter in Android Studio: Below is the example of GridView in Android, in which we show the Android logo's in the form of Grids. In this example

firstly we create an int type array for logo images and then call the Adapter to set the data in the GridView. In this we create a CustomAdapter by extending BaseAdapter in it. At Last we implement setOnItemClickListener event on GridView and on click of any item we send that item to another Activity and show the logo image in full size.

Below you can download code, see final output and step by step explanation of the example.



Step 1: Create a new Android project in Android Studio and fill all the required details. In our case we have named GridViewExample and package com.example.gourav.GridViewExample

Step 2: Open activity_main.xml and paste the below code. In this we have created a Grid view insideLinear Layout and also set number of columns to 3.

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

```

    android:orientation="vertical">
<!--
    GridView with 3 value for numColumns attribute
-->
<GridView
    android:id="@+id/simpleGridView"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:footerDividersEnabled="false"
    android:padding="1dp"
    android:numColumns="3" />
</LinearLayout>

```

Step 3: Create a new XML file and paste the below code. We have named activity_gridview.xml.

In this step we create a new XML file and add a ImageView in it. This file is used in CustomAdapter to set the logo images

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:padding="1dp"
    android:orientation="vertical">
    <ImageView
        android:id="@+id/icon"
        android:layout_width="match_parent"
        android:layout_height="120dp"
        android:scaleType="fitXY"
        android:layout_gravity="center_horizontal"
        android:src="@drawable/logo1" />
</LinearLayout>

```

Step 4: Now open drawable folder and save small size png images of different logo's and name them like logo1,logo2 and etc.

Step 5: Now open MainActivity.java and paste the below code. If your package name is different, don't copy it.

In this step firstly we get the reference of GridView and then create a int type array for Android logo's. After that we call the CustomAdapter and pass the array in it. At Last we implement setOnItemClickListener event on GridView and on click of any item we send that item to another Activity to show the logo image in Full Size. I have added comments in code to help you to understand the code easily so make you read the comments.

```

package com.example.gourav.GridViewExample;
import android.content.Intent;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.AdapterView;
import android.widget.GridView;
public class MainActivity extends AppCompatActivity {
    GridView simpleGrid;
    int logos[] = {R.drawable.logo1, R.drawable.logo2, R.drawable.logo3, R.drawable.logo4,
        R.drawable.logo5, R.drawable.logo6, R.drawable.logo7, R.drawable.logo8,
        R.drawable.logo9,
        R.drawable.logo10, R.drawable.logo11, R.drawable.logo12, R.drawable.logo13};
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        simpleGrid = (GridView) findViewById(R.id.simpleGridView); // init GridView
        // Create an object of CustomAdapter and set Adapter to GirdView
        CustomAdapter customAdapter = new CustomAdapter(getApplicationContext(), logos);
        simpleGrid.setAdapter(customAdapter);
        // implement setOnItemClickListener event on GridView
        simpleGrid.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view, int position, long id)
            {
                // set an Intent to Another Activity
                Intent intent = new Intent(MainActivity.this, SecondActivity.class);
                intent.putExtra("image", logos[position]); // put image data in Intent
                startActivity(intent); // start Intent
            }
        });
    }
}

```

Step 6:

Create a new class CustomAdapter and paste the below code.

In this step we create a CustomAdapter class by extending BaseAdapter in it. In this step we set the logo image's in the grid items. I have added comments in code to help you to understand the code easily so make you read the comments.

```

package com.example.gourav.GridViewExample;
import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.ImageView;
public class CustomAdapter extends BaseAdapter {
    Context context;
    int logos[];

```

```

LayoutInflater inflter;
public CustomAdapter(Context applicationContext, int[] logos) {
    this.context = applicationContext;
    this.logos = logos;
    inflter = (LayoutInflater.from(applicationContext));
}
@Override
public int getCount() {
    return logos.length;
}
@Override
public Object getItem(int i) {
    return null;
}
@Override
public long getItemId(int i) {
    return 0;
}
@Override
public View getView(int i, View view, ViewGroup viewGroup) {
    view = inflter.inflate(R.layout.activity_gridview, null); // inflate the layout
    ImageView icon = (ImageView) view.findViewById(R.id.icon); // get the reference of
    ImageView
    icon.setImageResource(logos[i]); // set logo images
    return view;
}
}
}

```

Step 7: Now Create a new XML file named activity_second and paste the below code in it.

In this step we create an XML file for our Second Activity to display the logo image in full size.

```

<?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"
    android:background="#fff"
    tools:context="com.example.gourav.GridViewExample.SecondActivity">
    <ImageView
        android:id="@+id/selectedImage"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:scaleType="fitXY" />
</RelativeLayout>

```

Step 8: Now Create a new Activity with name SecondActivity.class and add below code in it.

In this step we create a new Activity in which firstly we initiate the ImageView and then get the Image from our Previous Activity by using Intent object and set the same in the ImageView.

```
package com.example.gourav.GridViewExample;
import android.content.Intent;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.widget.ImageView;
public class SecondActivity extends AppCompatActivity {
    ImageView selectedImage;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_second);
        selectedImage = (ImageView) findViewById(R.id.selectedImage); // init a ImageView
        Intent intent = getIntent(); // get Intent which we set from Previous Activity
        selectedImage.setImageResource(intent.getIntExtra("image", 0)); // get image from
        Intent and set it in ImageView
    }
}
```

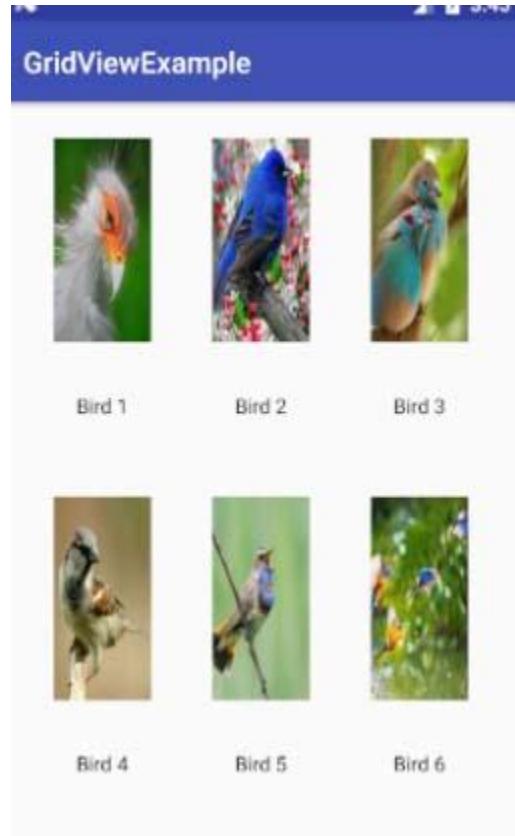
Output: Now run the App and you will see different Android images in GridView. Click on the any image and full size of it will open up.

3. GridView Example Using Custom ArrayAdapter In Android Studio:

ArrayAdapter is also an implementation of BaseAdapter so if we want more customization then we create a custom adapter and extend ArrayAdapter in that. Here we are creating GridView using custom array adapter.

Example of GridView using Custom Adapter : Example of Grid View using custom arrayadapter to show birds in the form of grids. Below is the code and final output:

Below you can download code, see final output and step by step explanation of the topic.



Step 1: Create a new project and name it GridViewExample.

Step 2: Now open app -> res -> layout -> activity_main.xml (or) main.xml and add following code :

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="abhiandroid.com.gridviewexample.MainActivity">

    <GridView
        android:id="@+id/simpleGridView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:numColumns="3" />

</RelativeLayout>
```

Step 3: Create a new layout Activity in app -> res-> layout-> new activity and name it grid_view_items.xml and add following code:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/grid_view_items"
    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="abhiandroid.com.gridviewexample.GridViewItems">

    <ImageView
        android:id="@+id/imageView"
        android:layout_width="150dp"
        android:layout_height="150dp"
        android:padding="5dp"
        android:scaleType="fitXY"
        android:src="@drawable/ic_launcher"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true" />

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="@dimen/activity_horizontal_margin"
        android:text="Demo"
        android:textColor="#000"
        android:layout_below="@+id/imageView"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="13dp" />
</RelativeLayout>
```

Step 4: Now open app -> java -> package -> MainActivity.java

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.GridView;
import java.util.ArrayList;

public class MainActivity extends AppCompatActivity {

    GridView simpleList;
```

```

ArrayList birdList=new ArrayList<>();
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    simpleList = (GridView) findViewById(R.id.simpleGridView);
    birdList.add(new Item("Bird 1",R.drawable.b1));
    birdList.add(new Item("Bird 2",R.drawable.b2));
    birdList.add(new Item("Bird 3",R.drawable.b3));
    birdList.add(new Item("Bird 4",R.drawable.b4));
    birdList.add(new Item("Bird 5",R.drawable.b5));
    birdList.add(new Item("Bird 6",R.drawable.b6));

    MyAdapter myAdapter=new MyAdapter(this,R.layout.grid_view_items,birdList);
    simpleList.setAdapter(myAdapter);
}
}

```

Step5 : Create a new Class src -> package -> MyAdapter.java and add the following code:

```

import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.ImageView;
import android.widget.TextView;

import java.util.ArrayList;

public class MyAdapter extends ArrayAdapter {

    ArrayList birdList = new ArrayList<>();

    public MyAdapter(Context context, int textViewResourceId, ArrayList objects) {
        super(context, textViewResourceId, objects);
        birdList = objects;
    }

    @Override
    public int getCount() {
        return super.getCount();
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {

        View v = convertView;
        LayoutInflater inflater = (LayoutInflater)
getContext().getSystemService(Context.LAYOUT_INFLATER_SERVICE);
        v = inflater.inflate(R.layout.grid_view_items, null);
        TextView textView = (TextView) v.findViewById(R.id.textView);

```

```
    ImageView imageView = (ImageView) v.findViewById(R.id.imageView);
    textView.setText(birdList.get(position).getbirdName());
    imageView.setImageResource(birdList.get(position).getbirdImage());
    return v;
}
}
```

Step 6: Create a new Class src -> package -> Item.java and add the below code:

```
public class Item {
    String birdListName;
    int birdListImage;

    public Item(String birdName,int birdImage)
    {
        this.birdListImage=birdImage;
        this.birdListName=birdName;
    }
    public String getbirdName()
    {
        return birdListName;
    }
    public int getbirdImage()
    {
        return birdListImage;
    }
}
```

Output:

Now run the App and you will see different bird images in GridView.

TextView

In Android, TextView displays text to the user and optionally allows them to edit it programmatically. TextView is a complete text editor, however basic class is configured to not allow editing but we can edit it.



View is the parent class of TextView. Being a subclass of view the text view component can be used in your app's GUI inside a ViewGroup, or as the content view of an activity.

We can create a TextView instance by declaring it inside a layout(XML file) or by instantiating it programmatically(Java Class).

TextView code in XML:

```
<TextView android:id="@+id/simpleTextView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="AbhiAndroid" />
```

TextView code in JAVA:

```
TextView textView = (TextView) findViewById(R.id.textView);
textView.setText("AbhiAndroid"); //set text for text view
```

Attributes of TextView:

Now let's we discuss about the attributes that helps us to configure a TextView in your xml file.

1. id: id is an attribute used to uniquely identify a text view. Below is the example code in which we set the id of a text view.

```
<TextView
    android:id="@+id/simpleTextView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
```

2. gravity: The gravity attribute is an optional attribute which is used to control the alignment of the text like left, right, center, top, bottom, center_vertical, center_horizontal etc.

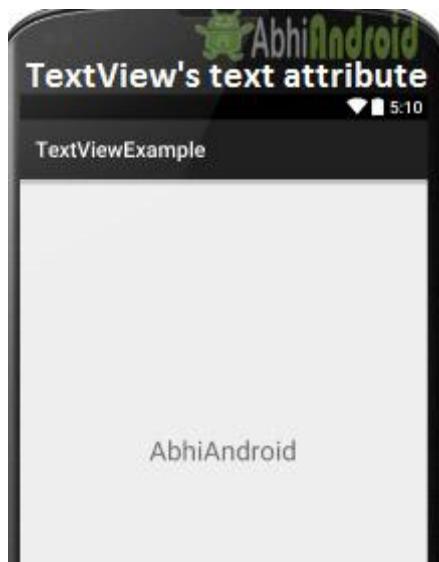
Below is the example code with explanation included in which we set the center_horizontal gravity for text of a TextView.

```
<TextView
    android:id="@+id/simpleTextView"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="AbhiAndroid"
    android:textSize="20sp"
    android:gravity="center_horizontal"/> <!--center horizontal gravity-->
```

3. text: text attribute is used to set the text in a text view. We can set the text in xml as well as in the java class.

Below is the example code with explanation included in which we set the text "AbhiAndroid" in a text view.

```
<TextView  
    android:id="@+id/simpleTextView"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_centerInParent="true"  
    android:textSize="25sp"  
    android:text="AbhiAndroid"/><!--Display Text as AbhiAndroid-->
```



In Java class:

Below is the example code in which we set the text in a textview programmatically means in java class.

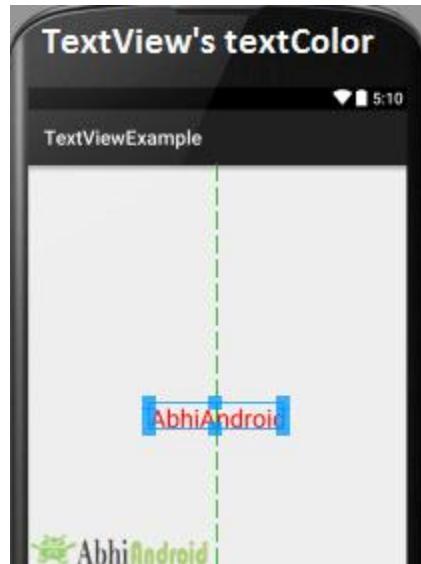
```
TextView textView = (TextView) findViewById(R.id.textView);  
textView.setText("AbhiAndroid"); //set text for text view
```

4. textColor: textColor attribute is used to set the text color of a text view. Color value is in the form of "#argb", "#rgb", "#rrggbb", or "#aarrggbb".

Below is the example code with explanation included in which we set the red color for the displayed text.

```
<TextView  
    android:id="@+id/simpleTextView"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="AbhiAndroid"  
    android:layout_centerInParent="true"
```

```
android:textSize="25sp"
android:textColor="#f00"/><!--red color for text view-->
```



In Java class:

Below is the example code in which we set the text color of a text view programmatically means in java class.

```
TextView textView = (TextView)findViewById(R.id.textView);
textView.setTextColor(Color.RED); //set red color for text view
```

5. textSize: textSize attribute is used to set the size of text of a text view. We can set the text size in sp(scale independent pixel) or dp(density pixel).

Below is the example code in which we set the 20sp size for the text of a text view.

```
<TextView
    android:id="@+id/simpleTextView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="AbhiAndroid"
    android:layout_centerInParent="true"
```

```
    android:textSize="40sp" /><!--Set size-->
```



In Java class:

Below is the example code in which we set the text size of a text view programmatically means in java class.

```
TextView textView = (TextView) findViewById(R.id.textView);
textView.setTextSize(20); //set 20sp size of text
```

6. textStyle: textStyle attribute is used to set the text style of a text view. The possible text styles are bold, italic and normal. If we need to use two or more styles for a text view then “|” operator is used for that.

Below is the example code with explanation included in which we set the bold and italic text styles for text.

```
<TextView
    android:id="@+id/simpleTextView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="AbhiAndroid"
    android:layout_centerInParent="true"
    android:textSize="40sp"
    android:textStyle="bold|italic"/><!--bold and italic text style of text-->
```



7. background: background attribute is used to set the background of a text view. We can set a color or a drawable in the background of a text view.

8. padding: padding attribute is used to set the padding from left, right, top or bottom. In above example code of background we also set the 10dp padding from all the side's of text view.

Below is the example code with explanation included in which we set the black color for the background, white color for the displayed text and set 10dp padding from all the side's for text view.

```
<TextView  
    android:id="@+id/simpleTextView"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="AbhiAndroid"  
    android:layout_centerInParent="true"  
    android:textSize="40sp"  
    android:padding="10dp"  
    android:textColor="#ffff"  
    android:background="#000"/> <!--red color for background of text view-->
```



In Java class:

Below is the example code in which we set the background color of a text view programmatically means in java class.

```
TextView textView = (TextView) findViewById(R.id.textView);
textView.setBackgroundColor(Color.BLACK); //set background color
```

Example of TextView:

Below is the example of TextView in which we display a text view and set the text in xml file and then change the text on button click event programmatically. Below is the final output and code:



Step 1: Create a new project and name it textViewExample.

Select File -> New -> New Project. Fill the forms and click "Finish" button.

Step 2: Open res -> layout -> xml (or) activity_main.xml and add following code. Here we will create a button and a textView in Relative Layout.

```
<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/simpleTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:text="Before Clicking"
        android:textColor="#f00"
        android:textSize="25sp"
        android:textStyle="bold|italic"
        android:layout_marginTop="50dp"/>

    <Button
        android:id="@+id/btnChangeText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:background="#f00"
        android:padding="10dp"
        android:text="Change Text"
        android:textColor="#fff"
        android:textStyle="bold" />
</RelativeLayout>
```

Step 3: Open app -> java -> package and open MainActivity.java and add the following code. Here we will change the text of TextView after the user click on Button.

```
package example.abhiandriod.textviewexample;

import android.graphics.Color;
import android.support.v7.app.AppCompatActivity;
```

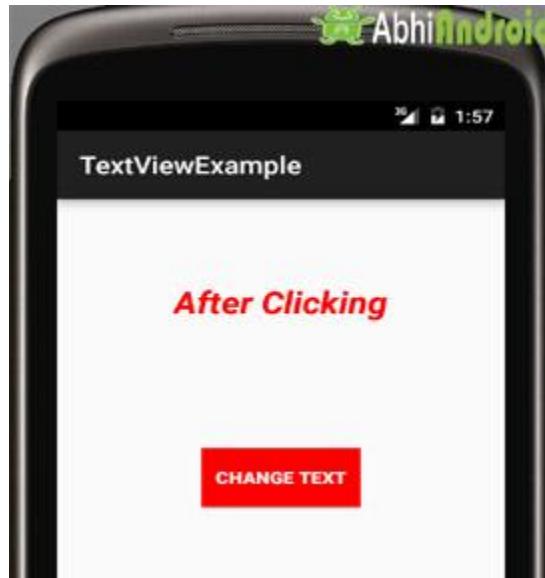
```
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main); //set the layout
        final TextView simpleTextView = (TextView) findViewById(R.id.simpleTextView); //get
the id for TextView
        Button changeText = (Button) findViewById(R.id.btnChangeText); //get the id for
button
        changeText.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                simpleTextView.setText("After Clicking"); //set the text after clicking
button
            }
        });
    }
}
```

Output:

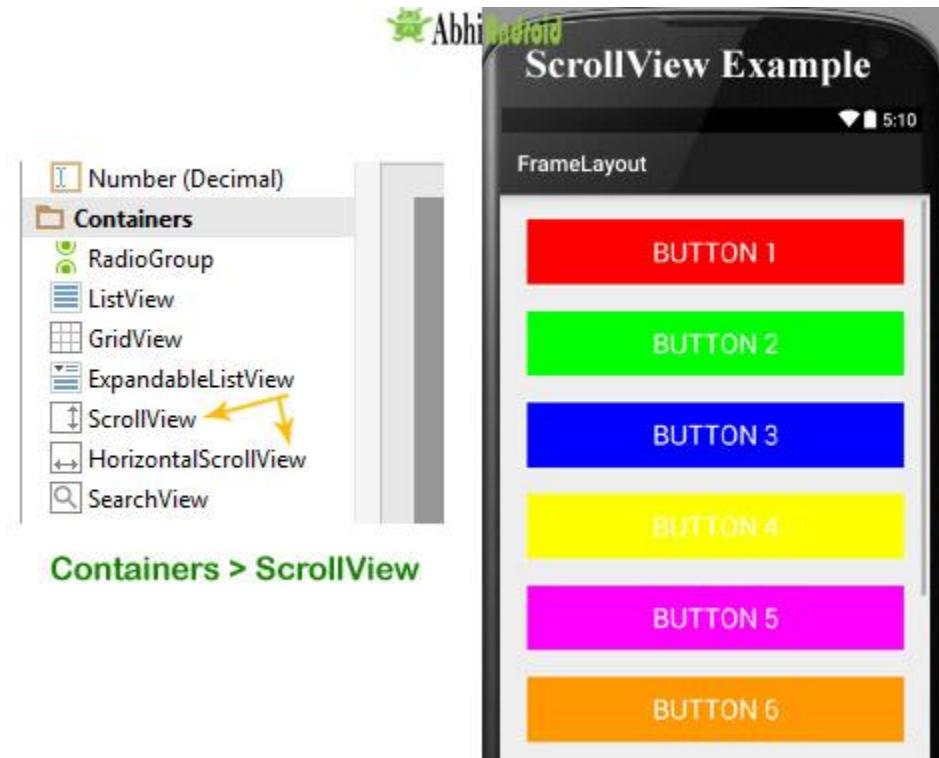
Now run the app in Emulator and click on the button. You will see text will change “After Clicking”.



ScrollView

In android ScrollView can hold only one direct child. This means that, if you have complex layout with more views(Buttons, TextViews or any other view) then you must enclose them inside another standard layout like Table Layout, Relative Layout or Linear Layout. You can specify layout_width and layout_height to adjust width and height of screen. You can specify height and width in dp(density pixel) or px(pixel). Then after enclosing them in a standard layout, enclose the whole layout in ScrollView to make all the element or views scrollable.

ScrollView in Android Studio Design: It is present inside Containers >> ScrollView or HorizontalScrollView



Important Note 1: We never use a Scroll View with a ListView because List View is default scrollable(i.e. vertical scrollable). More importantly, doing this affects all of the important optimizations in a List View for dealing with large lists(list items). Just because it effectively forces the List View to display its entire list of items to fill up the infinite container supplied by a ScrollView so we don't use it with List View.

Important Note 2: In android default ScrollView is used to scroll the items in vertical direction and if you want to scroll the items horizontally then you need to implement horizontal ScrollView.

ScrollView Syntax:

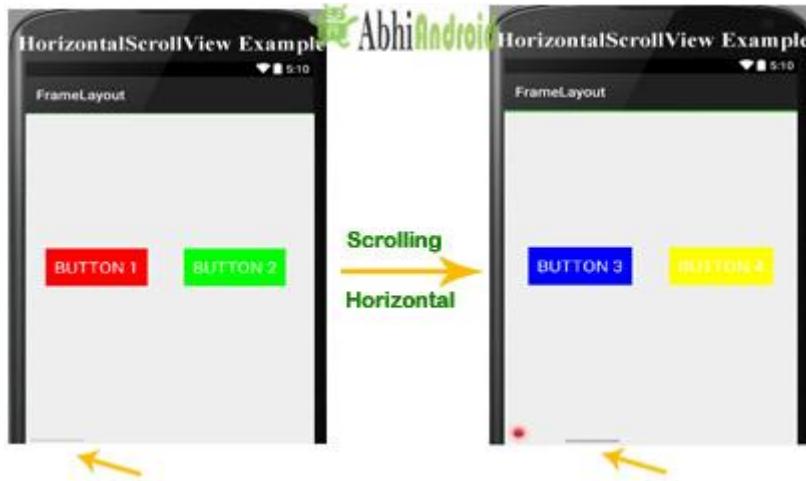
```
<ScrollView  
    android:id="@+id/scrollView"  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent">  
  
    <!-- add child view's here -->  
  
</ScrollView>
```

Here is how Scroll View looks in Android:



Horizontal ScrollView:

In android, You can scroll the elements or views in both vertical and horizontal directions. To scroll in Vertical we simply use ScrollView as we shown in the previous code of this article and to scroll in horizontal direction we need to use HorizontalScrollView.



Below is HorizontalScrollView syntax in Android is:

```
<HorizontalScrollView  
    android:id="@+id/horizontalscrollView"  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent">  
  
    <!-- add child view's here -->  
  
</HorizontalScrollView >
```

Attributes Of Scroll View:

ScrollView and HorizontalScrollView has same attributes, the only difference is scrollView scroll the child items in vertical direction while horizontal scroll view scroll the child items in horizontal direction.

Now let's we discuss about the attributes that helps us to configure a ScrollView and its child controls. Some of the most important attributes you will use with ScrollView include:

1. **id:** In android, id attribute is used to uniquely identify a ScrollView.

Below is id attribute's example code with explanation included.

```
<ScrollView  
    android:id="@+id/scrollView"  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent"  
/>
```

2. scrollbars: In android, scrollbars attribute is used to show the scrollbars in horizontal or vertical direction. The possible Value of scrollbars is vertical, horizontal or none. By default scrollbars is shown in vertical direction in scrollView and in horizontal direction in HorizontalScrollView.

Below is scrollbars attribute's example code in which we set the scrollbars in vertical direction.

```
< HorizontalScrollView  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent"  
    android:scrollbars="vertical"/><!--scrollbars in vertical direction-->
```

Example of ScrollView In Android Studio:

Example 1: In this example we will use 10 button and scroll them using ScrollView in vertical direction. Below is the code and final Output we will create:



Step 1: Create a new project in Android Studio and name it scrollviewExample.

Select File -> New -> New Project -> Android Application Project (or) Android Project. Fill the forms and click “Finish” button.

Step 2: Open res -> layout -> activity_main.xml (or) main.xml and add below code. Here we are creating a Relative Layout having 10 buttons which are nested in Linear Layout and then in ScrollView.

Important Note: Remember ScrollView can hold only one direct child. So we have to jointly put 10 buttons inside Linear Layout to make it one child. And then we put it inside ScrollView.

```
<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">

    <ScrollView
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:scrollbars="vertical">

        <LinearLayout
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:layout_margin="20dp"
            android:orientation="vertical">

            <Button
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:layout_gravity="center"
                android:background="#f00"
                android:text="Button 1"
                android:textColor="#fff"
                android:textSize="20sp" />

            <Button
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:layout_gravity="center"
                android:layout_marginTop="20dp"
                android:background="#0f0"
                android:text="Button 2"
                android:textColor="#fff"
                android:textSize="20sp" />

            <Button
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:layout_gravity="center"
                android:layout_marginTop="20dp"
                android:background="#00f"
                android:text="Button 3"
                android:textColor="#fff"
```

```
        android:textSize="20sp" />

    <Button
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginTop="20dp"
        android:background="#ff0"
        android:text="Button 4"
        android:textColor="#fff"
        android:textSize="20sp" />

    <Button
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginTop="20dp"
        android:background="#f0f"
        android:text="Button 5"
        android:textColor="#fff"
        android:textSize="20sp" />

    <Button
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginTop="20dp"
        android:background="#f90"
        android:text="Button 6"
        android:textColor="#fff"
        android:textSize="20sp" />

    <Button
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginTop="20dp"
        android:background="#f00"
        android:text="Button 7"
        android:textColor="#ff9"
        android:textSize="20sp" />

    <Button
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginTop="20dp"
        android:background="#444"
        android:text="Button 8"
        android:textColor="#fff"
        android:textSize="20sp" />

    <Button
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginTop="20dp"
```

```
    android:background="#ff002211"
    android:text="Button 9"
    android:textColor="#fff"
    android:textSize="20sp" />

    <Button
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginTop="20dp"
        android:background="#0f0"
        android:text="Button 10"
        android:textColor="#fff"
        android:textSize="20sp" />

</LinearLayout>
</ScrollView>
</RelativeLayout>
```

Step 3: Now Open src -> package -> MainActivity.java and paste the below code

```
package com.example.gourav.scrollviewExample;

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

public class MainActivity extends AppCompatActivity {

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

Step 4: Now open manifest.xml and paste the below code

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
```

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

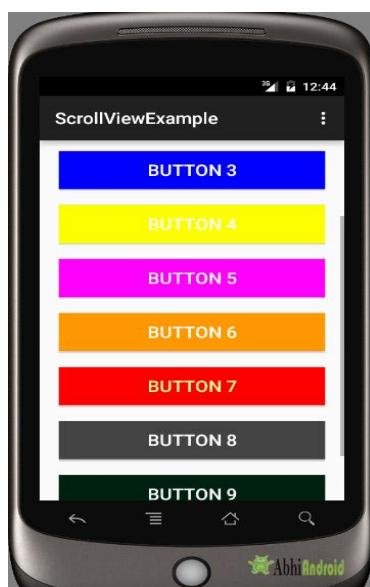
</manifest>
```

Step 5: Lastly open res ->values -> strings.xml and paste the below code

```
<resources>
<string name="app_name">ScrollViewExample</string>
<string name="hello_world">Hello world!</string>
<string name="action_settings">Settings</string>
</resources>
```

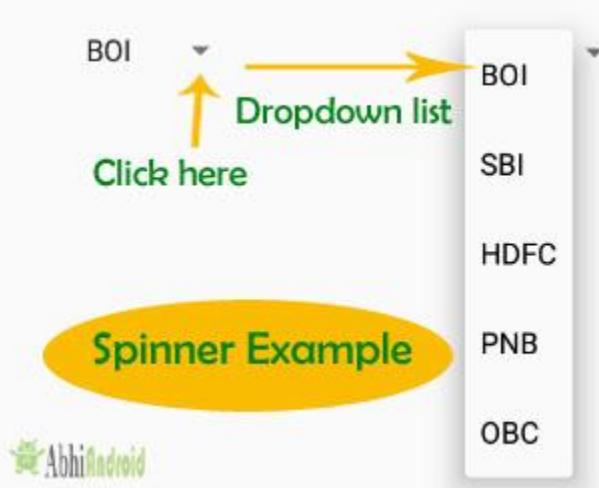
Output:

Now run the App in Emulator / AVD or in real device. You will see the 10 buttons which can be scrollable in vertical direction.



Spinner

In Android, Spinner provides a quick way to select one value from a set of values. Android spinners are nothing but the drop down-list seen in other programming languages. In a default state, a spinner shows its currently selected value. It provides an easy way to select a value from a list of values.



In Simple Words we can say that a spinner is like a combo box of AWT or swing where we can select a particular item from a list of items. Spinner is a sub class of AdapterView class.

Important Note: Spinner is associated with Adapter view so to fill the data in spinner we need to use one of the Adapter class.

Here is the XML basic code for Spinner:

```
<Spinner  
    android:id="@+id/simpleSpinner "  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content" />
```

Important Note: To fill the data in a spinner we need to implement an adapter class. A spinner is mainly used to display only text field so we can implement ArrayAdapter for that. We can also use Base Adapter and other custom adapters to display a spinner with more customize list. Suppose if we need to display a TextView and a ImageView in spinner item list

then array adapter is not enough for that. Here we have to implement custom adapter in our class. Below image of Spinner and Custom Spinner will make it more clear.



ArrayAdapter:

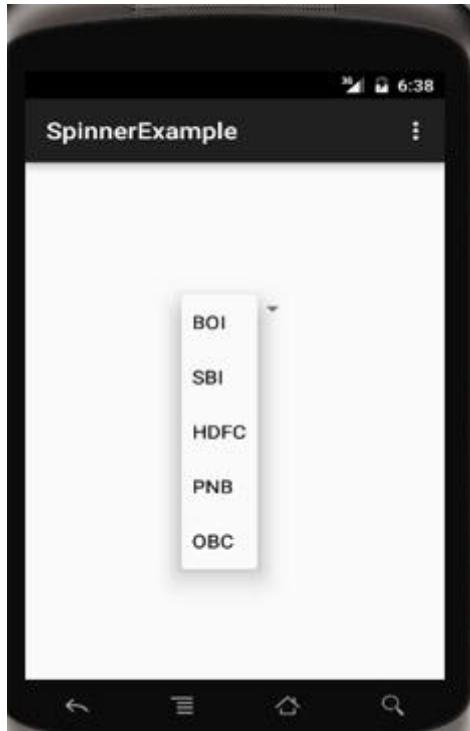
An adapter is a bridge between UI component and data source that helps us to fill data in UI component. It holds the data and send the data to adapter view then view can takes the data from the adapter view and shows the data on different views like as list view, grid view, spinner. Whenever you have a list of single items which is backed by an array, you can use Array Adapter.

Here is code of ArrayAdapter in Android:

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

Example of Spinner In Android Studio:

Example 1: Below is the example in which we display a list of bank names in a spinner and whenever you select an item the value will be displayed using toast on Mobile screen. Below is the final output and code:



Step 1: Create a new project in Android Studio and name it SpinnerExample.

Select File -> New -> New Project ->. Fill the forms and click "Finish" button.

Step 2: Open res -> layout -> activity_main.xml (or) main.xml and add following code. Here we will create a Spinner inside Relative Layout.

```
<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">

    <Spinner
        android:id="@+id/simpleSpinner"
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="100dp" />

</RelativeLayout>
```

Step 3: Now open app-> java -> package -> MainActivity.java and add the following code. Here we will use ArrayAdapter to fill the data in Spinner. Also we are using Toast to display when the item in Spinner is selected.

```
package example.abhiandriod.spinnerexample;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Spinner;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity implements
    AdapterView.OnItemSelectedListener{

    String[] bankNames={"BOI","SBI","HDFC","PNB","OBC"};
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //Getting the instance of Spinner and applying OnItemSelectedListener on it
        Spinner spin = (Spinner) findViewById(R.id.simpleSpinner);
        spin.setOnItemSelectedListener(this);

        //Creating the ArrayAdapter instance having the bank name list
        ArrayAdapter aa = new ArrayAdapter(this,android.R.layout.simple_spinner_item,bankNames);
        aa.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
        //Setting the ArrayAdapter data on the Spinner
        spin.setAdapter(aa);
    }

    //Performing action onItemSelected and onNothing selected
    @Override
    public void onItemSelected(AdapterView<?> arg0, View arg1, int position, long id) {
        Toast.makeText(getApplicationContext(), bankNames[position], Toast.LENGTH_LONG).show();
    }

    @Override
    public void onNothingSelected(AdapterView<?> arg0) {
        // TODO Auto-generated method stub
    }
}
```

```
}
```

Output:

Now run the program in Emulator and you will see options to choose among bank names present inside drop down list. You will also see Toast message displaying on the screen when you will select the particular bank.

Custom Spinner:

Custom Spinner is used to display a spinner item with image, text etc (i.e. creating more custom list item). It is achieved in Android using custom adapter like base adapter.

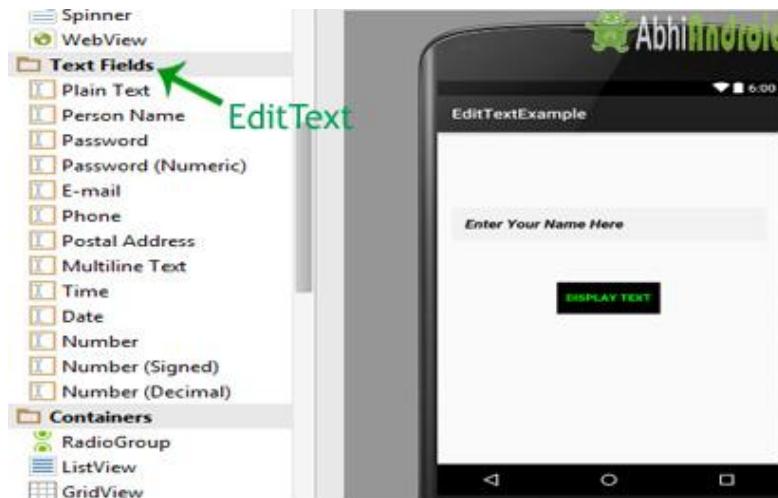


EditText

In Android, EditText is a standard entry widget in android apps. It is an overlay over TextView that configures itself to be editable. EditText is a subclass of TextView with text editing operations. We often use EditText in our applications in order to provide an input or text field, especially in forms. The most simple example of EditText is Login or Sign-in form.



Text Fields in Android Studio are basically EditText:



Important Note: An EditText is simply a thin extension of a TextView. An EditText inherits all the properties of a TextView.

EditText Code:

We can create a EditText instance by declaring it inside a layout(XML file) or by instantiating it programmatically (i.e. in Java Class).

EditText code in XML:

```
<EditText
    android:id="@+id/simpleEditText"
    android:layout_height="wrap_content"
    android:layout_width="match_parent"/>
```

Retrieving / Getting the Value From EditText In Java Class:

Below is the example code of EditText in which we retrieve the value from a EditText in Java class. We have used this code in the example you will find at the end of this post.

```
EditText simpleEditText = (EditText) findViewById(R.id.simpleEditText);
String editTextValue = simpleEditText.getText().toString();
```

Attributes of EditText:

Now let's we discuss few attributes that helps us to configure a EditText in your xml.

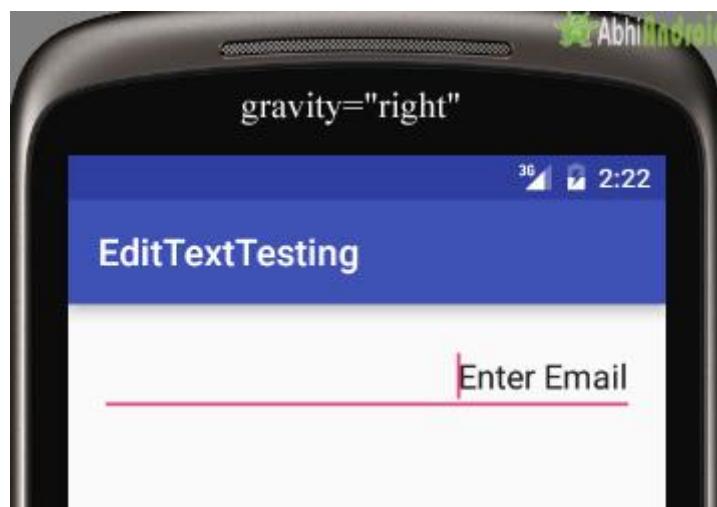
1. id: id is an attribute used to uniquely identify a text EditText. Below is the example code in which we set the id of a edit text.

```
<EditText  
    android:id="@+id/simpleEditText"  
    android:layout_height="wrap_content"  
    android:layout_width="match_parent"/>
```

2. gravity: The gravity attribute is an optional attribute which is used to control the alignment of the text like left, right, center, top, bottom, center_vertical, center_horizontal etc.

Below is the example code with explanation included in which we set the right gravity for text of a EditText.

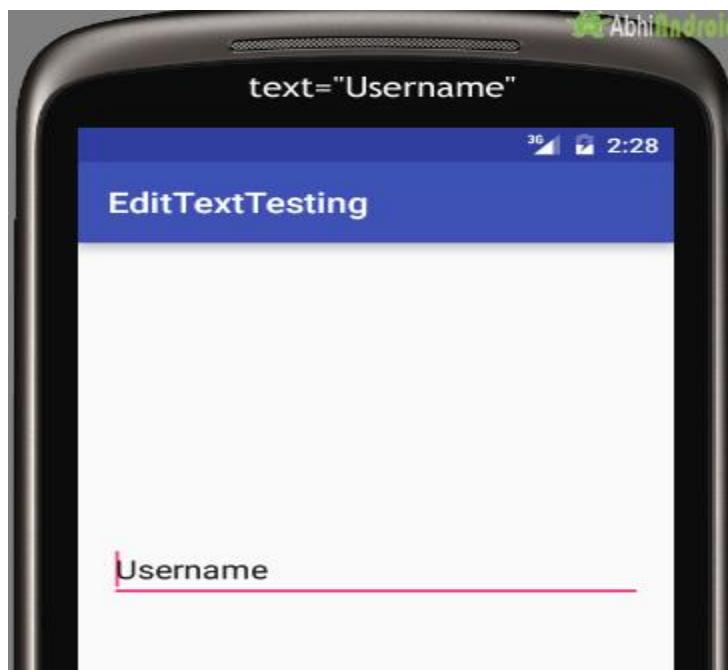
```
<EditText  
    android:id="@+id/simpleEditText"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:text="Enter Email"  
    android:gravity="right"/><!--gravity of a edit text-->
```



3. text: text attribute is used to set the text in a EditText. We can set the text in xml as well as in the java class.

Below is the example code in which we set the text “Username” in a edit text.

```
<EditText  
    android:id="@+id/simpleEditText"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:layout_centerInParent="true"  
    android:text="Username"/><!--set text in edit text-->
```



Setting text in EditText In Java class:

Below is the example code in which we set the text in a text view programmatically means in java class.

```
EditText editText = (EditText) findViewById(R.id.simpleEditText);  
editText.setText("Username");//set the text in edit text
```

4. hint: hint is an attribute used to set the hint i.e. what you want user to enter in this edit text. Whenever user start to type in edit text the hint will automatically disappear.

Below is the example code with explanation in which we set the hint of a edit text.

```
<EditText  
    android:id="@+id/simpleEditText"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:layout_centerInParent="true"  
    android:hint="Enter Your Name Here" /><!--display the hint-->
```



Setting hint in EditText In Java class:

Below is the example code in which we set the text in a text view programmatically means in java class.

```
EditText editText = (EditText) findViewById(R.id.simpleEditText);
```

```
editText.setHint("Enter Your Name Here");//display the hint
```

5. textColor: textColor attribute is used to set the text color of a text edit text. Color value is in the form of “#argb”, “#rgb”, “#rrggb”, or “#aarrggbb”.

Below is the example code with explanation included in which we set the red color for the displayed text of a edit text.

```
<EditText  
    android:id="@+id/simpleEditText"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:layout_centerInParent="true"  
    android:text="Password"  
    android:textColor="#f00"/><!--set the red text color-->
```



Setting textColor in EditText In Java class:

Below is the example code in which we set the text color of a edit text programmatically means in java class.

```
EditText simpleEditText=(EditText)findViewById(R.id.simpleEditText);  
simpleEditText.setTextColor(Color.RED);//set the red text color
```

6. **textColorHint:** textColorHint is an attribute used to set the color of displayed hint.

Below is the example code with explanation included in which we set the green color for displayed hint of a edit text.

```
<EditText  
    android:id="@+id/simpleEditText"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:layout_centerInParent="true"  
    android:hint="Enter Your Name Here"  
    android:textColorHint="#0f0"/><!--set the hint color green-->
```



Setting textColorHint in EditText In Java class:

Below is the example code in which we set the hint color of a edit text programmatically means in java class.

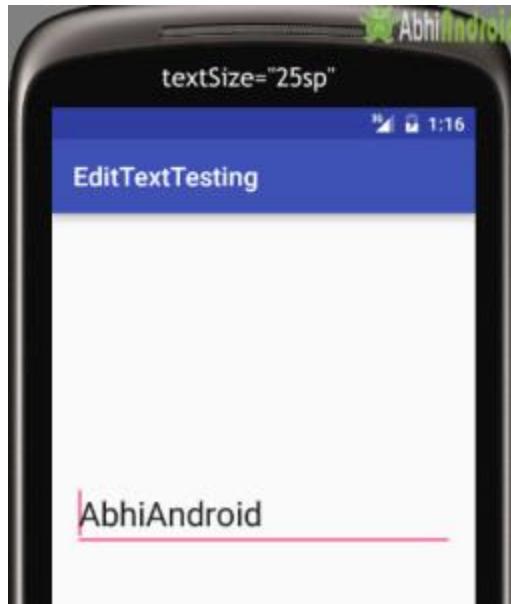
```
EditText simpleEditText=(EditText)findViewById(R.id.simpleEditText);
```

```
simpleEditText.setHintTextColor(Color.green(0)); //set the green hint color
```

7. textSize: textSize attribute is used to set the size of text of a edit text. We can set the text size in sp(scale independent pixel) or dp(density pixel).

Below is the example code in which we set the 25sp size for the text of a edit text.

```
<EditText  
    android:id="@+id/simpleEditText"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:layout_centerInParent="true"  
    android:text="AbhiAndroid"  
    android:textSize="25sp" /><!--set 25sp text size-->
```



Setting textSize in EditText in Java class:

Below is the example code in which we set the text size of a edit text programmatically means in java class.

```
EditText simpleEditText=(EditText)findViewById(R.id.simpleEditText);
simpleEditText.setTextSize(25); //set size of text
```

8. textStyle: textStyle attribute is used to set the text style of a edit text. The possible text styles are bold, italic and normal. If we need to use two or more styles for a edit text then “|” operator is used for that.

Below is the example code with explanation included, in which we set the bold and italic text styles for text.

```
<EditText
    android:id="@+id/simpleEditText"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_centerInParent="true"
    android:text="Email"
    android:textSize="25sp"
    android:textStyle="bold|italic"/><!--set bold and italic text style--&gt;</pre>
```



9. background: background attribute is used to set the background of a edit text. We can set a color or a drawable in the background of a edit text.

Below is the example code with explanation included in which we set the black color for the background, white color for the displayed hint and set 10dp padding from all the side's for edit text.

```
<EditText  
    android:id="@+id/simpleEditText"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:layout_centerInParent="true"  
    android:hint="Enter Your Name Here"  
    android:padding="15dp"  
    android:textColorHint="#fff"  
    android:textStyle="bold|italic"  
    android:background="#000"/><!--set background color black-->
```



Setting Background in EditText In Java class:

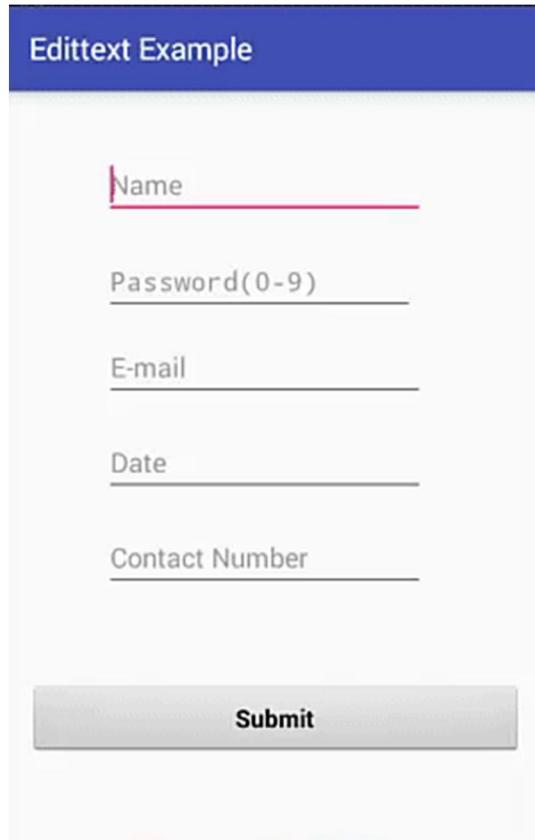
Below is the example code in which we set the background color of a edit text programmatically means in java class.

```
EditText simpleEditText=(EditText)findViewById(R.id.simpleEditText);  
simpleEditText.setBackgroundColor(Color.BLACK);//set black background color
```

10. padding: padding attribute is used to set the padding from left, right, top or bottom. In above example code of background we also set the 10dp padding from all the side's of edit text.

Example I – EditText in Android Studio

Below is the example of edit text in which we get the value from multiple edittexts and on button click event the Toast will show the data defined in Edittext.



Step 1: Create a new project in Android Studio and name it EditTextExample.

Step 2: Now Open res -> layout -> xml (or) activity_main.xml and add following code. In this code we have added multiple edittext and a button with onclick functionality.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
```

```
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.edittextexample.MainActivity">

    <EditText
        android:id="@+id/editText1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true"
        android:layout_marginLeft="50dp"
        android:layout_marginStart="50dp"
        android:layout_marginTop="24dp"
        android:ems="10"
        android:hint="@string/name"
        android:inputType="textPersonName"
        android:selectAllOnFocus="true" />

    <EditText
        android:id="@+id/editText2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/editText1"
        android:layout_alignStart="@+id/editText1"
        android:layout_below="@+id/editText1"
        android:layout_marginTop="19dp"
        android:ems="10"
        android:hint="@string/password_0_9"
        android:inputType="numberPassword" />

    <EditText
        android:id="@+id/editText3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/editText2"
        android:layout_alignStart="@+id/editText2"
        android:layout_below="@+id/editText2"
        android:layout_marginTop="12dp"
        android:ems="10"
        android:hint="@string/e_mail"
        android:inputType="textEmailAddress" />

    <EditText
        android:id="@+id/editText4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/editText3"
        android:layout_alignStart="@+id/editText3"
        android:layout_below="@+id/editText3"
        android:layout_marginTop="18dp"
        android:ems="10"
        android:hint="@string/date"
```

```
        android:inputType="date" />

    <EditText
        android:id="@+id/editText5"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/editText4"
        android:layout_alignStart="@+id/editText4"
        android:layout_below="@+id/editText4"
        android:layout_marginTop="18dp"
        android:ems="10"
        android:hint="@string/contact_number"
        android:inputType="phone" />

    <Button
        android:id="@+id/button"
        style="@android:style/Widget.Button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_below="@+id/editText5"
        android:layout_marginTop="62dp"
        android:text="@string/submit"
        android:textSize="16sp"
        android:textStyle="normal|bold" />
</RelativeLayout>
```

Step 3: Now open app -> java -> package -> MainActivity.java and add the below code.

In this we just fetch the text from the edittext, further with the button click event a toast will show the text fetched before.

```
package com.example.edittextexample;

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

public class MainActivity extends AppCompatActivity {

    Button submit;
    EditText name, password, email, contact, date;

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

```
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
name = (EditText) findViewById(R.id.editText1);
password = (EditText) findViewById(R.id.editText2);
email = (EditText) findViewById(R.id.editText3);
date = (EditText) findViewById(R.id.editText4);
contact = (EditText) findViewById(R.id.editText5);
submit = (Button) findViewById(R.id.button);

submit.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (name.getText().toString().isEmpty() ||
password.getText().toString().isEmpty() || email.getText().toString().isEmpty() || date.getText().toString().isEmpty()
            || contact.getText().toString().isEmpty()) {
            Toast.makeText(getApplicationContext(), "Enter the Data",
Toast.LENGTH_SHORT).show();
        } else {
            Toast.makeText(getApplicationContext(), "Name - " +
name.getText().toString() + "\n" + "Password - " + password.getText().toString()
                + "\n" + "E-Mail - " + email.getText().toString() + "\n" +
"Date - " + date.getText().toString()
                + "\n" + "Contact - " + contact.getText().toString(),
Toast.LENGTH_SHORT).show();
        }
    }
});
```

Output:

Now start the AVD in Emulator and run the App. You will see screen asking you to fill the data in required fields like name, password(numeric), email, date, contact number. Enter data and click on button. You will see the data entered will be displayed as Toast on screen.



Button

In Android, Button represents a push button. A Push buttons can be clicked, or pressed by the user to perform an action. There are different types of buttons used in android such as CompoundButton, ToggleButton, RadioButton.



Button is a subclass of TextView class and compound button is the subclass of Button class. **On a button we can perform different actions or events like click event, pressed event, touch event etc.**

Android buttons are GUI components which are sensible to taps (clicks) by the user. When the user taps/clicks on button in an Android app, the app can respond to the click/tap. These buttons can be divided into two categories: the first is Buttons with text on, and second is buttons with an image on. A button with images on can contain both an image and a text. Android buttons with images on are also called ImageButton.

Button code in XML:

The below code will create Button and write “Abhi Android” text on it.

```
<Button  
    android:id="@+id/simpleButton"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Abhi Android"/>
```



Attributes of Button in Android: Now let's we discuss some important attributes that helps us to configure a Button in your xml file (layout).

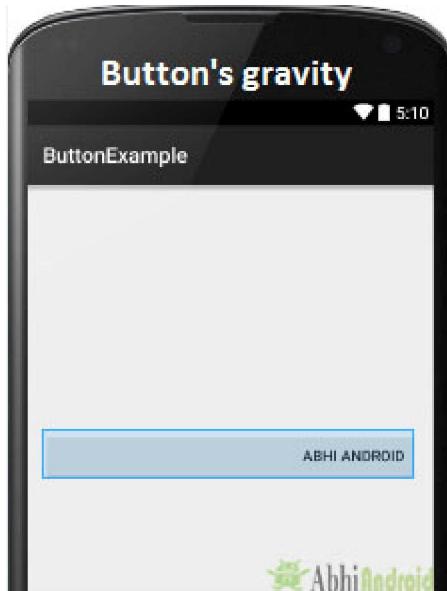
1. id: id is an attribute used to uniquely identify a text Button. Below is the example code in which we set the id of a Button.

```
<Button  
    android:id="@+id/simpleButton"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Abhi Android"/>
```

2. gravity: The gravity attribute is an optional attribute which is used to control the alignment of the text like left, right, center, top, bottom, center_vertical, center_horizontal etc.

Below is the example code with explanation included in which we set the right and center vertical gravity for text of a Button.

```
<Button  
    android:id="@+id/simpleButton"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:text="Abhi Android"  
    android:layout_centerInParent="true"  
    android:gravity="right|center_vertical"/><!-- set the gravity of button-->
```



3. text: text attribute is used to set the text in a Button. We can set the text in xml as well as in the java class.

Below is the example code with explanation included in which we set the text “Learning Android @ AbhiAndroid” in a Button.

```
<Button  
    android:id="@+id/simpleButton"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:layout_centerInParent="true"  
    android:text="Learn Android @ AbhiAndroid"/><!--display text on button-->
```



Setting Text Using Java class:

Below is the example code in which we set the text on Button programmatically means in java class. The output will be same as the above.

```
Button button = (Button) findViewById(R.id.simpleButton);
button.setText("Learn Android @ AbhiAndroid");//set the text on button
```

4.textColor: textColor attribute is used to set the text color of a Button. Color value is in the form of “#argb”, “#rgb”, “#rrggbb”, or “#aarrggb”.

Below is the example code with explanation included in which we set the red color for the displayed text of a Button.

```
<Button
    android:id="@+id/simpleButton"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_centerInParent="true"
    android:text="AbhiAndroid"
    android:textColor="#f00"/><!--red color for the text-->
```



Setting Text Color On Button Inside Java class:

Below is the example code in which we set the text color of a Button programmatically means in java class.

```
Button simpleButton=(Button) findViewById(R.id.simpleButton);
simpleButton.setTextColor(Color.RED); //set the red color for the text
```

5. textSize: textSize attribute is used to set the size of the text on Button. We can set the text size in sp(scale independent pixel) or dp(density pixel).

Below is the example code in which we set the 25sp size for the text of a Button.

```
<Button
    android:id="@+id/simpleButton"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_centerInParent="true"
    android:text="AbhiAndroid"
    android:textSize="25sp" /><!--25sp text size-->
```



Setting textSize In Java class:

Below is the example code in which we set the text size of a Button programmatically means in java class.

```
Button simpleButton=(Button) findViewById(R.id.simpleButton);
```

```
simpleButton.setTextSize(25); //set the text size of button
```

6. textStyle: textStyle attribute is used to set the text style of a Button. The possible text styles are bold, italic and normal. If we need to use two or more styles for a Button then “|” operator is used for that.

Below is the example code with explanation included, in which we set the bold and italic text styles for text of a button.

```
<Button
    android:id="@+id/simpleButton"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_centerInParent="true"
    android:text="AbhiAndroid"
    android:textSize="20sp"
    android:textStyle="bold|italic"/><!--bold and italic text style-->
```



7. background: background attribute is used to set the background of a Button. We can set a color or a drawable in the background of a Button.

Below is the example code in which we set the green color for the background, Black color for the displayed text and set 15dp padding from all the side's for Button.

```
<Button  
    android:id="@+id/simpleButton"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:layout_centerInParent="true"  
    android:text="Download"  
    android:textSize="20sp"  
    android:padding="15dp"  
    android:textStyle="bold|italic"  
    android:background="#147D03" /><!--Background green color-->
```



Setting background in Button In Java class:

Below is the example code in which we set the background color of a Button programmatically means in java class.

```
Button simpleButton=(Button)findViewById(R.id.simpleButton);  
simpleButton.setBackgroundColor(Color.BLACK); //set the black color of button background
```

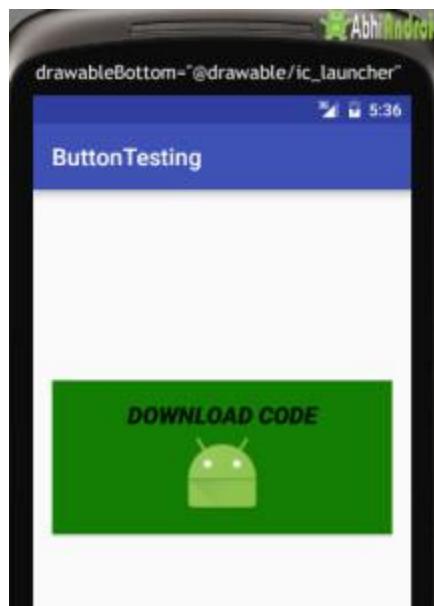
8. padding: padding attribute is used to set the padding from left, right, top or bottom. In above example code of background we also set the 10dp padding from all the side's of button.

9. drawableBottom: drawableBottom is the drawable to be drawn to the below of the text.

Below is the example code in which we set the icon to the below of the text.

Make sure you have image saved in your drawable folder name ic_launcher.

```
<Button  
    android:id="@+id/simpleButton"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:layout_centerInParent="true"  
    android:background="#147D03"  
    android:text="Download Code"  
    android:textSize="20sp"  
    android:padding="15dp"  
    android:textStyle="bold|italic"  
    android:drawableBottom="@drawable/ic_launcher"/><!--image drawable on button-->
```



10. drawableTop, drawableRight And drawableLeft: Just like the above attribute we can draw drawable to the left, right or top of text.

In the Below example we set the icon to the right of the text. In the same way you can do for other two attribute by your own:

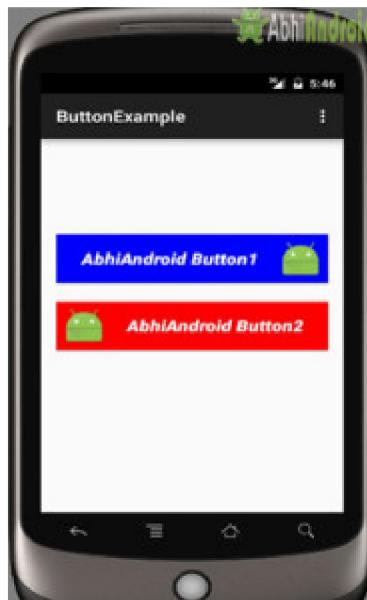
```
<Button  
    android:id="@+id/simpleButton"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:layout_centerInParent="true"  
    android:background="#147D03"  
    android:text="Download Code"  
    android:textSize="20sp"  
    android:padding="15dp"  
    android:textStyle="bold|italic"  
    android:drawableRight="@drawable/ic_launcher"/><!--image drawable on Right side of Text-->
```

on button-->



Button Example In Android Studio:

Below is the example of button in which we display two buttons with different background and whenever a user click on the button the text of the button will be displayed in a toast.



Step 1: Create a new project in Android Studio and name it ButtonExample.

Select File -> New -> New Project and Fill the forms and click "Finish" button.

Step 2: Now open res -> layout -> xml (or) activity_main.xml and add following code. Here we are designing the UI of two button in Relative Layout.

```
<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">

    <Button
        android:id="@+id/simpleButton1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="100dp"
        android:background="#00f"
        android:drawableRight="@drawable/ic_launcher"
        android:hint="AbhiAndroid Button1"
        android:padding="5dp"
        android:textColorHint="#fff"
        android:textSize="20sp"
        android:textStyle="bold|italic" />

    <Button
        android:id="@+id/simpleButton2"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:background="#f00"
        android:drawableLeft="@drawable/ic_launcher"
        android:hint="AbhiAndroid Button2"
        android:padding="5dp"
        android:textColorHint="#fff"
        android:textSize="20sp"
        android:textStyle="bold|italic" />

</RelativeLayout>
```

Step 3: Now Open app -> package -> MainActivity.java and the following code. Here using setOnClickListener() method on button and using Toast we will display which button is clicked by user.

```
package example.abhiandriod.buttonexample;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

    Button simpleButton1, simpleButton2;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        simpleButton1 = (Button) findViewById(R.id.simpleButton1); //get id of button 1
        simpleButton2 = (Button) findViewById(R.id.simpleButton2); //get id of button 2

        simpleButton1.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Toast.makeText(getApplicationContext(), "Simple Button 1",
                        Toast.LENGTH_LONG).show(); //display the text of button1
            }
        });
        simpleButton2.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Toast.makeText(getApplicationContext(), "Simple Button 2",
                        Toast.LENGTH_LONG).show(); //display the text of button2
            }
        });
    }

}
```

Output:

Now start the AVD in Emulator and run the App. You will see two button. Click on any button and you will see the message on screen which button is clicked.



ImageView

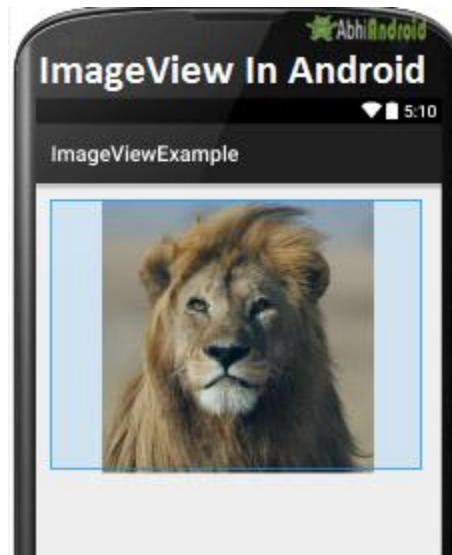
In Android, ImageView class is used to display an image file in application. Image file is easy to use but hard to master in Android, because of the various screen sizes in Android devices. An android is enriched with some of the best UI design widgets that allows us to build good looking and attractive UI based application.

Important Note: ImageView comes with different configuration options to support different scale types. Scale type options are used for scaling the bounds of an image to the bounds of the imageview. Some of them scaleTypes configuration properties are center, center_crop, fit_xy, fitStart etc.

Below is an ImageView code in XML:

Make sure to save lion image in drawable folder

```
<ImageView  
    android:id="@+id/simpleImageView"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:src="@drawable/lion" />
```



Attributes of ImageView:

Now let's we discuss some important attributes that helps us to configure a ImageView in your xml file.

1. id: id is an attribute used to uniquely identify a image view in android. Below is the example code in which we set the id of a image view.

```
<ImageView  
    android:id="@+id/simpleImageView"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
/>
```

2. src: src is an attribute used to set a source file or you can say image in your imageview to make your layout attractive.

Below is the example code in which we set the source of a imageview lion which is saved in drawable folder.

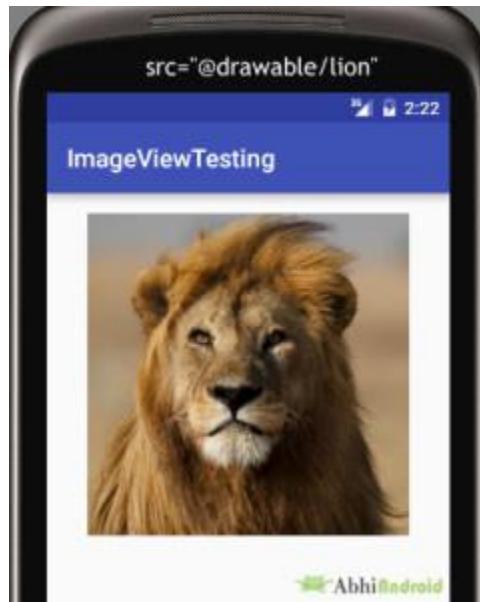
```
<ImageView  
    android:id="@+id/simpleImageView"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"
```

```
android:src="@drawable/lion" /><!--set the source of an image view-->
```

In Java:

We can also set the source image at run time programmatically in java class. For that we use setImageResource() method as shown in below example code.

```
/*Add in Oncreate() function after setContentView()*/
ImageView simpleImageView=(ImageView) findViewById(R.id.simpleImageView);
simpleImageView.setImageResource(R.drawable.lion); //set the source in java class
```



3. background: background attribute is used to set the background of a ImageView. We can set a color or a drawable in the background of a ImageView.

Below is the example code in which we set the black color in the background and an image in the src attribute of image view.

```
/*Add in Oncreate() function after setContentView()*/
ImageView simpleImageView=(ImageView) findViewById(R.id.simpleImageView);
simpleImageView.setBackgroundColor(Color.BLACK); //set black color in background of a image
view in java class
```

4. padding: padding attribute is used to set the padding from left, right, top or bottom of the Imageview.

paddingRight: set the padding from the right side of the image view.

paddingLeft: set the padding from the left side of the image view.

paddingTop: set the padding from the top side of the image view.

paddingBottom: set the padding from the bottom side of the image view.

padding: set the padding from the all side's of the image view.

Below is the example code of padding attribute in which we set the 30dp padding from all the side's of a image view.

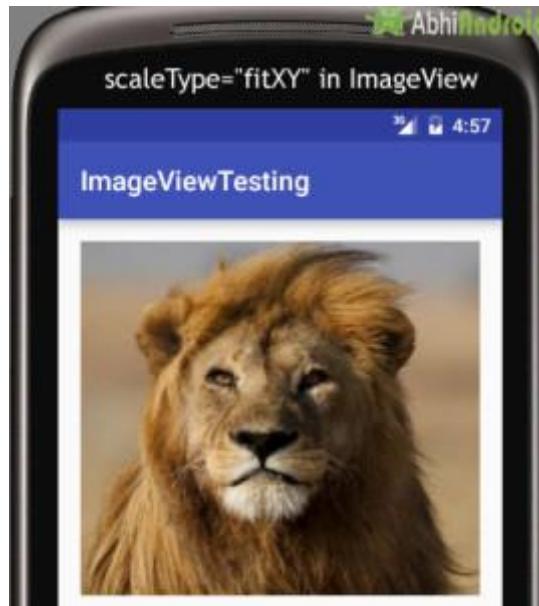
```
<ImageView
    android:id="@+id/simpleImageView"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:background="#000"
    android:src="@drawable/lion"
    android:padding="30dp"/><!--set 30dp padding from all the sides-->
```



5. scaleType: scaleType is an attribute used to control how the image should be re-sized or moved to match the size of this image view. **The value for scale type attribute can be fit_xy, center_crop, fitStart etc.**

Below is the example code of scale type in which we set the scale type of image view to fit_xy.

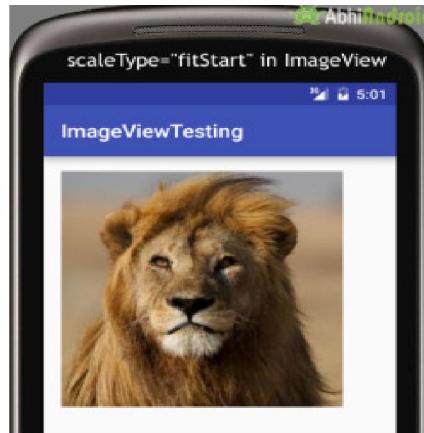
```
<ImageView  
    android:id="@+id/simpleImageView"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:src="@drawable/lion"  
    android:scaleType="fitXY"/><!--set scale type fit xy-->
```



Let's we take an another example of scale type to understand the actual working of scale type in a image view.

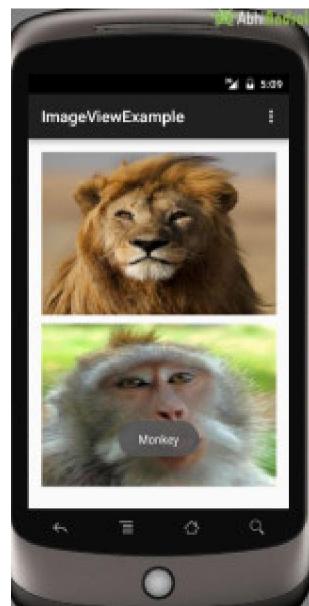
In below example code we set the value for scale type “fitStart” which is used to fit the image in the start of the image view as shown below:

```
<ImageView  
    android:id="@+id/simpleImageView"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:src="@drawable/lion"  
    android:scaleType="fitStart"/><!--set scale type fit start of image view-->
```



Example of ImageView:

Below is the example of imageview in which we display two animal images of Lion and Monkey. And whenever user click on an image Animal name is displayed as toast on screen. Below is the final output and code:



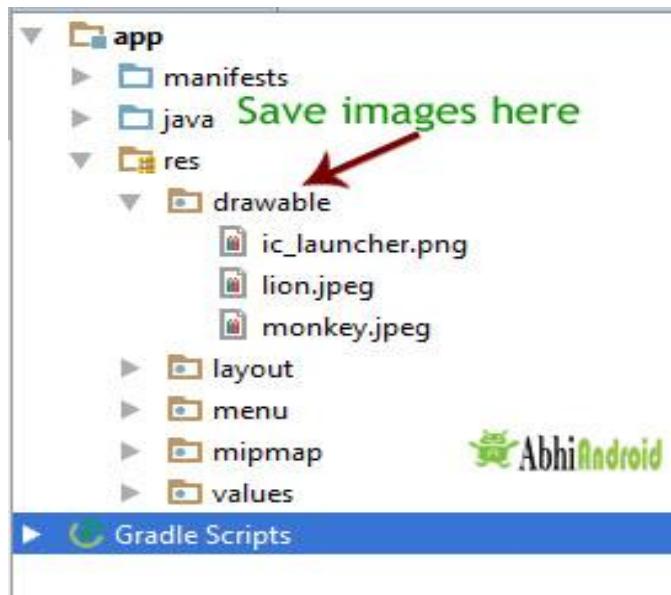
Step 1:

Create a new project and name it ImageViewExample.

In this step we create a new project in android studio by filling all the necessary details of the app like app name, package name, api versions etc.

Select File -> New -> New Project and Fill the forms and click "Finish" button.

Step 2: Download two images lion and monkey from the web. Now save those images in the drawable folder of your project.



Step 3: Now open res -> layout -> activity_main.xml (or) main.xml and add following code:

In this step we add the code for displaying an image view on the screen in a relative layout. Here make sure you have already saved two images name lion and monkey in your drawable folder.

```
<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">

    <ImageView
        android:id="@+id/simpleImageViewLion"
        android:layout_width="fill_parent"
        android:layout_height="200dp"
        android:scaleType="fitXY" />
```

```
        android:src="@drawable/lion" />

    <ImageView
        android:id="@+id/simpleImageViewMonkey"
        android:layout_width="fill_parent"
        android:layout_height="200dp"
        android:layout_below="@+id/simpleImageViewLion"
        android:layout_marginTop="10dp"
        android:scaleType="fitXY"
        android:src="@drawable/monkey" />

</RelativeLayout>
```

Step 4: Now open app -> java -> package -> MainActivity.java and add the following code:

In this step we add the code to initiate the image view's and then perform click event on them.

```
package example.abhiandriod.imageviewexample;

import android.graphics.Color;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.ImageView;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ImageView simpleImageViewLion = (ImageView)
findViewByViewId(R.id.simpleImageViewLion); //get the id of first image view
        ImageView simpleImageViewMonkey = (ImageView)
findViewByViewId(R.id.simpleImageViewMonkey); //get the id of second image view
        simpleImageViewLion.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Toast.makeText(getApplicationContext(), "Lion",
Toast.LENGTH_LONG).show(); //display the text on image click event
            }
        });
        simpleImageViewMonkey.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Toast.makeText(getApplicationContext(), "Monkey",
Toast.LENGTH_LONG).show(); //display the text on image click event
            }
        });
    }
}
```

```
    });
}

}
```

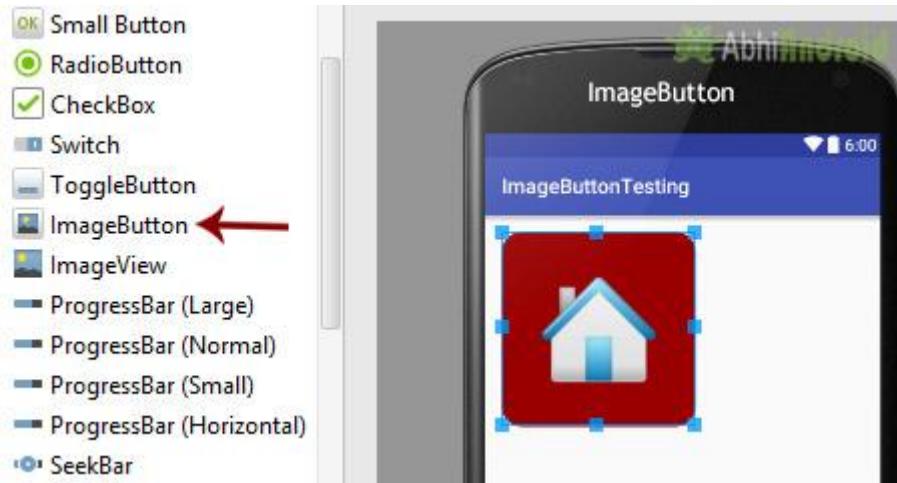
Output:

Now start AVD in Emulator and run the App. You will see the images of Lion and Monkey displayed on screen. Click on any Animal image and his name will appear on Screen. We clicked on Lion.



ImageButton

In Android, ImageButton is used to display a normal button with a custom image in a button. In simple words we can say, ImageButton is a button with an image that can be pressed or clicked by the users. By default it looks like a normal button with the standard button background that changes the color during different button states.



An image on the surface of a button is defined within a xml (i.e. layout) by using src attribute or within java class by using setImageResource() method. We can also set an image or custom drawable in the background of the image button.

Important Note: Standard button background image is displayed in the background of button whenever you create an image button. To remove that image, you can define your own

background image in xml by using background attribute or in java class by using setBackground() method.

Below is the code and image which shows how custom imagebutton looks in Android:

Important Note: ImageButton has all the properties of a normal button so you can easily perform any event like click or any other event which you can perform on a normal button.

ImageButton code in XML:

```
<!--Make Sure To Add Image Name home in Drawable Folder-->
<ImageButton
    android:id="@+id/simpleImageButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/home" />
```



Attributes of ImageButton:

Now let's we discuss some important attributes that helps us to configure a image button in your xml file (layout).

1. id: id is an attribute used to uniquely identify a image button. Below is the example code in which we set the id of a image button.

```
<ImageButton
    android:id="@+id/simpleImageButton"
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
```

2. src: src is an attribute used to set a source file of image or you can say image in your image button to make your layout look attractive.

Below is the example code in which we set the source of an image button. Make sure you have saved an image in drawable folder name home before using below code.

```
<ImageButton
    android:id="@+id/simpleImageButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/home"/> <!--src(source)file from drawable folder which display an
imagebutton-->
```

Setting Image Source In ImageButton Using Java class:

We can also set the source image at run time programmatically in java class. For that we use setImageResource() function as shown in below example code.

```
/*Add in Oncreate() funtion after setContentView()*/
ImageButton simpleImageButton = (ImageButton)findViewById(R.id.simpleImageButton);
simpleImageButton.setImageResource(R.drawable.home); //set the image programmatically
```

3. background: background attribute is used to set the background of an image button. We can set a color or a drawable in the background of a Button.

Below is the example code in which we set the black color for the background and an home image as the source of the image button.

```
<ImageButton
    android:id="@+id/simpleImageButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/home"
    android:background="#000"/> <!-- black background color for image button-->
```



Setting Background In ImageButton Using Java class:

Below is the example code in which we set the black background color of a image button programmatically means in java class.

```
/*Add in Oncreate() funtion after setContentView()*/
ImageButton simpleImageButton = (ImageButton) findViewById(R.id.simpleImageButton);
simpleImageButton.setBackgroundColor(Color.BLACK); //set black background color for image
button
```

4. padding: padding attribute is used to set the padding from left, right, top or bottom of the ImageButton.

paddingRight : set the padding from the right side of the image button.

paddingLeft : set the padding from the left side of the image button.

paddingTop : set the padding from the top side of the image button.

paddingBottom : set the padding from the bottom side of the image button.

padding : set the padding from the all side's of the image button.

Below is the example code of padding attribute in which we set the 20dp padding from all the side's of a image button.

```
<ImageButton  
    android:id="@+id/simpleImageButton"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:background="#000"  
    android:src="@drawable/home"  
    android:padding="30dp"/><!-- set 30dp padding from all the sides of the view-->
```



Example of ImageButton In Android Studio:

In the below example of ImageButton we display two custom image buttons with source and background. One is simple image button with simple background and other one is a round corner image button and whenever you click on an button, the name of the button will be displayed in a toast. Below is the code and final output:



Step 1:

Create a new project and name it ImageButtonExample

In this step we create a new project in android studio by filling all the necessary details of the app like app name, package name, api versions etc.

Select File -> New -> New Project and Fill the forms and click "Finish" button.

Step 2:

Right click on drawable -> New -> Drawable resource file and create new xml file name custom_imagebutton.xml and add following code

In this Step we create drawable xml in which we used solid and corner properties, solid is used to set the background color for the image button and corner is used to set the radius for button corners.

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android">
    <solid android:color="#900" /><!-- background color for imagebutton-->
    <corners android:radius="20dp" /><!-- round corners for imagebutton-->
</shape>
```

Step 3:

Open app -> layout -> activity_main.xml (or) main.xml and add following code:

In this step, we open an xml file and add the code which display two custom image buttons by using src, background, gravity and other attributes in Relative Layout.

```
<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">

    <!--Make sure to save two images home and youtube in drawable folder-->

    <ImageButton
        android:id="@+id/simpleImageButtonHome"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:background="@drawable/custom_image_button"
        android:padding="20dp"
        android:src="@drawable/home" />

    <ImageButton
        android:id="@+id/simpleImageButtonYouTube"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/simpleImageButtonHome"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="20dp"
        android:background="#005"
        android:padding="20dp"
        android:src="@drawable/youtube" />

</RelativeLayout>
```

Step 4: Open app -> package -> MainActivity.java

In this step, we add the code to initiate the image button's and then perform click event on them and display the text for selected item using a toast.

```
package example.abhiandriod.imagebuttonexample;

import android.app.Activity;
import android.graphics.Color;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
```

```
import android.widget.ImageButton;
import android.widget.Toast;

public class MainActivity extends Activity {

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

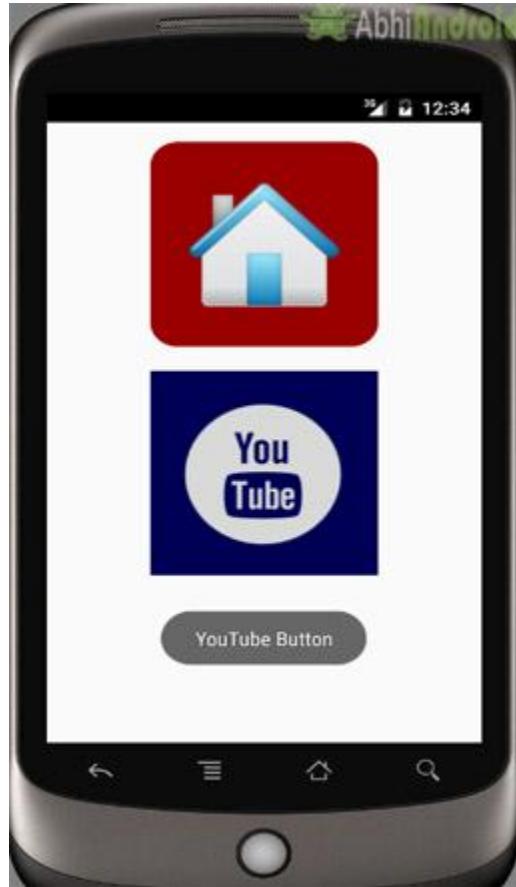
    // initiate view's
        ImageButton simpleImageButtonHome =
        (ImageButton)findViewById(R.id.simpleImageButtonHome);
        ImageButton simpleImageButtonYouTube =
        (ImageButton)findViewById(R.id.simpleImageButtonYouTube);

    // perform click event on button's
        simpleImageButtonHome.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Toast.makeText(getApplicationContext(),"Home
Button",Toast.LENGTH_LONG).show();// display the toast on home button click
            }
        );
        simpleImageButtonYouTube.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Toast.makeText(getApplicationContext(),"YouTube
Button",Toast.LENGTH_LONG).show();// display the toast on you tube button click
            }
        );
    }

}
```

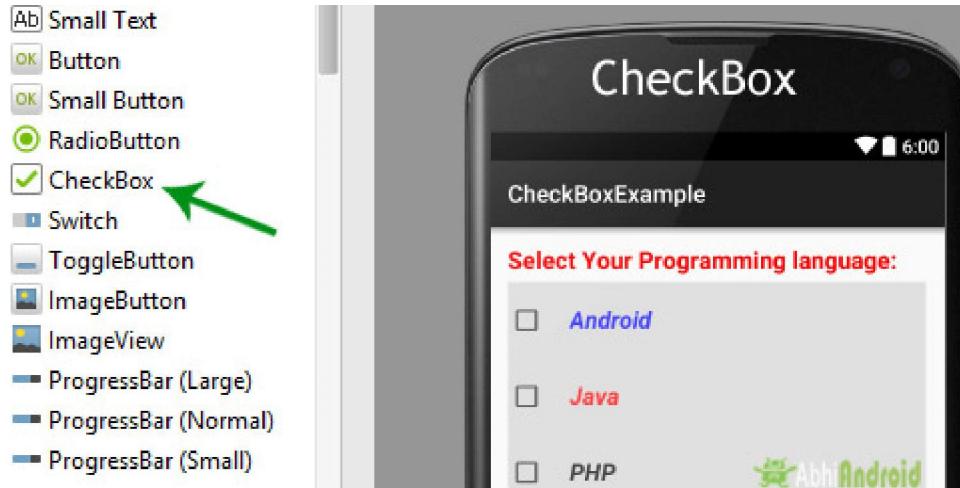
Output:

Now start the AVD in Emulator and run the App. You will see two ImageButton out of which top one is round corner. Click on any image and its name will be displayed on screen.



CheckBox

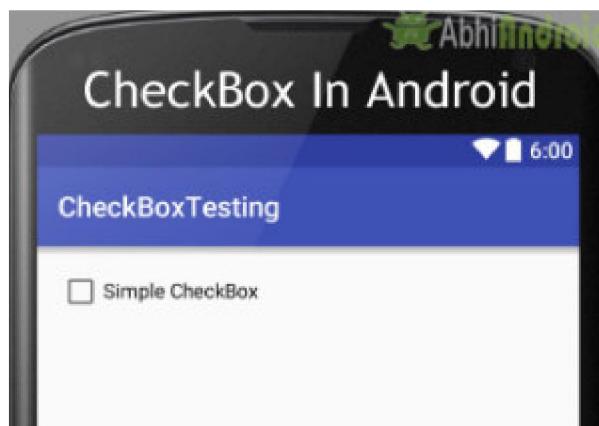
In Android, CheckBox is a type of two state button either unchecked or checked in Android. Or you can say it is a type of on/off switch that can be toggled by the users. You should use checkbox when presenting a group of selectable options to users that are not mutually exclusive. CompoundButton is the parent class of CheckBox class.



In android there is a lot of usage of check box. For example, to take survey in Android app we can list few options and allow user to choose using CheckBox. The user will simply checked these checkboxes rather than type their own option in EditText. Another very common use of CheckBox is as remember me option in Login form.

CheckBox code in XML:

```
<CheckBox  
    android:id="@+id/simpleCheckBox"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Simple CheckBox"/>
```



Important Note: You can check the current state of a check box programmatically by using isChecked() method. This method returns a Boolean value either true or false, if a check box is checked then it returns true otherwise it returns false. Below is an example code in which we checked the current state of a check box.

```
//initiate a check box
CheckBox simpleCheckBox = (CheckBox) findViewById(R.id.simpleCheckBox);

//check current state of a check box (true or false)
Boolean checkBoxState = simpleCheckBox.isChecked();
```

Attributes of CheckBox:

Now let's we discuss important attributes that helps us to configure a check box in XML file (layout).

1. id: id is an attribute used to uniquely identify a check box. Below we set the id of a image button.

```
<CheckBox
    android:id="@+id/simpleCheckBox"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Simple CheckBox"/>
```

2. checked: checked is an attribute of check box used to set the current state of a check box. The value should be true or false where true shows the checked state and false shows unchecked state of a check box. The default value of checked attribute is false. We can also set the current state programmatically.

Below is an example code in which we set true value for checked attribute that sets the current state to checked.

```
<CheckBox
    android:id="@+id/simpleCheckBox"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Simple CheckBox"
    android:checked="true"/> <!--set the current state of the check box-->
```



Setting Current State Of CheckBox In Java Class:

Below we set the current state of CheckBox in java class.

```
/*Add in Oncreate() funtion after setContentView()*/
// initiate a check box
CheckBox simpleCheckBox = (CheckBox) findViewById(R.id.simpleCheckBox);

// set the current state of a check box
simpleCheckBox.setChecked(true);
```

3. gravity: The gravity attribute is an optional attribute which is used to control the alignment of the text in CheckBox like left, right, center, top, bottom, center_vertical, center_horizontal etc.

Below we set the right and center_vertical gravity for the text of a check box.

```
<CheckBox
    android:id="@+id/simpleCheckBox"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Simple CheckBox"
    android:checked="true"
    android:gravity="right|center_vertical"/> <!-- gravity of the check box-->
```



4. text: text attribute is used to set the text in a check box. We can set the text in xml as well as in the java class.

Below is the example code with explanation included in which we set the text “Text Attribute Of Check Box” for a check box.

```
<CheckBox  
    android:id="@+id/simpleCheckBox"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:checked="true"  
    android:text="Text Attribute Of Check Box"/> <!--displayed text of the check box-->
```



Setting text in CheckBox In Java class:

Below is the example code in which we set the text of a check box programmatically means in java class.

```
/*Add in Oncreate() function after setContentView()*/
// initiate check box
CheckBox simpleCheckBox = (CheckBox) findViewById(R.id.simpleCheckBox);

// displayed text of the check box
simpleCheckBox.setText("Text Attribute Of Check Box");
```

5. textColor: textColor attribute is used to set the text color of a check box. Color value is in form of “#argb”, “#rgb”, “#rrggbb”, or “#aarrggbb”.

Below we set the red color for the displayed text of a check box.

```
<CheckBox
    android:id="@+id/simpleCheckBox"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Text is Red Color"
    android:textColor="#f00"
    android:checked="true"/> <!-- red color for the text of check box-->
```



Setting textColor in CheckBox In Java class:

Below we set the text color of a check box programmatically.

```
/*Add in Oncreate() function after setContentView()*/
//initiate the checkbox
CheckBox simpleCheckBox = (CheckBox) findViewById(R.id.simpleCheckBox);
```

```
//red color for displayed text  
simpleCheckBox.setTextColor(Color.RED);
```

6. textSize: textSize attribute is used to set the size of text of a check box. We can set the text size in sp(scale independent pixel) or dp(density pixel).

Below is the example code in which we set the 20sp size for the text of a check box.

```
<CheckBox  
    android:id="@+id/simpleCheckBox"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Text size Attribute Of Check Box"  
    android:textColor="#00f"  
    android:checked="false"  
    android:textSize="20sp"/><!--set Text Size of text in CheckBox-->
```



Setting Text Size in CheckBox In Java class:

Below we set the text size of a check box in java class.

```
/*Add in Oncreate() function after setContentView()*/  
CheckBox simpleCheckBox = (CheckBox) findViewById(R.id.simpleCheckBox);  
//set 20sp displayed text size  
simpleCheckBox.setTextSize(20);
```

7. textStyle: textStyle attribute is used to set the text style of the text of a check box. The possible text styles are bold, italic and normal. If we need to use two or more styles for a text view then “|” operator is used for that.

Below we set the bold and italic text styles for text of a check box.

```
<CheckBox
    android:id="@+id/simpleCheckBox"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Text Style Attribute Of Check Box"
    android:textColor="#44f"
    android:textSize="20sp"
    android:checked="false"
    android:textStyle="bold|italic"/>
```



8. background: background attribute is used to set the background of a check box. We can set a color or a drawable in the background of a check box.

Below we set the black color for the background, white color for the displayed text of a check box.

```
<CheckBox
    android:id="@+id/simpleCheckBox"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Text size Attribute Of Check Box"
    android:textColor="#fff"
    android:textSize="20sp"
    android:textStyle="bold|italic"
    android:checked="true"
    android:background="#000" /><!-- black background for the background of check box-->
```



Setting Background in CheckBox In Java class:

Below is the example code in which we set the background color of a check box programmatically means in java class.

```
/*Add in Oncreate() function after setContentView()*/
CheckBox simpleCheckBox = (CheckBox) findViewById(R.id.simpleCheckBox);
// set background in CheckBox
simpleCheckBox.setBackgroundColor(Color.BLACK);
```

9. padding: padding attribute is used to set the padding from left, right, top or bottom.

paddingRight :set the padding from the right side of the check box.

paddingLeft :set the padding from the left side of the check box.

paddingTop :set the padding from the top side of the check box.

paddingBottom :set the padding from the bottom side of the check box.

Padding :set the padding from the all side's of the check box.

Below is the example code of padding where we set the 30dp padding from all the side's of the check box.

```
<CheckBox  
    android:id="@+id/simpleCheckBox"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Padding Attribute Of Check Box"  
    android:textColor="#44f"  
    android:textSize="20sp"  
    android:textStyle="bold|italic"  
    android:checked="false"  
    android:padding="30dp"/> <!--30dp padding from all side's-->
```



Example of CheckBox In Android Studio:

Below is the example of CheckBox in Android, in which we display five check boxes using background and other attributes we discuss earlier in this post. Every check box represents a different subject name and whenever you click on a check box then text of that checked check box is displayed in a toast. Below is the final output, code and step by step explanation of the example:



Step 1:

Create a new project and name it CheckBoxExample

In this step we create a new project in android studio by filling all the necessary details of the app like app name, package name, api versions etc.

Select File -> New -> New Project and Fill the forms and click "Finish" button.

Step 2:

Now Open res -> layout -> activity_main.xml (or) main.xml and add the following code:

In this step we open an xml file and add the code for displaying five one TextView and check boxes.

```
<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:layout_width="wrap_content"  
        android:layout_height="wrap_content"
```

```
        android:text="Select Your Programming language: "
        android:textColor="#f00"
        android:textSize="20sp"
        android:textStyle="bold" />

    <LinearLayout
        android:id="@+id/linearLayout"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="30dp"
        android:background="#e0e0e0"
        android:orientation="vertical">

        <CheckBox
            android:id="@+id/androidCheckBox"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_centerHorizontal="true"
            android:checked="false"
            android:padding="20dp"
            android:text="@string/android"
            android:textColor="#44f"
            android:textSize="20sp"
            android:textStyle="bold|italic" />

        <CheckBox
            android:id="@+id/javaCheckBox"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_centerHorizontal="true"
            android:checked="false"
            android:padding="20dp"
            android:text="@string/java"
            android:textColor="#f44"
            android:textSize="20sp"
            android:textStyle="bold|italic" />

        <CheckBox
            android:id="@+id/phpCheckBox"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_centerHorizontal="true"
            android:checked="false"
            android:padding="20dp"
            android:text="@string/php"
            android:textColor="#444"
            android:textSize="20sp"
            android:textStyle="bold|italic" />

        <CheckBox
            android:id="@+id/pythonCheckBox"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_centerHorizontal="true"
            android:checked="false"
            android:padding="20dp"
            android:text="@string/python"
```

```
        android:textColor="#888"
        android:textSize="20sp"
        android:textStyle="bold|italic" />

    <CheckBox
        android:id="@+id/unityCheckBox"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:checked="false"
        android:padding="20dp"
        android:text="@string/unity"
        android:textColor="#101010"
        android:textSize="20sp"
        android:textStyle="bold|italic" />
    </LinearLayout>
</RelativeLayout>
```

Step 3:

Now Open app -> java-> package -> MainActivity.java

In this step we add the code to initiate the check boxes we created. And then we perform click event on button and display the text for selected check boxes using a toast.

```
package example.abhiandriod.checkboxexample;

import android.graphics.Color;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    CheckBox android, java, python, php, unity3D;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // initiate views
        android = (CheckBox) findViewById(R.id.androidCheckBox);
        android.setOnClickListener(this);
        java = (CheckBox) findViewById(R.id.javaCheckBox);
        java.setOnClickListener(this);
        python = (CheckBox) findViewById(R.id.pythonCheckBox);
        python.setOnClickListener(this);
        php = (CheckBox) findViewById(R.id.phpCheckBox);
        php.setOnClickListener(this);
        unity3D = (CheckBox) findViewById(R.id.unityCheckBox);
```

```

        unity3D.setOnClickListener(this);
    }

    @Override
    public void onClick(View view) {

        switch (view.getId()) {
            case R.id.androidCheckBox:
                if (android.isChecked())
                    Toast.makeText(getApplicationContext(), "Android",
Toast.LENGTH_LONG).show();
                break;
            case R.id.javaCheckBox:
                if (java.isChecked())
                    Toast.makeText(getApplicationContext(), "Java",
Toast.LENGTH_LONG).show();
                break;
            case R.id.phpCheckBox:
                if (php.isChecked())
                    Toast.makeText(getApplicationContext(), "PHP",
Toast.LENGTH_LONG).show();
                break;
            case R.id.pythonCheckBox:
                if (python.isChecked())
                    Toast.makeText(getApplicationContext(), "Python",
Toast.LENGTH_LONG).show();
                break;
            case R.id.unityCheckBox:
                if (unity3D.isChecked())
                    Toast.makeText(getApplicationContext(), "Unity 3D",
Toast.LENGTH_LONG).show();
                break;
        }
    }
}

```

Step 5: Open res ->values -> strings.xml

In this step we show string file which is used to store string data of an app.

```

<resources>
    <string name="app_name">CheckBoxExample</string>
    <string name="hello_world">Hello world!</string>
    <string name="action_settings">Settings</string>
    <string name="android">Android</string>
    <string name="java">Java</string>
    <string name="php">PHP</string>
    <string name="python">Python</string>
    <string name="unity">Unity 3D</string>
</resources>

```

Output:

Now start the AVD in Emulator and run the App. You will see 5 checkbox option asking you to choose your programming language. Select and the text of that particular CheckBox will appear on Screen.



Switch

In Android, Switch is a two-state toggle switch widget that can select between two options. It is used to display checked and unchecked state of a button providing slider control to user. Switch is a subclass of CompoundButton. It is basically an off/on button which indicate the current state of Switch. It is commonly used in selecting on/off in Sound, Bluetooth, WiFi etc.



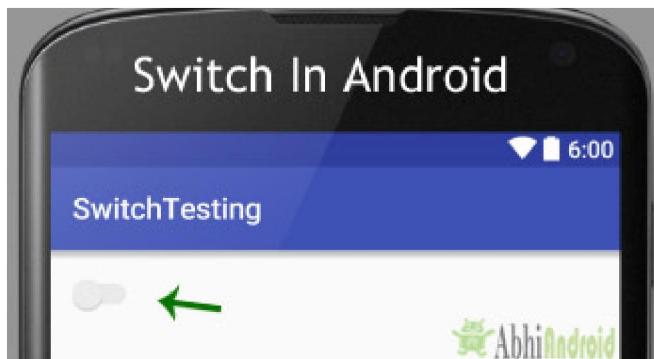
As compared to ToggleButton Switch provides user slider control. The user can simply tap on a switch to change its current state.

Switch allow the users to change the setting between two states like turn on/off wifi, Bluetooth etc from your phone's setting menu. It was introduced after Android 4.0 version (API level 14).

Important Note: Android Switch and ToggleButton both are the subclasses of CompoundButton class.

Switch code in XML:

```
<Switch
    android:id="@+id/simpleSwitch"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
```



Important Note: We can check the current state of a Switch programmatically by using isChecked() method. This method returns a Boolean value means true or false. If a Switch is checked then it returns true otherwise it returns false.

Below is an example code in which we checked the current state of a Switch.

```
// initiate a Switch
Switch simpleSwitch = (Switch) findViewById(R.id.simpleSwitch);

// check current state of a Switch (true or false).
Boolean switchState = simpleSwitch.isChecked();
```

Attributes of Switch:

Now let's we discuss important Switch attributes that helps us to configure a Switch in XML file(layout).

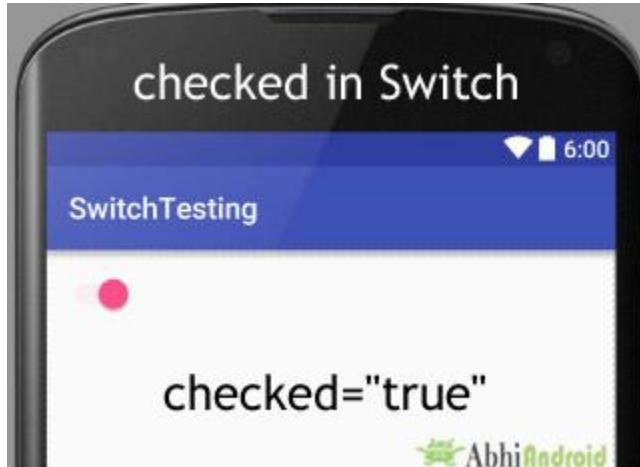
1. id: id is an attribute used to uniquely identify a Switch.

```
<Switch
    android:id="@+id/simpleSwitch"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
```

2. checked: checked attribute of Switch is used to set the current state of a Switch. The value can be either true or false where true shows the checked state and false shows unchecked state of a Switch. The default value of checked attribute is false. We can also set the current state programmatically.

Below we set "true" value for checked attribute that sets the current state to checked.

```
<Switch
    android:id="@+id/simpleSwitch"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:checked="true"/> <!-- set the current state of the Switch-->
```



Setting Check Attribute In Switch Using Java Class:

Below we set the current state of Switch in java class.

```
/*Add in Oncreate() function after setContentView()*/
// initiate a Switch
Switch simpleSwitch = (Switch) findViewById(R.id.simpleSwitch);

//set the current state of a Switch
simpleSwitch.setChecked(true);
```

3. text: text attribute is used to set the text in a Switch. We can set the text in xml as well as in the java class.

Below we set the text “Sound” for the Switch.

```
<Switch
    android:id="@+id/simpleSwitch"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:text="Sound" /> <!--displayed text of switch-->
```



Setting Text In Switch Using Java class:

In the below code we set text of Switch via java class.

```
/*Add in Oncreate() function after setContentView()*/
// initiate Switch
Switch simpleSwitch = (Switch) findViewById(R.id.simpleSwitch);

//displayed text of the Switch
simpleSwitch.setText("switch");
```

4. gravity: The gravity attribute is an optional attribute which is used to control the alignment of the text in Switch. We can set text left, right, center, top, bottom, center_vertical, center_horizontal etc in Switch.

In the below code we set the left gravity for the text in Switch.

```
<Switch
    android:id="@+id/simpleSwitch"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Sound"
    android:checked="true"
    android:gravity="left"/><!--gravity of the Switch-->
```



5. textOn And testOff: textOn attribute is used to set the text when Switch is in checked state (i.e. on state). We can set the textOn in XML as well as in the java class.

In the below example we set the textOn as “Yes” and textOff as “No” for a Switch

```
<Switch
    android:id="@+id/simpleSwitch"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:checked="true"
    android:text="Notification"
    android:textOff="No"
    android:textOn="Yes"/><!-- text to be displayed whenever current state is checked-->
```



Setting textOn And testOff of Switch In Java class:

Below we set the textOn and textOff of a Switch in java class

```
/*Add in Oncreate() function after setContentView()*/
Switch simpleSwitch = (Switch) findViewById(R.id.simpleSwitch); // initiate Switch

simpleSwitch.setTextOn("On"); // displayed text of the Switch whenever it is in checked or
```

```
on state
simpleSwitch.setTextOff("Off"); // displayed text of the Switch whenever it is in unchecked
i.e. off state
```

6. textColor: textColor attribute is used to set the text color of a Switch. Color value is in the form of “#argb”, “#rgb”, “#rrggb”, or “#aarrggbb”.

Below is the example code with explanation included in which we set the red color for the displayed text of a Switch.

```
<Switch
    android:id="@+id/simpleSwitch"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:checked="true"
    android:text="switch"
    android:layout_centerHorizontal="true"
    android:textOn="On"
    android:textOff="Off"
    android:gravity="center"
    android:textColor="#008F36"/><!-- Green color for displayed text-->
```



Setting textColor Of Switch text In Java class:

Below is the example code in which we set the text color of a Switch programmatically.

```
/*Add in Oncreate() funtion after setContentView()*/
Switch simpleSwitch = (Switch) findViewById(R.id.simpleSwitch); // initiate Switch
```

```
simpleSwitch.setTextColor(Color.GREEN); //red color for displayed text of Switch
```

7. textSize: textSize attribute is used to set the size of the text of a Switch. We can set the text size in sp(scale independent pixel) or dp(density pixel).

Below is the example code in which we set the 25sp size for the text of a Switch.

```
<Switch  
    android:id="@+id/simpleSwitch"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:checked="true"  
    android:text="switch"  
    android:layout_centerHorizontal="true"  
    android:textOn="On"  
    android:textOff="Off"  
    android:textColor="#f00"  
    android:gravity="center"  
    android:textSize="25sp"/> <!--25sp displayed text size-->
```



Setting textSize Of Switch Text In Java class:

Below is the example code in which we set the text size of a Switch programmatically.

```
Switch simpleSwitch = (Switch) findViewById(R.id.simpleSwitch); // initiate Switch  
simpleSwitch.setTextSize(25); // set 25sp displayed text size of Switch
```

9. textStyle: textStyle attribute is used to set the text style of the text of a Switch. The possible text styles are bold, italic and normal. If we need to use two or more styles for a text view then use “|” operator.

Below we set the bold and italic text styles for text of a Switch.

```
<Switch
    android:id="@+id/simpleSwitch"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:checked="false"
    android:text="switch"
    android:layout_centerHorizontal="true"
    android:textOn="On"
    android:textOff="Off"
    android:textColor="#f00"
    android:gravity="center"
    android:textSize="25sp"
    android:textStyle="bold|italic"/><!--bold and italic text style for displayed text-->
```



10. background: background attribute is used to set the background of a Switch. We can set a color or a drawable in the background of a Switch.

Below we set the black color for the background and red color for the displayed text of a Switch.

```
<Switch
    android:id="@+id/simpleSwitch"
    android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
    android:checked="true"
    android:text="switch"
    android:layout_centerHorizontal="true"
    android:textOn="On"
    android:textOff="Off"
    android:textColor="#f00"
    android:padding="20dp"
    android:gravity="center"
    android:textSize="25sp"
    android:background="#000"/><!-- black background color--&gt;</pre>
```



Setting Background In Java class:

Below we set the background color of a Switch programmatically.

```
Switch simpleSwitch = (Switch) findViewById(R.id.simpleSwitch);
simpleSwitch.setBackgroundColor(Color.BLACK);
```

11. padding: padding attribute is used to set the padding from left, right, top or bottom in Switch.

paddingRight: set the padding from the right side of the Switch.

paddingLeft: set the padding from the left side of the Switch.

paddingTop: set the padding from the top side of the Switch.

paddingBottom: set the padding from the bottom side of the Switch.

Padding: set the padding from the all side's of the Switch.

Below we set the 20dp padding from all the side's of the Switch.

```
<Switch  
    android:id="@+id/simpleSwitch"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:checked="true"  
    android:text="switch"  
    android:layout_centerHorizontal="true"  
    android:textOn="On"  
    android:textOff="Off"  
    android:textColor="#f00"  
    android:gravity="center"  
    android:textSize="25sp"  
    android:padding="40dp"/><!-- 20dp padding from all the side's-->
```



12. drawableBottom, drawableTop, drawableRight And drawableLeft: These attribute draw the drawable below, top, right and left of the text of Switch.

Below we set the icon to the below of the text of a Switch. You can try other three by yourself.

Before you try this make sure you have icon image in drawable folder

```
<Switch  
    android:id="@+id/simpleSwitch"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:checked="true"  
    android:text="switch"  
    android:layout_centerHorizontal="true"  
    android:textOn="On"  
    android:textOff="Off"  
    android:textColor="#f00"  
    android:gravity="center"  
    android:textSize="25sp"  
    android:drawableBottom="@drawable/ic_launcher"/><!--drawable icon to be displayed in the  
bottom of text-->
```



Example of Switch In Android Studio:

Below is the example of Switch in Android Studio. In this example, we display two Switches and one “submit” button using background & other attributes as discussed earlier in this post. Whenever you click on submit button the current state for both the Switch’s is displayed in a Toast. Below is the final output, code and example explained step by step:



Step 1:

Create a new project and name it SwitchExample

In this step we create a new project in android studio by filling all the necessary details of the app like app name, package name, api versions etc.

Select File -> New -> New Project Fill the forms and click "Finish" button

Step 2:

Open res -> layout -> activity_main.xml (or) main.xml and add following code:

In this step we open an xml file and add the code for displaying two Switch and one normal button.

```
<LinearLayout 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:orientation="vertical"
    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">

    <Switch
        android:id="@+id/simpleSwitch1"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:checked="false"
        android:text="switch 1"
        android:textOff="Off"
        android:textOn="On" />

    <Switch
        android:id="@+id/simpleSwitch2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginTop="20dp"
        android:checked="true"
        android:text="switch 2"
        android:textOff="Off"
        android:textOn="On" />

    <Button
        android:id="@+id/submitButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginTop="50dp"
        android:background="#009284"
        android:padding="10dp"
        android:text="Submit"
        android:textColor="#fff"
        android:textStyle="bold" />
</LinearLayout>
```

Step 3: Open app -> java -> package -> MainActivity.java

In this step we open MainActivity where we add the code to initiate the two Switch's and one normal button. And then we perform click event on button and display the text of current state of Switch by using a toast.

```
package example.abhiandriod.switchexample;

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

public class MainActivity extends AppCompatActivity {

    Switch simpleSwitch1, simpleSwitch2;
    Button submit;

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

```
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
// initiate view's
simpleSwitch1 = (Switch) findViewById(R.id.simpleSwitch1);
simpleSwitch2 = (Switch) findViewById(R.id.simpleSwitch2);
submit = (Button) findViewById(R.id.submitButton);
submit.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        String statusSwitch1, statusSwitch2;
        if (simpleSwitch1.isChecked())
            statusSwitch1 = simpleSwitch1.getTextOn().toString();
        else
            statusSwitch1 = simpleSwitch1.getTextOff().toString();
        if (simpleSwitch2.isChecked())
            statusSwitch2 = simpleSwitch2.getTextOn().toString();
        else
            statusSwitch2 = simpleSwitch2.getTextOff().toString();
        Toast.makeText(getApplicationContext(), "Switch1 :" + statusSwitch1 + "\n" +
"Switch2 :" + statusSwitch2, Toast.LENGTH_LONG).show(); // display the current state for
switch's
    }
});
}
```

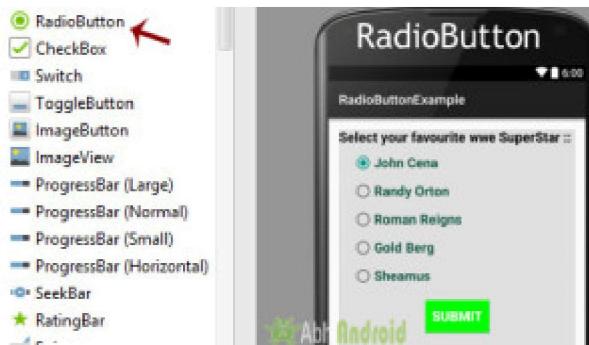
Output:

Now start the AVD in Emulator and run the App. You will see two switch and one Button. Now change the state of the switch (i.e. either checked or unchecked) and click on the Button. You will see the state displayed on screen as Toast.



RadioButton & RadioGroup

In Android, RadioButton are mainly used together in a RadioGroup. In RadioGroup checking the one radio button out of several radio button added in it will automatically unchecked all the others. It means at one time we can checked only one radio button from a group of radio buttons which belong to same radio group. The most common use of radio button is in Quiz App.



RadioButton is a two state button that can be checked or unchecked. If a radio button is unchecked then a user can check it by simply clicking on it. Once a RadioButton is checked by user it can't be unchecked by simply pressing on the same button. It will automatically unchecked when you press any other RadioButton within same RadioGroup.

Important Note: RadioGroup is a widget used in Android for the grouping of radio buttons and provide the feature of selecting only one radio button from the set. When a user try to select any other radio button within same radio group the previously selected radio button will be automatically unchecked.

RadioGroup And RadioButton code in XML:

```
<RadioGroup  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content">  
    <RadioButton  
        android:id="@+id/simpleRadioButton"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"/>  
    <RadioButton  
        android:id="@+id/simpleRadioButton1"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"/>  
</RadioGroup>
```



Checking Current State Of Radio Button:

You can check the current state of a radio button programmatically by using isChecked() method. This method returns a Boolean value either true or false. if it is checked then returns true otherwise returns false. Below is an example code with explanation in which we checked the current state of a radio button.

```
/*Add in Oncreate() function after setContentView()*/
RadioButton simpleRadioButton = (RadioButton) findViewById(R.id.simpleRadioButton); // initiate a radio button

Boolean RadioButtonState = simpleRadioButton.isChecked(); // check current state of a radio button (true or false).
```

Attributes of RadioButton In Android:

Now let's we discuss important attributes that helps us to create a beautiful radio button in xml file (layout).

1. id: id is an attribute used to uniquely identify a radio button.

```
<RadioButton
    android:id="@+id/simpleRadioButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
```

2. checked: checked attribute in radio button is used to set the current state of a radio button.

We can set it either true or false where true shows the checked state and false shows unchecked state of a radio button. As usual default value of checked attribute is false. We can also set the current state in JAVA.

Below we set true value for checked attribute which sets the current state to checked of a Button.

```
<RadioButton
    android:id="@+id/simpleRadioButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:checked="true"/> <!-- set the current state of the radio button-->
```



Setting checked State Of RadioButton In Java Class:

Below code set the current state of RadioButton to checked programmatically.

```
/*Add in Oncreate() funtion after setContentView()*/
// initiate a radio button
RadioButton simpleRadioButton = (RadioButton) findViewById(R.id.simpleRadioButton);

// set the current state of a radio button
simpleRadioButton.setChecked(true);
```

3. text: text attribute is used to set the text in a radio button. We can set the text both ways either in XML or in JAVA class.

Below is the example code with explanation included in which we set the text “I am a radiobutton” of a radio button.

```
<RadioButton
    android:id="@+id/simpleRadioButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:checked="true"
    android:layout_centerHorizontal="true"
    android:text="I am a radiobutton" /> <!-- displayed text of radio button-->
```

Setting text of RadioButton In Java class:

Below we set the text of a radio button programmatically:

```
/*Add in Oncreate() funtion after setContentView()*/
RadioButton simpleRadioButton=(RadioButton) findViewById(R.id.simpleRadioButton);
simpleRadioButton.setText("I am a radiobutton"); // displayed text of radio button
```



4. gravity: The gravity attribute is an optional attribute which is used to control the alignment of text like left, right, center, top, bottom, center_vertical, center_horizontal etc.

Below is the example code with explanation included in which we set the center gravity for the text of a radio button.

```
<RadioButton
    android:id="@+id/simpleRadioButton"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:checked="true"
    android:text="I am a Button"
    android:gravity="center"/> <!-- center gravity of the text-->
```



5. textColor: textColor attribute is used to set the text color of a radio button. Color value is in the form of “#argb”, “#rgb”, “#rrggbb”, or “#aarrggbb”.

Below we set the red color for the displayed text of a radio button.

```
<RadioButton
    android:id="@+id/simpleRadioButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:checked="true"
    android:layout_centerHorizontal="true"
    android:text="Male"
    android:textColor="#f00" /><!--red color for displayed text-->
```



Setting textColor of RadioButton text In Java class:

Below we set the text color of a radio button programmatically.

```
/*Add in Oncreate() function after setContentView()*/
RadioButton simpleRadioButton = (RadioButton) findViewById(R.id.simpleRadioButton); //initiate radio button
simpleRadioButton.setTextColor(Color.RED); //red color for displayed text of radio button
```

6. textSize: textSize attribute is used to set the size of the text of a radio button. We can set the text size in sp(scale independent pixel) or dp(density pixel).

Below we set the 25sp size for the text of a radio button.

```
<RadioButton
    android:id="@+id/simpleRadioButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:checked="true"
    android:layout_centerHorizontal="true"
```

```
    android:text="AbhiAndroid"
    android:textColor="#f00"
    android:textSize="25sp"/> <!--setting text size-->
```



Setting textSize Of RadioButton Text In Java class:

Below we set the text size of a radio button programmatically:

```
/*Add in Oncreate() funtion after setContentView()*/
RadioButton simpleRadioButton = (RadioButton) findViewById(R.id.simpleRadioButton); // initiate radio button

simpleRadioButton.setTextSize(25); // set 25sp displayed text size of radio button
```

7. textStyle: textStyle attribute is used to set the text style of the text of a radio button. The possible text styles are bold, italic and normal. If we need to use two or more styles for a text view then “|” operator is used for that.

Below is the example code with explanation included in which we set the bold and italic text styles for text of a radio button.

```
<RadioButton
    android:id="@+id/simpleRadioButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:checked="true"
    android:textSize="25sp"
    android:layout_centerHorizontal="true"
    android:text="Male"
    android:textColor="#f00"
    android:textStyle="bold|italic"/> <!-- bold and italic text style-->
```



8. background: background attribute is used to set the background of a radio button. We can set a color or a drawable in the background of a radio button.

Below we set the black color for the background and red color for the displayed text of a radio button

```
<RadioButton
    android:id="@+id/simpleRadioButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:checked="true"
    android:textSize="25sp"
    android:textStyle="bold|italic"
    android:padding="20dp"
    android:layout_centerHorizontal="true"
    android:text="Male"
    android:textColor="#f00"
    android:background="#000"/> <!-- black background for radio button-->
```



Setting Background Of RadioButton In Java class:

Below we set the background color of a radio button programmatically.

```
/*Add in Oncreate() function after setContentView()*/
RadioButton simpleRadioButton = (RadioButton) findViewById(R.id.simpleRadioButton);
simpleRadioButton.setBackgroundColor(Color.BLACK);
```

9. padding: padding attribute is used to set the padding from left, right, top or bottom.

paddingRight: set the padding from the right side of the radio button.

paddingLeft : set the padding from the left side of the radio button.

paddingTop : set the padding from the top side of the radio button.

paddingBottom: set the padding from the bottom side of the radio button.

Padding: set the padding from the all side's of the radio button.

Below we set padding attribute of 20dp padding from all the side's of the radio button.

```
<RadioButton
    android:id="@+id/simpleRadioButton"
    android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:checked="true"
        android:textSize="25sp"
        android:textStyle="bold|italic"
        android:layout_centerHorizontal="true"
        android:text="AbhiAndroid"
        android:textColor="#f00"
        android:padding="40dp"/> <!--40dp padding from all the sides of radio button-->
```



10. drawableBottom, drawableTop, drawableLeft And drawableRight: These attribute draw the drawable to the below of the text of RadioButton.

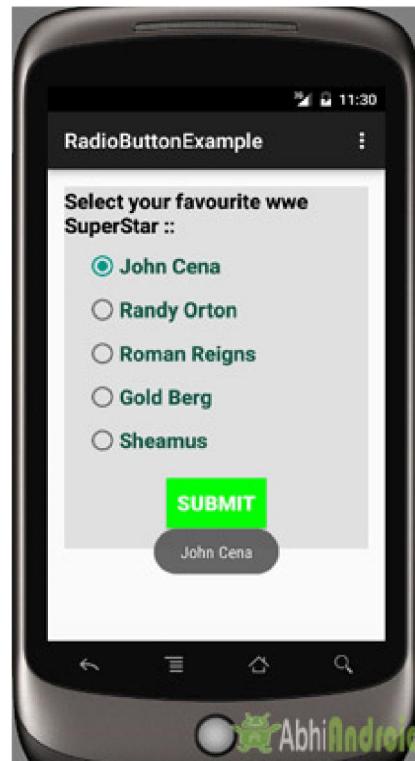
Below we set the icon to the right of the text of a RadioButton.

```
<RadioButton
    android:id="@+id/simpleRadioButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:checked="true"
    android:textSize="25sp"
    android:padding="20dp"
    android:layout_centerHorizontal="true"
    android:text="AbhiAndroid"
    android:textColor="#f00"
    android:drawableRight="@drawable/ic_launcher" /> <!-- drawable icon at the right of
radio button-->
```



Example Of RadioButton And RadioGroup in Android Studio:

Below is the example of Radiobutton in Android where we display five radio buttons with background and other attributes. The radio buttons are used to select your favorite WWE superstar with one “submit” button. Below is the final output, download code and step by step explanation of tutorial:



Step 1:

Create a new project and name it RadioButtonExample

Select File -> New -> New Project and Fill the forms and click “Finish” button.

Step 2:

Open res -> layout -> activity_main.xml (or) main.xml and add following code:

In this step we open an xml file and add the code for displaying 5 RadioButton and one normal button.

```
<LinearLayout 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:orientation="vertical"
    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">

    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="#e0e0e0"
        android:orientation="vertical">

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Select your favourite wwe SuperStar :: "
            android:textColor="#000"
            android:textSize="20sp"
            android:textStyle="bold" />

        <RadioGroup
            android:layout_width="wrap_content"
            android:layout_height="wrap_content">

            <RadioButton
                android:id="@+id/johnCena"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginLeft="20dp"
                android:layout_marginTop="10dp"
                android:checked="true"
                android:text="@string/johnCena"
                android:textColor="#154"
                android:textSize="20sp" />

            <RadioButton
                android:id="@+id/rock"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginLeft="20dp"
                android:layout_marginTop="10dp"
                android:checked="false"
                android:text="@string/rock"
                android:textColor="#154"
                android:textSize="20sp" />

            <RadioButton
                android:id="@+id/StoneCold"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginLeft="20dp"
                android:layout_marginTop="10dp"
                android:checked="false"
                android:text="@string/stoneCold"
                android:textColor="#154"
                android:textSize="20sp" />

            <RadioButton
                android:id="@+id/StonePapa"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginLeft="20dp"
                android:layout_marginTop="10dp"
                android:checked="false"
                android:text="@string/stonePapa"
                android:textColor="#154"
                android:textSize="20sp" />

            <RadioButton
                android:id="@+id/CMWZ"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginLeft="20dp"
                android:layout_marginTop="10dp"
                android:checked="false"
                android:text="@string/cmWZ"
                android:textColor="#154"
                android:textSize="20sp" />
        
    

```

```
        android:textStyle="bold" />

<RadioButton
    android:id="@+id/randyOrton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="20dp"
    android:layout_marginTop="10dp"
    android:checked="false"
    android:text="@string/randyOrton"
    android:textColor="#154"
    android:textSize="20sp"
    android:textStyle="bold" />

<RadioButton
    android:id="@+id/romanReigns"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="20dp"
    android:layout_marginTop="10dp"
    android:checked="false"
    android:text="@string/romanReigns"
    android:textColor="#154"
    android:textSize="20sp"
    android:textStyle="bold" />

<RadioButton
    android:id="@+id/goldBerg"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="20dp"
    android:layout_marginTop="10dp"
    android:checked="false"
    android:text="@string/goldBerg"
    android:textColor="#154"
    android:textSize="20sp"
    android:textStyle="bold" />

<RadioButton
    android:id="@+id/sheamus"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="20dp"
    android:layout_marginTop="10dp"
    android:checked="false"
    android:text="@string/sheamus"
    android:textColor="#154"
    android:textSize="20sp"
    android:textStyle="bold" />

</RadioGroup>

<Button
    android:id="@+id/submitButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
```

```
        android:layout_gravity="center"
        android:layout_margin="20dp"
        android:background="#0f0"
        android:padding="10dp"
        android:text="Submit"
        android:textColor="#fff"
        android:textSize="20sp"
        android:textStyle="bold" />
    </LinearLayout>

</LinearLayout>
```

Step 3:

Open src -> package -> MainActivity.java

In this step we open MainActivity and add the code to initiate the RadioButton and normal button. We also perform click event on button and display the selected superstar's name by using a Toast.

```
package example.gb.radioButtonexample;

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

public class MainActivity extends AppCompatActivity {

    RadioButton johnCena, randyOrton, goldBerg, romanReigns, sheamus;
    String selectedSuperStar;
    Button submit;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        johnCena = (RadioButton) findViewById(R.id.johnCena);
        randyOrton = (RadioButton) findViewById(R.id.randyOrton);
        goldBerg = (RadioButton) findViewById(R.id.goldBerg);
        romanReigns = (RadioButton) findViewById(R.id.romanReigns);
        sheamus = (RadioButton) findViewById(R.id.sheamus);
        submit = (Button) findViewById(R.id.submitButton);
        submit.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (randyOrton.isChecked()) {
```

```
        selectedSuperStar = randyOrton.getText().toString();
    } else if (sheamus.isChecked()) {
        selectedSuperStar = sheamus.getText().toString();
    } else if (johnCena.isChecked()) {
        selectedSuperStar = johnCena.getText().toString();
    } else if (romanReigns.isChecked()) {
        selectedSuperStar = romanReigns.getText().toString();
    } else if (goldBerg.isChecked()) {
        selectedSuperStar = goldBerg.getText().toString();
    }
    Toast.makeText(getApplicationContext(), selectedSuperStar,
Toast.LENGTH_LONG).show(); // print the value of selected super star
}
});
```

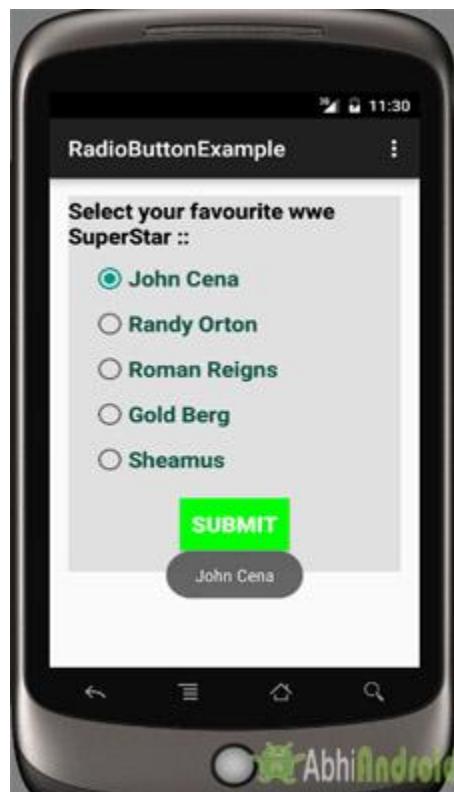
Step 4: Open res -> values -> strings.xml

In this step we open String file which is used to store string data of the app.

```
<resources>
    <string name="app_name">RadioButtonExample</string>
    <string name="hello_world">Hello world!</string>
    <string name="action_settings">Settings</string>
    <string name="randyOrton">Randy Orton</string>
    <string name="johnCena">John Cena</string>
    <string name="romanReigns">Roman Reigns</string>
    <string name="goldBerg">Gold Berg</string>
    <string name="sheamus">Sheamus</string>
</resources>
```

Run The App:

Now run the App in Emulator and you will see 5 RadioButton in RadioGroup listing WWE superstar name. Now choose your favorite one and click on Submit Button. The name will be displayed on Screen.



RatingBar

RatingBar is used to get the rating from the app user. A user can simply touch, drag or click on the stars to set the rating value. The value of rating always returns a floating point number which may be 1.0, 2.5, 4.5 etc.



In Android, RatingBar is an extension of ProgressBar and SeekBar which shows a rating in stars. RatingBar is a subclass of AbsSeekBar class.

The `getRating()` method of android RatingBar class returns the rating number.

RatingBar code in XML:

```
<RatingBar  
    android:id="@+id/simpleRatingBar"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content" />
```

Methods Used in Rating Bar:

`getRating():`

You can get the rating number from a RatingBar by using `getRating()` method. This method returns a Floating Point number. Below we get the current rating number from a RatingBar.

```
/*Add in Oncreate() function after setContentView()*/  
RatingBar simpleRatingBar = (RatingBar) findViewById(R.id.simpleRatingBar); // initiate a  
rating bar  
Float ratingNumber = simpleRatingBar.getRating(); // get rating number from a rating bar
```

getNumStars():

You can get the number of stars of a RatingBar by using getNumstars() method. This method returns int value. In below code we get the total number of stars of a RatingBar.

```
/*Add in Oncreate() function after setContentView()*/
RatingBar simpleRatingBar = (RatingBar) findViewById(R.id.simpleRatingBar); // initiate a rating bar
int numberOfStars = simpleRatingBar.getNumStars(); // get total number of stars of rating bar
```

Attributes Used in RatingBar:

Now let's we discuss some important attribute that helps us to configure a RatingBar in XML file (layout).

1. id: id is an attribute used to uniquely identify a rating bar.

```
<RatingBar
    android:id="@+id/simpleRatingBar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
```

2. background: background attribute is used to set the background of a RatingBar. We can set a color or a drawable in the background of a rating bar.

Below we set the red color for the background of a rating bar.

```
<RatingBar
    android:id="@+id/simpleRatingBar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="#f00"/> <!-- red color for the background of rating bar-->
```



Setting Background of RatingBar In Java class:

```
RatingBar simpleRatingBar = (RatingBar) findViewById(R.id.simpleRatingBar); // initiate a rating bar  
simpleRatingBar.setBackgroundColor(Color.RED); // set background color for a rating bar
```

3. numStars: numStars attribute is used to set the number of stars (or rating items) to be displayed in a rating bar. By default a rating bar shows five stars but we can change it using numStars attribute.

numStars must have a integer number like 1,2 etc.

Below we set num stars value to 7 of RatingBar.

```
<RatingBar  
    android:id="@+id/simpleRatingBar"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:numStars="7" /><!-- number of stars to be displayed-->
```



Setting numStars of RatingBar In Java class:

```
RatingBar simpleRatingBar = (RatingBar) findViewById(R.id.simpleRatingBar); // initiate a rating bar  
simpleRatingBar.setNumStars(7); // set total number of stars
```

4. rating: Rating attribute set the default rating of a rating bar. It must be a floating point number.

Below we set default rating to 3.5 for a rating bar.

```
<RatingBar  
    android:id="@+id/simpleRatingBar"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:rating="3.5" /> <!-- default rating-->
```



Setting Default Rating of RatingBar In Java class:

```
/*Add in Oncreate() function after setContentView()*/  
RatingBar simpleRatingBar = (RatingBar) findViewById(R.id.simpleRatingBar); // initiate a  
rating bar  
simpleRatingBar.setRating((float) 3.5); // set default rating
```

5. padding: padding attribute is used to set the padding from left, right, top or bottom.

paddingRight: set padding from the right side of the rating bar.

paddingLeft: set padding from the left side of the rating bar.

paddingTop: set padding from the top side of the rating bar.

paddingBottom: set the padding from the bottom side of the rating bar.

Padding: set the padding from the all side's of the rating bar.

Below we set the 20dp padding from all the side's of the rating bar.

```
<RatingBar  
    android:id="@+id/simpleRatingBar"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:rating="2.5"  
    android:numStars="6"  
    android:background="#f00"  
    android:padding="20dp"/> <!--20dp padding from all the sides of rating bar-->
```



RatingBar Example In Android Studio:

Below is the example of RatingBar in Android where we displayed a RatingBar with five stars and a submit button. Whenever a user click on the button value of total number of stars and value of rating is shown by using a Toast on screen. Below is the final output, download code and step by step tutorial:



Important Note: We can also create custom rating bar in Android in which we can change the star images like filled or empty star. We discussed this in next example, so don't miss it.

Step 1: Create a new project and name it RatingBarExample

Select File -> New -> New Project and Fill the forms and click "Finish" button.

Step 2: Open res -> layout -> activity_main.xml (or) main.xml and add following code

In this step we open an xml file and add the code for displaying a rating bar with five number of stars and “2” value for default rating and one submit button.

```
<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">

    <RatingBar
```

```
        android:id="@+id/simpleRatingBar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="60dp"
        android:background="#0f0"
        android:paddingLeft="5dp"
        android:paddingRight="5dp"
        android:rating="2" />

    <Button
        android:id="@+id/submitButton"
        android:layout_width="200dp"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:background="#f00"
        android:padding="10dp"
        android:text="Submit"
        android:textColor="#fff"
        android:textSize="20sp"
        android:textStyle="bold" />
</RelativeLayout>
```

Step 3: Open src -> package -> MainActivity.java

In this step we open MainActivity where we add the code to initiate the RatingBar & button and then we perform click event on button and display the total number of stars and rating by using a toast.

```
package example.gb.ratingbarexample;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.RatingBar;
import android.widget.Button;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
setContentView(R.layout.activity_main);
// initiate rating bar and a button
final RatingBar simpleRatingBar = (RatingBar) findViewById(R.id.simpleRatingBar);
Button submitButton = (Button) findViewById(R.id.submitButton);
// perform click event on button
submitButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // get values and then displayed in a toast
        String totalStars = "Total Stars:: " + simpleRatingBar.getNumStars();
        String rating = "Rating :: " + simpleRatingBar.getRating();
        Toast.makeText(getApplicationContext(), totalStars + "\n" + rating,
Toast.LENGTH_LONG).show();
    }
});
```

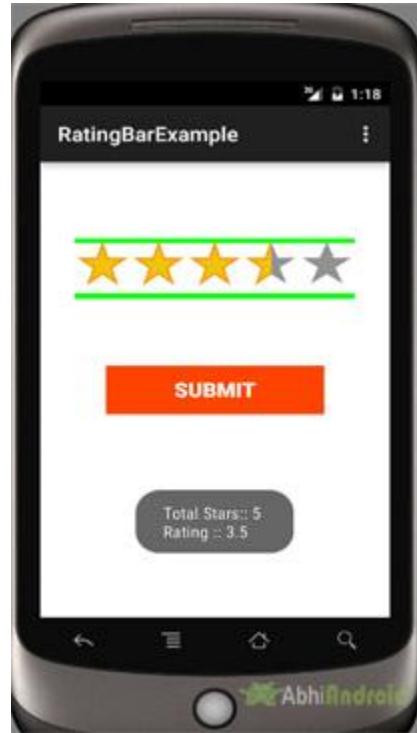
Output:

Now start the AVD in Emulator and run the App. You will see the rating option on screen. Select your rating and click on Submit. Your rating will be displayed on screen as Toast.



Custom RatingBar Example In Android Studio:

In the below example of custom rating bar in Android, we displayed a rating bar with five custom stars and a submit button. Whenever a user click on the button value of total number of stars and value of rating is shown by using a Toast. Below is the final output, download code and step by step explanation of Custom RatingBar.

**Step 1:** Create a new project and name it CustomRatingBarExample

Select File -> New -> New Project -> then Fill the forms and click "Finish" button.

Step 2: Open res -> layout -> activity_main.xml (or) main.xml and add following code

In this step we open an xml file and add the code for displaying a custom rating bar with five number of stars and “ 2.5 ” value for default rating and one submit button.

```
<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:background="#fff"
    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">

    <RatingBar
        android:id="@+id/simpleRatingBar"
        style="@style/customRatingBar"
        android:layout_width="wrap_content"
```

```

        android:layout_height="60dp"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="60dp"
        android:background="#0f0"
        android:rating="2.5" />

    <Button
        android:id="@+id/submitButton"
        android:layout_width="200dp"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:background="#f40"
        android:padding="10dp"
        android:text="Submit"
        android:textColor="#fff"
        android:textSize="20sp"
        android:textStyle="bold" />
</RelativeLayout>
```

Step 3: Open src -> package -> MainActivity.java

In this step we open MainActivity where we add the code to initiate the RatingBar & button and then perform click event on button and display the total number of stars and rating by using a Toast.

```

package example.gb.customratingbarexample;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.RatingBar;
import android.widget.Button;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // initiate rating bar and a button
        final RatingBar simpleRatingBar = (RatingBar) findViewById(R.id.simpleRatingBar);
        Button submitButton = (Button) findViewById(R.id.submitButton);
        // perform click event on button
        submitButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // get values and then displayed in a toast
                String totalStars = "Total Stars:: " + simpleRatingBar.getNumStars();
                String rating = "Rating :: " + simpleRatingBar.getRating();
                Toast.makeText(getApplicationContext(), totalStars + "\n" + rating,
                        Toast.LENGTH_LONG).show();
            }
        });
    }
}
```

```
        });
    }
}
```

Step 4:

Open res ->values ->styles.xml

In this step we add the code for displaying custom stars for rating. To do that in this styles.xml file we set an custom xml file from drawable and set the height and width for that item.

```
<resources>

    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- Customize your theme here. -->
    </style>
    <style name="customRatingBar" parent="@android:style/Widget.RatingBar">
        <item name="android:progressDrawable">@drawable/customratingstars</item>
        <item name="android:minHeight">20dip</item>
        <item name="android:maxHeight">20dip</item>
    </style>
</resources>
```

Step 5:

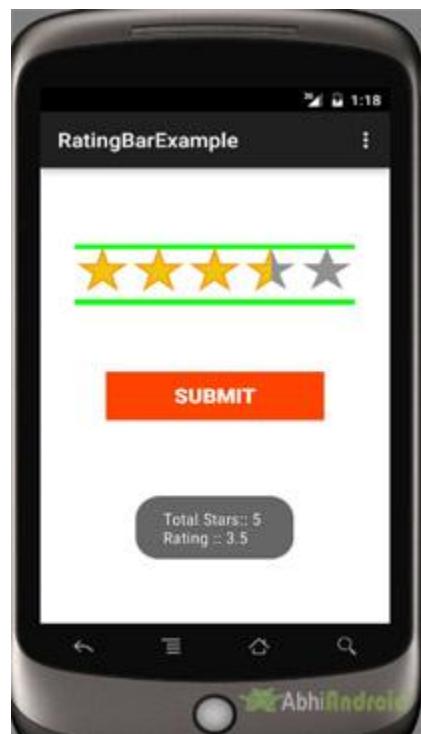
Create a new drawable XML file -> customratingstars.xml

In this step we create a new drawable XML file, in which we set the icons for filled and empty stars. As shown in below code snippet empty and filled are two different icons set from drawable.

```
<?xml version="1.0" encoding="utf-8"?>
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/background"
          android:drawable="@drawable/empty" />
    <item android:id="@+id/secondaryProgress"
          android:drawable="@drawable/empty" />
    <item android:id="@+id/progress"
          android:drawable="@drawable/filled" />
</layer-list>
```

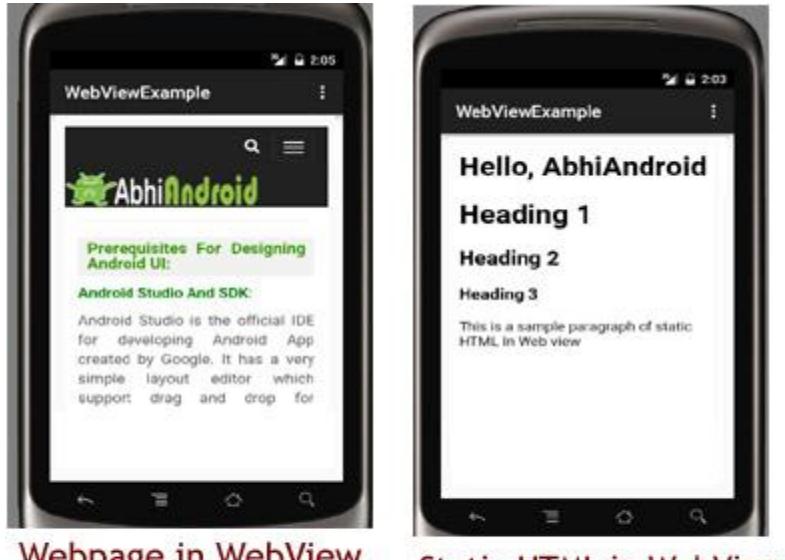
Output:

Now start the AVD in Emulator and run the App. You will see custom star rating option on screen.



WebView

In Android, WebView is a view used to display the web pages in application. This class is the basis upon which you can roll your own web browser or simply use it to display some online content within your Activity. We can also specify HTML string and can show it inside our application using a WebView. Basically, WebView turns application into a web application.



WebView

In order to add Web View in your application, you have to add <WebView> element to your XML(layout) file or you can also add it in java class.

```
<WebView  
    android:id="@+id/simpleWebView"  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent" />
```

Internet Permission Required For Webview:

Important Note: In order for Activity to access the Internet and load the web pages in a WebView, we must add the internet permissions to our Android Manifest file (Manifest.xml).

Below code define the internet permission in our manifest file to access the internet in our application.

```
<!--Add this before application tag in AndroidManifest.xml-->
<uses-permission android:name="android.permission.INTERNET" />
```



Methods of WebView In Android:

Let's discuss some common methods of a Webview which are used to configure a web view in our application.

loadUrl() – Load a web page in our WebView

```
loadUrl(String url)
```

This function is used to load a web page in a web view of our application. In this method we specify the url of the web page that should be loaded in a web view.

Below we load a url: <http://abhiandroid.com/ui/> in our application.

```
/*Add in Oncreate() function after setContentView()*/
// initiate a web view
WebView simpleWebView=(WebView) findViewById(R.id.simpleWebView);
// specify the url of the web page in loadUrl function
simpleWebView.loadUrl("http://abhiandroid.com/ui/");
```



2. loadData() – Load Static Html Data on WebView

```
loadData(String data, String mimeType, String encoding)
```

This method is used to load the static HTML string in a web view. loadData() function takes html string data, mime-type and encoding param as three parameters.

Below we load the static Html string data in our application of a web view.

```
/*Add in Oncreate() funtion after setContentView()*/
// initiate a web view
WebView webView = (WebView) findViewById(R.id.simpleWebView);
// static html string data
String customHtml = "<html><body><h1>Hello, AbhiAndroid</h1>" +
    "<h1>Heading 1</h1><h2>Heading 2</h2><h3>Heading 3</h3>" +
    "<p>This is a sample paragraph of static HTML In Web view</p>" +
    "</body></html>";
// load static html data on a web view
webView.loadData(customHtml, "text/html", "UTF-8");
```



3. Load Remote URL on WebView using WebClient:

WebviewClient help us to monitor event in a WebView. You have to Override the `shouldOverrideUrlLoading()` method. This method allow us to perform our own action when a particular url is selected. Once you are ready with the WebviewClient, you can set the WebviewClient in your WebView using the `setWebviewClient()` method.

Below we load a url by using web view client in a WebView.

```
/*Add in Oncreate() function after setContentView()*/
// initiate a web view
simpleWebView = (WebView) findViewById(R.id.simpleWebView);

// set web view client
simpleWebView.setWebViewClient(new MyWebViewClient());

// string url which you have to load into a web view
String url = "http://abhiandroid.com/ui/";
simpleWebView.getSettings().setJavaScriptEnabled(true);
simpleWebView.loadUrl(url); // load the url on the web view
}

// custom web view client class who extends WebviewClient
private class MyWebViewClient extends WebviewClient {
@Override
public boolean shouldOverrideUrlLoading(WebView view, String url) {
view.loadUrl(url); // load the url
return true;
}
```

4. **canGoBack()** – Move to one page back if a back history exist

This method is used to specify whether the web view has a back history item or not. This method returns a Boolean value either true or false. If it returns true then goBack() method is used to move one page back.

Below we check whether a web view has back history or not.

```
// initiate a web view
WebView simpleWebView=(WebView)findViewById(R.id.simpleWebView);
// checks whether a web view has a back history item or not
Boolean canGoBack=simpleWebView.canGoBack();
```

5. **canGoForward()** – Move one page forward if forward history exist

This method is used to specify whether the web view has a forward history item or not. This method returns a Boolean value either true or false. If it returns true then goForward() method is used to move one page forward.

Below we check whether a web view has forward history or not.

```
// initiate a web view
WebView simpleWebView=(WebView)findViewById(R.id.simpleWebView);
// checks whether a web view has a forward history item or not
Boolean canGoForward=simpleWebView.canGoForward();
```

6. **clearHistory()** – clear the WebView history

This method is used to clear the web view forward and backward history.

Below we clear the forward and backward history of a WebView.

```
WebView simpleWebView=(WebView)findViewById(R.id.simpleWebView); // initiate a web view
simpleWebView.clearHistory(); // clear the forward and backward history
```

WebView Example In Android Studio:

Here in this WebView example we show the use of web view in our application. To do that we display two buttons one is for displaying a web page and other is for displaying static HTML data in a web view. Below is the final output, download code and step by step explanation:



Step 1: Create a new project and name it WebViewExample

Select File -> New -> New Project... then Fill the forms and click "Finish" button.

Step 2: Open res -> layout -> activity_main.xml (or) main.xml and add following code :

In this step we open an XML file and add the code for displaying two buttons and a Webview in our xml file (layout)

```
<LinearLayout 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:orientation="vertical"
    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">

    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:weightSum="2">

        <Button
            android:id="@+id/loadWebPage"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_marginRight="10dp"
            android:layout_weight="1"
            android:background="#444"
            android:text="Load Web Page"
            android:textColor="#fff"
            android:textSize="14sp"
            android:textStyle="bold" />

        <Button
            android:id="@+id/loadFromStaticHtml"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_marginLeft="10dp"
            android:layout_weight="1"
            android:background="#444"
            android:text="Load Static HTML"
            android:textColor="#fff"
            android:textSize="14sp"
            android:textStyle="bold" />
    </LinearLayout>

    <WebView
        android:id="@+id/simpleWebView"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:layout_marginTop="20dp"
        android:scrollbars="none" />
</LinearLayout>
```

Step 3: Open src -> package -> MainActivity.java

In this step we open MainActivity and add the code to initiate the web view and two buttons. Out of those one button is used for displaying a web page in a webview and other one is used to load a static HTML page in webview.

```
package example.gb.webviewexample;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.support.v7.widget.Toolbar;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.webkit.WebView;
import android.webkit.WebViewClient;
import android.widget.Button;

public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    WebView simpleWebView;
    Button loadWebPage, loadFromStaticHtml;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // initiate buttons and a web view
        loadFromStaticHtml = (Button) findViewById(R.id.loadFromStaticHtml);
        loadFromStaticHtml.setOnClickListener(this);
        loadWebPage = (Button) findViewById(R.id.loadWebPage);
        loadWebPage.setOnClickListener(this);
        simpleWebView = (WebView) findViewById(R.id.simpleWebView);

    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.loadFromStaticHtml:
                // define static html text
                String customHtml = "<html><body><h1>Hello, AbhiAndroid</h1>" +
                    "<h1>Heading 1</h1><h2>Heading 2</h2><h3>Heading 3</h3>" +
                    "<p>This is a sample paragraph of static HTML In Web view</p>" +
                    "</body></html>";
                simpleWebView.loadData(customHtml, "text/html", "UTF-8"); // load html
                string data in a web view
                break;
            case R.id.loadWebPage:
                simpleWebView.setWebViewClient(new MyWebViewClient());
                String url = "http://abhiandroid.com/ui/";
                simpleWebView.getSettings().setJavaScriptEnabled(true);
                simpleWebView.loadUrl(url); // load a web page in a web view
                break;
        }
    }

    private class MyWebViewClient extends WebViewClient {
```

```
    @Override
    public boolean shouldOverrideUrlLoading(WebView view, String url) {
        view.loadUrl(url);
        return true;
    }
}
```

Step 4: Open manifests -> AndroidManifest.xml

In this step we open Manifest file and define the internet permission for our app.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="example.gb.webviewexample">
    <!-- define internet permission for our app -->
    <uses-permission android:name="android.permission.INTERNET" />
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme">
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Output:

Now start the AVD in Emulator and run the App. Choose either to open webpage or static HTML in WebView by clicking on Button. We open static HTML.

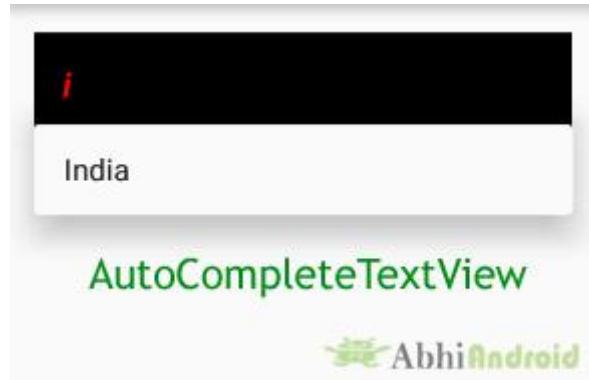


Convert Website Into App Using WebView:

Now you have learned WebView, so why not try converting any website into an Android App?

AutocompleteTextview

In Android, AutoCompleteTextView is a view i.e similar to EditText, except that it displays a list of completion suggestions automatically while the user is typing. A list of suggestions is displayed in drop down menu from which user can choose an item which actually replace the content of Editbox with that.



It is a subclass of EditText class so we can inherit all the properties of EditText in a AutoCompleteTextView.

AutoCompleteTextView code in XML:

```
<AutoCompleteTextView  
    android:id="@+id/simple.AutoCompleteTextView"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:text="This is an AutoCompleteTextView"/>
```

Using Array Adapter To Display Text Values In AutoCompleteTextView:

To display the Array content in an autocompletetextview we need to implement Adapter. In AutoCompleteTextView we mainly display text values so we use Array Adapter for that.

ArrayAdapter In Android:

ArrayAdapter is used when we need list of single type of items which is backed by an Array. For example, list of phone contacts, countries or names.

ArrayAdapter code:

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

Retrieving the Value From AutoCompleteTextView In Java Class:

Below code retrieve the value from a AutoCompleteTextView in Java class.

```
AutoCompleteTextView simple.AutoCompleteTextView = (AutoCompleteTextView)
findViewById(R.id.simple.AutoCompleteTextView);

String AutoCompleteTextViewValue = simple.AutoCompleteTextView.getText().toString();
```

Attributes of AutoCompleteTextView:

Now let's we discuss about the attributes that helps us to configure a AutoCompleteTextView in your xml file.

1. id: id is an attribute used to uniquely identify a text AutoCompleteTextView.

```
<AutoCompleteTextView
    android:id="@+id/simple.AutoCompleteTextView"
    android:layout_height="wrap_content"
    android:layout_width="match_parent"/>
```

2. text: text attribute is used to set the text in a AutoCompleteTextView. We can set the text in XML as well as in the java class.

Below we set the text "Country" in a AutoCompleteTextView.

```
<AutoCompleteTextView
    android:id="@+id/simple.AutoCompleteTextView"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Country"/><!--display text "Country"-->
```



3. gravity: The gravity attribute is an optional attribute which control the alignment of the text like left, right, center, top, bottom, center_vertical, center_horizontal etc.

Below we set the right gravity for text of a AutoCompleteTextView.

```
<AutoCompleteTextView  
    android:id="@+id/simple.AutoCompleteTextView"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:text="Country"  
    android:gravity="right"/><!--right gravity for text-->
```



Setting Text Of AutoCompleteTextView In Java class:

```
/*Add in Oncreate() funtion after setContentView()*/
```

```
AutoCompleteTextView autoCompleteTextView =
(AutoCompleteTextView)findViewById(R.id.simple.AutoCompleteTextView);

//display text Country
autoCompleteTextView.setText("Country");
```

4. hint: hint attribute gives the hint to the user that what should he Enter in this AutoCompleteTextView. Whenever he start to type in AutoCompleteTextView the hint will automatically disappear.

```
<AutoCompleteTextView
    android:id="@+id/simple.AutoCompleteTextView"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="Enter Your Country Name Here" /><!--display hint-->
```



Setting hint For AutoCompleteTextView In Java class:

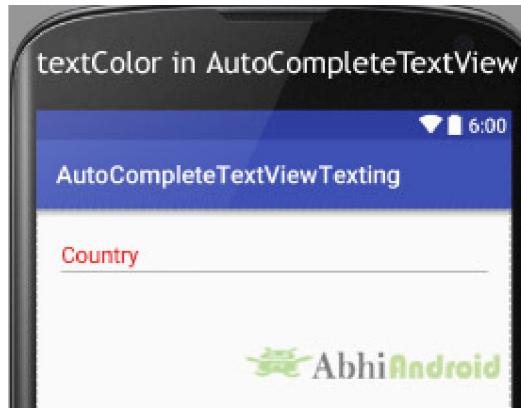
```
/*Add in Oncreate() funtion after setContentView()*/
AutoCompleteTextView autoCompleteTextView =
(AutoCompleteTextView)findViewById(R.id.simple.AutoCompleteTextView);

autoCompleteTextView.setHint("Enter Your Name Here");//display hint
```

5. textColor: This attribute set the color of a text in AutoCompleteTextView. Color value can be in the form of “#argb”, “#rgb”, “#rrggb”, or “#aarrggbb”.

```
<AutoCompleteTextView
    android:id="@+id/simple.AutoCompleteTextView"
    android:layout_width="fill_parent"
```

```
    android:layout_height="wrap_content"
    android:text="Country"
    android:textColor="#f00"/><!--red color for text--&gt;</pre>
```



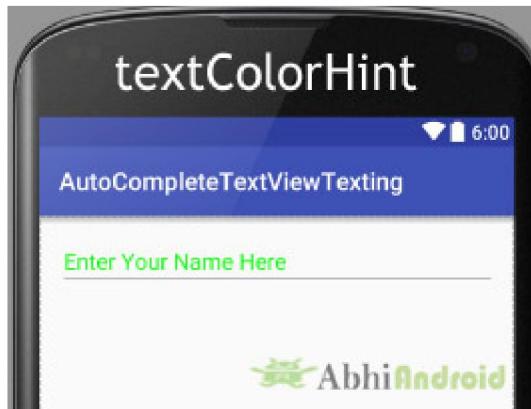
Setting TextColor of AutoCompleteView Text In Java class:

```
/*Add in Oncreate() function after setContentView()*/
AutoCompleteTextView
simple.AutoCompleteTextView=(AutoCompleteTextView)findViewById(R.id.simpleAutoCompleteTextVie
w);

simple.AutoCompleteTextView.setTextColor(Color.RED); //red color for text
```

6. textColorHint: This attribute is used to set the color of displayed hint.

```
<AutoCompleteTextView
    android:id="@+id/simpleAutoCompleteTextView"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="Enter Your Name Here"
    android:textColorHint="#0f0"/><!--green color for hint--&gt;</pre>
```



Setting Color to Hint Of AutoCompleteTextView In Java class:

```
/*Add in Oncreate() function after setContentView()*/
AutoCompleteTextView
simple.AutoCompleteTextView=(AutoCompleteTextView)findViewById(R.id.simple.AutoCompleteTextView);
//green color for displayed hint
simple.AutoCompleteTextView.setHintTextColor(Color.green(0));
```

7. textSize: This attribute set the size of text in AutoCompleteTextView. We can set the text size in sp(scale independent pixel) or dp(density pixel).

```
<AutoCompleteTextView
    android:id="@+id/simple.AutoCompleteTextView"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Country"
    android:textSize="25sp" /><!--set the text size-->
```



Setting Text Size In Java class:

```
/*Add in Oncreate() function after setContentView()*/
AutoCompleteTextView
simple.AutoCompleteTextView=(AutoCompleteTextView)findViewById(R.id.simple.AutoCompleteTextView);
//set the text size
simple.AutoCompleteTextView.setTextSize(25);
```

8. textStyle: textStyle attribute is used to give text style of a AutoCompleteTextView. We can add bold, italic and normal style. If we want to use two or more styles for a AutoCompleteTextView then “|” operator is used for that.

```
<AutoCompleteTextView
```

```
    android:id="@+id/simpleAutoCompleteTextView"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Country"
    android:textSize="25sp"
    android:textStyle="bold|italic"/><!--bold and italic text style--&gt;</pre>
```



9. background and padding: background attribute is used to set the background of a AutoCompleteTextView. We can set a color or a drawable in the background of a AutoCompleteTextView.

padding attribute is used to set the padding from left, right, top or bottom.

Below we set black color as background, white color as displayed hint and set 15dp padding from all the side's for AutoCompleteTextView.

```
<AutoCompleteTextView
    android:id="@+id/simpleAutoCompleteTextView"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:background="#000"
    android:hint="Enter Your Name Here"
    android:padding="15dp"
    android:textColorHint="#fff"
    android:textStyle="bold|italic" />
```



Setting Background of AutoCompleteTextView In Java class:

```
/*Add in Oncreate() funtion after setContentView()*/
AutoCompleteTextView
simple.AutoCompleteTextView=(AutoCompleteTextView)findViewById(R.id.simple.AutoCompleteTextVie
w);

simple.AutoCompleteTextView.setBackgroundColor(Color.BLACK); //set black background color
```

AutoCompleteTextView Example in Android Studio:

In the example of AutoCompleteTextView we display a auto complete text view with suggestion list which include country list. To fill the data in country list we implement an Array Adapter. Below is the final output, download code and step by step tutorial:



Step 1: Create a new project and name it AutoCompleteTextViewExample.

Step 2: Open res -> layout -> activity_mail.xml (or) main.xml and add following code:

In this step we open an xml file and add the code for displaying an auto complete text view by using different attributes as we discussed earlier in this article.

```
<?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="abhiandroid.com.autocompletetextviewexample.MainActivity">

    <AutoCompleteTextView
        android:id="@+id/simple.AutoCompleteTextView"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="#000"
        android:hint="Enter Your Name Here"
        android:padding="15dp"
        android:textColorHint="#fff"
        android:textStyle="bold|italic" />
</RelativeLayout>
```

Step 3: Open app -> package -> MainActivity.java

In this step we open MainActivity where we add the code to initiate the auto complete text view and then fill the data in the suggestion list by using an Array Adapter.

```
package example.abhiandriod.autocompletetextviewexample;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.ArrayAdapter;
import android.widget.AutoCompleteTextView;

public class MainActivity extends AppCompatActivity {

    String[] countryNameList = {"India", "China", "Australia", "New Zealand", "England",
    "Pakistan"};
    @Override
```

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
    //initiate an auto complete text view  
    AutoCompleteTextView simple.AutoCompleteTextView = (AutoCompleteTextView)  
findViewById(R.id.simple.AutoCompleteTextView);  
    ArrayAdapter adapter = new ArrayAdapter(this, android.R.layout.simple_list_item_1,  
countryNameList);  
  
    simple.AutoCompleteTextView.setAdapter(adapter);  
    simple.AutoCompleteTextView.setThreshold(1); //start searching from 1 character  
    simple.AutoCompleteTextView.setAdapter(adapter); //set the adapter for displaying  
country name list  
}  
  
}
```

Output:

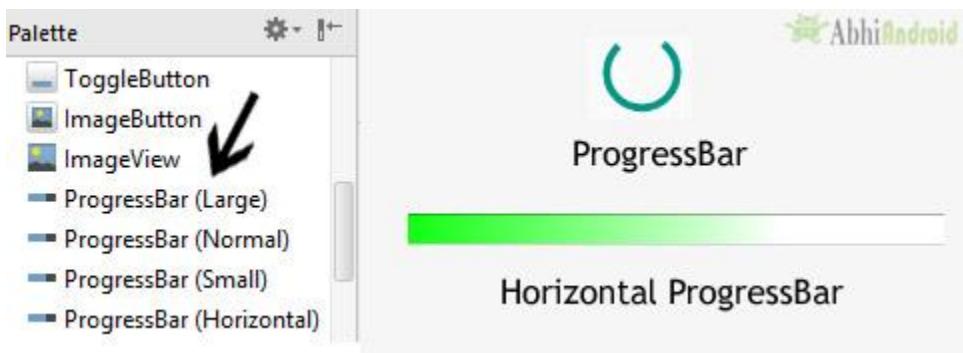
Now start the AVD in Emulator and run the App. Type ‘i’ & it will show ‘India’ as suggestion, type ‘A’ & it will show ‘Australia’ as suggestion and so on. This is done using AutoCompleteTextView.



ProgressBar

ProgressBar Tutorial With Example In Android Studio

In Android, ProgressBar is used to display the status of work being done like analyzing status of work or downloading a file etc. In Android, by default a progress bar will be displayed as a spinning wheel but If we want it to be displayed as a horizontal bar then we need to use style attribute as horizontal. It mainly use the “android.widget.ProgressBar” class.



Important Note: A progress bar can also be made indeterminate. In this mode a progress bar shows a cyclic animation without an indication of progress. This mode is used in application when we don't know the amount of work to be done.

To add a progress bar to a layout (xml) file, you can use the <ProgressBar> element. By default, a progress bar is a spinning wheel (an indeterminate indicator). To change to a horizontal progress bar, apply the progress bar's horizontal style.

ProgressBar code:

```
<ProgressBar  
    android:id="@+id/simpleProgressBar"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content" />
```

Horizontal ProgressBar code:

```
<ProgressBar  
    android:id="@+id/simpleProgressBar"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    style="@style/Widget.AppCompat.ProgressBar.Horizontal"/>
```

Important Methods Used In ProgressBar:

1. getMax() – returns the maximum value of progress bar

We can get the maximum value of the progress bar in java class. This method returns a integer value. Below is the code to get the maximum value from a Progress bar.

```
ProgressBar simpleProgressBar=(ProgressBar) findViewById(R.id.simpleProgressBar);  
// initiate the progress bar  
int maxValue=simpleProgressBar.getMax(); // get maximum value of the progress bar
```

2. getProgress() – returns current progress value

We can get the current progress value from a progress bar in java class. This method also returns a integer value. Below is the code to get current progress value from a Progress bar.

```
ProgressBar simpleProgressBar=(ProgressBar)findViewById(R.id.simpleProgressBar); // initiate  
the progress bar  
int progressValue=simpleProgressBar.getProgress(); // get progress value from the progress  
bar
```

Attributes of ProgressBar In Android:

Now let's discuss important attributes that helps us to configure a Progress bar in xml file (layout).

1. id: id is an attribute used to uniquely identify a Progress bar.

```
<ProgressBar  
    android:id="@+id/simpleProgressBar"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    style="@style/Widget.AppCompat.ProgressBar.Horizontal"/>
```

2. max: max is an attribute used in android to define maximum value of the progress can take.

It must be an integer value like 100, 200 etc.

Below we set 100 maximum value for a progress bar.

```
<ProgressBar  
    android:id="@+id/simpleProgressBar"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    style="@style/Widget.AppCompat.ProgressBar.Horizontal"  
    android:max="100" /><!--set 100 maximum value for the progress bar-->
```

Set Max Value of ProgressBar In Java Class :

```
ProgressBar simpleProgressBar=(ProgressBar) findViewById(R.id.simpleProgressBar); //  
initiate the progress bar  
simpleProgressBar.setMax(100); // 100 maximum value for the progress bar
```



3. progress: progress is an attribute used in android to define the default progress value between 0 and max. It must be an integer value.

Below we set the 100 max value and then set 50 default progress.

```
<ProgressBar
    android:id="@+id/simpleProgressBar"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:max="100"
    style="@style/Widget.AppCompat.ProgressBar.Horizontal"
    android:progress="50"/><!--// 50 default progress value-->
```



Setting Progress Value of ProgressBar In Java Class :

```
ProgressBar simpleProgressBar=(ProgressBar)findViewById(R.id.simpleProgressBar); // initiate
the progress bar
simpleProgressBar.setMax(100); // 100 maximum value for the progress value
simpleProgressBar.setProgress(50); // 50 default progress value for the progress bar
```

4. progressDrawable: progress drawable is an attribute used in Android to set the custom drawable for the progress mode.

Below we set a custom gradient drawable for the progress mode of a progress bar. Before you try below code make sure to download a progress icon from the web and add in your drawable folder.

Step 1: Add this code in activity_main.xml or main.xml inside layout.

```
<ProgressBar
    android:id="@+id/simpleProgressBar"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:max="100"
    android:progress="60"
    android:layout_marginTop="100dp"
    style="@style/Widget.AppCompat.ProgressBar.Horizontal"
    android:progressDrawable="@drawable/custom_progress"/><!--custom progress drawable for
progress mode--&gt;</pre>
```

Step 2: Create a new drawable resource xml in drawable folder and name it custom_progress.

Here add the below code which creates gradient effect in progressbar.

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

    <item>
        <shape>
            <gradient
                android:endColor="#ffff"
                android:startColor="#1f1"
                android:useLevel="true" />
        </shape>
    </item>

</layer-list>
```



5. background: background attribute is used to set the background of a Progress bar. We can set a color or a drawable in the background of a Progress bar.

Below we set the black color for the background of a Progress bar.

```
<ProgressBar
    android:id="@+id/simpleProgressBar"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:max="100"
    android:progress="50"
    style="@style/Widget.AppCompat.ProgressBar.Horizontal"
    android:background="#000"/><!-- black background color for progress bar-->
```



6. indeterminate: indeterminate attribute is used in Android to enable the indeterminate mode. In this mode a progress bar shows a cyclic animation without an indication of progress. This mode is used in application when we don't know the amount of work to be done. In this mode the actual working will not be shown.

In below code we set the indeterminate to true.

```
<ProgressBar
    android:id="@+id/simpleProgressBar"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:max="100"
    android:progress="50"
    android:background="#000"
    android:padding="20dp" style="@style/Widget.AppCompat.ProgressBar.Horizontal"
    android:indeterminate="true"/><!--true value for indeterminate-->
```



Setting indeterminate of ProgressBar In Java class”

```
ProgressBar simpleProgressBar=(ProgressBar)findViewById(R.id.simpleProgressBar); // initiate  
the progress bar  
simpleProgressBar.setBackgroundColor(Color.BLACK); // black background color for the  
progress bar
```

7. padding: padding attribute is used to set the padding from left, right, top or bottom of ProgressBar.

paddingRight: set the padding from the right side of the Progress bar.

paddingLeft: set the padding from the left side of the Progress bar.

paddingTop: set the padding from the top side of the Progress bar.

paddingBottom: set the padding from the bottom side of the Progress bar.

Padding: set the padding from the all side's of the Progress bar.

Below we set the 20dp padding from all the side's of the Progress bar.

```
<ProgressBar
```

```
    android:id="@+id/simpleProgressBar"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:max="100"
    android:progress="50"
    android:background="#000"
    style="@style/Widget.AppCompat.ProgressBar.Horizontal"
    android:padding="20dp"/> <!--// 20dp padding from all the sides of the progress bar-->
```



ProgressBar Example In Android Studio:

In the first example of ProgressBar we displayed a default spinning wheel progress bar and a start button whenever a user click on the button the progress bar is displayed. Below is the final output, download code and step by step explanation:



Step 1: Create a new project and name it ProgressBarExample.

Select File -> New -> New Project... then Fill the forms and click "Finish" button.

Step 2: Open res -> layout -> activity_main.xml (or) main.xml and add following code:

In this step we open an xml file and add the code for displaying a progress bar and set its visibility to invisible and one start button.

```
<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">

    <ProgressBar
        android:id="@+id/simpleProgressBar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

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

    <Button
        android:id="@+id/startButton"
        android:layout_width="200dp"
        android:layout_height="wrap_content"
        android:text="Start"
        android:textSize="20sp"
        android:textStyle="bold"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="100dp"
        android:padding="10dp"
        android:background="#0f0"
        android:textColor="#fff"/>

</RelativeLayout>
```

Step 3:

Now Open src -> package -> MainActivity.java

In this step we open MainActivity where we add the code to initiate the progress bar & button and then perform click event on button which display the progress bar.

```
package example.gb.progressbarexample;

import android.graphics.Color;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.ProgressBar;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // initiate progress bar and start button
        final ProgressBar simpleProgressBar = (ProgressBar)
        findViewById(R.id.simpleProgressBar);
        Button startButton = (Button) findViewById(R.id.startButton);
        // perform click event on button
        startButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // visible the progress bar
                simpleProgressBar.setVisibility(View.VISIBLE);
            }
        });
    }
}
```

```
}
```

Output:

Now start the AVD in Emulator and run the App. Click on the start button and Progress Bar will be displayed on screen.



Horizontal ProgressBar Example In Android Studio:

In the second example we display a horizontal progress bar with drawable background and a start button. Here whenever a user clicks on button a thread is used to start the progress. Below is the final output, download code and step by step explanation:



Step 1: Create a new project and name it ProgressBarExample.

Select File -> New -> New Project... then Fill the forms and click "Finish" button.

Step 2: Open res -> layout -> activity_main.xml (or) main.xml and add following code:

In this step we open an xml file and add the code for displaying a horizontal progress bar by using style property, drawable progress xml and one start button.

```
<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:background="#000"
    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">

    <ProgressBar
        android:id="@+id/simpleProgressBar"
        style="@style/Widget.AppCompat.ProgressBar.Horizontal"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="70dp"
        android:max="100"
        android:progress="0"
        android:progressDrawable="@drawable/custom_progress" />

    <Button
```

```

        android:id="@+id/startButton"
        android:layout_width="200dp"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="120dp"
        android:background="#0f0"
        android:padding="10dp"
        android:text="Start"
        android:textColor="#fff"
        android:textSize="20sp"
        android:textStyle="bold" />

    </RelativeLayout>

```

Step 3:

Create an xml file in drawable -> custom_progress.xml

In this step we create a custom drawable xml for the progress bar. In this xml we create a layer list in which we create an item and then set the gradient colors for our custom progress bar.

```

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

    <item>
        <shape>
            <gradient
                android:endColor="#fff"
                android:startColor="#1f1"
                android:useLevel="true" />

        </shape>
    </item>

</layer-list>

```

Step 4:

Open app -> package -> MainActivity.java

In this step we open MainActivity and add the code to initiate the progress bar, button and then perform click event on button. After that we start the progressing in a progress bar using a thread.

```

package example.gb.progressbarexample;

import android.app.Dialog;
import android.content.DialogInterface;
import android.content.Intent;
import android.graphics.Color;
import android.provider.Settings;
import android.support.v7.app.AlertDialog;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

```

```
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.ProgressBar;
import android.widget.Button;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

    int progress = 0;
    ProgressBar simpleProgressBar;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // initiate progress bar and start button
        simpleProgressBar = (ProgressBar) findViewById(R.id.simpleProgressBar);
        Button startButton = (Button) findViewById(R.id.startButton);
        // perform click event on button
        startButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // call a function
                setProgressValue(progress);

            }
        });
    }

    private void setProgressValue(final int progress) {

        // set the progress
        simpleProgressBar.setProgress(progress);
        // thread is used to change the progress value
        Thread thread = new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                setProgressValue(progress + 10);
            }
        });
        thread.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.menu_main, menu);
        return true;
    }
}
```

```
@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
    // Handle action bar item clicks here. The action bar will  
    // automatically handle clicks on the Home/Up button, so long  
    // as you specify a parent activity in AndroidManifest.xml.  
    int id = item.getItemId();  
  
    //noinspection SimplifiableIfStatement  
    if (id == R.id.action_settings) {  
        return true;  
    }  
  
    return super.onOptionsItemSelected(item);  
}
```

Output:

Now run the App in AVD, click on start button and you will see horizontal progressbar.



TimePicker

In Android, TimePicker is a widget used for selecting the time of the day in either AM/PM mode or 24 hours mode. The displayed time consist of hours, minutes and clock format. If we need to show this view as a Dialog then we have to use a TimePickerDialog class.



TimePicker code:

```
<TimePicker
    android:id="@+id/simpleTimePicker"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:timePickerMode="spinner"/>
```

Methods of TimePicker:

Let's discuss some common methods of a time picker, which are used to configure a time picker in our application.

1. **setCurrentHour(Integer currentHour):**

This method is used to set the current hours in a time picker.

setHour(Integer hour): **setCurrentHour()** method was deprecated in API level 23. From api level 23 we have to use **setHour(Integer hour)**. In this method there is only one parameter of integer type which is used to set the value for hours.

Below we set the 5 value for the current hours.

```
TimePicker simpleTimePicker=(TimePicker)findViewById(R.id.simpleTimePicker); // initiate a  
time picker  
// set the value for current hours  
simpleTimePicker.setCurrentHour(5); // before api level 23  
simpleTimePicker.setHour(5); // from api level 23
```



2. **setCurrentMinute(Integer currentMinute):**

This method is used to set the current minutes in a time picker.

setMinute(Integer minute): setCurrentMinute() method was deprecated in API level 23. From api level 23 we have to use `setMinute(Integer minute)`. In this method there is only one parameter of integer type which set the value for minutes.

Below we set the 35 value for the current minutes.

```
TimePicker simpleTimePicker=(TimePicker)findViewById(R.id.simpleTimePicker); // initiate a  
time picker  
// set the value for current hours  
simpleTimePicker.setCurrentMinute(35); // before api level 23  
simpleTimePicker.setMinute(35); // from api level 23
```



3. **getCurrentHour()**:

This method is used to get the current hours from a time picker.

getCurrentHour(): **getCurrentHour()** method was deprecated in API level 23. From api level 23 you have to use **getHour()**. This method returns an integer value.

Below we get the value of hours from a timepicker set by user.

```
TimePicker simpleTimePicker = (TimePicker) findViewById(R.id.simpleTimePicker); // initiate a time
pickerint hours =simpleTimePicker.getCurrentHour(); // before api level 23
int hours =simpleTimePicker.getHour(); // after api level 23
```

4. **getCurrentMinute()**:

This method is used to get the current minutes from a time picker.

getMinute(): **getCurrentMinute()** method was deprecated in API level 23. From api level 23 we have to use **getMinute()**. This method returns an integer value.

Below we get the value of minutes from a time picker.

```
TimePicker simpleTimePicker = (TimePicker) findViewById(R.id.simpleTimePicker); // initiate a  
time picker  
int minutes = simpleTimePicker.getCurrentMinute(); // before api level 23  
int minutes = simpleTimePicker.getMinute(); // after api level 23
```

5. **setIs24HourView(Boolean is24HourView):**

This method is used to set the mode of the Time picker either 24 hour mode or AM/PM mode. In this method we set a Boolean value either true or false. True value indicate 24 hour mode and false value indicate AM/PM mode.

Below we set the current mode of the time picker.

```
TimePicker simpleTimePicker = (TimePicker) findViewById(R.id.simpleTimePicker); // initiate a  
time picker  
simpleTimePicker.setIs24HourView(true); // set 24 hours mode for the time picker
```



6. **is24HourView():**

This method is used to check the current mode of the time picker. This method returns true if its 24 hour mode or false if AM/PM mode is set.

Below we get the current mode of the time picker:

```
TimePicker simpleTimePicker = (TimePicker) findViewById(R.id.simpleTimePicker); // initiate a  
time picker  
Boolean mode=simpleTimePicker.is24HourView(); // check the current mode of the time picker
```

7. **setOnTimeChangedListener(TimePicker.OnTimeChangedListener onTimeChangedListener):**

This method is used to set the callback that indicates the time has been adjusted by the user. onTimeChanged(TimePicker view, int hourOfDay, int minute) is an override function of this listener in which we have three parameters first is for TimePicker, second for getting hour of the day and last is for getting the minutes after changing the time of the time picker.

Below we show the use of on time changed listener of a time picker.

```
TimePicker simpleTimePicker = (TimePicker) findViewById(R.id.simpleTimePicker); // initiate a  
time picker  
  
simpleTimePicker.setOnTimeChangedListener(new TimePicker.OnTimeChangedListener() {  
    @Override  
    public void onTimeChanged(TimePicker view, int hourOfDay, int minute) {  
        }  
});
```

Attributes of TimePicker:

Now let's we discuss about the attributes that helps us to configure a time picker in your xml file (layout).

1. **id:** id is an attribute used to uniquely identify a time picker.

```
<TimePicker  
    android:id="@+id/simpleTimePicker"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"/> <!-- id of a time picker -->
```

2. timePickerMode: time picker mode is an attribute of time picker used to set the mode either spinner or clock. Default mode is clock but this mode is no longer used after api level 21, so from api level 21 you have to set the mode to spinner.

Below we set the mode to spinner.

```
<TimePicker
    android:id="@+id/simpleTimePicker"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:timePickerMode="spinner" /> <!-- time picker mode of a time picker -->
```



timePickerMode="spinner"

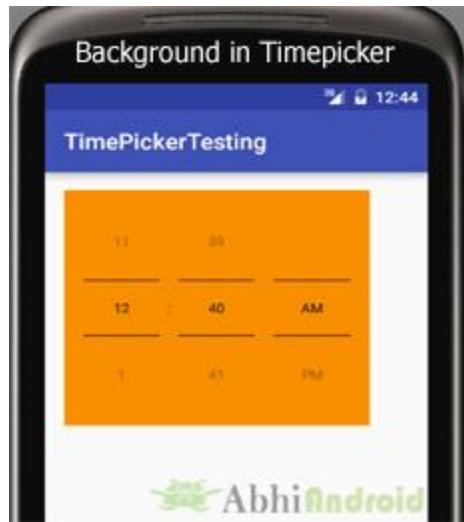
timePickerMode="clock"

3. background: background attribute is used to set the background of a time picker. We can set a color or a drawable image in the background. We can also set the background color programmatically means in java class.

Below we set the orange color for the background of a time picker.

```
<TimePicker
    android:id="@+id/simpleTimePicker"
    android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
    android:timePickerMode="spinner"
    android:background="#F88F00" /> <!-- orange background color for the time picker -->
```



Setting TimePicker Background In Java Class:

```
TimePicker simpleTimePicker=(TimePicker) findViewById(R.id.simpleTimePicker); //initiate a
time picker
simpleTimePicker.setBackgroundColor(Color.YELLOW); //Yellow background color for the
background of a time picker
```

4. padding: padding attribute is used to set the padding from left, right, top or bottom for a time picker.

paddingRight: set the padding from the right side of the time picker.

paddingLeft: set the padding from the left side of the time picker.

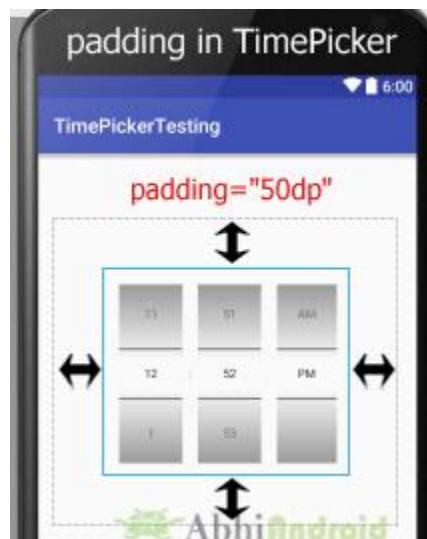
paddingTop: set the padding from the top side of the time picker.

paddingBottom: set the padding from the bottom side of the time picker.

Padding: set the padding from the all side's of the time picker.

Below example we set the 20dp padding from all the side's of the time picker.

```
<TimePicker
    android:id="@+id/simpleTimePicker"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:timePickerMode="spinner"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="50dp"
    android:padding="20dp"/> <!-- 20dp padding from all the sides of a time picker -->
```



Example of TimePicker in Android Studio:

Example 1: In the below example of time picker we will show you the use of time picker in our application. For that we display simple time picker and a textview in our xml file and perform `setOnTimeChangedListener()` event, so that whenever a user adjust the time the current displayed time of time picker is displayed by using a Toast and also displayed in the textview. Below is the final output, download project code and step by step explanation:



Step 1: Create a new project and name it TimePickerExample

Step 2: Open res -> layout -> activity_main.xml (or) main.xml and add following code:

In this step we open an xml file and add the code for displaying a time picker with spinner mode and textview for displaying time of time picker.

```
<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">

    <TimePicker
        android:id="@+id/simpleTimePicker"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="50dp"
        android:background="#090"
        android:padding="20dp"
        android:timePickerMode="spinner" />

    <TextView
        android:id="@+id/time"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
```

```
    android:text="Time Is ::"
    android:textColor="#090"
    android:textSize="20sp"
    android:textStyle="bold" />

</RelativeLayout>
```

Step 3: Open app -> package -> MainActivity.java

In this step we open MainActivity where we add the code to initiate the time picker and a text view to display time of time picker and then we perform setOnTimeChangedListener() event so whenever a user adjust the time the current displayed time of time picker is displayed by using a Toast and also displayed in the textview.

```
package example.gb.timepickerexample;

import android.app.TimePickerDialog;
import android.graphics.Color;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.TextView;
import android.widget.TimePicker;
import android.widget.Toast;

import org.w3c.dom.Text;

import java.util.Calendar;

public class MainActivity extends AppCompatActivity {

    TextView time;
    TimePicker simpleTimePicker;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // initiate the view's
        time = (TextView) findViewById(R.id.time);
        simpleTimePicker = (TimePicker) findViewById(R.id.simpleTimePicker);
        simpleTimePicker.setIs24HourView(false); // used to display AM/PM mode
        // perform set on time changed listener event
        simpleTimePicker.setOnTimeChangedListener(new TimePicker.OnTimeChangedListener() {
            @Override
            public void onTimeChanged(TimePicker view, int hourOfDay, int minute) {
                // display a toast with changed values of time picker
                Toast.makeText(getApplicationContext(), hourOfDay + " : " + minute,
                        Toast.LENGTH_SHORT).show();
                time.setText("Time is :: " + hourOfDay + " : " + minute); // set the current
            }
        });
    }
}
```

```
    });
}
```

Output:

Now run the App in AVD and you will see TimePicker on the screen. Change the time and it will be displayed as Toast and also in TextView.



Example of TimePickerDialog in Android Studio:

Example 2: In the second example of TimePicker we will show the use of time picker dialog in our application. To do that we will display edittext in our xml file and perform a click listener event on it, so whenever a user click on it time picker dialog will appear and from there user can adjust the time and after selecting the time it will be displayed in the edittext.



Step 1: Create a new project and name it TimePickerExample

Step 2: Open res -> layout -> activity_main.xml (or) main.xml and add following code:

In this step we open an xml file and add the code for displaying a edittext to display time of time picker.

```
<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">

    <EditText
        android:id="@+id/time"
        android:layout_width="200dp"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:hint="Select Time..."
        android:textColor="#090"
        android:textColorHint="#090"
        android:background="#d4d4d4"
        android:padding="15dp"
        android:textSize="20sp"/>
```

```
    android:textStyle="bold" />  
  
  </RelativeLayout>
```

Step 3: Open src -> package -> MainActivity.java

In this step we open MainActivity where we add the code to initiate the edittext to display time of time picker and perform click event on edittext, so whenever a user clicks on edittext a time picker dialog will appear from there user can set the time. And finally then the time will be displayed in the edit text.

```
package example.gb.timepickerexample;  
  
import android.app.TimePickerDialog;  
import android.graphics.Color;  
import android.support.v7.app.AppCompatActivity;  
import android.os.Bundle;  
import android.view.Menu;  
import android.view.MenuItem;  
import android.view.View;  
import android.widget.EditText;  
import android.widget.TextView;  
import android.widget.TimePicker;  
import android.widget.Toast;  
  
import org.w3c.dom.Text;  
  
import java.util.Calendar;  
  
public class MainActivity extends AppCompatActivity {  
  
    EditText time;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        // initiate the edit text  
        time = (EditText) findViewById(R.id.time);  
        // perform click event listener on edit text  
        time.setOnClickListener(new View.OnClickListener() {  
            @Override  
            public void onClick(View v) {  
                Calendar mcurrentTime = Calendar.getInstance();  
                int hour = mcurrentTime.get(Calendar.HOUR_OF_DAY);  
                int minute = mcurrentTime.get(Calendar.MINUTE);  
                TimePickerDialog mTimePicker;  
                mTimePicker = new TimePickerDialog(MainActivity.this, new  
                TimePickerDialog.OnTimeSetListener() {  
                    @Override
```

```
public void onTimeSet(TimePicker timePicker, int selectedHour,  
int selectedMinute) {  
    time.setText(selectedHour + ":" + selectedMinute);  
}  
}, hour, minute, true);//Yes 24 hour time  
mTimePicker.setTitle("Select Time");  
mTimePicker.show();  
  
});  
}  
}
```

Output:

Now run the App in AVD and you will see edittext asking user to select time. When user click on it, timepicker dialog will open from there user can select time. And then this time will be displayed in EditText.



CalendarView

In Android, Calendar View widget was added in API level 11(Android version 3.0) which means this view is only supported in the device that are running on Android 3.0 and higher version. It is used for displaying and selecting dates.



The supported range of dates of this calendar is configurable. User can select a date by taping/clicking on it and also can scroll & find the calendar to a desired date. Developer can also set minimum and maximum date shown in calendar view.

Basic Calendar View XML Code:

```
<CalendarView  
    android:id="@+id/simpleCalendarView"  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent" />
```



Important Methods Of Calendar View:

Let's we discuss some important methods of Calendar View that may be called in order to manage the CalendarView.

1. getDate(): This method is used to get the selected date of CalendarView in milliseconds since January 1, 1970 00:00:00 in user's preferred time zone. This method returns long type value for selected date.

Below we get the selected of CalendarView in milliseconds.

```
CalendarView simpleCalendarView = (CalendarView) findViewById(R.id.simpleCalendarView); //  
get the reference of CalendarView  
long selectedDate = simpleCalendarView.getDate(); // get selected date in milliseconds
```

2. setDate(long date): This method is used to set the selected date in milliseconds since January 1, 1970 00:00:00 in user's preferred time zone.

Below we set the selected date in milliseconds for a CalendarView.

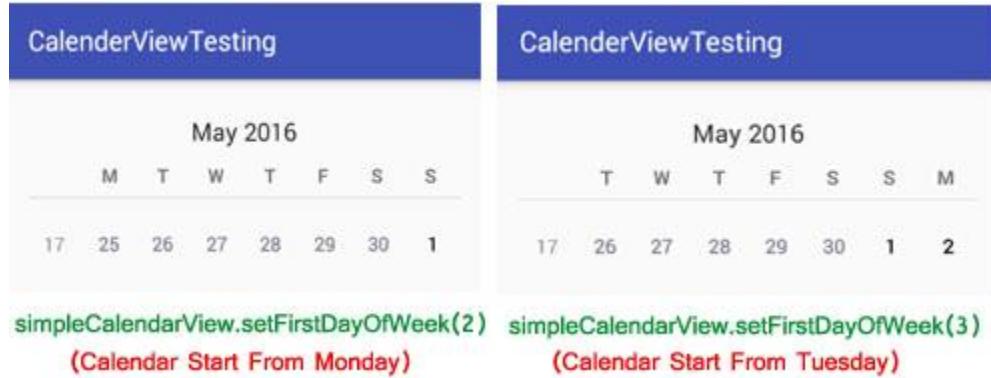
```
CalendarView simpleCalendarView = (CalendarView) findViewById(R.id.simpleCalendarView); //  
get the reference of CalendarView  
simpleCalendarView.setDate(1463918226920L); // set selected date 22 May 2016 in milliseconds
```



3. `setFirstDayOfWeek(int firstDayOfWeek)`: This method is used to set the first day of the week.

Below we set the 2 value means Monday as the first day of the week.

```
CalendarView simpleCalendarView = (CalendarView) findViewById(R.id.simpleCalendarView); //  
get the reference of CalendarView  
simpleCalendarView.setFirstDayOfWeek(2); // set Monday as the first day of the week
```



4. `getFirstDayOfWeek()`: This method is used to get the first day of week. This method returns an int type value.

Below we get the first day of the week of CalendarView.

```
CalendarView simpleCalendarView = (CalendarView) findViewById(R.id.simpleCalendarView); //  
get the reference of CalendarView  
int firstDayOfWeek= simpleCalendarView.getFirstDayOfWeek(); // get first day of the week
```

5. `setMaxDate(long maxDate)`: This method is used to set the maximal date supported by thisCalendarView in milliseconds since January 1, 1970 00:00:00 in user's preferred time zone.

Below we set the long value for maximal date supported by the CalendarView.

```
CalendarView simpleCalendarView = (CalendarView) findViewById(R.id.simpleCalendarView); //  
get the reference of CalendarView  
simpleCalendarView.setMaxDate(1463918226920L); // set 22nd May 2016 as max date supported by  
this CalendarView
```



6. getMaxDate(): This method is used to get the maximal date supported by this CalendarView in milliseconds since January 1, 1970 00:00:00 in user's preferred time zone. This method returns long type value for maximal date supported by this CalendarView.

Below we firstly set the long value for the maximal date and then get the maximal value supported by the CalendarView.

```
CalendarView simpleCalendarView = (CalendarView) findViewById(R.id.simpleCalendarView); //  
get the reference of CalendarView  
simpleCalendarView.setMaxDate(1463918226920L); // set max date supported by this  
CalendarView  
long maxDate= simpleCalendarView.getMaxDate(); // get max date supported by this  
CalendarView
```

7. setMinDate(long minDate): This method is used to to set the minimal date supported by this CalendarView in milliseconds since January 1, 1970 00:00:00 in user's preferred time zone.

Below we set the long value for minimal date supported by the CalendarView.

```
CalendarView simpleCalendarView = (CalendarView) findViewById(R.id.simpleCalendarView); //  
get the reference of CalendarView  
simpleCalendarView.setMinDate(1463918226920L); // set min date supported by this  
CalendarView
```

8. getMinDate(): This method is used to to get the minimal date supported by thisCalendarView in milliseconds since January 1, 1970 00:00:00 in user's preferred time zone. This method returns long type value for minimal date supported by this CalendarView.

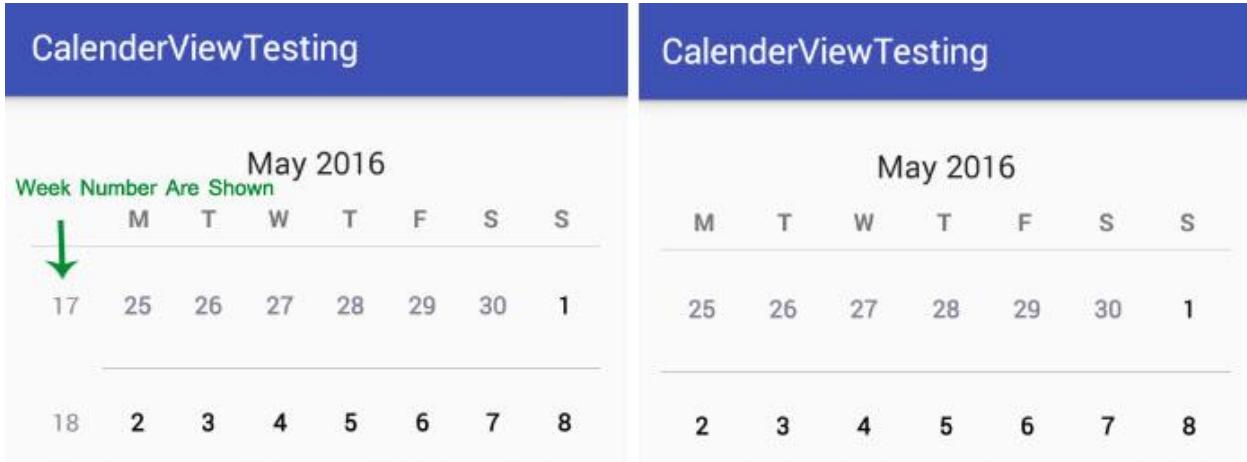
Below we firstly set the long value for the minimal date and then get the minimal value supported by the CalendarView.

```
CalendarView simpleCalendarView = (CalendarView) findViewById(R.id.simpleCalendarView); //  
get the reference of CalendarView  
simpleCalendarView.setMinDate(1463918226920L); // set min date supported by this  
CalendarViewlong minDate= simpleCalendarView.getMinDate(); // get min date supported by this  
CalendarView
```

9. setShowWeekNumber(boolean showWeekNumber): This method is used to show or hide the week number of CalendarView. In this method we set Boolean type value means true or false.

Below we set the true value for showing the week numbers of CalendarView.

```
CalendarView simpleCalendarView = (CalendarView) findViewById(R.id.simpleCalendarView); //  
get the reference of CalendarView  
simpleCalendarView.setShowWeekNumber(true); // set true value for showing the week numbers.
```



10. `getShowWeekNumber()`: This method is used to check whether the week number are shown or not. This method returns Boolean type value means true or false. True indicates week numbers are shown and false indicates week numbers are currently hidden.

Below we checks whether the week number are currently showing or not.

```
CalendarView simpleCalendarView = (CalendarView) findViewById(R.id.simpleCalendarView); //  
get the reference of CalendarView  
simpleCalendarView.getShowWeekNumber(); // checks whether the week number are shown or not.
```

11. `getSelectedDateVerticalBar()`: This method is used to get the drawable i.e used for the vertical bar shown at the beginning and at the end of the selected date. This method was deprecated in API level 23 so no longer used by Material style CalendarView.

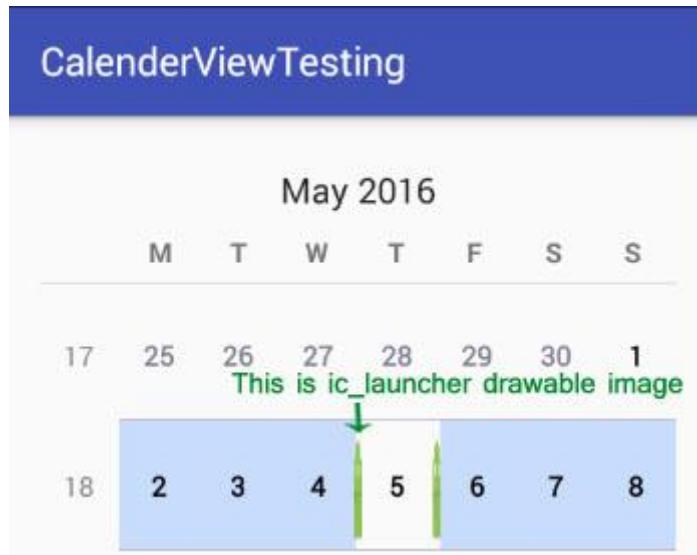
Below we get the applied drawable for the vertical bar shown at the beginning and at the end of the selected date.

```
CalendarView simpleCalendarView = (CalendarView) findViewById(R.id.simpleCalendarView); //  
get the reference of CalendarView  
Drawable verticalBar=simpleCalendarView.getSelectedDateVerticalBar(); // get the applied  
drawable for the vertical bar
```

12. setSelectedDateVerticalBar(Drawabledrawable): This method is used to set the drawable for the vertical bar shown at the beginning and at the end of the selected date. This method was deprecated in API level 23 so no longer used by Material style CalendarView.

Below we set the drawable for the vertical bar shown at the beginning and at the end of the selected date.

```
CalendarView simpleCalendarView = (CalendarView) findViewById(R.id.simpleCalendarView); //  
get the reference of CalendarView  
simpleCalendarView.setSelectedDateVerticalBar(getResources().getDrawable(R.drawable.ic_launcher)); // set the drawable for the vertical bar
```



13. setSelectedDateVerticalBar(int resourceId): This method is used to set the color for the vertical bar shown at the beginning and at the end of the selected date. This method was deprecated in API level 23 so no longer used by Material style CalendarView.

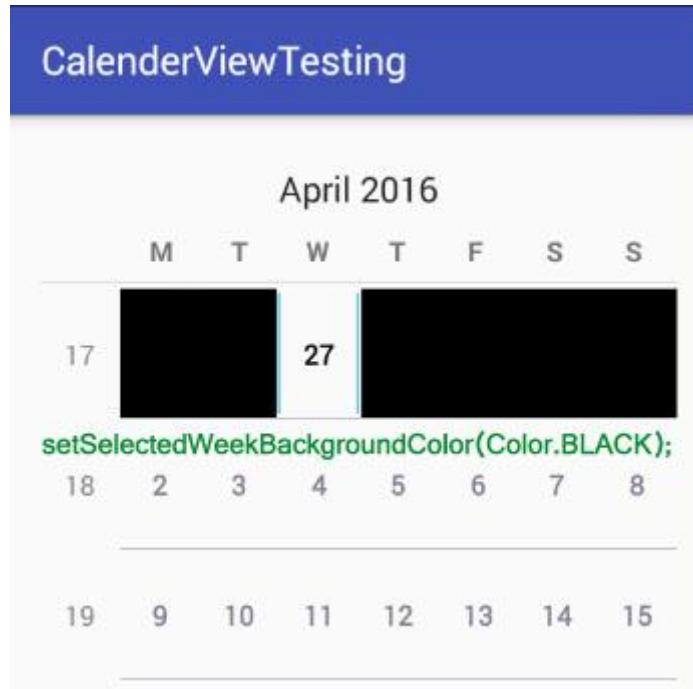
Below we set the blue color for the vertical bar shown at the beginning and at the end of the selected date.

```
CalendarView simpleCalendarView = (CalendarView) findViewById(R.id.simpleCalendarView); //  
get the reference of CalendarView  
simpleCalendarView.setSelectedDateVerticalBar(Color.BLUE); // set the color for the vertical  
bar
```

14. setSelectedWeekBackgroundColor(int color): This method is used to set the color in the background of selected week of CalendarView. This method was deprecated in API level 23 so no longer used by Material style CalendarView.

Below we set the black color in the background of selected week of CalendarView.

```
CalendarView simpleCalendarView = (CalendarView) findViewById(R.id.simpleCalendarView); //  
get the reference of CalendarView  
simpleCalendarView.setSelectedWeekBackgroundColor(Color.BLACK); // set black color in the  
background of selected week
```



15. getSelectedWeekBackgroundColor(): This method is used to get the background color of selected week of CalendarView. This method returns an int type value. This method was deprecated in API level 23 so no longer used by Material style CalendarView.

Below we get the applied color in the background of selected week of CalendarView.

```
CalendarView simpleCalendarView = (CalendarView) findViewById(R.id.simpleCalendarView); //  
get the reference of CalendarView  
int backColor=simpleCalendarView.getSelectedWeekBackgroundColor(); // get applied color in  
the background of selected week
```

16. `getWeekNumberColor()`: This method is used to get the color of week numbers. This method returns an int type value. This method was deprecated in API level 23 so no longer used by Material style CalendarView.

Below we get the applied color of week numbers.

```
CalendarView simpleCalendarView = (CalendarView) findViewById(R.id.simpleCalendarView); //  
get the reference of CalendarView  
int weekNumberColor=simpleCalendarView.getWeekNumberColor(); // get applied color of week  
number
```

17. `setWeekNumberColor(int color)`: This method is used to set the color for the week numbers. This method was deprecated in API level 23 so no longer used by Material style CalendarView.

Below we set the red color of week numbers.

```
CalendarView simpleCalendarView = (CalendarView) findViewById(R.id.simpleCalendarView); //  
get the reference of CalendarView  
simpleCalendarView.setWeekNumberColor(Color.RED); // set red color for the week number
```

18. `setWeekSeparatorLineColor(int color)`: This method is used to set the color for the week separator line. This method was deprecated in API level 23 so no longer used by Material style CalendarView.

Below we set the green color for the week separator line.

```
CalendarView simpleCalendarView = (CalendarView) findViewById(R.id.simpleCalendarView); //  
get the reference of CalendarView  
simpleCalendarView.setWeekSeparatorLineColor(Color.GREEN); // set green color for the week  
separator line.
```

19. `getWeekSeparatorLineColor()`: This method is used to get the color of week separator line. This method returns an int type value. This method was deprecated in API level 23 so no longer used by Material style Calendar View.

Below we get the applied color of week separator line.

```
CalendarView simpleCalendarView = (CalendarView) findViewById(R.id.simpleCalendarView); //  
get the reference of CalendarView  
int weekSeparatorLineColor=simpleCalendarView.getWeekSeparatorLineColor(); // get applied  
color of week separator line
```

20. `setUnfocusedMonthDateColor(int color)`: This method is used to set the color for the dates of an unfocused month. This method was deprecated in API level 23 so no longer used by Material style Calendar View.

Below we set the gray color for the dates of an unfocused month.

```
CalendarView simpleCalendarView = (CalendarView) findViewById(R.id.simpleCalendarView); //  
get the reference of CalendarView  
simpleCalendarView.setUnfocusedMonthDateColor(Color.GRAY); // set gray color for the dates  
of an unfocused month
```

21. `getUnfocusedMonthDateColor()`: This method is used to get the color for the dates of an unfocused month. This method returns int type value. This method was deprecated in API level 23 so no longer used by Material style CalendarView.

Below we get the applied color for the dates of an unfocused month.

```
CalendarView simpleCalendarView = (CalendarView) findViewById(R.id.simpleCalendarView); //  
get the reference of CalendarView  
int unfocusedMonthDateColor=simpleCalendarView.getUnfocusedMonthDateColor(); // get the  
color for the dates of an unfocused month
```

22. `setFocusedMonthDateColor(int color)`: This method is used to set the color for the dates of an focused month. This method was deprecated in API level 23 so no longer used by Material style CalendarView.

Below we set the yellow color for the dates of an focused month.

```
CalendarView simpleCalendarView = (CalendarView) findViewById(R.id.simpleCalendarView); //  
get the reference of CalendarView  
simpleCalendarView.setFocusedMonthDateColor(Color.YELLOW); // set yellow color for the dates  
of focused month
```

23. `getFocusedMonthDateColor()`: This method is used to get the color for the dates of an focused month. This method returns int type value. This method was deprecated in API level 23 so no longer used by Material style CalendarView.

Below we get the applied color for the dates of an focused month.

```
CalendarView simpleCalendarView = (CalendarView) findViewById(R.id.simpleCalendarView); //  
get the reference of CalendarView  
int focusedMonthDateColor=simpleCalendarView.getFocusedMonthDateColor(); // get the color  
for the dates of focused month
```

24. `setOnDateChangeListener(OnDateChangeListener listener)`: This method is used to set the listener to be notified upon selected date change.

Below we show how to use `setOnDateChangeListener` event of `CalendarView`.

```
CalendarView simpleCalendarView = (CalendarView) findViewById(R.id.simpleCalendarView); //  
get the reference of CalendarView  
// perform setOnDateChangeListener event on CalendarView  
simpleCalendarView.setOnDateChangeListener(new CalendarView.OnDateChangeListener() {  
@Override  
public void onSelectedDayChange(CalendarView view, int year, int month, int dayOfMonth) {  
// add code here  
}  
});
```

Attributes of CalendarView:

Now let's we discuss some common attributes of a CalendarView that helps us to configure it in our layout (xml).

1. id: id attribute is used to uniquely identify a CalendarView.

Below we set the id of the CalendarView that is used to uniquely identify it.

```
<CalendarView  
    android:id="@+id/simpleCalendarView"  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent" /> <!-- id of the CalendarView that is used to uniquely  
    identify it -->
```

2. firstDayOfWeek: This attributes is used to set the first day of week according to Calendar.

We can also set this programmatically means in java class using setFirstDayOfWeek(int firstDayOfWeek) method

Below we set the 2 value means Monday as the first day of the week.

```
<CalendarView  
    android:id="@+id/simpleCalendarView"  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent"  
    android:firstDayOfWeek="2" />  
<!-- set the 2 value means Monday as the first day of the week -->
```

3. focusedMonthDateColor: This attribute is used to set the color for the dates of the focused month. We can also set this programmatically means in java class using setFocusedMonthDateColor(int color) method.

Below we set the yellow color for the dates of an focused month.

```
<CalendarView  
    android:id="@+id/simpleCalendarView"  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent"  
    android:firstDayOfWeek="2"
```

```
    android:focusedMonthDateColor="#ff0" />
    <!-- set the yellow color for the dates of focused month -->
```

4. unfocusedMonthDateColor: This attribute is used to set the color for the dates of the unfocused month. We can also set this programmatically means in java class using setUnfocusedMonthDateColor(int color) method.

Below we set the Red color for the dates of an focused month.

```
<CalendarView
    android:id="@+id/simpleCalendarView"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:firstDayOfWeek="2"
    android:unfocusedMonthDateColor="#f00" />
    <!-- set the yellow color for the dates of an unfocused month -->
```

5. maxDate: This attribute is used to set the maximal date supported by this CalendarView. This attribute use mm/dd/yyyy format. We can also set this programmatically means in java class using setMaxDate(long maxDate)

Below we set the 05/22/2017 as maximal date supported by the CalendarView.

```
<CalendarView
    android:id="@+id/simpleCalendarView"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:firstDayOfWeek="2"
    android:maxDate="05/22/2017" />
    <!-- set maximal date supported by this CalendarView -->
```

6. minDate: This attribute is used to set the minimal date supported by this CalendarView. This attribute use mm/dd/yyyy format. We can also set this programmatically means in java class using setMinDate(long minDate)

Below we set the 05/22/2016 as minimal date supported by the CalendarView.

```
<CalendarView
    android:id="@+id/simpleCalendarView"
```

```
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:firstDayOfWeek="2"
    android:minDate="05/22/2016" />
    <!-- set minimal date supported by this CalendarView -->
```

7. selectedDateVerticalBar: This attribute is used to set the drawable/color for the vertical bar shown at the beginning and at the end of the selected date. We can also set this programmatically means in java class using setSelectedDateVerticalBar(int resourceId) or setSelectedDateVerticalBar(Drawable drawable) method.

Below we set the black color for the vertical bar shown at the beginning and at the end of the selected date.

```
<CalendarView
    android:id="@+id/simpleCalendarView"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:firstDayOfWeek="2"
    android:selectedDateVerticalBar="@color/black" />
    <!-- set black color for the vertical bar shown at the beginning and at the end of the selected date -->
```

8. selectedWeekBackgroundColor: This attribute is used to set the color in the background of selected week of CalendarView. We can also set this programmatically in java class using setSelectedWeekBackgroundColor(int color) method.

Below we set the red color in the background of selected week of CalendarView.

```
<CalendarView
    android:id="@+id/simpleCalendarView"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:firstDayOfWeek="2"
    android:selectedWeekBackgroundColor="#f00" />
    <!-- set red in the background of selected week of CalendarView -->
```

9. showWeekNumber: This attribute is used to show or hide the week number of CalendarView. In this method we set Boolean type value means true or false. We can also set

this programmatically means in java class using setShowWeekNumber(boolean showWeekNumber) method.

Below we set the false value for hiding the week numbers of CalendarView.

```
<CalendarView
    android:id="@+id/simpleCalendarView"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:firstDayOfWeek="2"
    android:showWeekNumber="false" />
    <!-- set the false value for hiding the week numbers of CalendarView. -->
```

10. weekNumberColor: This attribute is used to set the color for the week numbers.

Below we set the red color of week numbers. We can also set this programmatically means in java class using setWeekNumberColor(int color) method.

```
<CalendarView
    android:id="@+id/simpleCalendarView"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:firstDayOfWeek="2"
    android:weekNumberColor="#f00" />
    <!-- set red color for the week numbers of CalendarView. -->
```

11. weekSeparatorLineColor: This attribute is used to set the color for the week separator line. We can also set this programmatically means in java class using setWeekSeparatorLineColor(int color) method.

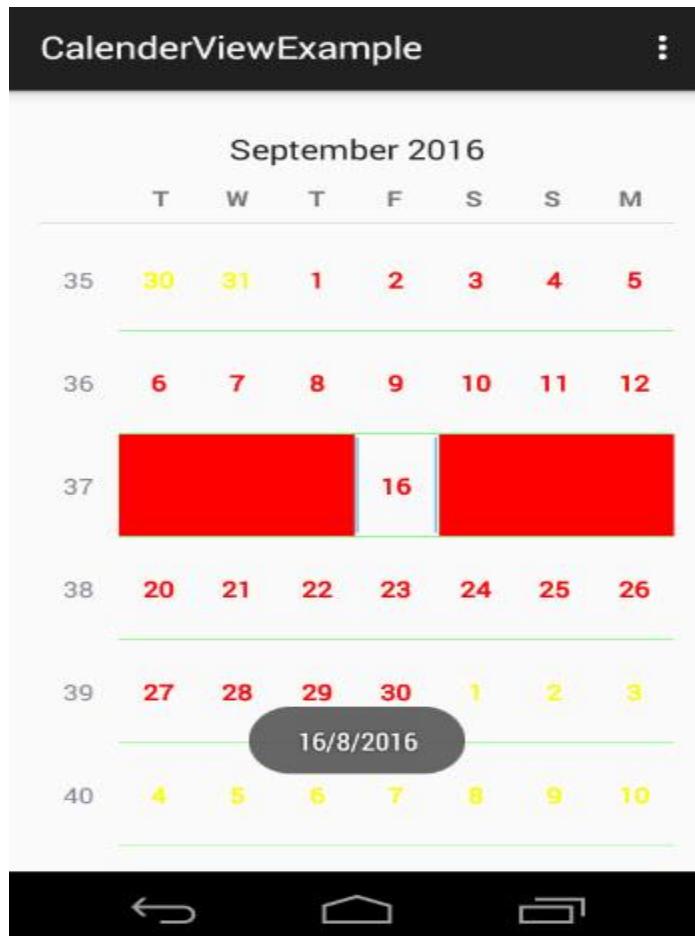
Below we set the green color for the week separator line.

```
<CalendarView
    android:id="@+id/simpleCalendarView"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:firstDayOfWeek="2"
    android:weekSeparatorLineColor="#0f0" />
    <!-- set green color for the week seperator line -->
```

Calendar View Example In Android Studio:

Below is the example of CalendarView in which we display a CalendarView for a minimal and maximal supported date. In this we set 01/01/2016 minimal and 01/01/2018 as maximal date for this calendarView. We set focused and unfocused month date color and also used some other functions and attribute for more customization of CalendarView. Finally we perform setOnDateChangeListener event to be notified upon selected date change. Whenever a user tap/click on any date the selected date will be displayed by using a Toast.

Below you can download the Android Studio project code, see final output and step by step explanation:



Step 1: Create a new project and name it CalendarViewExample

Step 2: Open res -> layout ->activity_main.xml (or) main.xml and add following code:

```
<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">

    <!-- CalendarView with monday as first day and minmal and maximal day -->
    <CalendarView
        android:id="@+id/simpleCalendarView"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:firstDayOfWeek="2"
        android:maxDate="01/01/2018"
        android:minDate="01/01/2016" />

</RelativeLayout>
```

Step 3: Open src -> package -> MainActivity.java

In this step we open MainActivity and add the code to initiate the Calendar View. In this we set selected week background color and week separator line color and finally perform setOnDateChangeListener event to be notified upon selected date change. Whenever a user tap/click on any date the selected date will be displayed by using a Toast.

```
package example.abhiandroid.calenderviewexample;

import android.graphics.Color;
import android.graphics.drawable.Drawable;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.CalendarView;
import android.widget.Toast;
```

```
public class MainActivity extends AppCompatActivity {

    CalendarView simpleCalendarView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        simpleCalendarView = (CalendarView) findViewById(R.id.simpleCalendarView); // get
        the reference of CalendarView
        simpleCalendarView.setFocusedMonthDateColor(Color.RED); // set the red color for the
        dates of focused month
        simpleCalendarView.setUnfocusedMonthDateColor(Color.BLUE); // set the yellow color
        for the dates of an unfocused month
        simpleCalendarView.setSelectedWeekBackgroundColor(Color.RED); // red color for the
        selected week's background
        simpleCalendarView.setWeekSeparatorLineColor(Color.GREEN); // green color for the
        week separator line
        // perform setDateChangeListener event on CalendarView
        simpleCalendarView.setOnDateChangeListener(new CalendarView.OnDateChangeListener() {
            @Override
            public void onSelectedDayChange(CalendarView view, int year, int month, int
dayOfMonth) {
                // display the selected date by using a toast
                Toast.makeText(getApplicationContext(), dayOfMonth + "/" + month + "/" +
year, Toast.LENGTH_LONG).show();
            }
        });
    }

}
```

Output:

Now run the app and you will see Calendar open. Now click on any date and it will displayed on the screen as Toast. Also try scrolling up and down to see the maximum and minimum date set for Calendar.

ExpandableListView

In Android, ExpandableListView is a View that shows items in a vertically scrolling two level list. Different from the listview by allowing two level groups which can individually be expanded to show its children. Each group can be expanded or collapsed individually to show or hide its children items.



ExpandableListView in Android



We can attach listeners events to the ExpandableListView to listen for OnClick or any other events on the Group or the individual children. Adapters are used to supply or control the data that will be displayed in an ExpandableListView.

Important Note: You cannot use the value wrap_content for the height attribute of a ExpandableListView in XML if the parent's size is not strictly specified. In other words we mean if the parent were ScrollView then you could not specify wrap_content since it can be of any

length. However, you can use wrap content if the ExpandableListView parent has a specific size, such as 200 pixels.

ExpandableListView code in XML:

```
<ExpandableListView  
    android:id="@+id/simpleExpandableListView"  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent"/>
```



Attributes of ExpandableListView In Android

Now let's we discuss about some important attributes that helps us to configure a ExpandableListView in XML file(layout).

1. id: id is an attribute used to uniquely identify a Expandable List View.

```
<ExpandableListView  
    android:id="@+id/simpleExpandableListView"  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent"/> <!-- id of an attribute used to uniquely identify a  
    expandable list view -->
```

2. divider: This is a drawable or color to draw between different group list items.

Below we draw red color divider between different group items.

```
<ExpandableListView  
    android:id="@+id/simpleExpandableListView"  
    android:layout_width="match_parent"  
    android:layout_height="fill_parent"  
    android:divider="#f00"  
    android:dividerHeight="1dp" /> <!-- red color divider with 1dp height between the  
    groups items of expandable list view -->
```



3. dividerHeight: This specify the height of the divider between group list items. This could be in dp (density pixel), sp(scale independent pixel) or px (pixel).

In above example of divider we also set the divider height 1dp between the list items. The height should be in dp, sp or px.

4. listSelector: This property is used to set the selector of the expandable list View. It is generally orange or Sky blue color mostly but you can also define your own custom color or an image as a list selector as per your design.

Below selector color is green, when you select any list item then that item's background color is green.

```
<ExpandableListView
    android:id="@+id/simpleExpandableListView"
    android:layout_width="match_parent"
    android:layout_height="fill_parent"
    android:divider="#f00"
    android:dividerHeight="1dp"
    android:listSelector="#0f0" /> <!-- green color for the list selector item -->
```



5. childDivider: This is a drawable or color to draw between different child list items of a expandable list view.

Below we draw green color divider between different child items of a group.

```
<ExpandableListView
    android:id="@+id/simpleExpandableListView"
    android:layout_width="match_parent"
    android:layout_height="fill_parent"
    android:divider="#f00"
    android:dividerHeight="1dp"
    android:childDivider="#0f0" /> <!-- green color divider between the child items of
expandable list view -->
```

The below image is from the ExpandableListView example which is explained at the end of this post. In this we have set green color as Child divider and red color as divider. The reason we have used this example image because we need fill data using Adapter to show you childDivider attribute in action.



6. padding: padding attribute is used to set the padding from left, right, top or bottom.

paddingRight: set the padding from the right side of the expandable list view.

paddingLeft: set the padding from the left side of the Progress bar.

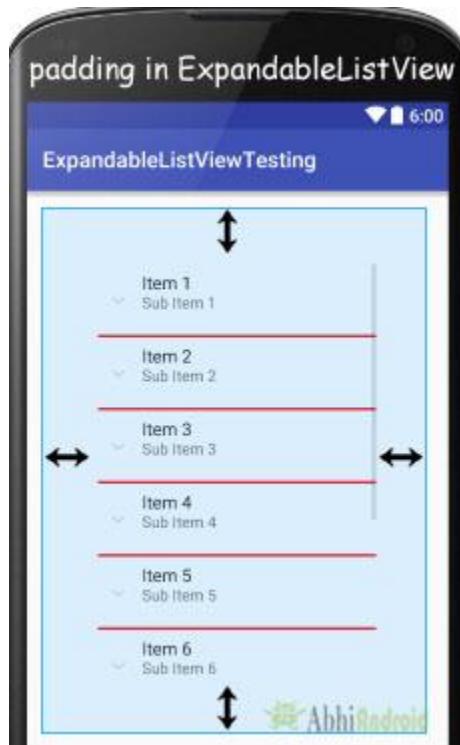
paddingTop: set the padding from the top side of the expandable list view.

paddingBottom: set the padding from the bottom side of the expandable list view.

Padding: set the padding from the all side's of the expandable list view.

Below we set the 50dp padding from all the side's of the expandable list view.

```
<ExpandableListView  
    android:id="@+id/simpleExpandableListView"  
    android:layout_width="match_parent"  
    android:layout_height="fill_parent"  
    android:divider="#f00"  
    android:dividerHeight="2dp"  
    android:childDivider="#0f0"  
    android:padding="50dp" /> <!-- 50 dp padding from all the sides of a expandable list view -->
```



Adapter Used In ExpandableListView In Android:

An adapter is a bridge between UI component and data source that helps us to fill data in UI component. It holds the data and send the data to Adapter view then view can takes the data from the adapter view and shows the data on different views like as ExpandableListView or other Views. The implementation of this interface will provide access to the data of the children (categorized by groups), and also instantiate views for the children and groups.

In Android for supplying data in an ExpandableListView following adapters are used.

1. [ExpandableListAdapter](#)
2. [BaseExpandableListAdapter](#)
3. [SimpleExpandableListAdapter](#)

Now we explain these three adapters in detail:

1. ExpandableListAdapter:

ExpandableListAdapter is an Adapter that links a ExpandableListView with the underlying data. The implementation of this interface will provide the data for the children and also initiate the views for the children and groups. For customization of list we need to implement ExpandableListAdapter in our custom adapter.

Below is the example code of ExpandableListAdapter in which we create CustomAdapter class and then implements ExpandableListAdapter in that class.

```
public class CustomAdapter implements ExpandableListAdapter {  
    @Override  
    public void registerDataSetObserver(DataSetObserver observer) {  
    }  
  
    @Override  
    public void unregisterDataSetObserver(DataSetObserver observer) {  
    }  
  
    @Override  
    public int getGroupCount() {  
        return 0;  
    }  
  
    @Override  
    public int getChildrenCount(int groupPosition) {  
        return 0;  
    }  
  
    @Override  
    public Object getGroup(int groupPosition) {  
        return null;  
    }  
  
    @Override  
    public Object getChild(int groupPosition, int childPosition) {  
        return null;  
    }  
  
    @Override  
    public long getGroupId(int groupPosition) {  
        return 0;  
    }
```

```
}

@Override
public long getChildId(int groupPosition, int childPosition) {
    return 0;
}

@Override
public boolean hasStableIds() {
    return false;
}

@Override
public View getGroupView(int groupPosition, boolean isExpanded, View convertView, ViewGroup parent) {
    return null;
}

@Override
public View getChildView(int groupPosition, int childPosition, boolean isLastChild, View convertView, ViewGroup parent) {
    return null;
}

@Override
public boolean isChildSelectable(int groupPosition, int childPosition) {
    return false;
}

@Override
public boolean areAllItemsEnabled() {
    return false;
}

@Override
public boolean isEmpty() {
    return false;
}

@Override
public void onGroupExpanded(int groupPosition) {
}

@Override
public void onGroupCollapsed(int groupPosition) {
}

@Override
public long getCombinedChildId(long groupId, long childId) {
    return 0;
}

@Override
public long getCombinedGroupId(long groupId) {
    return 0;
}
```

```
}
```

Read [ExpandableListAdapter Tutorial With Example In Android Studio](#) for more details.

2. BaseExpandableListAdapter:

BaseExpandableListAdapter is a base class for the expandable list adapter used to provide data and Views from some data to ExpandableListView. For Creating a custom ExpandableListView we need to create a custom class and then extends BaseExpandableListAdapter class in that class.

Below is an example code of BaseExpandableListAdapter in which we create custom adapter class and then extends BaseExpandableListAdapter in that class.

```
public class CustomAdapter extends BaseExpandableListAdapter {

    @Override
    public int getGroupCount() {
        return 0;
    }

    @Override
    public int getChildrenCount(int groupPosition) {
        return 0;
    }

    @Override
    public Object getGroup(int groupPosition) {
        return null;
    }

    @Override
    public Object getChild(int groupPosition, int childPosition) {
        return null;
    }

    @Override
    public long getGroupId(int groupPosition) {
        return 0;
    }

    @Override
    public long getChildId(int groupPosition, int childPosition) {
        return 0;
    }

    @Override
```

```

public boolean hasStableIds() {
    return false;
}

@Override
public View getGroupView(int groupPosition, boolean isExpanded, View convertView, ViewGroup parent) {
    return null;
}

@Override
public View getChildView(int groupPosition, int childPosition, boolean isLastChild, View convertView, ViewGroup parent) {
    return null;
}

@Override
public boolean isChildSelectable(int groupPosition, int childPosition) {
    return false;
}
}

```

Read [BaseExpandableListAdapter With Example In Android Studio](#) for explanation of all these function.

3. SimpleExpandableListAdapter:

SimpleExpandableListAdapter is an adapter that is used to map the static data to group and child views defined in our XML (layout) file. We can separately specify the data backing to the group as a List of Maps. Each entry in a ArrayList corresponds to one group in the Expandable List. The maps contains the data for each row. We can also specify an XML file that defines the views used to display a group, and a mapping from keys in the Map to specific views. This process is similar for a child, except it is one level deeper so the data backing is specified as a List<list>, where the first List is corresponds to the group of the child and the second List corresponds to the position of the child within that group, and finally the Map holds the data for the particular child.

```

public SimpleExpandableListAdapter (Context context, List<? extends Map<String, ?>>
groupData, int groupLayout, String[]groupFrom, int[] groupTo, List<? extends List<? extends
Map<String, ?>>> childData, int childLayout, String[] childFrom, int[] childTo)

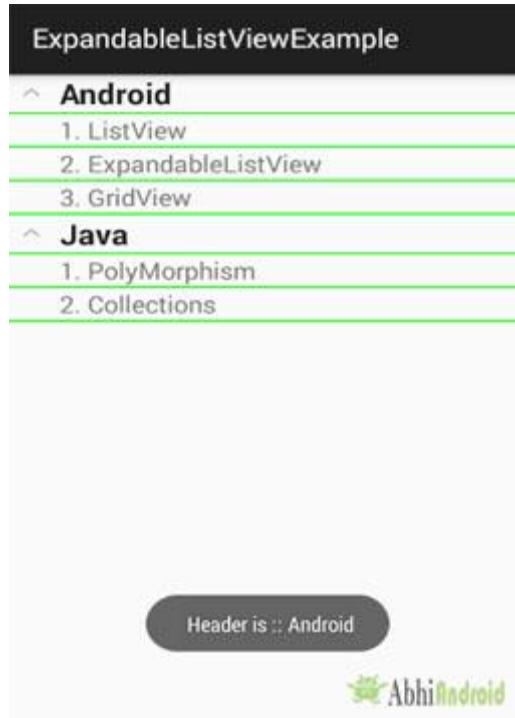
```

Read [SimpleExpandableListAdapter tutorial](#) for explanation of all these parameter.

ExpandableListView using BaseExpandableListAdapter Example In Android Studio

Below is the example of ExpandableListView in android where we display an expandable list with subject name and their topics. In this example we display subject names as Group items and their topic names as child items for a particular group. In this we implement setOnChildClickListener() and setOnGroupClickListener() events and whenever a user clicks on a child or a group item the name of the item is displayed by using a Toast.

Below you can download code, see final output and step by step explanation of Example in Android Studio:



Step 1: Create a new project and name it ExpandableListViewExample.

Step 2: Open res -> layout ->activity_main.xml (or) main.xml and add following code:

In this step we open an XML file and add the code for displaying a ExpandableListView by using its different attributes.

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

    <ExpandableListView
        android:id="@+id/simpleExpandableListView"
        android:layout_width="match_parent"
        android:layout_height="fill_parent"
        android:divider="#f00"
        android:childDivider="#0f0"
        android:dividerHeight="1dp" />

</RelativeLayout>
```

Step 3: Create a new xml file for group items Open res -> layout -> group_items.xml and add following code:

In this step we add the code for displaying a TextView subject names.

```
<?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="55dip"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/heading"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:paddingLeft="35sp"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:textStyle="bold" />

</LinearLayout>
```

Step 4: Create a new xml file for group items Open res -> layout -> child_items.xml and add following code:

In this step we add the code for displaying two TextView i.e. one for sequence of topics and another for topic name

```

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

    <TextView
        android:id="@+id/sequence"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:paddingLeft="35sp"
        android:textAppearance="?android:attr/textAppearanceMedium" />

    <TextView
        android:id="@+id/childItem"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_toRightOf="@+id/sequence"
        android:textAppearance="?android:attr/textAppearanceMedium" />

</RelativeLayout>

```

Step 5: Open src -> package -> MainActivity.Java

In this step we open MainActivity and add the code to initiate the ExpandableListView and add the data to lists for displaying in an ExpandableListView using model classes and then set the adapter which fills the data in the ExpandableListView. In this we implement setOnChildClickListener() and setOnGroupClickListener() events. Whenever a user clicks on a child or a group item the name of the item is display by using a Toast.

```

package example.abhiandroid.expandablelistviewexample;

import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.ExpandableListView;
import android.widget.Toast;

import java.util.ArrayList;
import java.util.LinkedHashMap;

public class MainActivity extends AppCompatActivity{

    private LinkedHashMap<String, GroupInfo> subjects = new LinkedHashMap<String,
    GroupInfo>();
    private ArrayList<GroupInfo> deptList = new ArrayList<GroupInfo>();

```

```
private CustomAdapter listAdapter;
private ExpandableListView simpleExpandableListView;

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

    // add data for displaying in expandable list view
    loadData();

    //get reference of the ExpandableListView
    simpleExpandableListView = (ExpandableListView)
        findViewById(R.id.simpleExpandableListView);
    // create the adapter by passing your ArrayList data
    listAdapter = new CustomAdapter(MainActivity.this, deptList);
    // attach the adapter to the expandable list view
    simpleExpandableListView.setAdapter(listAdapter);

    //expand all the Groups
    expandAll();

    // setOnChildClickListener listener for child row click
    simpleExpandableListView.setOnChildClickListener(new
        ExpandableListView.OnChildClickListener() {
            @Override
            public boolean onChildClick(ExpandableListView parent, View v, int
                groupPosition, int childPosition, long id) {
                //get the group header
                GroupInfo headerInfo = deptList.get(groupPosition);
                //get the child info
                ChildInfo detailInfo = headerInfo.getProductList().get(childPosition);
                //display it or do something with it
                Toast.makeText(getApplicationContext(), " Clicked on :: " + headerInfo.getName()
                    + "/" + detailInfo.getName(), Toast.LENGTH_LONG).show();
                return false;
            }
        });
    // setOnGroupClickListener listener for group heading click
    simpleExpandableListView.setOnGroupClickListener(new
        ExpandableListView.OnGroupClickListener() {
            @Override
            public boolean onGroupClick(ExpandableListView parent, View v, int
                groupPosition, long id) {
                //get the group header
                GroupInfo headerInfo = deptList.get(groupPosition);
                //display it or do something with it
                Toast.makeText(getApplicationContext(), " Header is :: " + headerInfo.getName(),
                    Toast.LENGTH_LONG).show();

                return false;
            }
        });
}
```

```
//method to expand all groups
private void expandAll() {
    int count = listAdapter.getGroupCount();
    for (int i = 0; i < count; i++){
        simpleExpandableListView.expandGroup(i);
    }
}

//method to collapse all groups
private void collapseAll() {
    int count = listAdapter.getGroupCount();
    for (int i = 0; i < count; i++){
        simpleExpandableListView.collapseGroup(i);
    }
}

//load some initial data into our list
private void loadData(){

    addProduct("Android","ListView");
    addProduct("Android","ExpandableListView");
    addProduct("Android","GridView");

    addProduct("Java","PolyMorphism");
    addProduct("Java","Collections");
}

//here we maintain our products in various departments
private int addProduct(String department, String product){

    int groupPosition = 0;

    //check the hash map if the group already exists
    GroupInfo headerInfo = subjects.get(department);
    //add the group if doesn't exists
    if(headerInfo == null){
        headerInfo = new GroupInfo();
        headerInfo.setName(department);
        subjects.put(department, headerInfo);
        deptList.add(headerInfo);
    }

    //get the children for the group
    ArrayList<ChildInfo> productList = headerInfo.getProductList();
    //size of the children list
    int listSize = productList.size();
    //add to the counter
    listSize++;

    //create a new child and add that to the group
    ChildInfo detailInfo = new ChildInfo();
    detailInfo.setSequence(String.valueOf(listSize));
    detailInfo.setName(product);
    productList.add(detailInfo);
}
```

```
        headerInfo.setProductList(productList);

        //find the group position inside the list
        groupPosition = deptList.indexOf(headerInfo);
        return groupPosition;
    }
}
```

Step 6: Create a New Class Open -> package – > GroupInfo.Java and add the following code.

In this step, we create a class for setting and getting the group item name and child items info according to a particular group. GroupInfo is a model class used to set the name of the group item and child items information from your main activity and then get the information within Adapter class. Finally set the value to ExpandableListView.

```
package example.abhiandroid.expandablelistviewexample;

import java.util.ArrayList;

public class GroupInfo {

    private String name;
    private ArrayList<ChildInfo> list = new ArrayList<ChildInfo>();

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public ArrayList<ChildInfo> getProductList() {
        return list;
    }

    public void setProductList(ArrayList<ChildInfo> productList) {
        this.list = productList;
    }
}
```

Step 7: Create a New Class Open -> package – > ChildInfo.Java and add the following code.

In this step, we create a class for setting and getting the name and sequence for the child items. ChildInfo is a model class used to set the name of the child item and the sequence of the child item from your main activity and then get the name and sequence within adapter class. Finally set the value to expandable list view.

```
package example.abhiandroid.expandablelistviewexample;

public class ChildInfo {

    private String sequence = "";
    private String name = "";

    public String getSequence() {
        return sequence;
    }

    public void setSequence(String sequence) {
        this.sequence = sequence;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

Step 8: Create a New Class Open -> package – > CustomAdapter.Java and add the following code.

In this step, we create a CustomAdapter class and then extends BaseExpandableListAdapter in that class. Finally set the data in the ExpandableListView from GroupInfo and ChildInfo model class.

```
package example.abhiandroid.expandablelistviewexample;

import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseExpandableListAdapter;
import android.widget.TextView;
```

```
import java.util.ArrayList;

/**
 * Created by Gourav on 08-03-2016.
 */
public class CustomAdapter extends BaseExpandableListAdapter {

    private Context context;
    private ArrayList<GroupInfo> deptList;

    public CustomAdapter(Context context, ArrayList<GroupInfo> deptList) {
        this.context = context;
        this.deptList = deptList;
    }

    @Override
    public Object getChild(int groupPosition, int childPosition) {
        ArrayList<ChildInfo> productList = deptList.get(groupPosition).getProductList();
        return productList.get(childPosition);
    }

    @Override
    public long getChildId(int groupPosition, int childPosition) {
        return childPosition;
    }

    @Override
    public View getChildView(int groupPosition, int childPosition, boolean isLastChild,
                           View view, ViewGroup parent) {

        ChildInfo detailInfo = (ChildInfo) getChild(groupPosition, childPosition);
        if (view == null) {
            LayoutInflater infalInflater = (LayoutInflater)
context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
            view = infalInflater.inflate(R.layout.child_items, null);
        }

        TextView sequence = (TextView) view.findViewById(R.id.sequence);
        sequence.setText(detailInfo.getSequence().trim() + ". ");
        TextView childItem = (TextView) view.findViewById(R.id.childItem);
        childItem.setText(detailInfo.getName().trim());

        return view;
    }

    @Override
    public int getChildrenCount(int groupPosition) {

        ArrayList<ChildInfo> productList = deptList.get(groupPosition).getProductList();
        return productList.size();
    }

    @Override
    public Object getGroup(int groupPosition) {
```

```
        return deptList.get(groupPosition);
    }

    @Override
    public int getGroupCount() {
        return deptList.size();
    }

    @Override
    public long getGroupId(int groupPosition) {
        return groupPosition;
    }

    @Override
    public View getGroupView(int groupPosition, boolean isLastChild, View view,
                           ViewGroup parent) {

        GroupInfo headerInfo = (GroupInfo) getGroup(groupPosition);
        if (view == null) {
            LayoutInflater inf = (LayoutInflater)
context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
            view = inf.inflate(R.layout.group_items, null);
        }

        TextView heading = (TextView) view.findViewById(R.id.heading);
        heading.setText(headerInfo.getName().trim());

        return view;
    }

    @Override
    public boolean hasStableIds() {
        return true;
    }

    @Override
    public boolean isChildSelectable(int groupPosition, int childPosition) {
        return true;
    }
}
```

Output:

Now run the App and you will see main topics and sub-topics listed in ExpandableListView.

Chronometer

In Android, Chronometer is a class that implements a simple timer. Chronometer is a subclass of TextView. This class helps us to add a timer in our app.



You can give Timer start time in the `elapsedRealTime()` timebase and it start counting from that. If we don't give base time then it will use the time at which time we call `start()` method. By default a chronometer displays the current timer value in the form of MM:SS or H:MM:SS. We can set our own format into an arbitrary string.

Chronometer code in XML:

```
<Chronometer  
    android:id="@+id/simpleChronometer"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content" />
```

Methods Of Chronometer In Android:

Let's discuss some important methods of chronometer that may be called in order to manage the chronometer.

1. start(): start function of chronometer is used to start the counting up.

Below we start the counting up of a chronometer.

```
Chronometer simpleChronometer = (Chronometer) findViewById(R.id.simpleChronometer); //  
initiate a chronometer  
  
simpleChronometer.start(); // start a chronometer
```



2. stop(): stop function of chronometer is used to stop the counting up.

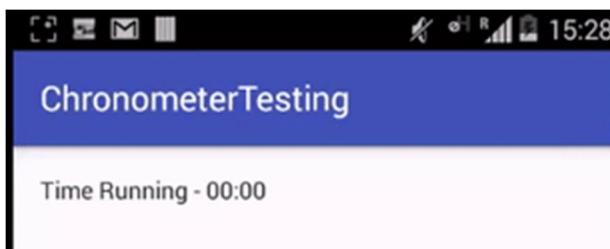
Below code stop the counting of a chronometer.

```
Chronometer simpleChronometer = (Chronometer) findViewById(R.id.simpleChronometer); //  
initiate a chronometer  
  
simpleChronometer.stop(); // stop a chronometer
```

3. setFormat(String format): set format function of chronometer is used to set the format string used to display. In other words we can say it is used to display text, numbers etc along-with chronometer.

In the below example code we set the string format for displaying a chronometer.

```
Chronometer simpleChronometer = (Chronometer) findViewById(R.id.simpleChronometer); //  
initiate a chronometer  
simpleChronometer.start(); // start a chronometer  
simpleChronometer.setFormat("Time Running - %s"); // set the format for a chronometer
```



4. getformat(): This function of chronometer is used for getting the current format string. This methods returns a string type value.

Below we get the current format string from a chronometer.

```
simpleChronometer = (Chronometer) findViewById(R.id.simpleChronometer); // initiate a  
chronometer  
  
String formatType=simpleChronometer.getFormat(); // get the format from a chronometer
```

5. setOnChronometerTickListener(Chronometer.OnChronometerTickListener listener): This is a listener event which is automatically called when the chronometer changes.

Below we show the use of setOnChronometerTickListener() for a chronometer.

```
Chronometer simpleChronometer = (Chronometer) findViewById(R.id.simpleChronometer); //  
initiate a chronometer  
// perform set on chronometer tick listener event of chronometer  
  
simpleChronometer.setOnChronometerTickListener(new Chronometer.OnChronometerTickListener() {  
  
    @Override  
  
    public void onChronometerTick(Chronometer chronometer) {  
  
        // do something when chronometer changes  
    }  
  
});
```

6. setBase(long base): set base function of chronometer is used to set the time that count up time is in reference to. You can give it a start time in the elapsedRealtime() timebase, and it counts up from that, or if you don't give it a base time, it will use the time at which you call start().

`SystemClock.elapsedRealtime()` is the number of milliseconds since the device was turned on.

Below we set the base time for a chronometer.

```
Chronometer simpleChronometer = (Chronometer) findViewById(R.id.simpleChronometer); //  
initiate a chronometer  
  
simpleChronometer.setBase(SystemClock.elapsedRealtime()); // set base time for a chronometer
```

7. getBase(): get base function is used to get the base time from a chronometer. This method returns a base time as set through the setBase() function. This method return long value.

Below we get the base time from a chronometer.

```
Chronometer simpleChronometer = (Chronometer) findViewById(R.id.simpleChronometer);  
// initiate a chronometer  
  
long base=simpleChronometer.getBase(); // get base time from a chronometer
```

Attributes of Chronometer In Android:

Now let's we discuss some common attributes of a chronometer that helps us to configure it in our layout (xml).

1. Id: id attribute is used to uniquely identify a chronometer.

```
<Chronometer  
    android:id="@+id/simpleChronometer"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content" />
```

2. gravity: The gravity attribute is an optional attribute which is used to control the alignment of the text like left, right, center, top, bottom, center_vertical, center_horizontal etc.

Below we set the center_horizontal gravity for text of a chronometer.

```
<Chronometer  
    android:id="@+id/simpleChronometer"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:gravity="center_horizontal" /><!-- center horizontal gravity for the text of  
chronometer -->
```



3. textColor: textColor attribute is used to set the text color of chronometer. Color value is in the form of "#argb", "#rgb", "#rrggbb", or "#aarrggbb".

Below we set the red color for the displayed text of a chronometer.

```
<Chronometer  
    android:id="@+id/simpleChronometer"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:gravity="center_horizontal"  
    android:textColor="#f00"/><!-- red color for the text of chronometer -->
```



Setting textColor of Chronometer In Java class:

```
Chronometer simpleChronometer = (Chronometer) findViewById(R.id.simpleChronometer);  
// initiate a chronometer  
  
simpleChronometer.setTextColor(Color.RED); // set red color for the displayed text of a  
chronometer
```

4. textSize: textSize attribute is used to set the size of text of chronometer. We can set the text size in sp (scale independent pixel) or dp (density pixel).

Below we set the 25sp size for the text of a chronometer.

```
<Chronometer  
    android:id="@+id/simpleChronometer"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:gravity="center_horizontal"  
    android:textColor="#f00"  
    android:textSize="25sp"/><!-- 25 sp size for displayed text of chronometer -->
```



Setting textSize of Chronometer In Java class:

```
Chronometer simpleChronometer = (Chronometer) findViewById(R.id.simpleChronometer); //  
initiate a chronometer  
  
simpleChronometer.setTextSize(25); // set text size of a chronometer
```

5. textStyle: textStyle attribute is used to set the text style of a chronometer. The possible text styles are bold, italic and normal. If we need to use two or more styles for a chronometer text then “|” operator is used for that.

Below we set the bold and italic text styles for text of a chronometer.

```
<Chronometer  
    android:id="@+id/simpleChronometer"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_centerHorizontal="true"  
    android:textColor="#f00"  
    android:textSize="25sp"  
    android:textStyle="bold|italic"/><!-- bold and italic style for displayed text of  
chronometer -->
```



6. background: background attribute is used to set the background of a chronometer. You can set a color or a drawable in the background of a text view. You can also set the background color programmatically means in java class.

Below we set the green color for the background of a chronometer.

```
<Chronometer  
    android:id="@+id/simpleChronometer"  
    android:layout_width="50dp"  
    android:layout_height="wrap_content"  
    android:layout_centerHorizontal="true"  
    android:textColor="#f00"  
    android:textSize="25sp"  
    android:textStyle="bold|italic"  
    android:background="#0f0"/><!-- green background color of chronometer -->
```



Setting background of Chronometer In Java class:

```
Chronometer simpleChronometer = (Chronometer) findViewById(R.id.simpleChronometer); //  
initiate a chronometer  
  
simpleChronometer.setBackgroundColor(Color.GREEN); // set green color for the background of  
a chronometer
```

7. padding: padding attribute is used to set the padding from left, right, top or bottom.

paddingRight : set the padding from the right side of the chronometer.

paddingLeft : set the padding from the left side of the chronometer.

paddingTop : set the padding from the top side of the chronometer.

paddingBottom : set the padding from the bottom side of the

Padding : set the padding from the all side's of the chronometer.

Below we set 20dp padding from the top of a chronometer.

```
<Chronometer  
  
    android:id="@+id/simpleChronometer"  
    android:layout_width="50dp"  
    android:layout_height="wrap_content"  
    android:layout_centerHorizontal="true"  
    android:textColor="#f00"  
    android:textSize="25sp"  
    android:textStyle="bold|italic"  
    android:background="#0f0"  
    android:paddingTop="20dp"/><!-- 20 dp padding from the top of a chronometer -->
```



8. drawableBottom: drawableBottom is the drawable to be drawn to the below of the text of chronometer.

9. drawableTop: drawableTop is the drawable to be drawn to the top of the text of a chronometer.

10. drawableRight: drawableRight is the drawable to be drawn to the right side of the text of a chronometer.

11. drawableLeft: drawableLeft is the drawable to be drawn to the left side of the text of a chronometer.

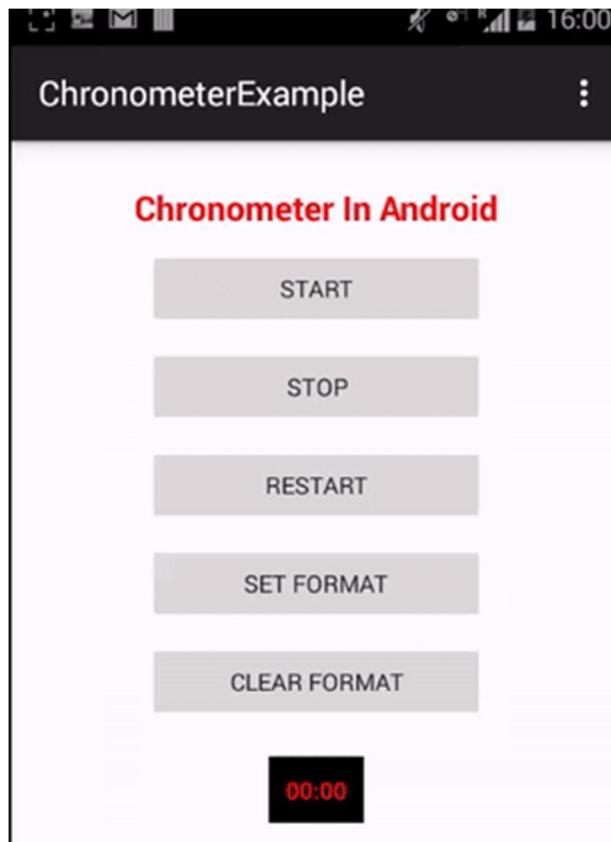
Below we set the icon to the left side of the text of chronometer. Make sure to save ic_launcher image in drawable folder.

```
<Chronometer  
    android:id="@+id/simpleChronometer"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_centerHorizontal="true"  
    android:textColor="#f00"  
    android:textSize="25sp"  
    android:textStyle="bold|italic"  
    android:background="#0f0"  
    android:gravity="center"  
    android:padding="10dp"  
    android:drawableLeft="@drawable/ic_launcher"/><!-- drawable left of a chronometer -->
```



Chronometer Example in Android Studio:

In the example of chronometer we display chronometer with five buttons and perform click events on that buttons. Buttons are used to start, stop, restart, setFormat and clearFormat of chronometer. Below is the final output, download project code and step by step explanation.



Step 1: Create a new project and name it ChronometerExample.

Step 2: Open res -> layout -> activity_main.xml (or) main.xml and add following code:

In this step we open xml file and add the code for displaying a chronometer with five buttons to perform operations on chronometer.

```
<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:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="10dp"
        android:text="@string/chronometerInAndroid"
        android:textColor="#FF0000"
        android:textSize="20dp"
        android:textStyle="bold" />

    <Button
        android:id="@+id/startButton"
        android:layout_width="200dp"
        android:layout_height="wrap_content"
        android:layout_below="@+id/textView1"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="10dp"
        android:text="@string/start" />

    <Button
        android:id="@+id/stopButton"
        android:layout_width="200dp"
        android:layout_height="wrap_content"
        android:layout_below="@+id/startButton"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="10dp"
        android:text="@string/stop" />

    <Button
        android:id="@+id/restartButton"
        android:layout_width="200dp"
        android:layout_height="wrap_content"
        android:layout_below="@+id/stopButton"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="10dp"
        android:text="@string/restart" />

    <Button
        android:id="@+id/setFormat"
        android:layout_width="200dp"
        android:layout_height="wrap_content"
        android:layout_below="@+id/restartButton"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="10dp"
        android:text="@string/setFormat" />

    <Button
        android:id="@+id/clearFormat"
        android:layout_width="200dp"
        android:layout_height="wrap_content"
        android:layout_below="@+id/setFormat"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="10dp"
        android:text="@string/clearFormat" />
    <!-- chronometer with black background and red text color -->
    <Chronometer
        android:id="@+id/simpleChronometer"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
    android:layout_below="@+id/clearFormat"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="20dp"
    android:background="#000"
    android:gravity="center"
    android:padding="10dp"
    android:textColor="#f00"
    android:textStyle="bold" />

</RelativeLayout>
```

Step 3: Open src -> package -> MainActivity.java

In this step we open MainActivity and add the code to initiate the chronometer, five button and then perform on click listener event on buttons to perform start, stop and other operations on chronometer as discussed earlier in this post.

```
package example.abhiandroid.chronometerexample;

import android.os.Bundle;
import android.os.SystemClock;
import android.support.v7.app.AppCompatActivity;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.Chronometer;

public class MainActivity extends AppCompatActivity {

    Chronometer simpleChronometer;
    Button start, stop, restart, setFormat, clearFormat;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // initiate views
        simpleChronometer = (Chronometer) findViewById(R.id.simpleChronometer);
        start = (Button) findViewById(R.id.startButton);
        stop = (Button) findViewById(R.id.stopButton);
        restart = (Button) findViewById(R.id.restartButton);
        setFormat = (Button) findViewById(R.id.setFormat);
        clearFormat = (Button) findViewById(R.id.clearFormat);
        // perform click event on start button to start a chronometer
        start.setOnClickListener(new View.OnClickListener() {

            @Override
            public void onClick(View v) {
                // TODO Auto-generated method stub
            }
        });
    }
}
```

```
        simpleChronometer.start();
    }
});

// perform click event on stop button to stop the chronometer
stop.setOnClickListener(new View.OnClickListener() {

    @Override
    public void onClick(View v) {
        // TODO Auto-generated method stub

        simpleChronometer.stop();
    }
});

// perform click event on restart button to set the base time on chronometer
restart.setOnClickListener(new View.OnClickListener() {

    @Override
    public void onClick(View v) {
        // TODO Auto-generated method stub

        simpleChronometer.setBase(SystemClock.elapsedRealtime());
    }
});

// perform click event on set Format button to set the format of chronometer
setFormat.setOnClickListener(new View.OnClickListener() {

    @Override
    public void onClick(View v) {
        // TODO Auto-generated method stub

        simpleChronometer.setFormat("Time (%s)");
    }
});

// perform click event on clear button to clear the current format of chronometer as
you set through set format
clearFormat.setOnClickListener(new View.OnClickListener() {

    @Override
    public void onClick(View v) {
        // TODO Auto-generated method stub

        simpleChronometer.setFormat(null);
    }
}
}
}
```

Step 4: Open res -> values -> strings.xml

In this step we shows the list of strings that will be used in our example.

```
<resources>
    <string name="app_name">ChronometerExample</string>
    <string name="hello_world">Hello world!</string>
    <string name="action_settings">Settings</string>
    <string name="chronometerInAndroid">Chronometer In Android</string>
    <string name="start">Start</string>
    <string name="stop">Stop</string>
    <string name="restart">Restart</string>
    <string name="setFormat">Set Format</string>
    <string name="clearFormat">Clear Format</string>
</resources>
```

Output:

Now run the App and you will see 5 buttons to start, stop, restart, set format and clear format of Chronometer.

Zoom Controls

In Android, Zoom Controls class display simple set of controls that is used for zooming and provides callback to register for events. Zoom Controls has two buttons ZoomIn and ZoomOut which are used to control the zooming functionality.



Zoom Controls code in XML:

```
<ZoomControls  
    android:id="@+id/simpleZoomControl"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content" />
```

Important Methods Of Zoom Controls:

Now let's discuss some common methods which are used to configure ZoomControls in our application.

- 1. hide():** This method is used to hide the ZoomControls from the screen. In some cases we need to hide the ZoomControls from the screen so that we use this function.
- 2. show():** This method is used to show the ZoomControls which we hide from the screen by using hide method.

Below we show the use of hide and show methods of ZoomControls:

Step 1: In this example first in xml file we display ZoomControls with two buttons hide and show which are used to hide and show the ZoomControls.



xml code of above image UI:

```
<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">

    <ZoomControls
        android:id="@+id/simpleZoomControl"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="50dp" />
```

```

<Button
    android:id="@+id/show"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_centerInParent="true"
    android:layout_margin="20dp"
    android:background="#0f0"
    android:text="Show"
    android:textColor="#fff" />

<Button
    android:id="@+id/hide"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_below="@+id/show"
    android:layout_margin="20dp"
    android:background="#f00"
    android:text="Hide"
    android:textColor="#fff" />

</RelativeLayout>

```

Step 2: Now in Java Class we use hide() and show() methods to hide and show Zoom Controls option.

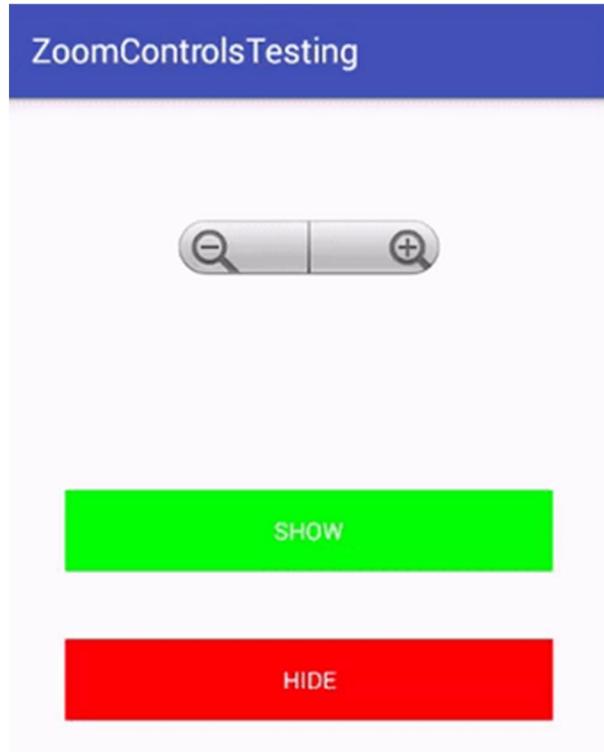
```

/*Add below setContentView() method in Oncreate()*/
    final ZoomControls simpleZoomControls = (ZoomControls)
findViewById(R.id.simpleZoomControl); // initiate a ZoomControls
    Button show = (Button) findViewById(R.id.show); // initiate show Button
    Button hide = (Button) findViewById(R.id.hide); // initiate hide Button
// perform setOnClickListener on show button
    show.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
// show a ZoomControls
            simpleZoomControls.show();
        }
    });
// perform setOnClickListener on hide button
    hide.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
// hide a ZoomControls
            simpleZoomControls.hide();
        }
    });

```

Output:

Now run the App and you can see ZoomControls can be hided and showed again from user.



3. setOnZoomInClickListener(OnClickListener listener): This is a listener event automatically called when we click on the Zoom In button of ZoomControls. In this listener we add the code to zoom in image.

Below we show the use of setOnZoomInClickListener in android.

```
final ZoomControls simpleZoomControls = (ZoomControls) findViewById(R.id.simpleZoomControl);
// initiate a ZoomControls

// perform setOnZoomInClickListener event on ZoomControls
simpleZoomControls.setOnZoomInClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
// add zoom in code here
}
});
```

4. setOnZoomOutClickListener(OnClickListenerlistener): This is a listener event automatically called when we click on the Zoom Out button of ZoomControls. In this listener we add the code for zoom out a image.

Below we show the use of setOnZoomOutClickListener in android.

```
final ZoomControls simpleZoomControls = (ZoomControls) findViewById(R.id.simpleZoomControl);  
// initiate a ZoomControls  
  
// perform setOnZoomOutClickListener event on ZoomControls  
simpleZoomControls.setOnZoomOutClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        // add zoom out code here  
    }  
});
```

5. setIsZoomInEnabled(boolean isEnabled): This method is used to enable or disable the zoom In button of ZoomControls. In this method we set a Boolean value either true or false. By default it has true value but sometime after a limit of zoom in we need to disable the zoom in functionality i.e. after that we didn't need more zoom in.

Below we set the false value for setIsZoomInEnabled that disable zoom in button of ZoomControls.

```
ZoomControls simpleZoomControls = (ZoomControls) findViewById(R.id.simpleZoomControl); //  
initiate a ZoomControls  
simpleZoomControls.setIsZoomInEnabled(false); // disable zoom in button of ZoomControls
```

6. setIsZoomOutEnabled(boolean isEnabled): This method is used to enable or disable the zoom Out button of ZoomControls. In this method we set a Boolean value means true or false. By default it has true value but sometime after a limit of zoom out we need to disable the zoom out functionality means at that time we didn't need more zoom out.

Below we set the false value for setIsZoomOutEnabled that disable zoom out button of ZoomControls.

```
ZoomControls simpleZoomControls = (ZoomControls) findViewById(R.id.simpleZoomControl); //  
initiate a ZoomControls  
simpleZoomControls.setIsZoomOutEnabled(false); // disable zoom out button of ZoomControls
```

Attributes Of Zoom Controls in Android:

Now let's we discuss some important attributes that helps us to configure a ZoomControls in our xml file.

1. id: This attribute is used to uniquely identify a ZoomControls.

```
<ZoomControls  
    android:id="@+id/simpleZoomControl"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content" /> <!-- id of ZoomControls used to uniquely identify  
it -->
```

2. background: This attribute is used to set the background of a ZoomControls. We can set a color or a drawable in the background of a ZoomControls.

Below we set the black color for the background of ZoomControls.

```
<ZoomControls  
    android:id="@+id/simpleZoomControl"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:background="#000" />  
<!-- set black color in the background of a ZoomControls -->
```



Setting background in ZoomControls In Java class:

```
ZoomControls simpleZoomControls = (ZoomControls) findViewById(R.id.simpleZoomControl); //  
initiate a ZoomControls  
simpleZoomControls.setBackgroundColor(Color.BLACK); // set black color in the background of  
ZoomControls
```

3. padding: This attribute is used to set the padding from left, right, top or bottom side of a ZoomControls .

paddingRight: set the padding from the right side of a ZoomControls.

paddingLeft: set the padding from the left side of a ZoomControls.

paddingTop: set the padding from the top side of a ZoomControls.

paddingBottom: set the padding from the bottom side of a ZoomControls.

Padding: set the padding from the all side's of a ZoomControls.

Below we set the 20dp padding from all the sides of a ZoomControls.

```
<ZoomControls  
    android:id="@+id/simpleZoomControl"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:background="#000"
```

```
    android:padding="20dp"/><!-- 20dp padding from all the sides of a ZoomControls -->
```



Zoom Controls Example In Android Studio

Below is an example of ZoomControls in which we display an ImageView and a ZoomControls. In this first we hide the ZoomControls from the screen and show it on the touch event of image. We also perform setOnZoomInClickListener and setOnZoomOutClickListener events for implementing zoom in and zoom out functionality. After zoom in and zoom out the ZoomControls is automatically hide from the screen and re-shown if user click on the image. A Toast is displayed to show the zoom in and zoom out message on the screen.

Below you can download complete Android Studio project code, final output and step by step explanation of the example:



Step 1: Create a new project and name it ZoomControlsExample.

Step 2: Open res -> layout ->activity_main.xml (or) main.xml and add following code :

In this step we add the code for displaying a ImageView and ZoomControls.

```
<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:background="#000"
    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">

    <ImageView
        android:id="@+id/image"
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:src="@drawable/image" />

    <ZoomControls
        android:id="@+id/simpleZoomControl"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="50dp" />

</RelativeLayout>
```

Step 3: Open src -> package -> MainActivity.java

In this step firstly we initiate the ImageView and ZoomControls and then hide the ZoomControls from the screen and show it on the touch event of image.

We also perform setOnZoomInClickListener and setOnZoomOutClickListener events for implementing zoom in and zoom out functionality. After zoom in and zoom out the ZoomControls is automatically hide from the screen and for re-showing it user need to click on image. A Toast is displayed to show the zoom in and zoom out message on the screen.

```
package example.abhiandroid.zoomcontrolsexample;

import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.Menu;
import android.view.MenuItem;
import android.view.MotionEvent;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.Toast;
import android.widget.ZoomControls;

public class MainActivity extends AppCompatActivity {

    ImageView image;
    ZoomControls simpleZoomControls;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        image = (ImageView) findViewById(R.id.image); // initiate a ImageView
        simpleZoomControls = (ZoomControls) findViewById(R.id.simpleZoomControl); // initiate a ZoomControls
        simpleZoomControls.hide(); // initially hide ZoomControls from the screen
        // perform setOnTouchListener event on ImageView
        image.setOnTouchListener(new View.OnTouchListener() {
```

```
    @Override
    public boolean onTouch(View v, MotionEvent event) {
        // show Zoom Controls on touch of image
        simpleZoomControls.show();
        return false;
    }
});
// perform setOnZoomInClickListener event on ZoomControls
simpleZoomControls.setOnZoomInClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // calculate current scale x and y value of ImageView
        float x = image.getScaleX();
        float y = image.getScaleY();
        // set increased value of scale x and y to perform zoom in functionality
        image.setScaleX((float) (x + 1));
        image.setScaleY((float) (y + 1));
        // display a toast to show Zoom In Message on Screen
        Toast.makeText(getApplicationContext(),"Zoom In",Toast.LENGTH_SHORT).show();
        // hide the ZoomControls from the screen
        simpleZoomControls.hide();
    }
});
// perform setOnZoomOutClickListener event on ZoomControls
simpleZoomControls.setOnZoomOutClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // calculate current scale x and y value of ImageView
        float x = image.getScaleX();
        float y = image.getScaleY();
        // set decreased value of scale x and y to perform zoom out functionality
        image.setScaleX((float) (x - 1));
        image.setScaleY((float) (y - 1));
        // display a toast to show Zoom Out Message on Screen
        Toast.makeText(getApplicationContext(),"Zoom
Out",Toast.LENGTH_SHORT).show();
        // hide the ZoomControls from the screen
        simpleZoomControls.hide();
    }
});
}
}
```

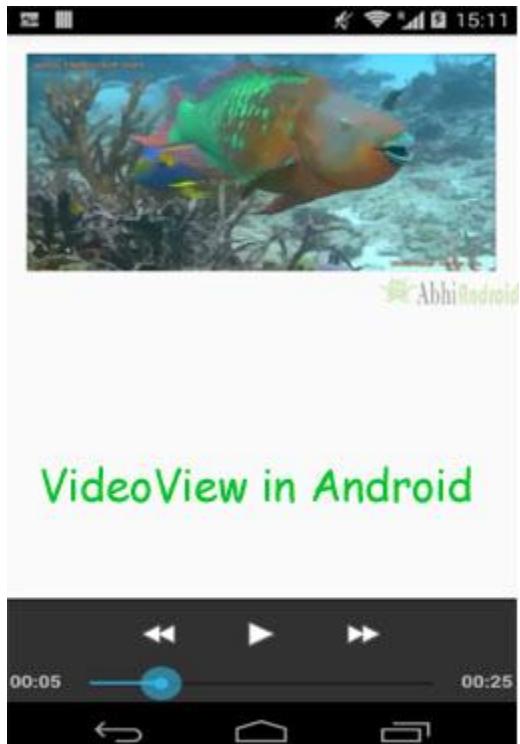
Output:

Now run the App and you can see a image on the screen. Click on the image and Zoom Controls button will appear on the screen. Now do Zoom In or Zoom Out the image as you wish.

VideoView

In Android, VideoView is used to display a video file. It can load images from various sources (such as content providers or resources) taking care of computing its measurement from the

video so that it can be used for any layout manager, providing display options such as scaling and tinting.



Important Note: VideoView does not retain its full state when going into the background. In particular it does not restore the current play position and play state. Applications should save and restore these in `onSaveInstanceState(Bundle)` and `onRestoreInstanceState(Bundle)`.

VideoView code In XML Android:

```
<VideoView  
    android:id="@+id/simpleVideoView"  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent" />
```

Methods Used in VideoView:

Let's we discuss some important methods of VideoView that may be called in order to manage the playback of video:

1. setVideoUri(Uri uri): This method is used to set the absolute path of the video file which is going to be played. This method takes a Uri object as an argument.

Below we set the uri of video which is saved in Android Studio:

Step 1: Create a new directory in res folder and name it raw

Step 2: Save a video name fishvideo in raw folder

Step 3: Now use the below code to set the path for the video using setVideoUri() method in VideoView.

```
// initiate a video view
VideoView simpleVideoView = (VideoView) findViewById(R.id.simpleVideoView);
simpleVideoView.setVideoURI(Uri.parse("android.resource://" + getPackageName() + "/" +
R.raw.fishvideo));
```

Setting Video From Online Web Source:

Step 1: First add internet permision in Manifest.xml file. We will need to add this so as to access the video through Internet. Open AndroidManifest.xml and add the below code

```
<!--Add this before application tag in AndroidManifest.xml-->
<uses-permission android:name="android.permission.INTERNET" />
```



Step 2: Add the basic VideoVideo XML code in activity_main.xml or activity.xml

Step 3: Use the below code to access the Video from our website

```
package abhiandroid.com.videofromwebsource;

import android.app.ProgressDialog;
import android.net.Uri;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.VideoView;

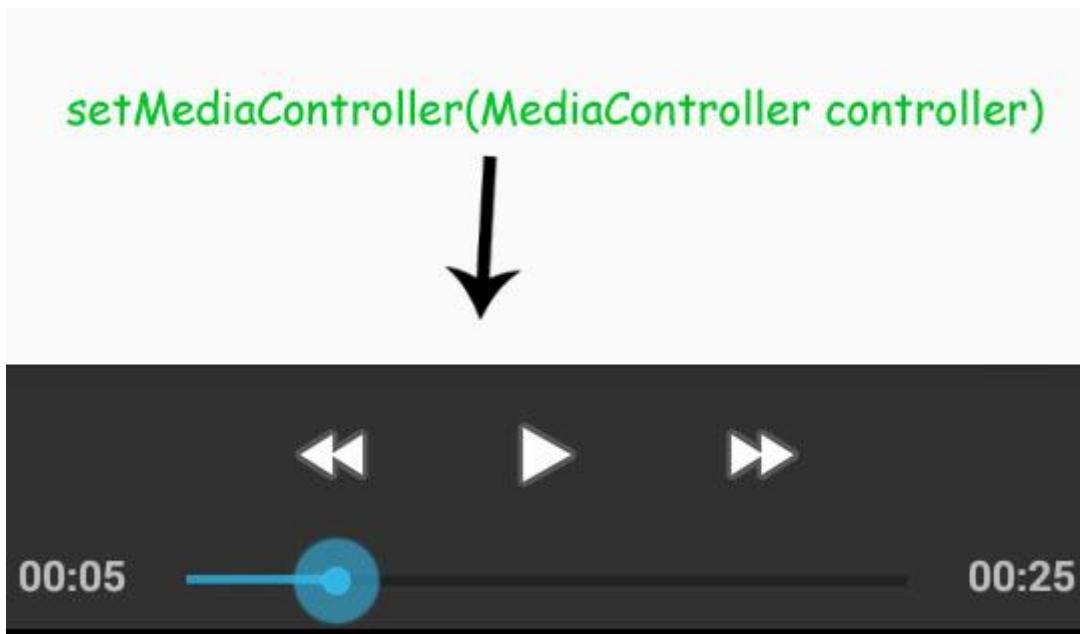
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Uri uri = Uri.parse("http://abhiandroid.jobxfryqt.netdna-cdn.com/ui/wp-content/uploads/2016/04/videoviewtestingvideo.mp4");
        VideoView simpleVideoView = (VideoView) findViewById(R.id.simpleVideoView); // initiate a video view
        simpleVideoView.setVideoURI(uri);
        simpleVideoView.start();
    }
}
```

2. setMediaController(MediaController controller): This method of VideoView is used to set the controller for the controls of video playback.

Below we show how to set the media controller object for a video view.

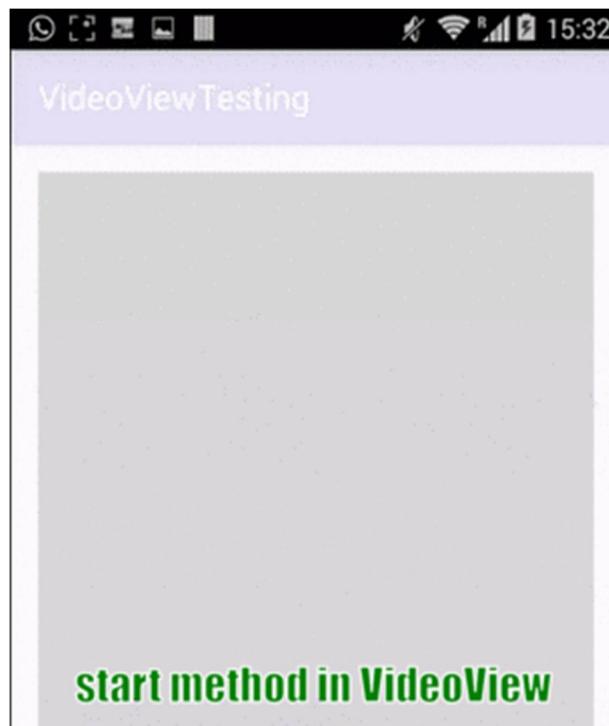
```
// create an object of media controller
MediaController mediaController = new MediaController(this);
// initiate a video view
VideoView simpleVideoView = (VideoView) findViewById(R.id.simpleVideoView);
// set media controller object for a video view
simpleVideoView.setMediaController(mediaController);
```



3. `start()`: This method of `VideoView` is used to start the playback of video file.

Below we show how to start a video in video view

```
VideoView simpleVideoView = (VideoView) findViewById(R.id.simpleVideoView); // initiate a
video view
simpleVideoView.start(); // start a video
```



4. pause(): This method of video view is used to pause the current playback.

Below we shows how to pause a video in video view.

```
VideoView simpleVideoView = (VideoView) findViewById(R.id.simpleVideoView); // initiate a  
video view  
simpleVideoView.pause(); // pause a video
```



5. canPause(): This method will tell whether VideoView is able to pause the video. This method returns a Boolean value means either true or false. If a video can be paused then it returns true otherwise it returns false.

Below we checks whether a video is able to pause or not.

```
VideoView simpleVideoView = (VideoView) findViewById(R.id.simpleVideoView); // initiate a  
video view  
Boolean canPauseVideo = simpleVideoView.canPause(); // check whether a video is able to  
pause or not
```

6. canSeekForward(): This method will tell whether video is able to seek forward. This method returns a Boolean value i.e. true or false. If a video can seek forward then it returns true otherwise it returns false.

Below we checks whether a video is able to seek forward or not.

```
VideoView simpleVideoView = (VideoView) findViewById(R.id.simpleVideoView); // initiate a  
video view  
Boolean canSeekForward = simpleVideoView.canSeekForward(); // checks whether a video view is  
able to seek forward or not
```

7. canSeekBackward(): This method will tell whether video is able to seek backward. This method returns a Boolean value i.e. true or false. If a video can seek backward then it return true otherwise it return false.

Below we checks whether a video is able to seek backward or not.

```
VideoView simpleVideoView = (VideoView) findViewById(R.id.simpleVideoView); // initiate a  
video view  
Boolean canSeekBackword = simpleVideoView.canSeekBackward(); // checks whether a video view  
is able to seek backword or not
```

8. getDuration(): This method is used to get the total duration of VideoView. This methods return an integer value.

Below we get the total duration of a video view.

```
VideoView simpleVideoView = (VideoView) findViewById(R.id.simpleVideoView); // initiate a  
video view  
int duration =simpleVideoView.getDuration(); // get the total duration of the video
```

9. getCurrentPosition(): This method is used to get the current position of playback. This method returns an integer value.

Below we get the current position of a playback.

10. isPlaying(): This method tells whether a video is currently playing or not. This method returns a Boolean value. It returns true if video is playing or false if it's not.

Below we check whether a video view is currently playing or not

```
VideoView simpleVideoView = (VideoView) findViewById(R.id.simpleVideoView); // initiate a  
video view  
Boolean isPlaying = simpleVideoView.isPlaying(); // check whether a video view is currently  
playing or not
```

11. stopPlayback(): This method of VideoView is used to stop the video playback.

Below we show how to stop a video in video view.

```
VideoView simpleVideoView = (VideoView) findViewById(R.id.simpleVideoView); // initiate a  
video view  
simpleVideoView.stopPlayback(); // stop a video
```

12. setOnPreparedListener(MediaPlayer.OnPreparedListener): This is a listener which allows a callback method to be called when the video is ready to play.

Below we show the use of setOnPreparedListener event of a video view.

```
VideoView simpleVideoView = (VideoView) findViewById(R.id.simpleVideoView); // initiate a
video view

// perform set on prepared listener event on video view
simpleVideoView.setOnPreparedListener(new MediaPlayer.OnPreparedListener() {
@Override
public void onPrepared(MediaPlayer mp) {

// do something when video is ready to play
}
});
```

13. setOnErrorListener(MediaPlayer.OnErrorListener): This listener allows a callback method to be called when an error occurs during the video playback.

Below we show the use of setOnErrorListener event of a video view.

```
VideoView simpleVideoView = (VideoView) findViewById(R.id.simpleVideoView); // initiate a
video view

// perform set on error listener event on video view
simpleVideoView.setOnErrorListener(new MediaPlayer.OnErrorListener() {
@Override
public boolean onError(MediaPlayer mp, int what, int extra) {

// do something when an error is occur during the video playback
return false;
}
});
```

14. setOnCompletionListener(MediaPlayer.OnCompletionListener): This listener allow a callback method to be called when the end of the video is reached.

Below we shows the use of setOnCompletionListener event of a video view.

```
VideoView simpleVideoView = (VideoView) findViewById(R.id.simpleVideoView); // initiate a
video view
// perform set on completion listener event on video view
simpleVideoView.setOnCompletionListener(new MediaPlayer.OnCompletionListener() {
@Override
public void onCompletion(MediaPlayer mp) {
// do something when the end of the video is reached
}
});
```

MediaController In VideoView

MediaController is a class which is used to provide the controls for the video playback. If a video is simply played using the VideoView class then the user will not be given any control over the playback of the video which will run until the end of the video is reached. This issue can be addressed by attaching an instance of the MediaController class to the VideoView instance. The Media Controller will then provide a set of controls allowing the user to manage the playback (such as seeking backwards/forwards and pausing in the video timeline).

Methods Of MediaController:

Let's we discuss some important methods of MediaController class that may be called in order to control for the playback.

1. setAnchorView(View view): setAnchorView is used to designates the view to which the controller is to be anchored. This controls the location of the controls on the screen.

Below we show how to use setanchorview() method of a MediaController class.

```
MediaController mediaController = new MediaController(this); // create an object of media
controller
mediaController.setAnchorView(simpleVideoView); // set anchor view for video view
```

2. show(): This method is used to show the controller on the screen.

Below we show the controller on the screen.

```
MediaController mediaController = new MediaController(this); // create an object of media controller  
mediaController.show(); // show the controller on the screen
```

3. show(int timeout): This method is used to set the time to show the controller on the screen.

Below we set the time for showing the controller on the screen.

4. hide(): This method is used to hide the controls from the screen.

Below we hide the control from the screen

```
MediaController mediaController = new MediaController(this); // create an object of media controller  
mediaController.hide(); // hide the control from the screen
```

5. isShowing(): This method returns a Boolean value indicating whether the controls are currently visible to the user or not.

Below we checks whether the controls are currently visible or not.

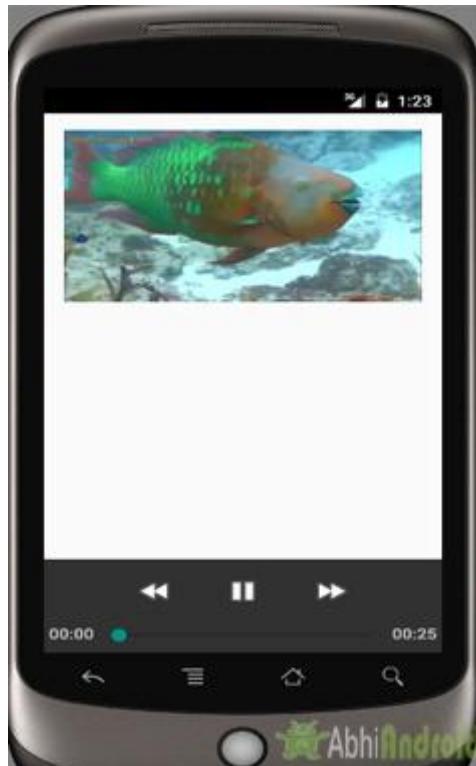
```
MediaController mediaController = new MediaController(this); // create an object of media controller  
Boolean isShowing = mediaController.isShowing(); // checks whether the controls are currently visible or not
```

VideoView Example In Android Studio:

Below is the example of VideoView in Android in which we play a video in a video view by using Media Controller and perform set on error and completion listener events and display Toast when the video is completed or an error occur while playing thee video.

In this example we create a folder named raw in our project and store the video file in that folder and then set the uri for the video in our activity in which we display the video view.

Below is the final output, download code and step by step explanation:



Step 1: Create a new project in Android Studio and name it VideoViewExample

Step 2: Open res -> layout -> xml (or) main.xml and add following code :

In this step we open an xml file and add the code to display a VideoView in our activity.

```
<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">

    <VideoView
```

```
    android:id="@+id/simpleVideoView"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />

</RelativeLayout>
```

Step 3: Open src -> package -> MainActivity.java

In this step we open MainActivity and add the code to initiate the video view and create an object of MediaController to control the video playback.

In this class we also set the uri for the video and perform set on error and completion listener events and display Toast message when video is completed or an error is occur while playing thee video.

Also make sure to create a new directory in res folder and name it raw. Save a video name fishvideo in raw folder. We will be setting path to this Video in setVideoURI() method.

```
package example.abhiandroid.videoviewexample;

import android.app.Activity;
import android.app.ProgressDialog;
import android.content.res.Configuration;
import android.media.MediaPlayer;
import android.media.MediaPlayer.OnPreparedListener;
import android.net.Uri;
import android.os.Bundle;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.MediaController;
import android.widget.Toast;
import android.widget.VideoView;

public class MainActivity extends Activity {

    VideoView simpleVideoView;
    MediaController mediaControls;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // Find your VideoView in your video_main.xml layout
        simpleVideoView = (VideoView) findViewById(R.id.simpleVideoView);
```

```
if (mediaControls == null) {
    // create an object of media controller class
    mediaControls = new MediaController(MainActivity.this);
    mediaControls.setAnchorView(simpleVideoView);
}
// set the media controller for video view
simpleVideoView.setMediaController(mediaControls);
// set the uri for the video view
simpleVideoView.setVideoURI(Uri.parse("android.resource://" + getPackageName() + "/"
+ R.raw.fishvideo));
// start a video
simpleVideoView.start();

// implement on completion listener on video view
simpleVideoView.setOnCompletionListener(new MediaPlayer.OnCompletionListener() {
    @Override
    public void onCompletion(MediaPlayer mp) {
        Toast.makeText(getApplicationContext(), "Thank You...!!!",
Toast.LENGTH_LONG).show(); // display a toast when an video is completed
    }
});
simpleVideoView.setOnErrorListener(new MediaPlayer.OnErrorListener() {
    @Override
    public boolean onError(MediaPlayer mp, int what, int extra) {
        Toast.makeText(getApplicationContext(), "Oops An Error Occur While Playing
Video...!!!", Toast.LENGTH_LONG).show(); // display a toast when an error is occurred while
playing an video
        return false;
    }
});
}
```

Output:

Now run the App and you will see Video playing as the App open. Click on the Video and Media Controller will appear on the screen.

SearchView

In Android, SearchView widget provide search user interface where users can enter a search query and then submit a request to search provider. It shows a list of query suggestions or results if available and allow the users to pick a suggestion or result to launch into.



Basic SearchView code in XML:

```
<SearchView  
    android:id="@+id/simpleSearchView"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content" />
```

SearchView Methods In Android:

Let's we discuss some important methods of search view that may be called in order to manage the search view.

1. `getQuery()`: This function is used to get the query string currently in the text field of a search view. This method returns CharSequence type value.

Below we get the query String from a search view.

```
SearchView simpleSearchView = (SearchView) findViewById(R.id.simpleSearchView); //  
initiate a search view  
CharSequence query = simpleSearchView.getQuery(); // get the query string currently in the  
text field
```

2. `getQueryHint()`: This function is used for getting the hint text that will be displayed in the query text field. This method returns a CharSequence type value.

Below is an example code which get the hint text that will be displayed in the query text field of a search view.

```
SearchView simpleSearchView = (SearchView) findViewById(R.id.simpleSearchView); // initiate  
a search view  
CharSequence queryHint = simpleSearchView.getQueryHint(); // get the hint text that will be  
displayed in the query text field
```

3. `isIconifiedByDefault()`: This method returns the default iconified state of the search field. This method returns a Boolean value either true or false.

Below we get the default state of the search field.

```
SearchView simpleSearchView = (SearchView) findViewById(R.id.simpleSearchView); // initiate  
a search view  
boolean isIconified=simpleSearchView.isIconifiedByDefault(); // checks default iconified state  
of the search field
```

4. `setIconifiedByDefault(Boolean iconify)`: This method is used to set the default or resting state of the search field. In this method we set Boolean value true or false.

Important Note: When a SearchView is used in an Action Bar as an action view for collapsible menu item then it needs to be set to iconified by default using `setIconifiedByDefault(true)` function. If you want the search field to always be visible, then call

`setIconifiedByDefault(false)`. `true` is the default value for this function. You can also set iconified from xml by using `iconifiedByDefault` property to true or false.

Below we set the iconified by default value to false .

```
SearchView simpleSearchView = (SearchView) findViewById(R.id.simpleSearchView); // initiate  
a search view  
simpleSearchView.setIconifiedByDefault(false); // set the default or resting state of the  
search field
```

5. `setQueryHint(CharSequence hint)`: This method is used to set the hint text to display in the query text field. This method support `CharSequence` type value.

Below we set the query hint for a SearchView.

```
SearchView simpleSearchView = (SearchView) findViewById(R.id.simpleSearchView); // initiate  
a search view  
simpleSearchView.setQueryHint("Search View"); // set the hint text to display in the query  
text field
```

6. `setOnQueryTextFocusChangeListener(OnFocusChangeListenerlistener)`: This listener inform when the focus of the query text field changes.

In the below code we show the use of `setOnQueryTextFocusChangeListener()` of `SearchView`.

```
SearchView simpleSearchView = (SearchView) findViewById(R.id.simpleSearchView); // initiate  
a search view  
  
// perform set on query text focus change listener event  
simpleSearchView.setOnQueryTextFocusChangeListener(new View.OnFocusChangeListener() {  
    @Override  
    public void onFocusChange(View v, boolean hasFocus) {  
        // do something when the focus of the query text field changes  
    }  
});
```

7. `setOnQueryTextListener(OnQueryTextListenerlistener)`: It is a user action within the `SearchView`.

Below we show the use of setOnQueryTextListener() of search view.

```
SearchView simpleSearchView = (SearchView) findViewById(R.id.simpleSearchView); // initiate  
a search view  
  
// perform set on query text listener event  
simpleSearchView.setOnQueryTextListener(new SearchView.OnQueryTextListener() {  
    @Override  
    public boolean onQueryTextSubmit(String query) {  
        // do something on text submit  
        return false;  
    }  
  
    @Override  
    public boolean onQueryTextChange(String newText) {  
        // do something when text changes  
        return false;  
    }  
});
```

Attributes of SearchView:

Now let's we discuss some common attributes of a searchview that helps us to configure it in our layout (xml).

1. Id: id attribute is used to uniquely identify a search view.

```
<!-- id of an search view used to uniquely identify it -->  
<SearchView  
    android:id="@+id/simpleSearchView"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content" />
```

2. queryHint: This attribute ia used to set optional query hint string which will be displayed in the empty query field. Query hint is same as hint attribute for edittext.

Below we set the query hint of an search view.

```
<SearchView  
    android:id="@+id/simpleSearchView"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:iconifiedByDefault="false"
```

```
    android:queryHint="Search Here" /><!-- set query string of an search view -->
```



3. iconifiedByDefault: This attribute of searchview is used to set the default or resting state of the search field. You can set a Boolean value for this attribute and default value is true. True value indicates you can iconifies or expands the search view.

Below we set the false value for this attribute.

```
<SearchView
    android:id="@+id/simpleSearchView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:iconifiedByDefault="false"
    android:queryHint="Search here"/><!-- set iconified by default to false -->
```



Setting

iconifiedByDefault="true"

4. background: background attribute is used to set the background of a search view. You can set a color or a drawable in the background of a search view. You can also set the background color in java class.

Below we set the red color for the background of a search view.

```
<SearchView  
    android:id="@+id/simpleSearchView"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:iconifiedByDefault="false"  
    android:queryHint="Search here"  
    android:background="#f00"/><!-- red color for the background of search view -->
```



Setting background of SearchView In Java class:

```
SearchView simpleSearchView=(SearchView)findViewById(R.id.simpleSearchView);
simpleSearchView.setBackgroundColor(Color.RED);
```

5. padding: padding attribute is used to set the padding from left, right, top or bottom.

paddingRight: set the padding from the right side of the search view.

paddingLeft: set the padding from the left side of the search view.

paddingTop: set the padding from the top side of the search view.

paddingBottom: set the padding from the bottom side of the search view

Padding: set the padding from the all side's of the search view.

Below we set 40dp padding from all the sides of a search view.

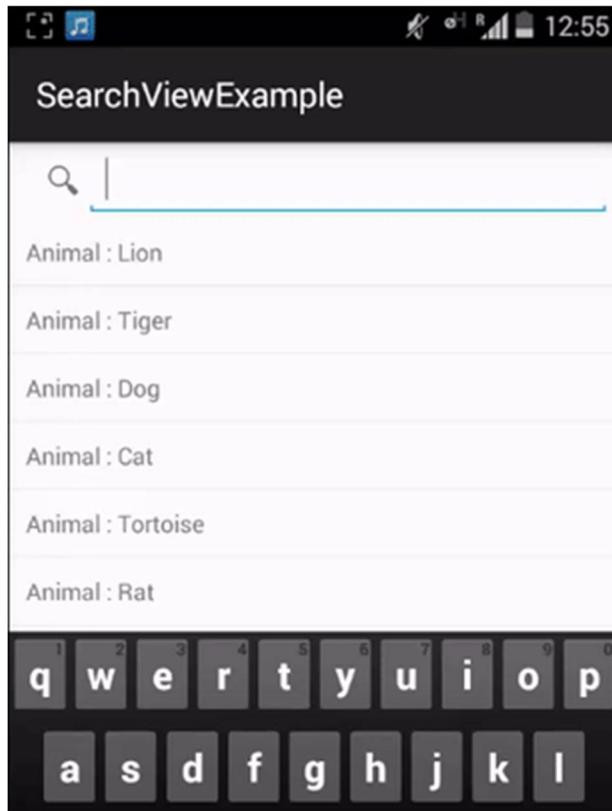
```
<SearchView
    android:id="@+id/simpleSearchView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:iconifiedByDefault="false"
    android:queryHint="Search View"
    android:background="#f00"
    android:padding="40dp"/> <!-- set 40 dp padding from all the sides of a search view -->
```



SearchView Example In Android Studio:

In the below example of SearchView we display SearchView and ListView. In this we create an animal name list and then set the Adapter to fill the data in ListView. Finally we implement SearchView.OnQueryTextListener to filter the animal list according to search query.

Below you can download complete SearchView Android Studio project code, see final output and step by step explanation of example:



Step 1: Create a new project and name it SearchViewExample

Step 2: Open res -> layout ->activity_main.xml (or) main.xml and add following code:

In this step we open an xml file and add the code for displaying a SearchView and ListView by using its different attributes.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent">  
  
    <SearchView  
        android:id="@+id/search"  
        android:layout_width="fill_parent"  
        android:layout_height="wrap_content"  
        android:iconifiedByDefault="false">  
  
        <requestFocus />  
    </SearchView>
```

```
<ListView  
    android:id="@+id/listview"  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent"  
    android:layout_below="@+id/search" />  
  
</RelativeLayout>
```

Step 3: Open src -> package -> MainActivity.java

In this step we open MainActivity and add the code to initiate SearchView and ListView. In this we create an Animal name list and then set the adapter to fill the data in ListView. In this we also implement SearchView.OnQueryTextListener to filter the animal list according to search query.

```
package example.abhiandroid.searchviewexample;  
  
import android.os.Bundle;  
import android.support.v7.app.AppCompatActivity;  
import android.widget.ListView;  
import android.widget.SearchView;  
  
import java.util.ArrayList;  
  
public class MainActivity extends AppCompatActivity implements  
SearchView.OnQueryTextListener {  
  
    // Declare Variables  
    ListView list;  
    ListViewAdapter adapter;  
    SearchView editsearch;  
    String[] animalNameList;  
    ArrayList<AnimalNames> arraylist = new ArrayList<AnimalNames>();  
  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        // Generate sample data  
  
        animalNameList = new String[]{"Lion", "Tiger", "Dog",  
            "Cat", "Tortoise", "Rat", "Elephant", "Fox",  
            "Cow", "Donkey", "Monkey"};  
  
        // Locate the ListView in listview_main.xml  
        list = (ListView) findViewById(R.id.listview);  
  
        for (int i = 0; i < animalNameList.length; i++) {
```

```
        AnimalNames animalNames = new AnimalNames(animalNameList[i]);
        // Binds all strings into an array
        arraylist.add(animalNames);
    }

    // Pass results to ListViewAdapter Class
    adapter = new ListViewAdapter(this, arraylist);

    // Binds the Adapter to the ListView
    list.setAdapter(adapter);

    // Locate the EditText in listview_main.xml
    editsearch = (SearchView) findViewById(R.id.search);
    editsearch.setOnQueryTextListener(this);
}

@Override
public boolean onQueryTextSubmit(String query) {

    return false;
}

@Override
public boolean onQueryTextChange(String newText) {
    String text = newText;
    adapter.filter(text);
    return false;
}
}
```

Step 4: Now create New Class. Go to app -> java -> right click on package-> New -> Java Class and create ListViewAdapter.java and add following code. Here we extends BaseAdapter in ListViewAdapter class and then set the data in the ListView by using Modal class.

```
package example.abhiandroid.searchviewexample;

import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.TextView;
import java.util.ArrayList;
import java.util.List;
import java.util.Locale;
```

```
public class ListViewAdapter extends BaseAdapter {

    // Declare Variables

    Context mContext;
    LayoutInflator inflater;
    private List<AnimalNames> animalNamesList = null;
    private ArrayList<AnimalNames> arraylist;

    public ListViewAdapter(Context context, List<AnimalNames> animalNamesList) {
        mContext = context;
        this.animalNamesList = animalNamesList;
        inflater = LayoutInflator.from(mContext);
        this.arraylist = new ArrayList<AnimalNames>();
        this.arraylist.addAll(animalNamesList);
    }

    public class ViewHolder {
        TextView name;
    }

    @Override
    public int getCount() {
        return animalNamesList.size();
    }

    @Override
    public AnimalNames getItem(int position) {
        return animalNamesList.get(position);
    }

    @Override
    public long getItemId(int position) {
        return position;
    }

    public View getView(final int position, View view, ViewGroup parent) {
        final ViewHolder holder;
        if (view == null) {
            holder = new ViewHolder();
            view = inflater.inflate(R.layout.listview_item, null);
            // Locate the TextViews in listview_item.xml
            holder.name = (TextView) view.findViewById(R.id.name);
            view.setTag(holder);
        } else {
            holder = (ViewHolder) view.getTag();
        }
        // Set the results into TextViews
        holder.name.setText(animalNamesList.get(position).getAnimalName());
        return view;
    }

    // Filter Class
    public void filter(String charText) {
        charText = charText.toLowerCase(Locale.getDefault());
        animalNamesList.clear();
        if (charText.length() == 0) {
```

```
        animalNamesList.addAll(arraylist);
    } else {
        for (AnimalNames wp : arraylist) {
            if (wp.getAnimalName().toLowerCase(Locale.getDefault()).contains(charText))
{
                animalNamesList.add(wp);
            }
        }
    }
    notifyDataSetChanged();
}
```

Step 5: Now Create new a new layout Activity. Go to res-> right click on layout -> New -> Activity -> Blank Activity and create list_view_items.xml and add following code. Here we are creating items view that will be displayed inside each row.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:padding="10dp">

    <TextView
        android:id="@+id/nameLabel"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Animal : " />

    <TextView
        android:id="@+id/name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toRightOf="@+id/nameLabel" />

</RelativeLayout>
```

Step 6: Now create New Class. Go to app -> java -> right click on package-> New -> Java Class and create AnimalNames.java and add following code. Here we have a constructor for setting the animal name and a function to get the animal name.

```
package example.abhiandroid.searchviewexample;

public class AnimalNames {
    private String animalName;

    public AnimalNames(String animalName) {
        this.animalName = animalName;
    }

    public String getAnimalName() {
        return this.animalName;
    }
}
```

Output:

Now run the App, you will see different Animal names listed. Now type first character of Animal that came to your mind in SearchView and you will the list is sorted.

Toast

In Android, Toast is used to display information for a period of time. It contains a message to be displayed quickly and disappears after specified period of time. It does not block the user interaction. Toast is a subclass of Object class. In this we use two constants for setting the duration for the Toast. Toast notification in android always appears near the bottom of the screen. We can also create our custom toast by using custom layout(xml file).

Simple Toast In Android

Toast

Custom Toast In Android

Custom Toast With Image

Special Note: In Android, Toast is used when we required to notify user about an operation without expecting any user input. It displays a small popup for message and automatically fades out after timeout.

Important Methods Of Toast:

Let's we discuss some important methods of Toast that may be called in order to manage the Toast.

1. `makeText(Context context, CharSequence text, int duration)`: This method is used to initiate the Toast. This method take three parameters First is for the application Context, Second is text message and last one is duration for the Toast.

Constants of Toast: Below is the constants of Toast that are used for setting the duration for the Toast.

1. `LENGTH_LONG`: It is used to display the Toast for a long period of time. When we set this duration the Toast will be displayed for a long duration.

2. `LENGTH_SHORT`: It is used to display the Toast for short period of time. When we set this duration the Toast will be displayed for short duration.

Below we show the use of `makeText()` method of Toast in which we set application context, a text message and duration for the Toast.

```
Toast toast = Toast.makeText(getApplicationContext(), "Simple Toast", Toast.LENGTH_LONG); //  
initiate the Toast with context, message and duration for the Toas
```

2. `show()`: This method is used to display the Toast on the screen. This method is display the text which we create using `makeText()` method of Toast.

Below we Firstly initiate the Toast and then display it using `show()` method.

```
Toast toast = Toast.makeText(getApplicationContext(), "Simple Toast In Android",  
Toast.LENGTH_LONG); // initiate the Toast with context, message and duration for the Toast  
toast.show(); // display the Toast
```

3. `setGravity(int,int,int)`: This method is used to set the gravity for the Toast. This method accepts three parameters: a Gravity constant, an x-position offset, and a y-position offset.

Below we Firstly initiate the Toast, set top and left gravity and then display it using `show()` method.

```
Toast toast = Toast.makeText(getApplicationContext(), "Simple Toast In Android",
    Toast.LENGTH_LONG); // initiate the Toast with context, message and duration for the Toast
toast.setGravity(Gravity.TOP | Gravity.LEFT, 0, 0); // set gravity for the Toast.
toast.show(); // display the Toast
```

4. setText(CharSequence s): This method is used to set the text for the Toast. If we use makeText() method and then we want to change the text value for the Toast then we use this method.

Below we firstly create a new Toast using makeText() method and then set the text for the Toast.

```
Toast toast = Toast.makeText(getApplicationContext(), "Simple Toast In Android",
    Toast.LENGTH_LONG); // initiate the Toast with context, message and duration for the Toast
toast.setGravity(Gravity.TOP | Gravity.LEFT, 0, 0); // set gravity for the Toast.
toast.setText("Changed Toast Text"); // set the text for the Toast
toast.show(); // display the Toast
```

5. setDuration(int duration): This method is used to set the duration for the Toast. If we use makeText() method and then we want to change the duration for the Toast then we use this method.

Below we firstly create a new Toast using makeText() method and then set the duration for the Toast.

```
Toast toast = Toast.makeText(getApplicationContext(), "Simple Toast In Android",
    Toast.LENGTH_LONG); // initiate the Toast with context, message and duration for the Toast
toast.setGravity(Gravity.TOP | Gravity.LEFT, 0, 0); // set gravity for the Toast.
toast.setDuration(Toast.LENGTH_SHORT); // set the duration for the Toast.
toast.show(); // display the Toast
```

6. inflate(int, ViewGroup): This method is used to inflate the layout from the xml. In this method first parameter is the layout resource ID and the second is the root View.

Below we retrieve the Layout Inflater and then inflate the layout from the xml file.

```
// Retrieve the Layout Inflater and inflate the layout from xml
LayoutInflater inflater = getLayoutInflater();
View layout = inflater.inflate(R.layout.custom_toast_layout,
```

```
(ViewGroup) findViewById(R.id.toast_layout_root));
```

7. setView(View): This method is used to set the view for the Toast. In this method we pass the inflated layout which we inflate using inflate() method.

Below we firstly retrieve the layout inflator and then inflate the layout and finally create a new Toast and pass the inflated layout in the setView() method.

```
// Retrieve the Layout Inflater and inflate the layout from xml
LayoutInflater inflater = getLayoutInflater();
View layout = inflater.inflate(R.layout.custom_toast_layout,
(ViewGroup) findViewById(R.id.toast_layout_root));

// create a new Toast using context
Toast toast = new Toast(getApplicationContext());
toast.setDuration(Toast.LENGTH_LONG); // set the duration for the Toast
toast.setView(layout); // set the inflated layout
toast.show(); // display the custom Toast
```

Custom Toast in Android:

In Android, Sometimes simple Toast may not be satisfactory, and then we can go for customizing a Toast. For creating a custom layout, define a View layout, in XML and pass the root View object to the setView(View) method.

Steps for Implementation of Custom Toast In Android:

Step 1: Firstly Retrieve the Layout Inflater with `getLayoutInflator()` (or `getSystemService()`) and then inflate the layout from XML using `inflate(int, ViewGroup)`. In inflate method first parameter is the layout resource ID and the second is the root View.

Step 2: Create a new Toast with `Toast(Context)` and set some properties of the Toast, such as the duration and gravity.

Step 3: Call `setView(View)` and pass the inflated layout in this method.

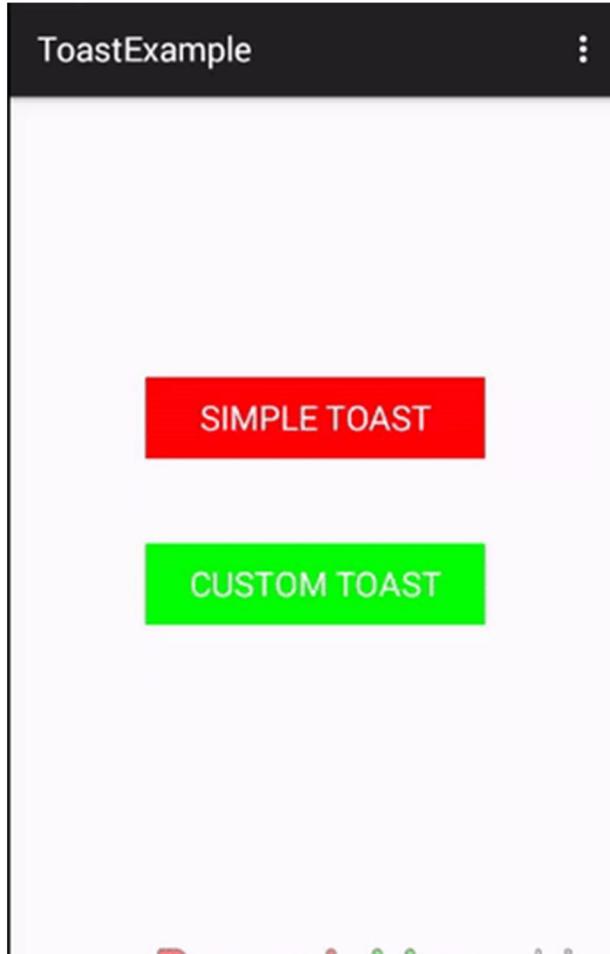
Step 4: Display the Toast on the screen using `show()` method of `Toast`.

In the below example we have shown the functioning of Toast and custom Toast both.

Toast And Custom Toast Example In Android Studio:

Below is the example of Toast and Custom Toast in Android. In this example we display two Button's one for Simple Toast and other for Custom Toast and perform click event on them. Whenever a user click on simple Toast Button a Toast with message "Simple Toast In Android" displayed on the screen and when a user clicks on custom toast Button a message "Custom Toast In Android" with a image displayed on the screen. For Creating a custom toast we firstly retrieve the layout inflater and then inflate the custom toast layout from the xml file. After that we get the reference of TextView and ImageView from the inflated layout and set the text and image in the TextView and ImageView. Finally we create a new Toast and pass the inflated layout in the setView() method and then display the Toast by using show() method of Toast.

Below is the final output, download Android Studio code and step by step explanation of the example:



Step 1: Create a new project and name it ToastExample

Step 2: Open res -> layout ->activity_main.xml (or) main.xml and add following code:

```
<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">
    <!-- Button's for simple and custom Toast -->
    <Button
        android:id="@+id/simpleToast"
        android:layout_width="200dp"
        android:layout_height="wrap_content"
```

```
    android:layout_centerHorizontal="true"
    android:layout_marginTop="150dp"
    android:background="#f00"
    android:text="Simple Toast"
    android:textColor="#fff"
    android:textSize="20sp" />

    <Button
        android:id="@+id/customToast"
        android:layout_width="200dp"
        android:layout_height="wrap_content"
        android:layout_below="@+id/simpleToast"
        android:layout_centerHorizontal="true"
        android:layout_margin="50dp"
        android:background="#0f0"
        android:text="Custom Toast"
        android:textColor="#fff"
        android:textSize="20sp" />

</RelativeLayout>
```

Step 3: Now create a xml layouts by right clicking on res/layout -> New -> Layout Resource File and name it custom_toast_layout.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/toast_layout_root"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#DAAA"
    android:orientation="horizontal"
    android:padding="8dp">
    <!-- ImageView and TextView for custom Toast -->
    <ImageView
        android:id="@+id/toastImageView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginRight="8dp" />

    <TextView
        android:id="@+id/toastTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textColor="#FFF" />
</LinearLayout>
```

Step 4: Open src -> package -> MainActivity.java

In this step we open MainActivity and add the code for initiate the Button's and perform click event on Button's. Whenever a user click on simple Toast Button a Toast with message "Simple Toast In Android" displayed on the screen and when a user clicks on custom toast Button a

message “Custom Toast In Android” with a image displayed on the screen. For Creating a custom toast we firstly retrieve the layout inflater and then inflate the custom toast layout from the xml file. After that we get the reference of TextView and ImageView from the inflated layout and set the text and image in the TextView and ImageView. Finally we create a new Toast and pass the inflated layout in the setView() method and then display the Toast by using show() method of Toast.

```
package com.abhiandroid.toastexample;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Gravity;
import android.view.LayoutInflater;
import android.view.View;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;
import android.widget.Button;
import android.view.ViewGroup;

public class MainActivity extends AppCompatActivity {

    Button simpleToast, customToast;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // get the reference of Button's
        simpleToast = (Button) findViewById(R.id.simpleToast);
        customToast = (Button) findViewById(R.id.customToast);
        // perform setOnClickListener event on simple Toast Button
        simpleToast.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // initiate a Toast with message and duration
                Toast toast = Toast.makeText(getApplicationContext(), "Simple Toast In
Android", Toast.LENGTH_LONG); // initiate the Toast with context, message and duration for
the Toast
                toast.setGravity(Gravity.BOTTOM | Gravity.CENTER_HORIZONTAL, 0, 0); // set gravity for the Toast.
                toast.show(); // display the Toast
            }
        });
        // perform setOnClickListener event on custom Toast Button
        customToast.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Retrieve the Layout Inflater and inflate the layout from xml
```

```
        LayoutInflator inflater = getLayoutInflator();
        View layout = inflater.inflate(R.layout.custom_toast_layout,
                (ViewGroup) findViewById(R.id.toast_layout_root));
        // get the reference of TextView and ImageView from inflated layout
        TextView toastTextView = (TextView) layout.findViewById(R.id.toastTextView);
        ImageView toastImageView = (ImageView)
        layout.findViewById(R.id.toastImageView);
        // set the text in the TextView
        toastTextView.setText("Custom Toast In Android");
        // set the Image in the ImageView
        toastImageView.setImageResource(R.drawable.ic_launcher);
        // create a new Toast using context
        Toast toast = new Toast(getApplicationContext());
        toast.setDuration(Toast.LENGTH_LONG); // set the duration for the Toast
        toast.setView(layout); // set the inflated layout
        toast.show(); // display the custom Toast
    }
});
```

Intent in Android

Android uses Intent for communicating between the components of an Application and also from one application to another application.

Intent are the objects which is used in android for passing the information among Activities in an Application and from one app to another also. Intent are used for communicating between the Application components and it also provides the connectivity between two apps.

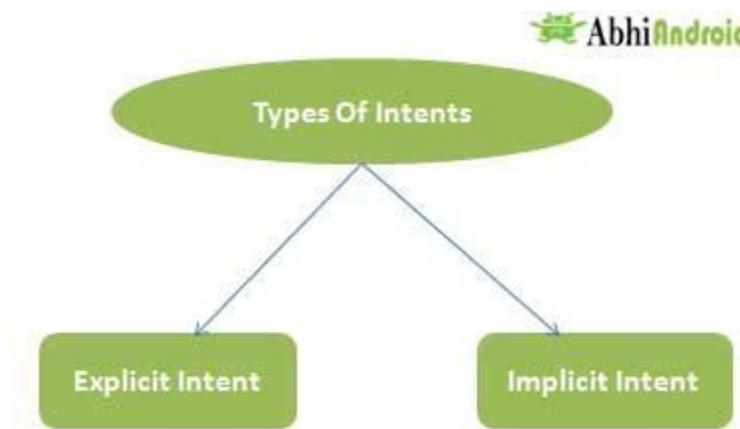
For example: Intent facilitate you to redirect your activity to another activity on occurrence of any event. By calling, `startActivity()` you can perform this task.

```
Intent intent = new Intent(getApplicationContext(), SecondActivity.class);
startActivity(intent);
```

In the above example, foreground activity is getting redirected to another activity i.e. `SecondActivity.java`. `getApplicationContext()` returns the context for your foreground activity.

Types of Intents:

Intent are of two types: Explicit Intent and Implicit Intent



Explicit Intent:

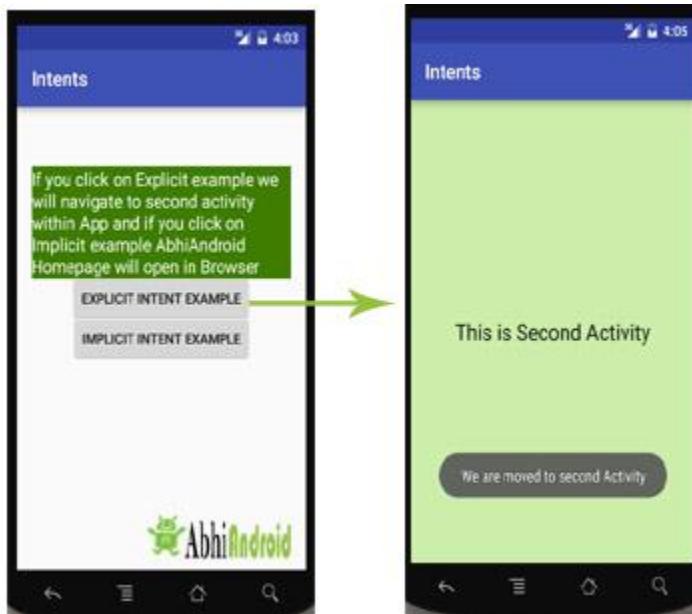
Explicit Intents are used to connect the application internally.

In Explicit we use the name of component which will be affected by Intent. For Example: If we know class name then we can navigate the app from One Activity to another activity using Intent. In the similar way we can start a service to download a file in background process.

Explicit Intent work internally within an application to perform navigation and data transfer. The below given code snippet will help you understand the concept of Explicit Intents

```
Intent intent = new Intent(getApplicationContext(), SecondActivity.class);
startActivity(intent);
```

Here SecondActivity is the JAVA class name where the activity will now be navigated. Example with code in the end of this post will make it more clear.



Implicit Intent:

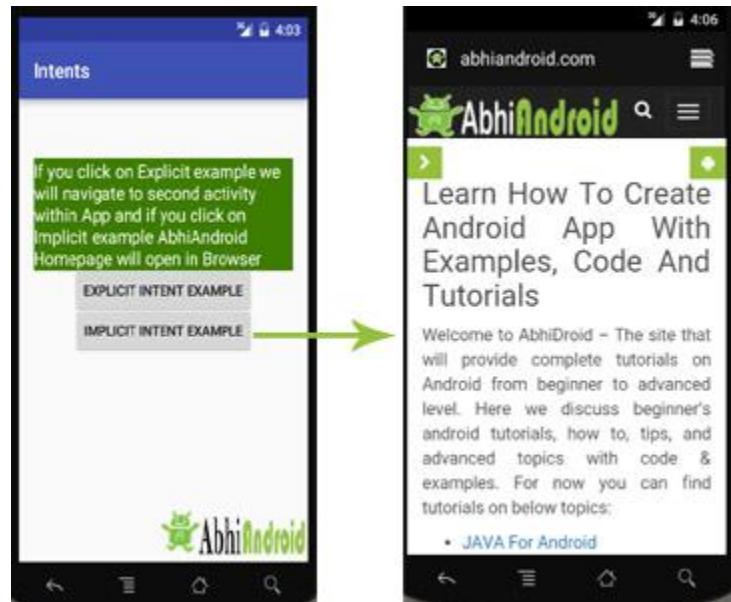
In Implicit Intents we do need to specify the name of the component. We just specify the Action which has to be performed and further this action is handled by the component of another application.

The basic example of implicit Intent is to open any web page

Let's take an example to understand Implicit Intents more clearly. We have to open a website using intent in your application. See the code snippet given below

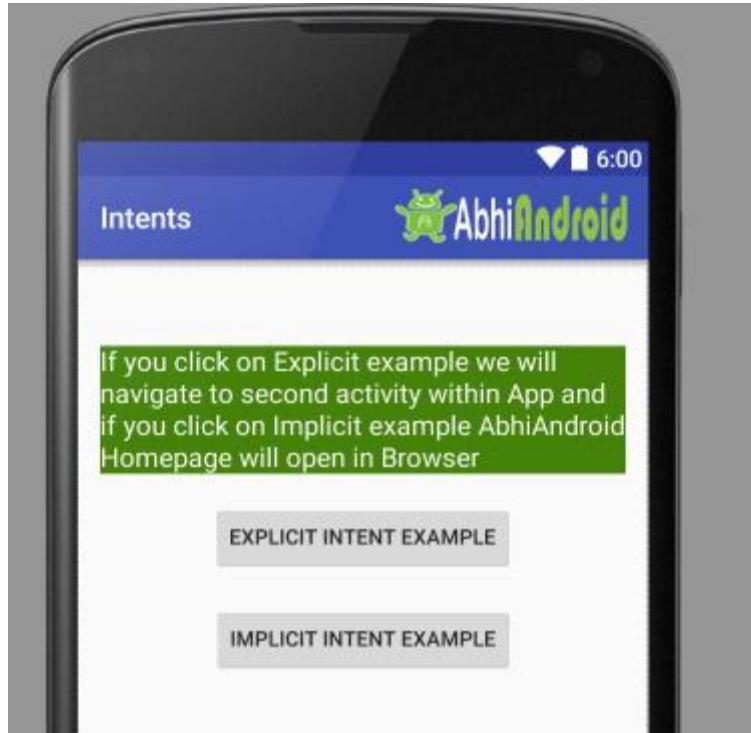
```
Intent intentObj = new Intent(Intent.ACTION_VIEW);
intentObj.setData(Uri.parse("http://www.abhiandroid.com"));
startActivity(intentObj);
```

Unlike Explicit Intent you do not use any class name to pass through Intent(). In this example we has just specified an action. Now when we will run this code then Android will automatically start your web browser and it will open AbhiAndroid home page.



Intent Example In Android:

Let's implement Intent for a very basic use. In the below example we will Navigate from one Activity to another and open a web homepage of AbhiAndroid using Intent. The example will show you both implicit and explicit Intent together. Below is the final output:



Create a project in Android Studio and named it “Intents”. Make an activity, which would consists Java file; MainActivity.java and an xml file for User interface which would be activity_main.xml

Step 1: Let's design the UI of activity_main.xml:

- First design the text view displaying basic details of the App
- Second design the two button of Explicit Intent Example and Implicit Intent Example

Below is the complete code of activity_main.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@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:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:text="If you click on Explicit example we will navigate to second activity
within App and if you click on Implicit example AbhiAndroid Homepage will open in Browser"
        android:id="@+id/textView2"
        android:clickable="false"
        android:layout_alignParentTop="true"
        android:layout_alignParentStart="true"
        android:layout_marginTop="42dp"
        android:background="#3e7d02"
        android:textColor="#ffffff" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Explicit Intent Example"
        android:id="@+id/explicit_Intent"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="147dp" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Implicit Intent Example"
        android:id="@+id/implicit_Intent"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true" />

</RelativeLayout>
```

Step 2: Design the UI of second activity activity_second.xml

Now lets design UI of another activity where user will navigate after he click on Explicit Example button. Go to layout folder, create a new activity and name it activity_second.xml.

In this activity we will simply use TextView to tell user he is now on second activity.

Below is the complete code of activity_second.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
```

```
        android:paddingBottom="@dimen/activity_vertical_margin"
        android:background="#CCEAA"
        tools:context="com.example.android.intents.SecondActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="This is Second Activity"
        android:id="@+id/textView"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true" />
</RelativeLayout>
```

Step 3: Implement onClick event for Implicit And Explicit Button inside MainActivity.java

Now we will use setOnClickListener() method to implement OnClick event on both the button. Implicit button will open AbhiAndroid.com homepage in browser and Explicit button will move to SecondActivity.java.

Below is the complete code of MainActivity.java

```
package com.example.android.intents;

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

        explicit_btn = (Button) findViewById(R.id.explicit_Intent);
        implicit_btn = (Button) findViewById(R.id.implicit_Intent);

        //implement Onclick event for Explicit Intent
    }
}
```

```
explicit_btn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        Intent intent = new Intent(getApplicationContext(), SecondActivity.class);
        startActivity(intent);

    }
});  
  
//implement onClick event for Implicit Intent  
  
implicit_btn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        Intent intent = new Intent(Intent.ACTION_VIEW);
        intent.setData(Uri.parse("http://www.abhiandroid.com"));
        startActivity(intent);
    }
});  
  
}  
}
```

Step 4: Create A New JAVA class name SecondActivity

Now we need to create another SecondActivity.java which will simply open the layout of activity_second.xml . Also we will use Toast to display message that he is on second activity.

Below is the complete code of SecondActivity.java:

```
package com.example.android.intents;  
  
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.Toast;  
  
public class SecondActivity extends AppCompatActivity {  
  
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_second);
```

```
    Toast.makeText(getApplicationContext(), "We are moved to second
Activity",Toast.LENGTH_LONG).show();
}
}
```

Step 5: Manifest file:

Make sure Manifest file has both the MainActivity and SecondActivity listed it. Also here MainActivity is our main activity which will be launched first. So make sure intent-filter is correctly added just below MainActivity.

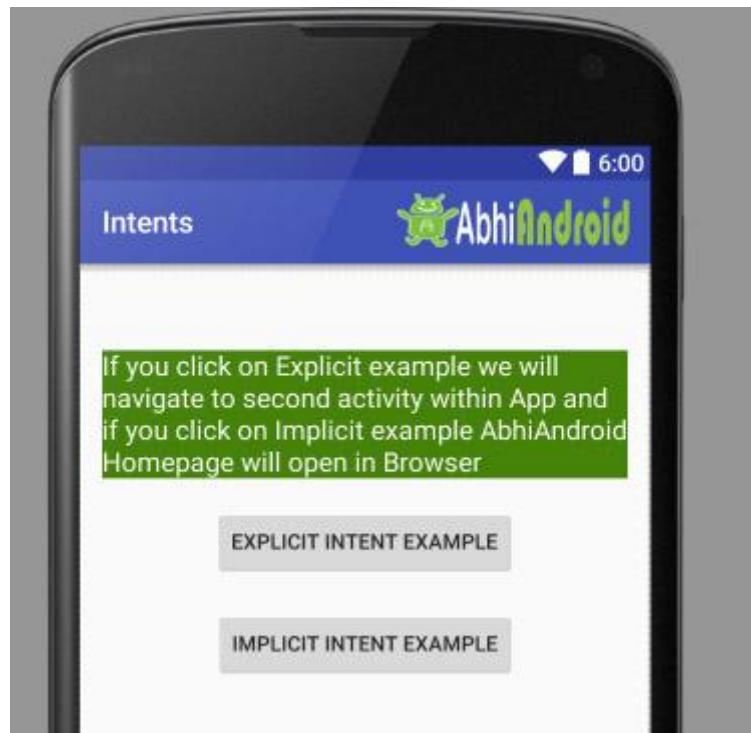
Below is the code of Manifest file:

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

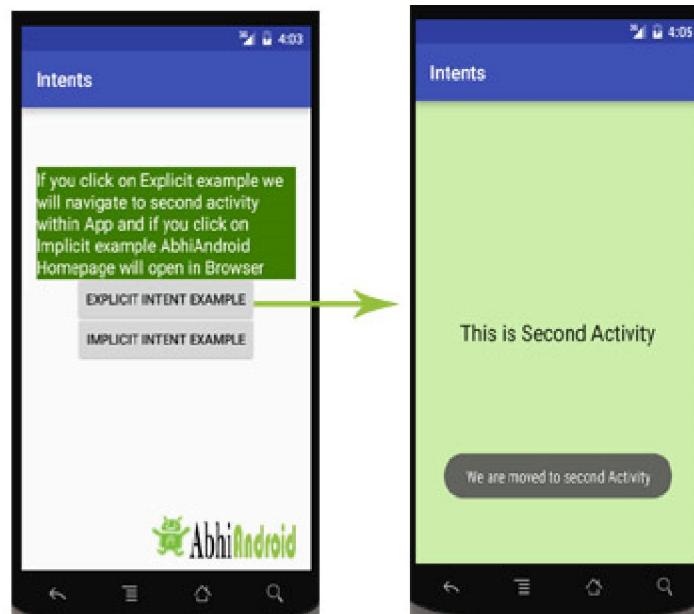
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme" >
        <activity android:name=".MainActivity" >
            <intent-filter>
              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

Output:

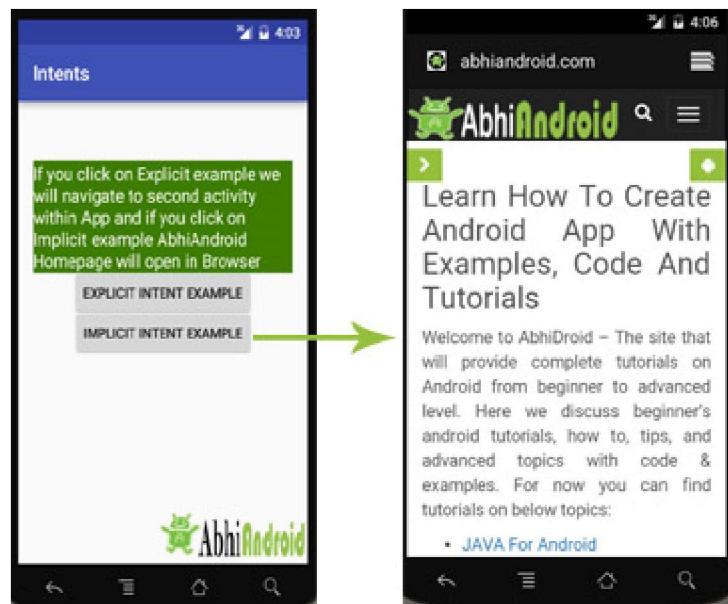
Now run the above program in your Emulator. The App will look like this:



First Click on Explicit Intent Example. The SecondActivity will be open within the App:



Now go back in Emulator and click on Implicit Intent Example. The AbhiAndroid.com homepage will open in Browser (make sure you have internet):



Intent Uses In Android:

Android uses Intents for facilitating communication between its components like Activities, Services and Broadcast Receivers.

Intent for an Activity:

Every screen in Android application represents an activity. To start a new activity you need to pass an Intent object to `startActivity()` method. This Intent object helps to start a new activity and passing data to the second activity.

Intent for Services:

Services work in background of an Android application and it does not require any user Interface. Intents could be used to start a Service that performs one-time task(for example: Downloading some file) or for starting a Service you need to pass Intent to `startService()` method.

Intent for Broadcast Receivers:

There are various message that an app receives, these messages are called as Broadcast Receivers. (For example, a broadcast message could be initiated to intimate that the file downloading is completed and ready to use). Android system initiates some broadcast message on several events, such as System Reboot, Low Battery warning message etc.

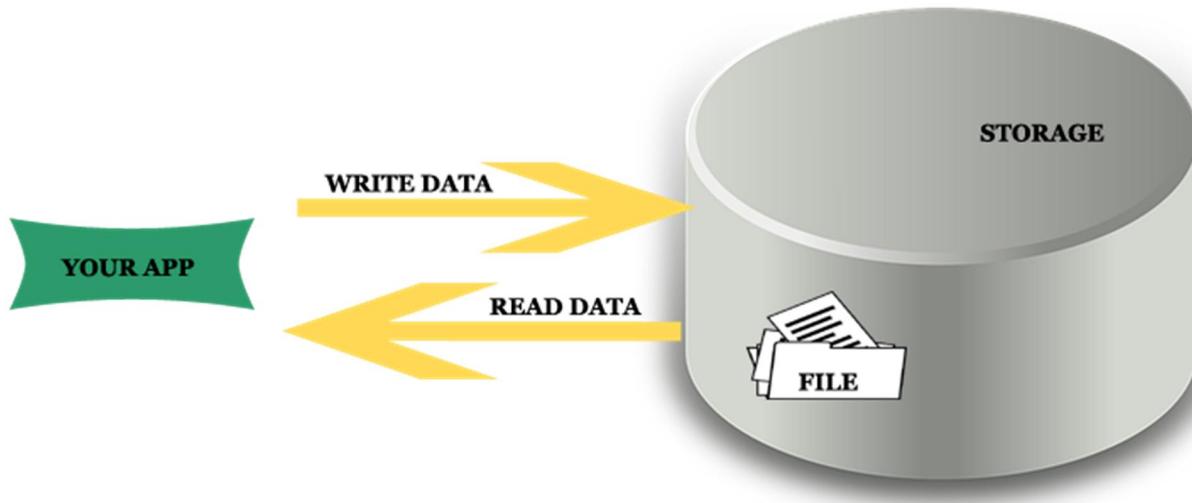
Importance of using Intents in Android Applications:

Whenever you need to navigate to another activity of your app or you need to send some information to next activity then we can always prefer to Intents for doing so.

Intents are really easy to handle and it facilitates communication of components and activities of your application. Moreover you can communicate to another application and send some data to another application using Intents.

Internal Storage

In this tutorial we are going to learn about internal storage of data/files in Android App using example or you can say the primary memory of your phone. It is also important to note that the data is stored in a file specified by the user but user can not access that file, also this file can only be accessed by the application itself.



Important Points About Internal Storage In Android:

1. The stored data in memory is allowed to read and write files.
2. When files are stored in internal storage these file can only be accessed by the application itself not by other applications.
3. These files in storage exist till the application stays over the device, as you uninstall associated files get removed automatically.
4. The files are stored in directory data/data which is followed by the application package name.
5. User can explicitly grant the permission to other apps to access files.
6. To make the data private i.e you can use MODE_PRIVATE as discussed below about the modes.
7. Technique is best suited when data can only be access by the application neither by the user nor by other apps.
8. The data is stored in a file which contains data in bit format so it's required to convert data before adding it to a file or before extracting from a file.

Modes of Internal Storage

MODE_PRIVATE — In private mode the data stored earlier is always overridden by the current data i.e every time you try to commit a new write to a file which removes or override the previous content. We have used MODE_PRIVATE in the example at the end of this article.

MODE_APPEND — In this mode the data is append to the existing content i.e keep adding data.

Write data to file in Internal Storage:

- Define the filename as string and also define data you wanna write to file as string or in any format generated from app or any other source.
- Use FileOutputStream method by creating its object as defined.
- Convert data into byte stream before writing over file because file accepts only byte format further close the file using file object.

```
String File_Name= "Demo.txt"; //gives file name
String Data="Hello!!"; //define data

FileOutputStream fileobj = openFileOutput( File_Name, Context.MODE_PRIVATE);
byte[] ByteArray = Data.getBytes(); //Converts into bytes stream
fileobj.write(ByteArray); //writing to file
fileobj.close(); //File closed
```

Internal Storage Example In Android Studio

Below is the example to show how user can used internal memory for data storage. Here we are creating two activities, the first activity contain the form that will store data in file and second is used to load data that is saved before.



Step 1: Create a new project and name it InternalStorageDemo.

Step 2: Open res -> layout -> activity_main.xml (or) main.xml and add following code:

In this code simply add textView , editText and button with onclick functionality.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.internalstoragedemo.MainActivity">

    <TextView
        android:text="@string/name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:layout_alignParentTop="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_marginLeft="51dp"
        android:layout_marginStart="51dp"
        android:layout_marginTop="59dp"
        android:id="@+id/txtname"
        android:textStyle="bold|italic"
        android:textSize="18sp" />

<TextView
    android:text="@string/password"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/txtname"
    android:layout_alignLeft="@+id/txtname"
    android:layout_alignStart="@+id/txtname"
    android:layout_marginTop="56dp"
    android:id="@+id/txtpass"
    android:textStyle="bold|italic"
    android:textSize="18sp" />

<EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:inputType="textPersonName"
    android:ems="8"
    android:layout_alignParentTop="true"
    android:layout_toRightOf="@+id/txtpass"
    android:layout_toEndOf="@+id/txtpass"
    android:layout_marginLeft="21dp"
    android:layout_marginStart="21dp"
    android:layout_marginTop="48dp"
    android:id="@+id/editName" />

<EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:inputType="textPassword"
    android:ems="10"
    android:layout_below="@+id/editName"
    android:layout_alignLeft="@+id/editName"
    android:layout_alignStart="@+id/editName"
    android:layout_marginTop="35dp"
    android:id="@+id/editPass" />

<Button
    android:text="@string/save"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/editPass"
    android:layout_alignLeft="@+id/txtpass"
    android:layout_alignStart="@+id/txtpass"
    android:layout_marginTop="86dp"
    android:id="@+id/button"
    android:onClick="save"/> // OnClick "save"
```

```
<Button
    android:text="@string/next"
    android:la
    android:layout_height="wrap_content"
    android:layout_alignTop="@+id/button"
    android:layout_alignRight="@+id/editName"
    android:layout_alignEnd="@+id/editName"
    android:layout_marginRight="25dp"
    android:layout_marginEnd="25dp"
    android:id="@+id/button2"
    android:onClick="next"/> // OnClick "next"
</RelativeLayout>
```

Step 3: Open src -> package -> MainActivity.java

In this step we open MainActivity and add the functions defined over button onclick i.e save or next. The save function get the data from edittext and save it in byte format inside file. Here we also used Toast to display the path where file is stored with file name. The next function uses intent to move to the next activity associated with it.

```
package com.example.internalstoragedemo;

import android.content.Context;
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;

public class MainActivity extends AppCompatActivity {
    EditText editname,editpass;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        editname = (EditText) findViewById(R.id.editName);
        editpass= (EditText) findViewById(R.id.editPass);
    }

    public void save(View view) // SAVE
    {
        File file= null;
        String name = editname.getText().toString();
        String password = editpass.getText().toString();

        FileOutputStream fileOutputStream = null;
```

```
try {
    name = name + " ";
    file = getFilesDir();
    fileOutputStream = openFileOutput("Code.txt", Context.MODE_PRIVATE); //MODE
PRIVATE
    fileOutputStream.write(name.getBytes());
    fileOutputStream.write(password.getBytes());
    Toast.makeText(this, "Saved \n" + "Path --" + file + "\tCode.txt",
Toast.LENGTH_SHORT).show();
    editname.setText("");
    editpass.setText("");
    return;
} catch (Exception ex) {
    ex.printStackTrace();
} finally {
    try {
        fileOutputStream.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}

public void next( View view) //NEXT
{
    Toast.makeText(this,"NEXT", Toast.LENGTH_SHORT).show();
    Intent intent= new Intent(this, Main2Activity.class);
    startActivity(intent);
}
```

Step 4: Open res -> layout -> activity_main2.xml (or) main2.xml and add following code:

In this activity the layout is just similar with the main.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main2"
    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="com.example.internalstoragedemo.Main2Activity">

    <TextView
        android:text="@string/getname"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="10dp" />

```

```
        android:layout_alignParentTop="true"
        android:layout_alignRight="@+id/button3"
        android:layout_alignEnd="@+id/button3"
        android:layout_marginRight="11dp"
        android:layout_marginEnd="11dp"
        android:layout_marginTop="76dp"
        android:id="@+id/textView3"
        android:textSize="18sp"
        android:textStyle="bold|italic" />

<TextView
    android:text="@string/getpassword"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/textView3"
    android:layout_alignRight="@+id/textView3"
    android:layout_alignEnd="@+id/textView3"
    android:layout_marginTop="33dp"
    android:id="@+id/textView4"
    android:textStyle="bold|italic"
    android:textSize="18sp" />

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_above="@+id/textView4"
    android:layout_alignLeft="@+id/button4"
    android:layout_alignStart="@+id/button4"
    android:id="@+id/getname"
    android:textStyle="bold|italic"
    android:textSize="18sp" />

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBottom="@+id/textView4"
    android:layout_alignLeft="@+id/getname"
    android:layout_alignStart="@+id/getname"
    android:id="@+id/getpass"
    android:textStyle="bold|italic"
    android:textSize="18sp" />

<Button
    android:text="@string/load"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/button3"
    android:layout_marginLeft="35dp"
    android:layout_marginStart="35dp"
    android:onClick="load"
    android:layout_below="@+id/textView4"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_marginTop="80dp" />

<Button
    android:text="@string/back"
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginRight="54dp"
    android:layout_marginEnd="54dp"
    android:id="@+id/button4"
    android:onClick="back"
    android:layout_alignBaseline="@+id/button3"
    android:layout_alignBottom="@+id/button3"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true" />

</RelativeLayout>
```

Step 5:

Open src -> package -> MainActivity2.java

In this step we open MainActivity2 and add the functions defined over button's onclick i.e load or back. The load function retrieve the data from the file, add it to the StringBuffer and further set the text over the textView's. The back function contain intent to move back to main activity.

```
package com.example.internalstoragedemo;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.TextView;
import android.widget.Toast;
import java.io.FileInputStream;

public class Main2Activity extends AppCompatActivity {

    TextView getname, getpass;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main2);
        getname = (TextView) findViewById(R.id.getname);
        getpass = (TextView) findViewById(R.id.getpass);
    }
    public void load(View view)
    {
        try {
            FileInputStream fileInputStream = openFileInput("Code.txt");
            int read = -1;
            StringBuffer buffer = new StringBuffer();
            while (read > -1) {
                read = fileInputStream.read();
                if (read > -1) {
                    buffer.append((char) read);
                }
            }
            getname.setText(buffer.toString());
            getpass.setText(buffer.toString());
        } catch (Exception e) {
            Log.e("Error", e.getMessage());
        }
    }
}
```

```
        while((read =fileInputStream.read())!= -1){
            buffer.append((char)read);
        }
        Log.d("Code", buffer.toString());
        String name = buffer.substring(0,buffer.indexOf(" "));
        String pass = buffer.substring(buffer.indexOf(" ")+1);
        getname.setText(name);
        getpass.setText(pass);
    } catch (Exception e) {
        e.printStackTrace();
    }
    Toast.makeText(this,"Loaded", Toast.LENGTH_SHORT).show();
}

public void back( View view)
{
    Toast.makeText(this,"Back", Toast.LENGTH_SHORT).show();
    Intent intent= new Intent(this, MainActivity.class);
    startActivity(intent);
}
}
```

Output:

Now run the app and you will see login form on the screen. Add data in the fields and save it. Further click next to move to next activity and load data that is saved before.

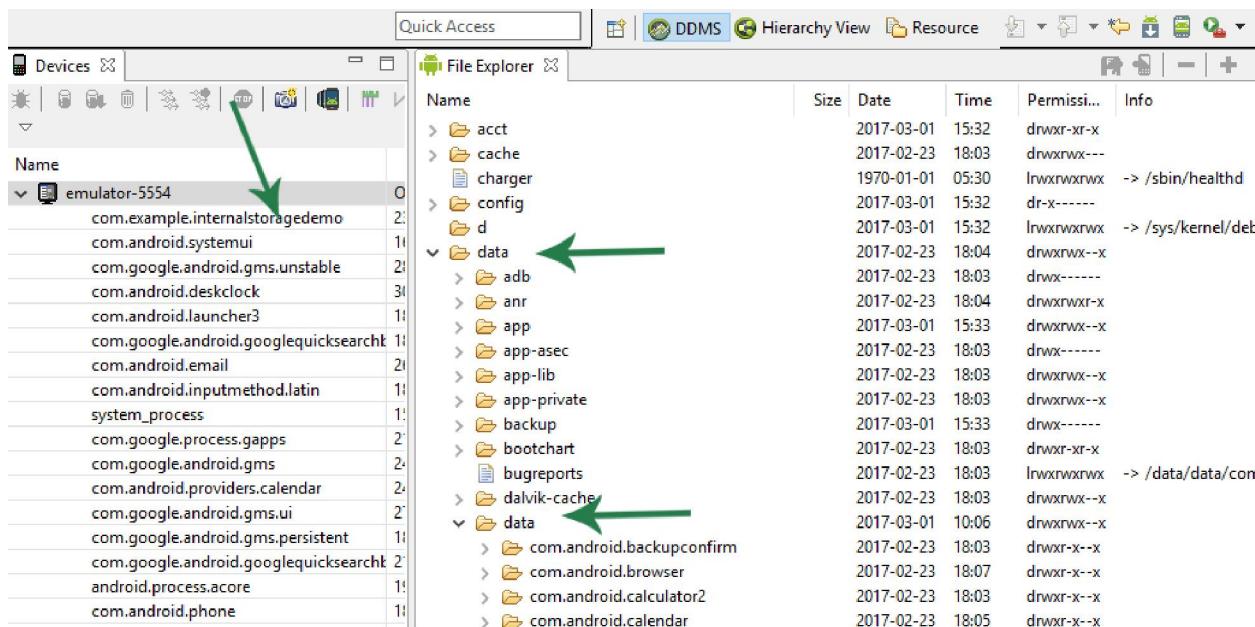
Bonus Tip

You can view the file which contain data that you stored, if you noticed that while saving a toast appears with the path where file is stored let's locate that file.

Important Note: The file will only be viewable if you run the App using Android Studio emulator and not in genymotion or any other external alternative emulator.

Move to Tools -> Android -> Android Device Monitor and open **Android Device Monitor**.

Select the process i.e internalstoragedemo, select File Explorer find
data/data/[package_name]/files/[file.txt]

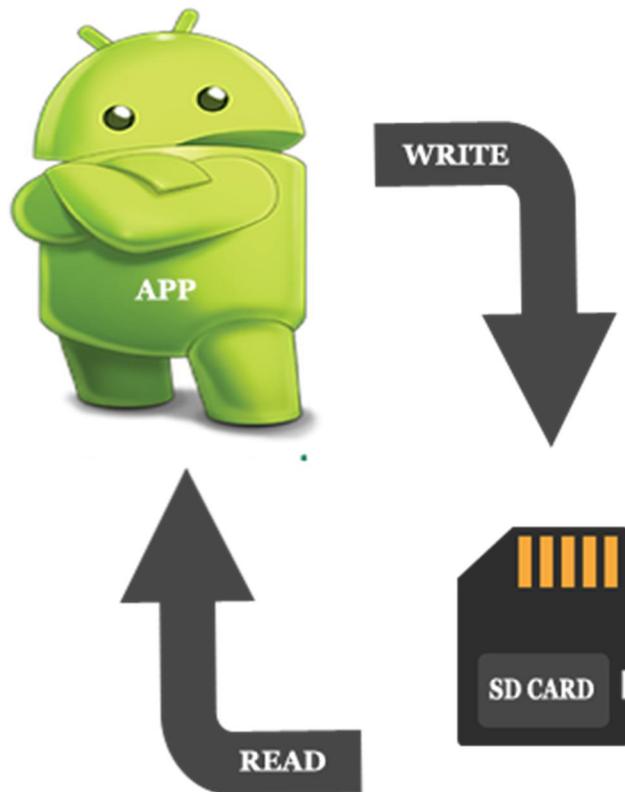


In given example it is data/data/com.example.internalstoragedemo/files/Code.txt.

> com.example.android.softkeyboard	2017-02-23	18:05	drwxr-x--x
> com.example.internalstoragedemo	2017-03-01	10:14	drwxr-x--x
> cache	2017-03-01	10:06	drwxrwx--x
> code_cache	2017-03-01	10:06	drwxrwx--x
> files	2017-03-01	10:14	drwxrwx--x
Code.txt	1	2017-03-01	16:01
> com.google.android.apps.maps	2017-03-01	15:32	drwxr-x--x
> com.google.android.apps.photos	2017-03-01	15:32	drwxr-x--x
> com.google.android.gms	2017-03-01	15:33	drwxr-x--x
> com.google.android.googlequicksea	2017-03-01	15:32	drwxr-x--x
> com.google.android.gsf	2017-02-23	18:05	drwxr-x--x

External Storage

In this tutorial we gonna study about storage of data/files in android external storage or you can say the secondary memory/SD card of your phone. The data is stored in a file specified by the user itself and user can access these file. These files are only accessible till the application exits or you have SD card mounted on your device.



Important Note: It is necessary to add external storage the permission to read and write. For that you need to add permission in android Manifest file.

Open AndroidManifest.xml file and add permissions to it just after the package name.

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

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

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

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

    <application
        android:allowBackup="true"
```

External Storage Availability In Android Studio

To avoid crashing app it is required to check before whether storage SD card is available for read & write operations. `getExternalStorageState()` method is used to determine the state of the storage media i.e SD card is mounted, is it readable , it is writable etc.. all this kind of information.

```
boolean isAvailable= false;
boolean isWritable= false;
boolean isReadable= false;
String state = Environment.getExternalStorageState();
if(Environment.MEDIA_MOUNTED.equals(state)){
    // Read and write operation possible
    isAvailable= true;
    isWritable= true;
    isReadable= true;
} else if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)){
    // Read operation possible
    isAvailable= true;
    isWritable= false;
    isReadable= true;
} else {
    // SD card not mounted
    isAvailable = false;
    isWritable= false;
    isReadable= false; }
```

Methods to Store Data In Android:

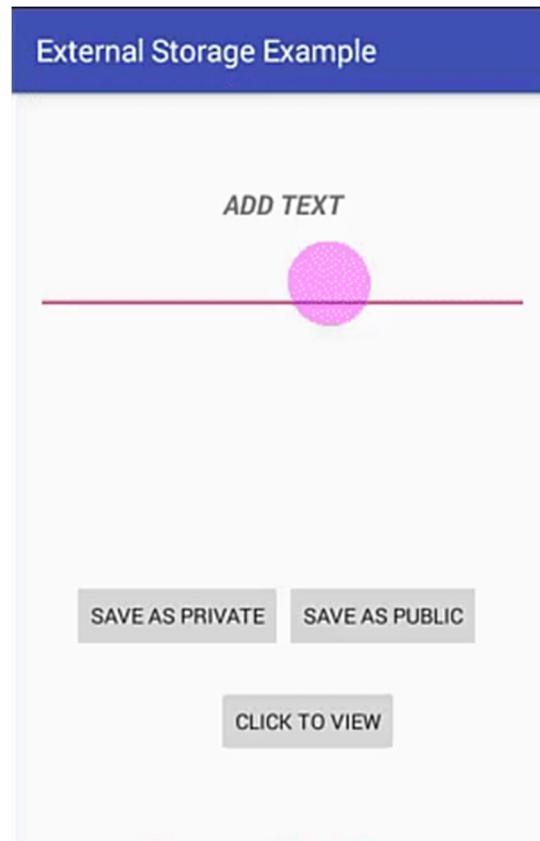
getExternalStorageDirectory() – Older way to access external storage in API Level less than 7. It is absolute now and not recommended. It directly get the reference to the root directory of your external storage or SD Card.

getExternalFilesDir(String type) – It is recommended way to enable us to create private files specific to app and files are removed as app is uninstalled. Example is app private data.

getExternalStoragePublicDirectory() : This is current recommended way that enable us to keep files public and are not deleted with the app uninstallation. Example images clicked by the camera exists even we uninstall the camera app.

External Storage Example In Android Studio

Below is the example to show how user can use external memory for data storage. Here we are creating two activities, the first activity contain the form that will store data in file and second is used to load data which is saved.



Step 1: Create a new project and name it ExternalStorageExample.

Step 2: Open res -> layout -> activity_main.xml (or) main.xml and add following code:

In this code simply add textView , edittext and button for onclick functionality.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.externalstorageexample.MainActivity">

    <TextView
        android:id="@+id/textView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_marginTop="46dp"
        android:gravity="center"
        android:text="@string/add_text"
        android:textSize="18sp"
        android:textStyle="bold|italic" />

    <Button
        android:id="@+id/button4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/button"
        android:layout_alignParentBottom="true"
        android:layout_alignStart="@+id/button" />
```

```
        android:layout_marginBottom="52dp"
        android:layout_marginLeft="96dp"
        android:layout_marginStart="96dp"
        android:onClick="next"
        android:text="@string/click_to_view" />

<Button
    android:id="@+id/button2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_above="@+id/button4"
    android:layout_alignParentEnd="true"
    android:layout_alignParentRight="true"
    android:layout_marginBottom="22dp"
    android:layout_marginEnd="32dp"
    android:layout_marginRight="32dp"
    android:onClick="savePublic"
    android:text="@string/save_as_public" />

<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/button2"
    android:layout_alignBottom="@+id/button2"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_marginLeft="24dp"
    android:layout_marginStart="24dp"
    android:onClick="savePrivate"
    android:text="@string/save_as_private" />

<EditText
    android:id="@+id/editText2"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_below="@+id/textView"
    android:layout_marginTop="16dp"
    android:ems="10"
```

```
    android:gravity="center_vertical|center"
    android:inputType="textMultiLine" />

</RelativeLayout>
```

Step 3:

Open src -> package -> MainActivity.java

In this step we open MainActivity and add the functions defined over buttons onclick. The savePrivate and savePublic function get the data from edittext and save it in byte format in file. Also toast is used to display the path where file is stored with file name. The next function uses intent to move to the next activity associated with it.

Important Note: There is a difference while retrieving data with different API level. The permission concept has changed since API 23. Before API level 23 the user was asked during installation, after API level 23 the user is asked during runtime. So an additional runtime permission is added in the application operating over the API level above 23.

```
package com.example.externalstorageexample;

import android.Manifest;
import android.content.Intent;
import android.os.Environment;
import android.support.v4.app.ActivityCompat;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;

public class MainActivity extends AppCompatActivity {
    EditText editText;
    private int STORAGE_PERMISSION_CODE = 23;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        editText = (EditText) findViewById(R.id.editText2);
    }
}
```

```

public void next(View view) {
    Intent intent = new Intent(MainActivity.this, Main2Activity.class);
    startActivity(intent);
}

public void savePublic(View view) {
    //Permission to access external storage
    ActivityCompat.requestPermissions(this, new
String[]{Manifest.permission.READ_EXTERNAL_STORAGE}, STORAGE_PERMISSION_CODE);
    String info = editText.getText().toString();
    File folder =
Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_ALARMS);// Folder Name
    File myFile = new File(folder, "myData1.txt");// Filename
    writeData(myFile, info);
    editText.setText("");
}

public void savePrivate(View view) {
    String info = editText.getText().toString();
    File folder = getExternalFilesDir("AbhiAndroid");// Folder Name
    File myFile = new File(folder, "myData2.txt");// Filename
    writeData(myFile, info);
    editText.setText("");
}

private void writeData(File myFile, String data) {
    FileOutputStream fileOutputStream = null;
    try {
        fileOutputStream = new FileOutputStream(myFile);
        fileOutputStream.write(data.getBytes());
        Toast.makeText(this, "Done" + myFile.getAbsolutePath(),
Toast.LENGTH_SHORT).show();
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        if (fileOutputStream != null) {
            try {
                fileOutputStream.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
}

```

Step 4: Open res -> layout -> activity_main2.xml (or) main2.xml and add following code:

In this activity the layout is just similar with the main.xml.

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

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main2"
    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="com.example.externalstorageexample.Main2Activity">

    <TextView
        android:id="@+id/getText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true"
        android:layout_marginTop="33dp"
        android:gravity="center"
        android:text=""
        android:textSize="18sp"
        android:textStyle="bold|italic" />

    <Button
        android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_above="@+id/button5"
        android:layout_alignParentEnd="true"
        android:layout_alignParentRight="true"
        android:onClick="getPublic"
        android:text="@string/show_public_data" />

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_below="@+id/getText"
        android:layout_marginTop="178dp"
        android:onClick="getPrivate"
        android:text="@string/show_private_data"
        tools:ignore="UnknownId" />

    <Button
        android:id="@+id/button5"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/button2"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="59dp"
        android:onClick="back"
        android:text="@string/back" />

</RelativeLayout>
```

Step 5: Open src -> package -> MainActivity2.java

In this step we open MainActivity2 and add the functions defined over button's onclick. The getPrivate and getPublic function retrieve the data from the file, add it to the buffer and further set the text over the textView's. The back function contain intent to move back to main activity.

```
package com.example.externalstorageexample;

import android.content.Intent;
import android.os.Environment;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;
import java.io.*;

public class Main2Activity extends AppCompatActivity {
    TextView showText;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main2);
        showText = (TextView) findViewById(R.id.getText);

    }

    public void back(View view) {
        Intent intent = new Intent(Main2Activity.this, MainActivity.class);
        startActivity(intent);
    }

    public void getPublic(View view) {
        File folder =
Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_ALARMS); // Folder Name
        File myFile = new File(folder, "myData1.txt"); // Filename
        String text = getdata(myFile);
        if (text != null) {
            showText.setText(text);
        } else {
            showText.setText("No Data");
        }
    }

    public void getPrivate(View view) {
        File folder = getExternalFilesDir("AbhiAndroid"); // Folder Name
        File myFile = new File(folder, "myData2.txt"); // Filename
```

```
String text = getdata(myFile);
if (text != null) {
    showText.setText(text);
} else {
    showText.setText("No Data");
}

private String getdata(File myfile) {
    FileInputStream fileInputStream = null;
    try {
        fileInputStream = new FileInputStream(myfile);
        int i = -1;
        StringBuffer buffer = new StringBuffer();
        while ((i = fileInputStream.read()) != -1) {
            buffer.append((char) i);
        }
        return buffer.toString();
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        if (fileInputStream != null) {
            try {
                fileInputStream.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
    return null;
}
```

Output:

Now run the app and you will see form on the screen. Add data in the field and save it. Further click next to move to next activity and get data that is saved before as public or private.

Shared Preference

Shared Preference in Android are used to save data based on key-value pair. If we go deep into understanding of word: shared means to distribute data within and preference means something important or preferable, so SharedPreferences data is shared and preferred data.

Shared Preference can be used to save primitive data type: string, long, int, float and Boolean.

What Is Preference File?

Before we begin explaining shared preference, it is important to understand preference file.

A preference file is actually a xml file saved in internal memory of device. Every application have some data stored in memory in a directory data/data/application package name i.e data/data/com.abhiandroid.abhiapp so whenever getSharedPreferences(String name,int mode)

function get called it validates the file that if it exists or it doesn't then a new xml is created with passed name.

Two Ways To Save Data Through Shared Preference:

There are two different ways to save data in Android through Shared Preferences – One is using Activity based preferences and other is creating custom preferences.

Activity Preferences:

- For activity preferences developer have to call function `getPreferences (int mode)` available in Activity class
- Use only when one preference file is needed in Activity
- It doesn't require name as it will be the only preference file for this activity
- Developer doesn't usually prefer using this even if they need only one preference file in Activity. They prefer using custom `getSharedPreferences(String name,int mode)`.

To use activity preferences developer have to call function `getPreferences (int mode)` available in Activity class. The function `getPreferences(int mode)` call the other function used to create custom preferences i.e `getSharedPreferences(String name,int mode)`. Just because Activity contains only one preference file so `getPreferences(int mode)` function simply pass the name of Activity class to create a preference file.

Important Note: Mode are discussed in Custom preferences.

Custom Preferences:

Developer needs to use `getSharedPreferences(String name,int mode)` for custom preferences

Used in cases when more than one preference file required in Activity

Name of the preference file is passed in first parameter

Custom Preferences can be created by calling function `getSharedPreferences(String name,int mode)`, it can be called from anywhere in the application with reference of Context. Here name is any preferred name for example: User, Book etc. and mode is used to set kind of privacy for file.

Mode And Its Type In Shared Preference:

To create activity based preferences or custom preferences developer have to pass mode to let the system know about privacy of preference file.

Before we begin a brief discussion on Context

Context is an abstract class used to get global information in Android Application resources like strings, values, drawables, assets etc. Here modes are declared and static final constant in Context class.

There are three types of Mode in Shared Preference:

Context.MODE_PRIVATE – default value (Not accessible outside of your application)

Context.MODE_WORLD_READABLE – readable to other apps

Context.MODE_WORLD_WRITEABLE – read/write to other apps

MODE_PRIVATE – It is a default mode. MODE_PRIVATE means that when any preference file is created with private mode then it will not be accessible outside of your application. This is the most common mode which is used.

MODE_WORLD_READABLE – If developer creates a shared preference file using mode world readable then it can be read by anyone who knows its name, so any other outside application can easily read data of your app. This mode is very rarely used in App.

MODE_WORLD_WRITEABLE – It's similar to mode world readable but with both kind of accesses i.e read and write. This mode is never used in App by Developer.

Important Google Recommend Naming Convention For Preference File:

Please make sure to follow recommendation by Google while giving preference file name.
Google recommends to give name as application package name + preferred name.

For example: if your application package name is `com.abhiandroid.abhiapp` then preference could be `com.abhiandroid.abhiapp.User` where `com.abhiandroid.abhiapp.User` is the name of preference file.

Creating Shared Preference:

As discussed above, to create shared preference developer needs to call `getPreferences(int mode)` or `getSharedPreferences(String name,int mode)` method. Both of the function return `SharedPreferences` object to deal with save and get values from preference file.

`SharedPreferences` is a singleton class means this class will only have a single object throughout the application lifecycle. However there can multiple preference files so all the preference files can be read and write using `SharedPreferences` class.

Save Data In SharedPreferences:

Step 1:

To save data in `SharedPreferences` developer need to called `edit()` function of `SharedPreferences` class which returns `Editor` class object.

Step 2:

`Editor` class provide different function to save primitive time data. For example, `Editor` have all primitive data function including String type data as `.putInt(String keyName,int value)`.

Common primitive function format is: `put + Primitive Type name (String keyname, Primitive Type value)`

Step 3:

Now make sure that key name should be unique for any values you save otherwise it will be overrided. To exactly save the values you put in different primitive functions you must call commit() function of Editor.

Read or get data from SharedPreferences

Step 1: To read data first developer have to get reference of SharedPreferences object by calling getPreferences (int mode) or getSharedPreferences (String name,int mode).

Step 2: Then developer can get values using SharedPreferences object by calling different primitive type function starting with get+Primitive Type name. Here every function has an additional parameter for every Primitive Type as default value in case there is no value found corresponding to passed key.

For example, to get saved int value just call **getInt("GameScore",-1)** function where "GameScore" is key and -1 is a default value in case no value was saved for GameScore.

Remove Data Or Clear All Data

Similar to save data there is a function remove(String key) in SharedPreferences.Editor to remove a particular key based data from preference file

To clear all data just call function clear() available in SharedPreferences.Editor. Make sure to call commit() method to save changes. So clear data will never delete the preference file but make it empty.

Code Of Saving & Retrieving Data in Shared Preference:

```
public static final String PREFS_GAME = "com.abhiandroid.abhiapp.GamePlay";
public static final String GAME_SCORE= "GameScore";

//===== Code to save data =====

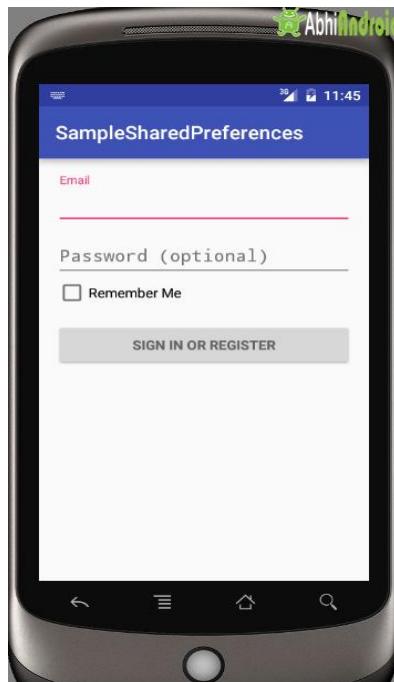
SharedPreferences sp = getSharedPreferences(PREFS_GAME , Context.MODE_PRIVATE);
```

```
sp.edit().putInt(GAME_SCORE,100).commit();  
  
//===== Code to get saved/ retrieve data ======  
  
SharedPreferences sp = getSharedPreferences(PREFS_GAME ,Context.MODE_PRIVATE);  
int sc = sp.getInt(GAME_SCORE,0);  
Log.d("AbhiAndroid","achieved score is "+sc);
```

Shared Preference Example:

Let's use Shared Preference for a very basic purpose of saving login details of a person i.e. email and password on his device. So he don't have to re-enter his login details every time he opens the App.

Below is the final output we will create and use Shared Preference to save Signin Details:



Step 1: Create a new project and create an login Activity activity_login.xml. In this create a login UI asking user email and password with an option of remember me checkbox. Also a button displaying Signin or Register.

Below is the complete code login UI activity_login.xml

```
<LinearLayout 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:gravity="center_horizontal"
    android:orientation="vertical"
    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="com.samplesharedpreferences.LoginActivity">

    <!-- Login progress -->
    <ProgressBar
        android:id="@+id/login_progress"
        style="?android:attr/progressBarStyleLarge"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="8dp"
        android:visibility="gone"/>

    <ScrollView
        android:id="@+id/login_form"
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <LinearLayout
            android:id="@+id/email_login_form"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="vertical">

            <android.support.design.widget.TextInputLayout
                android:layout_width="match_parent"
                android:layout_height="wrap_content">

                <EditText
                    android:id="@+id/email"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:hint="@string/prompt_email"
                    android:inputType="textEmailAddress"
                    android:maxLines="1"
                    android:singleLine="true"/>

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

            <android.support.design.widget.TextInputLayout
                android:layout_width="match_parent"
                android:layout_height="wrap_content">

                <EditText
                    android:id="@+id/password"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:inputType="textPassword"/>

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

        </LinearLayout>
    </ScrollView>

```

```
        android:hint="@string/prompt_password"
        android:imeActionId="@+id/login"
        android:imeActionLabel="@string/action_sign_in_short"
        android:imeOptions="actionUnspecified"
        android:inputType="textPassword"
        android:maxLines="1"
        android:singleLine="true"/>

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

    <CheckBox
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Remember Me"
        android:id="@+id/checkBoxRememberMe"/>

    <Button
        android:id="@+id/email_sign_in_button"
        style="?android:textAppearanceSmall"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"
        android:text="@string/action_sign_in"
        android:textStyle="bold"/>

</LinearLayout>
</ScrollView>
</LinearLayout>
```

Step 2: Now lets code the LoginActivity.java where we will create a Login form. Below is the complete code of LoginActivity.java with explanation included in comment.

```
package com.samplesharedpreferences;

import android.animation.Animator;
import android.animation.AnimatorListenerAdapter;
import android.annotation.TargetApi;
import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.pm.PackageManager;
import android.support.annotation.NonNull;
import android.support.design.widget.Snackbar;
import android.support.v7.app.AppCompatActivity;
import android.app.LoaderManager.LoaderCallbacks;
import android.content.ContentResolver;
import android.content.CursorLoader;
import android.content.Loader;
import android.database.Cursor;
import android.net.Uri;
import android.os.AsyncTask;
import android.os.Build.VERSION;
```

```
import android.os.Build;
import android.os.Bundle;
import android.provider.ContactsContract;
import android.text.TextUtils;
import android.view.KeyEvent;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.inputmethod.EditorInfo;
import android.widget.ArrayAdapter;
import android.widget.AutoCompleteTextView;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.EditText;
import android.widget.TextView;

import java.util.ArrayList;
import java.util.List;

import static android.Manifest.permission.READ_CONTACTS;

/**
 * A login screen that offers login via email/password.
 */
public class LoginActivity extends AppCompatActivity {

    // UI references.
    private EditText mEmailView;
    private EditText mPasswordView;

    private CheckBox checkBoxRememberMe;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login);
        // Set up the login form.
        mEmailView = (EditText) findViewById(R.id.email);
        mPasswordView = (EditText) findViewById(R.id.password);
        mPasswordView.setOnEditorActionListener(new TextView.OnEditorActionListener() {
            @Override
            public boolean onEditorAction(TextView textView, int id, KeyEvent keyEvent) {
                if (id == R.id.login || id == EditorInfo.IME_NULL) {
                    attemptLogin();
                    return true;
                }
                return false;
            }
        });

        Button mEmailSignInButton = (Button) findViewById(R.id.email_sign_in_button);
        mEmailSignInButton.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View view) {
                attemptLogin();
            }
        });
    }

    private void attemptLogin() {
        String email = mEmailView.getText().toString();
        String password = mPasswordView.getText().toString();

        if (TextUtils.isEmpty(email)) {
            mEmailView.setError("Email is required.");
            return;
        }

        if (TextUtils.isEmpty(password)) {
            mPasswordView.setError("Password is required.");
            return;
        }

        mEmailView.setError(null);
        mPasswordView.setError(null);

        boolean cancel = false;
        DialogInterface.OnClickListener listener = new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int which) {
                if (which == DialogInterface.BUTTON_POSITIVE) {
                    cancel = true;
                }
            }
        };

        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setMessage("Do you want to log in?")
            .setPositiveButton("Yes", listener)
            .setNegativeButton("No", listener);
        AlertDialog dialog = builder.create();
        dialog.show();
        if (cancel) {
            return;
        }

        // ...
    }
}
```

```
checkBoxRememberMe = (CheckBox) findViewById(R.id.checkBoxRememberMe);
//Here we will validate saved preferences
if (!new PrefManager(this).isUserLogedOut()) {
    //user's email and password both are saved in preferences
    startHomeActivity();
}

}

/***
 * Attempts to sign in or register the account specified by the login form.
 * If there are form errors (invalid email, missing fields, etc.), the
 * errors are presented and no actual login attempt is made.
 */
private void attemptLogin() {

    // Reset errors.
    mEmailView.setError(null);
    mPasswordView.setError(null);

    // Store values at the time of the login attempt.
    String email = mEmailView.getText().toString();
    String password = mPasswordView.getText().toString();

    boolean cancel = false;
    View focusView = null;

    // Check for a valid password, if the user entered one.
    if (!TextUtils.isEmpty(password) && !isPasswordValid(password)) {
        mPasswordView.setError(getString(R.string.error_invalid_password));
        focusView = mPasswordView;
        cancel = true;
    }

    // Check for a valid email address.
    if (TextUtils.isEmpty(email)) {
        mEmailView.setError(getString(R.string.error_field_required));
        focusView = mEmailView;
        cancel = true;
    } else if (!isValidEmail(email)) {
        mEmailView.setError(getString(R.string.error_invalid_email));
        focusView = mEmailView;
        cancel = true;
    }

    if (cancel) {
        // There was an error; don't attempt login and focus the first
        // form field with an error.
        focusView.requestFocus();
    } else {
        // save data in local shared preferences
        if (checkBoxRememberMe.isChecked())
            saveLoginDetails(email, password);
        startHomeActivity();
    }
}
```

```
}

private void startHomeActivity() {
    Intent intent = new Intent(this, HomeActivity.class);
    startActivity(intent);
    finish();
}

private void saveLoginDetails(String email, String password) {
    new PrefManager(this).saveLoginDetails(email, password);
}

private boolean isEmailValid(String email) {
    //TODO: Replace this with your own logic
    return email.contains("@");
}

private boolean isPasswordValid(String password) {
    //TODO: Replace this with your own logic
    return password.length() > 4;
}
}
```

Step 3: Create a new java class for Shared Preference PrefManager. Below is the code of PrefManager.java

```
package com.samplesharedpreferences;

import android.content.Context;
import android.content.SharedPreferences;

/**
 * Class for Shared Preference
 */
public class PrefManager {

    Context context;

    PrefManager(Context context) {
        this.context = context;
    }

    public void saveLoginDetails(String email, String password) {
        SharedPreferences sharedPreferences = context.getSharedPreferences("LoginDetails",
Context.MODE_PRIVATE);
        SharedPreferences.Editor editor = sharedPreferences.edit();
        editor.putString("Email", email);
        editor.putString("Password", password);
        editor.commit();
    }
}
```

```
public String getEmail() {
    SharedPreferences sharedpreferences = context.getSharedPreferences("LoginDetails",
Context.MODE_PRIVATE);
    return sharedpreferences.getString("Email", "");
}

public boolean isUserLogedOut() {
    SharedPreferences sharedpreferences = context.getSharedPreferences("LoginDetails",
Context.MODE_PRIVATE);
    boolean isEmpty = sharedpreferences.getString("Email", "").isEmpty();
    boolean isPasswordEmpty = sharedpreferences.getString("Password", "").isEmpty();
    return isEmpty || isPasswordEmpty;
}
}
```

Step 4: Design the simple Home UI where we will display Welcome message along with his saved email address in Shared preference. Below is the code of content_home.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
    tools:context="com.samplesharedpreferences.HomeActivity"
    tools:showIn="@layout/activity_home">

    <TextView
        android:id="@+id/textViewUser"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:gravity="center"
        android:text="Welcome"/>

</RelativeLayout>
```

Step 5: Manifest file

Make sure in your AndroidManifest.xml Intent Filter for Main and Launcher is set inside .LoginActivity because then only that Activity will open first.

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

    <!-- To auto-complete the email text field in the login form with the user's emails -->
    <uses-permission android:name="android.permission.GET_ACCOUNTS"/>
    <uses-permission android:name="android.permission.READ_PROFILE"/>
    <uses-permission android:name="android.permission.READ_CONTACTS"/>

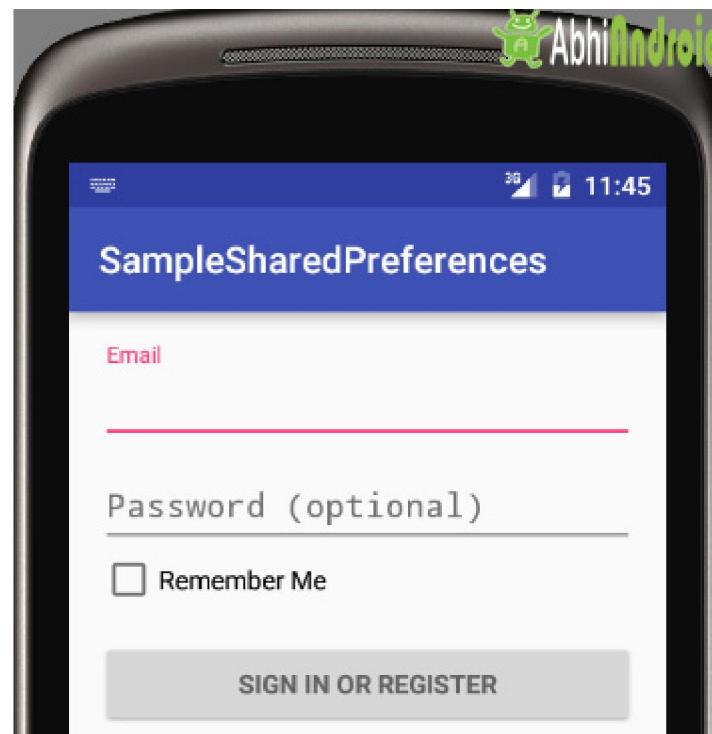
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity
            android:name=".LoginActivity"
            android:label="@string/app_name">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

                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <activity
            android:name=".HomeActivity"
            android:label="@string/title_activity_home"
            android:theme="@style/AppTheme.NoActionBar">
        </activity>
    </application>

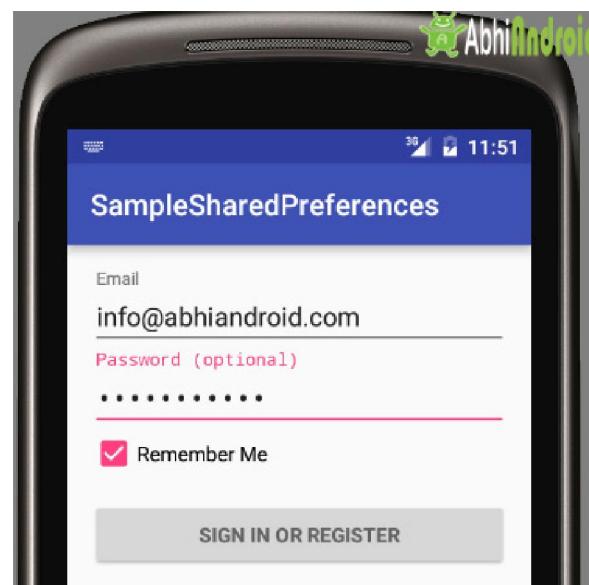
</manifest>
```

Output:

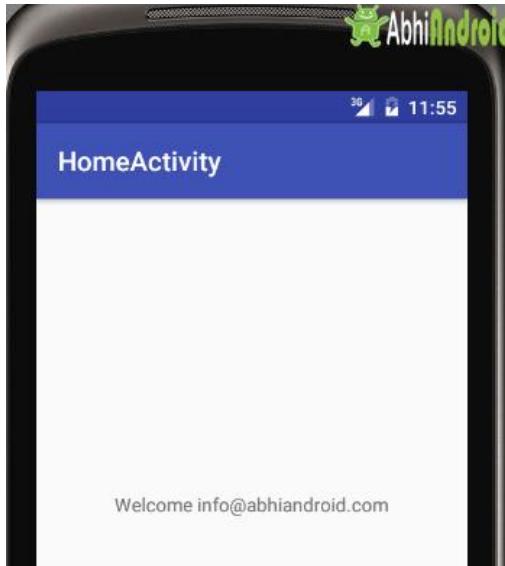
Step 1: Now run the App in Emulator. You will see the below output screen



Step 2: Fill the email, password and click on remember me checkbox. In our case we have filled info@abhiandroid.com and abhiandroid.



Step 3: It will display you the welcome message along with your email ID. Now close the App and reopen it. You will see the same message of Welcome and your email address printed. So here we have stored your email address on your device itself using Shared Preference.

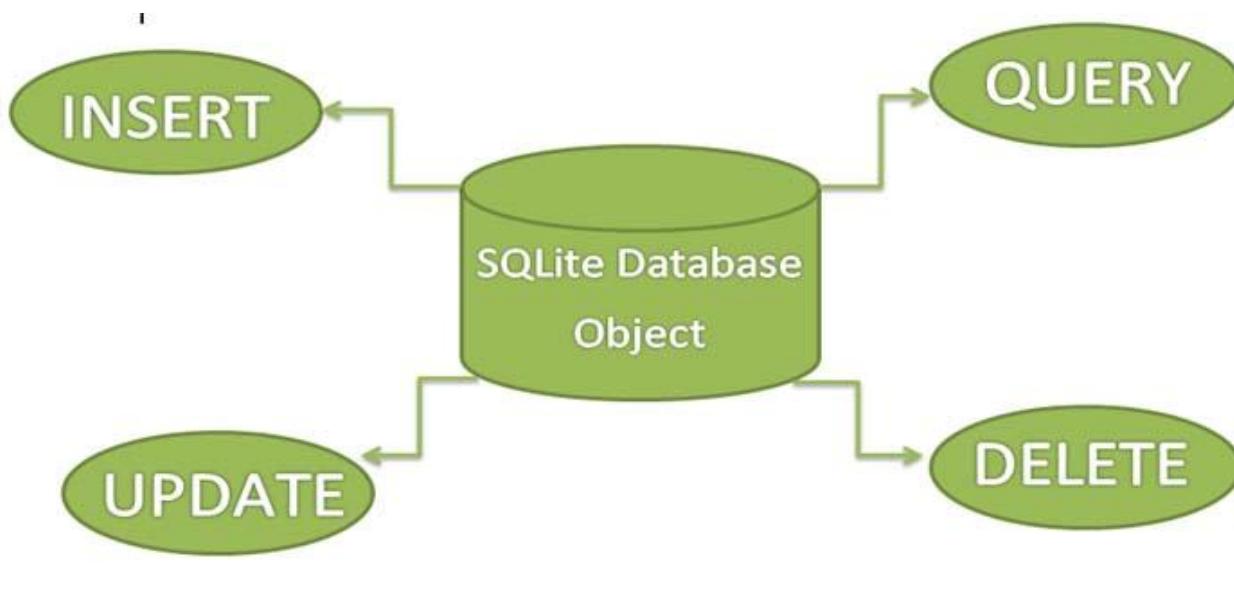


Importance:

During application development there are many situations where we need to store or save some data in a file. So rather than creating our own file and then managing it, Android provides a mechanism to save data using SharedPreferences. To save some little amount of data SharedPreferences can be used like username and password of user. However if there is a need to save large amounts of data, it is always preferable to save data using sqlite.

Sqlite

SQLite is a Structure query base database, open source, light weight, no network access and standalone database. It support embedded relational database features.



Whenever an application needs to store large amount of data then using sqlite is more preferable than other repository system like SharedPreferences or saving data in files.

Android has built in SQLite database implementation. It is available locally over the device(mobile & tablet) and contain data in text format. It carry light weight data and suitable with many languages. So, it doesn't required any administration or setup procedure of the database.

Important Note – The database created is saved in a directory:

data/data/APP_Name/databases/DATABASE_NAME.

Creating And Updating Database In Android

For creating, updating and other operations you need to create a subclass or SQLiteOpenHelper class. SQLiteOpenHelper is a helper class to manage database creation and version management. It provides two methods `onCreate(SQLiteDatabase db)`, `onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)`.

The SQLiteOpenHelper is responsible for opening database if exist, creating database if it does not exists and upgrading if required. The SQLiteOpenHelper only require the `DATABASE_NAME` to create database. After extending SQLiteOpenHelper you will need to implement its methods `onCreate`, `onUpgrade` and constructor.

onCreate(SQLiteDatabase sqLiteDatabase) method is called only once throughout the application lifecycle. It will be called whenever there is a first call to `getReadableDatabase()` or `getWritableDatabase()` function available in super SQLiteOpenHelper class. So SQLiteOpenHelper class call the `onCreate()` method after creating database and instantiate `SQLiteDatabase` object. Database name is passed in constructor call.

onUpgrade(SQLiteDatabase db,int oldVersion, int newVersion) is only called whenever there is a updation in existing version. So to update a version we have to increment the value of version variable passed in the superclass constructor.

In `onUpgrade` method we can write queries to perform whatever action is required. In most example you will see that existing table(s) are being dropped and again `onCreate()` method is being called to create tables again. But it's not mandatory to do so and it all depends upon your requirements.

We have to change database version if we have added a new row in the database table. If we have requirement that we don't want to lose existing data in the table then we can write alter table query in the onUpgrade(SQLiteDatabase db,int oldVersion, int newVersion) method.

For more details read: [Insert, Read, Delete & Update Operation In SQLite](#)

SQLite Example In Android Studio

Get Better Understanding of Sqlite Before You Read Example – To get better understanding of SQLite database, it is recommended you read below article first:

- Introduction To SQLite And Installation
- List Of All Operators In SQLite
- Data Type And Commands In SQLite
- List Of All Clauses In SQLite For Defining Specific Condition
- Insert, Read, Delete & Update Operation In SQLite

In this example we simply want to illustrate the insert, update, delete and more operations of SQLite over a table in Android Studio. We created a activity having textView, button and editText over it. Another class which extends SQLiteOpenHelper where the create and insert operations will be carried out. The example contain proper validation like you need to enter data before executing any operation.

Below you can download code, see final output and step by step explanation:



Step 1: Create a New Project and Name it SQLiteOperations.

Step 2: Open res -> layout -> activity_main.xml (or) main.xml and add following code:

In this step we create a layout in our XML file adding textbox, buttons, edittext. On button onclick is defined which associate it with related function.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.sqliteoperations.MainActivity"
    android:background="@android:color/holo_blue_dark">

    <TextView
```

```
        android:text="@string/username"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_marginTop="12dp"
        android:id="@+id/textView"
        android:textSize="18sp"
        android:textStyle="bold|italic"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:gravity="center" />

<EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="textPersonName"
        android:ems="10"
        android:id="@+id/editName"
        android:textStyle="bold|italic"
        android:layout_below="@+id/textView"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true"
        android:hint="Enter Name"
        android:gravity="center_vertical|center" />

<TextView
        android:text="@string/password"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="13dp"
        android:id="@+id/textView2"
        android:textStyle="bold|italic"
        android:textSize="18sp"
        android:layout_below="@+id/editName"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true"
        android:gravity="center"
        android:hint="Enter Password" />

<Button
        android:text="@string/view_data"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/button2"
        android:textSize="18sp"
        android:onClick="viewdata"
        android:textStyle="bold|italic"
        android:layout_alignBaseline="@+id/button"
        android:layout_alignBottom="@+id/button"
        android:layout_alignRight="@+id/button4"
        android:layout_alignEnd="@+id/button4" />

<Button
        android:text="@string/add_user"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/button"
```

```
        android:textStyle="bold|italic"
        android:textSize="18sp"
        android:onClick="addUser"
        android:layout_marginLeft="28dp"
        android:layout_marginStart="28dp"
        android:layout_below="@+id/editPass"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_marginTop="23dp" />

    <Button
        android:text="@string/update"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/button3"
        android:onClick="update"
        android:textStyle="normal|bold"
        android:layout_below="@+id/editText3"
        android:layout_alignLeft="@+id/button4"
        android:layout_alignStart="@+id/button4"
        android:layout_marginTop="13dp" />

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:inputType="textPersonName"
        android:ems="10"
        android:id="@+id/editText6"
        android:layout_alignTop="@+id/button4"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:freezesText="false"
        android:hint="Enter Name to Delete Data"
        android:layout_toLeftOf="@+id/button2"
        android:layout_toStartOf="@+id/button2" />

    <Button
        android:text="@string/delete"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginRight="21dp"
        android:layout_marginEnd="21dp"
        android:id="@+id/button4"
        android:onClick="delete"
        android:textStyle="normal|bold"
        tools:ignore="RelativeOverlap"
        android:layout_marginBottom="41dp"
        android:layout_alignParentBottom="true"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true" />

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:inputType="textPersonName"
        android:ems="10"
        android:layout_marginTop="47dp"
```

```
        android:id="@+id/editText3"
        android:textStyle="bold|italic"
        android:textSize="14sp"
        android:layout_below="@+id/button"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_marginLeft="7dp"
        android:layout_marginStart="7dp"
        android:hint="Current Name" />

<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:inputType="textPassword"
    android:ems="10"
    android:layout_marginTop="11dp"
    android:id="@+id/editPass"
    android:hint="Enter Password"
    android:gravity="center_vertical|center"
    android:textSize="18sp"
    android:layout_below="@+id/textView2"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:textAllCaps="false"
    android:textStyle="normal|bold" />

<EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:inputType="textPersonName"
    android:ems="10"
    android:id="@+id/editText5"
    android:textStyle="bold|italic"
    android:textSize="14sp"
    android:hint="New Name"
    android:layout_alignTop="@+id/button3"
    android:layout_alignLeft="@+id/editText3"
    android:layout_alignStart="@+id/editText3"
    android:layout_marginTop="32dp" />
</RelativeLayout>
```

Step 3 : Now open app -> java -> package -> MainActivity.java and add the below code.

In this step we used the functions that linked via the button click. These functions are defined in other class and are used here. Each function return value that define no of rows updated, using that we defined whether operation is successful or not. Also user need to define valid data to perform operation empty fields will not be entertained and return error .

```
package com.example.sqliteoperations;

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

public class MainActivity extends AppCompatActivity {
    EditText Name, Pass , updateold, updatenew, delete;
    myDbAdapter helper;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Name= (EditText) findViewById(R.id.editName);
        Pass= (EditText) findViewById(R.id.editPass);
        updateold= (EditText) findViewById(R.id.editText3);
        updatenew= (EditText) findViewById(R.id.editText5);
        delete = (EditText) findViewById(R.id.editText6);

        helper = new myDbAdapter(this);
    }
    public void addUser(View view)
    {
        String t1 = Name.getText().toString();
        String t2 = Pass.getText().toString();
        if(t1.isEmpty() || t2.isEmpty())
        {
            Message.message(getApplicationContext(),"Enter Both Name and Password");
        }
        else
        {
            long id = helper.insertData(t1,t2);
            if(id<=0)
            {
                Message.message(getApplicationContext(),"Insertion Unsuccessful");
                Name.setText("");
                Pass.setText("");
            } else
            {
                Message.message(getApplicationContext(),"Insertion Successful");
                Name.setText("");
                Pass.setText("");
            }
        }
    }

    public void viewdata(View view)
    {
        String data = helper.getData();
        Message.message(this,data);
    }

    public void update( View view)
    {
        String u1 = updateold.getText().toString();
        String u2 = updatenew.getText().toString();
```

```
if(u1.isEmpty() || u2.isEmpty())
{
    Message.message(getApplicationContext(),"Enter Data");
}
else
{
    int a= helper.updateName( u1, u2);
    if(a<=0)
    {
        Message.message(getApplicationContext(),"Unsuccessful");
        updateold.setText("");
        updatenew.setText("");
    } else {
        Message.message(getApplicationContext(),"Updated");
        updateold.setText("");
        updatenew.setText("");
    }
}

public void delete( View view)
{
    String uname = delete.getText().toString();
    if(uname.isEmpty())
    {
        Message.message(getApplicationContext(),"Enter Data");
    }
    else{
        int a= helper.delete(uname);
        if(a<=0)
        {
            Message.message(getApplicationContext(),"Unsuccessful");
            delete.setText("");
        }
        else
        {
            Message.message(this, "DELETED");
            delete.setText("");
        }
    }
}
```

Step 4:

In this step create a java class myDbAdapter.java.

In this we define the functions that are used to perform the operations insert, update and delete operations in SQLite. Further this class create another class that will extend the SQLiteOpenHelper. Each function carry equivalent methods that perform operations.

Important Note – According to naming convention it is suggested to define primary key starting with underscore example: `_id`.

```
package com.example.sqliteoperations;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

public class myDbAdapter {
    myDbHelper myhelper;
    public myDbAdapter(Context context)
    {
        myhelper = new myDbHelper(context);
    }

    public long insertData(String name, String pass)
    {
        SQLiteDatabase dbb = myhelper.getWritableDatabase();
        ContentValues contentValues = new ContentValues();
        contentValues.put(myDbHelper.NAME, name);
        contentValues.put(myDbHelper.MyPASSWORD, pass);
        long id = dbb.insert(myDbHelper.TABLE_NAME, null , contentValues);
        return id;
    }

    public String getData()
    {
        SQLiteDatabase db = myhelper.getWritableDatabase();
        String[] columns = {myDbHelper.UID,myDbHelper.NAME,myDbHelper.MyPASSWORD};
        Cursor cursor =db.query(myDbHelper.TABLE_NAME,columns,null,null,null,null,null);
        StringBuffer buffer= new StringBuffer();
        while (cursor.moveToNext())
        {
            int cid =cursor.getInt(cursor.getColumnIndex(myDbHelper.UID));
            String name =cursor.getString(cursor.getColumnIndex(myDbHelper.NAME));
            String password
=cursor.getString(cursor.getColumnIndex(myDbHelper.MyPASSWORD));
            buffer.append(cid+ " " + name + " " + password +" \n");
        }
        return buffer.toString();
    }

    public int delete(String uname)
    {
        SQLiteDatabase db = myhelper.getWritableDatabase();
        String[] whereArgs ={uname};

        int count =db.delete(myDbHelper.TABLE_NAME ,myDbHelper.NAME+" = ?",whereArgs);
        return count;
    }
}
```

```
}

    public int updateName(String oldName , String newName)
    {
        SQLiteDatabase db = myhelper.getWritableDatabase();
        ContentValues contentValues = new ContentValues();
        contentValues.put(myDbHelper.NAME,newName);
        String[] whereArgs= {oldName};
        int count =db.update(myDbHelper.TABLE_NAME,contentValues, myDbHelper.NAME+" = ? ",whereArgs );
        return count;
    }

    static class myDbHelper extends SQLiteOpenHelper
    {
        private static final String DATABASE_NAME = "myDatabase";      // Database Name
        private static final String TABLE_NAME = "myTable";      // Table Name
        private static final int DATABASE_Version = 1; .           // Database Version
        private static final String UID=_id;           // Column I (Primary Key)
        private static final String NAME = "Name";          //Column II
        private static final String MyPASSWORD= "Password";     // Column III
        private static final String CREATE_TABLE = "CREATE TABLE "+TABLE_NAME+
                " ("+UID+" INTEGER PRIMARY KEY AUTOINCREMENT, "+NAME+" VARCHAR(255) ,"+
MyPASSWORD+" VARCHAR(225));";
        private static final String DROP_TABLE ="DROP TABLE IF EXISTS "+TABLE_NAME;
        private Context context;

        public myDbHelper(Context context) {
            super(context, DATABASE_NAME, null, DATABASE_Version);
            this.context=context;
        }

        public void onCreate(SQLiteDatabase db) {

            try {
                db.execSQL(CREATE_TABLE);
            } catch (Exception e) {
                Message.message(context,""+e);
            }
        }

        @Override
        public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
            try {
                Message.message(context,"OnUpgrade");
                db.execSQL(DROP_TABLE);
                onCreate(db);
            }catch (Exception e) {
                Message.message(context,""+e);
            }
        }
    }
}
```

Step 5: In this step create another java class Message.class

In this just simply add toast for displaying message. This is optional, it is just added to again and again defining toast in the example.

```
package com.example.sqliteoperations;

import android.content.Context;
import android.widget.Toast;

public class Message {
    public static void message(Context context, String message) {
        Toast.makeText(context, message, Toast.LENGTH_LONG).show();
    }
}
```

Output

Now run the app and view the functionality added over the buttons.



Add & Retrieve Image From SQLite Database:

To understand how to add or retrieve image from phone external storage to application using SQLite Database.

JSON Parsing

JSON stands for JavaScript Object Notation. It is structured, light weight, human readable and easy to parse. It's a best alternative to XML when our android app needs to interchange data from server. XML parsing is very complex as compare to JSON parsing.

JSON is shorter, quicker and easier way to interchange data from server. JSON is great success and most of the API available support JSON format.

Android Provide us four different classes to manipulate JSON data. These classes are `JSONObject`, `JSONArray`, `JSONStringer` and `JSONTokenizer`.

Sample JSON format:

Below is the sample code of JSON. Its very simple JSON code which gives us the list of users where each object contain the information like user id, name, email, gender and different contact numbers.

```
{  
    "users": [  
        {  
            "id": "1087",  
            "name": "Abhishek Saini",  
            "email": "info@abhiandroid.com",  
            "gender": "male",  
            "contact": {  
                "mobile": "+91 0000000000",  
                "home": "00 000000",  
                "office": "00 000000"  
            }  
        },  
        {  
            "id": "1088",  
            "name": "Gourav",  
            "email": "gourav9188@gmail.com",  
            "gender": "male",  
            "contact": {  
                "mobile": "+91 0000000000",  
                "home": "00 000000",  
                "office": "00 000000"  
            }  
        },  
        .  
        .  
        .  
        .  
    ]  
}
```

JSON Elements In Android:

In Android, JSON consist of many components. Below we define some common components.

Array(): In a JSON, square bracket ([]) represents a JSONArray. JSONArray values may be any mix of JSONObject, other JSONArray, Strings, Booleans, Integers, Longs, Doubles, null or NULL. Values may not be NaNs, infinities, or of any type not listed here.

Objects({}): In a JSON, curly bracket ({}) represents a JSONObject. A JSONObject represents the data in the form of key and value pair. JSONObject values may be any mix of other JSONObjects,

JSONArray, Strings, Booleans, Integers, Longs, Doubles, null or NULL. Values may not be NaNs, infinities, or of any type not listed here.

Key: A JSONObject contains a key that is in string format. A pair of key and value creates a JSONObject.

Value: Each key has a value that could be primitive datatype(integer, float, String etc).

JSON Parsing In Android:

Usually, JSON contain two types of nodes JSONArray and JSONObject so while parsing we have to use the appropriate method. If JSON starts from square bracket ([) we use getJSONArray() method and if it start from curly bracket ({}) then we should use the getJSONObject() method. Apart from these there are some other methods for better parsing JSON data.

JSONObject Parsing methods:

Below we define some important methods of JSONObject parsing which are mainly used for parsing the data from JSONObject.

1. get(String name): This method is used to get the value from JSONObject. It returns the value of object type. We pass the String type key and it returns the value of Object type if exists otherwise it throws JSONException.

2. getBoolean(String name): This method is used to get the Boolean value from JSONObject. We pass the String type key and it returns the value of Boolean type if exists otherwise it throws JSONException.

3. `getDouble(String name)`: This method is used to get the double type value from JSONObject. We pass the String type key and it returns the value in double type if exists otherwise it throws JSONException.

4. `getInt(String name)`: This method is used to get the int type value from JSONObject. We pass the string type key and it returns the value in int type if exists otherwise it throws JSONException.

5. `getJSONArray(String name)`: This method is used to get the JSONArray type value. We pass the String type key and it returns JSONArray if exists otherwise it throws JSONException.

6. `getJSONObject(String name)`: This method is used to get the JSONObject type value. We pass the String type key and it returns the JSONObject value if exists otherwise it throws JSONException.

7. `getLong(String name)`: This method is used to get the long type value from JSONObject. We pass the String type key and it returns the value in long type if exists otherwise it throws JSONException.

8. `getString(String name)`: This method is used to get the String type value from JSONObject. We pass the String type key and it returns the value in String type if exists otherwise it throws JSONException.

9. `length()`: This method is used to get the number of name/value mappings in this object.

10. `keys()`: This method is used to get the iterator of String names in the Object.

11. opt(String name): This method is used to get the value from JSONObject. It returns the value of Object type. We pass the String type key and it returns the value of Object type if exists otherwise it returns null.

12. optBoolean(String name): This method is used to get the Boolean value from JSONObject. We pass the String type key and it returns the value of Boolean type if exists otherwise it returns false.

13. optDouble(String name): This method is used to get the double type value from JSONObject. We pass the String type key and it returns the value in double type if exists otherwise it returns NaN.

14. optInt(String name): This method is used to get the int type value from JSONObject. We pass the string type key and it returns the value in int type if exists otherwise it returns 0.

15. optJSONArray(String name): This method is used to get the JSONArray type value from JSONObject. We pass the String type key and it returns JSONArray if exists otherwise it returns null.

16. optJSONObject(String name): This method is used to get the other JSONObject type value from JSONObject. We pass the String type key and it returns the JSONObject value if exists otherwise it returns null.

17. optLong(String name): This method is used to get the long type value from JSONObject. We pass the String type key and it returns the value in long type if exists otherwise it returns 0.

18. optString(String name): This method is used to get the String type value from JSONObject. We pass the String type key and it returns the value in String type if exists otherwise it returns empty("") string.

JSONArray Parsing methods:

Below we define some important methods of JSONArray parsing which are mainly used for parsing the data from JSONArray.

1. get(int index): This method is used to get the value from JSONArray. It returns the value of object type. We pass the index and it returns the value of object type if exist otherwise it throws JSONException.

2. getBoolean(int index): This method is used to get the Boolean value from JSONArray. We pass the index and it returns the value of Boolean type if exists otherwise it throws JSONException.

3. getDouble(int index): This method is used to get the double type value from JSONArray. We pass the index and it returns the value in double type if exists otherwise it throws JSONException.

4. getInt(int index): This method is used to get the int type value from JSONArray. We pass the index and it returns the value in int type if exists otherwise it throws JSONException.

5. getJSONArray(int index): This method is used to get the JSONArray type value. We pass the index and it returns JSONArray if exists otherwise it throws JSONException.

6. getJSONObject(int index): This method is used to get the JSONObject type value. We pass the index and it returns the JSONObject value if exists otherwise it throws JSONException.

7. getLong(int index): This method is used to get the long type value from JSONArray. We pass the index and it returns the value in long type if exists otherwise it throws JSONException.

8. getString(int index): This method is used to get the String type value from JSONArray. We pass the index and it returns the value in String type if exists otherwise it throws JSONException.

9. length(): This method is used to get the number of values in this Array.

10. opt(int index): This method is used to get the value from JSONArray. It returns the value of Object type. We pass the index and it returns the value at index of Object type if exists otherwise it returns null.

11. optBoolean(int index): This method is used to get the Boolean value from JSONArray. We pass the index and it returns the value of Boolean type if exists otherwise it returns false.

12. optDouble(int index): This method is used to get the double type value from JSONArray. We pass the index and it returns the value in double type if exists otherwise it returns NaN.

13. optInt(int index): This method is used to get the int type value from JSONArray. We pass the index and it returns the value in int type if exists otherwise it returns 0.

14. optJSONArray(int index): This method is used to get the other JSONArray type value from JSONArray. We pass the index and it returns JSONArray if exists otherwise it returns null.

15. optJSONObject(int index): This method is used to get the JSONObject type value from JSONArray. We pass the index and it returns the JSONObject value if exists otherwise it returns null.

16. optLong(int index): This method is used to get the long type value from JSONArray. We pass the index and it returns the value in long type if exists otherwise it returns 0.

17. optString(int index): This method is used to get the String type value from JSONArray. We pass the index and it returns the value in String type if exists otherwise it returns empty("") string.

Example 1 of Simple JSON Parsing In Android Studio:

Below is the example of JSON parsing in Android, In this example we parse the data from JSON and then display it in the UI.In this, we have employee name and salary stored in JSON format.

Firstly we create two TextView's in our XML file and then in our Activity we parse the data using JSONObject methods and set it in the TextView's.

Below you can download code, see final output and step by step explanation of the example:



Step 1: Create a new project and name it JSONParsingExample.

Step 2: Open res -> layout -> activity_main.xml (or) main.xml and add following code:

In this step we create two TextView's for displaying employee name and salary.

```
<?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"
    tools:context="abhiandroid.com.jsonparsingexample.MainActivity">

    <TextView
        android:id="@+id/name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="50dp"
        android:text="Name"
        android:textColor="#000"
        android:textSize="20sp" />

    <TextView
        android:id="@+id/salary"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="80dp"
        android:text="Salary"
        android:textColor="#000"
        android:textSize="20sp" />
</RelativeLayout>
```

Step 3 : Now open app -> java -> package -> MainActivity.java and add the below code.

In this step firstly we get the reference of both TextView's then we parse the JSON using JSONObject methods and finally we set the data in TextView's.

```
package abhiandroid.com.jsonparsingexample;

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

import org.json.JSONException;
import org.json.JSONObject;

public class MainActivity extends AppCompatActivity {

    String JSON_STRING = "{\"employee\":{\"name\":\"Abhishek Saini\",\"salary\":65000}}";
    String name, salary;
    TextView employeeName, employeeSalary;

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

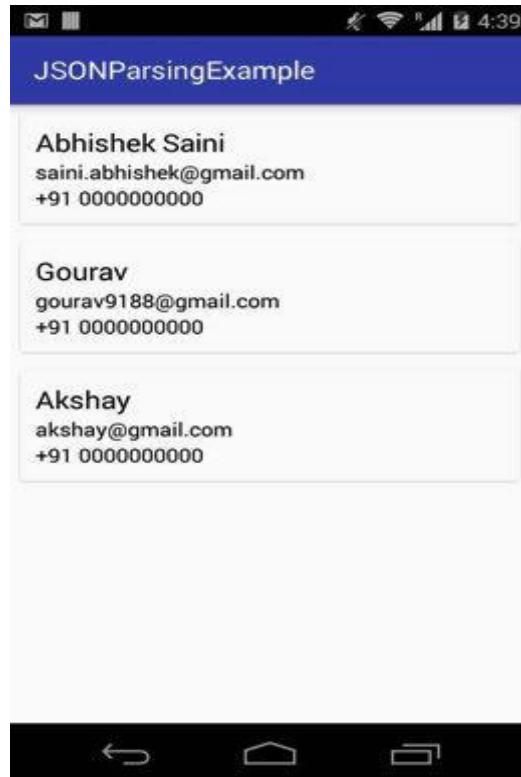
```
// get the reference of TextView's
employeeName = (TextView) findViewById(R.id.name);
employeeSalary = (TextView) findViewById(R.id.salary);

try {
    // get JSONObject from JSON file
    JSONObject obj = new JSONObject(JSON_STRING);
    // fetch JSONObject named employee
    JSONObject employee = obj.getJSONObject("employee");
    // get employee name and salary
    name = employee.getString("name");
    salary = employee.getString("salary");
    // set employee name and salary in TextView's
    employeeName.setText("Name: "+name);
    employeeSalary.setText("Salary: "+salary);

} catch (JSONException e) {
    e.printStackTrace();
}
}
```

JSON Parsing File Example 2 In Android Studio:

Below is the 2nd example of JSON parsing In Android Studio. In this example we create a JSON file and store it in assets folder of Android. In this JSON file we have list of users where each object contain the information like user id, name, email, gender and different contact numbers. In this we are using LinearLayoutManager with vertical orientation to display the array items. Firstly we declare a RecyclerView in our XML file and then get the reference of it in our Activity. After that we fetch the JSON file then parse the JSONArray data and finally set the Adapter to show the items in RecyclerView. Whenever a user clicks on an item the name of the Person is displayed on the screen with the help of Toast.



Step 1: Create a new project and name it JSONParsingExample.

Step 2: Open Gradle Scripts > build.gradle and add RecyclerView and CardView Library dependency in it.

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

android {
    compileSdkVersion 24
    buildToolsVersion "24.0.1"

    defaultConfig {
        applicationId "abhiandroid.com.jsonparsingexample"
        minSdkVersion 16
        targetSdkVersion 24
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
```

```
        minifyEnabled false
        proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-
rules.pro'
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:24.1.1'
    compile "com.android.support:recyclerview-v7:23.0.1" // dependency file for RecyclerView
    compile 'com.android.support:cardview-v7:23.0.1' // dependency file for CardView
}
```

Step 3: Open res -> layout -> activity_main.xml (or) main.xml and add following code:

In this step we create a RecyclerView in our XML file.

Step 4: Create a new XML file rowlayout.xml for item of RecyclerView and paste the following code in it.

In this step we create a new xml file for item row in which we create a TextView to show the data.

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

<android.support.v7.widget.CardView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/card_view"
    android:layout_width="match_parent"
    android:layout_margin="5dp"
    android:layout_height="wrap_content">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:padding="10dp">
        <!--
            items for a single row of RecyclerView
        -->
        <TextView
            android:id="@+id/name"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Name"
        android:textColor="#000"
        android:textSize="20sp" />

    <TextView
        android:id="@+id/email"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="email@email.com"
        android:textColor="#000"
        android:textSize="15sp" />

    <TextView
        android:id="@+id/mobileNo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="e9999999999"
        android:textColor="#000"
        android:textSize="15sp" />
</LinearLayout>
</android.support.v7.widget.CardView>
```

Step 5 : Now open app -> java -> package -> MainActivity.java and add the below code.

In this step firstly we get the reference of RecyclerView. After that we fetch the JSON file from assets and parse the JSON data using JSONArray and JSONObject methods and then set a LayoutManager and finally we set the Adapter to show the items in RecyclerView.

```
package abhiandroid.com.jsonparsingexample;

import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.LinearLayoutManager;
import android.support.v7.widget.RecyclerView;
import android.util.Log;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.Arrays;

public class MainActivity extends AppCompatActivity {

    // ArrayList for person names, email Id's and mobile numbers
    ArrayList<String> personNames = new ArrayList<>();
```

```
ArrayList<String> emailIds = new ArrayList<>();
ArrayList<String> mobileNumbers = new ArrayList<>();

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    // get the reference of RecyclerView
    RecyclerView recyclerView = (RecyclerView) findViewById(R.id.recyclerView);
    // set a LinearLayoutManager with default vertical orientation
    LinearLayoutManager linearLayoutManager = new
    LinearLayoutManager(getApplicationContext());
    recyclerView.setLayoutManager(linearLayoutManager);

    try {
        // get JSONObject from JSON file
        JSONObject obj = new JSONObject(loadJSONFromAsset());
        // fetch JSONArray named users
        JSONArray userArray = obj.getJSONArray("users");
        // implement for loop for getting users list data
        for (int i = 0; i < userArray.length(); i++) {
            // create a JSONObject for fetching single user data
            JSONObject userDetail = userArray.getJSONObject(i);
            // fetch email and name and store it in arraylist
            personNames.add(userDetail.getString("name"));
            emailIds.add(userDetail.getString("email"));
            // create a object for getting contact data from JSONObject
            JSONObject contact = userDetail.getJSONObject("contact");
            // fetch mobile number and store it in arraylist
            mobileNumbers.add(contact.getString("mobile"));
        }
    } catch (JSONException e) {
        e.printStackTrace();
    }

    // call the constructor of CustomAdapter to send the reference and data to Adapter
    CustomAdapter customAdapter = new CustomAdapter(MainActivity.this, personNames,
emailIds, mobileNumbers);
    recyclerView.setAdapter(customAdapter); // set the Adapter to RecyclerView
}

public String loadJSONFromAsset() {
    String json = null;
    try {
        InputStream is = getAssets().open("users_list.json");
        int size = is.available();
        byte[] buffer = new byte[size];
        is.read(buffer);
        is.close();
        json = new String(buffer, "UTF-8");
    } catch (IOException ex) {
        ex.printStackTrace();
        return null;
    }
    return json;
}
```

Step 6: Create a new class CustomAdapter.java inside package and add the following code.

In this step we create a CustomAdapter class and extends RecyclerView.Adapter class with ViewHolder in it. After that we implement the overrided methods and create a constructor for getting the data from Activity, In this custom Adapter two methods are more important first is onCreateViewHolder in which we inflate the layout item xml and pass it to View Holder and other is onBindViewHolder in which we set the data in the view's with the help of ViewHolder.

```
package abhiandroid.com.jsonparsingexample;

import android.content.Context;
import android.support.v7.widget.RecyclerView;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;
import android.widget.Toast;

import java.util.ArrayList;

public class CustomAdapter extends RecyclerView.Adapter<CustomAdapter.MyViewHolder> {

    ArrayList<String> personNames;
    ArrayList<String> emailIds;
    ArrayList<String> mobileNumbers;
    Context context;

    public CustomAdapter(Context context, ArrayList<String> personNames, ArrayList<String>
emailIds, ArrayList<String> mobileNumbers) {
        this.context = context;
        this.personNames = personNames;
        this.emailIds = emailIds;
        this.mobileNumbers = mobileNumbers;
    }

    @Override
    public MyViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        // infalte the item Layout
        View v = LayoutInflater.from(parent.getContext()).inflate(R.layout.rowlayout,
parent, false);
        MyViewHolder vh = new MyViewHolder(v); // pass the view to View Holder
        return vh;
    }

    @Override
    public void onBindViewHolder(MyViewHolder holder, final int position) {
        // set the data in items
    }
}
```

```
holder.name.setText(personNames.get(position));
holder.email.setText(emailIds.get(position));
holder.mobileNo.setText(mobileNumbers.get(position));
// implement setOnClickListener event on item view.
holder.itemView.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        // display a toast with person name on item click
        Toast.makeText(context, personNames.get(position),
Toast.LENGTH_SHORT).show();
    }
});

@Override
public int getItemCount() {
    return personNames.size();
}

public class MyViewHolder extends RecyclerView.ViewHolder {
    TextView name, email, mobileNo;// init the item view's

    public MyViewHolder(View itemView) {
        super(itemView);

        // get the reference of item view's
        name = (TextView) itemView.findViewById(R.id.name);
        email = (TextView) itemView.findViewById(R.id.email);
        mobileNo = (TextView) itemView.findViewById(R.id.mobileNo);

    }
}
```

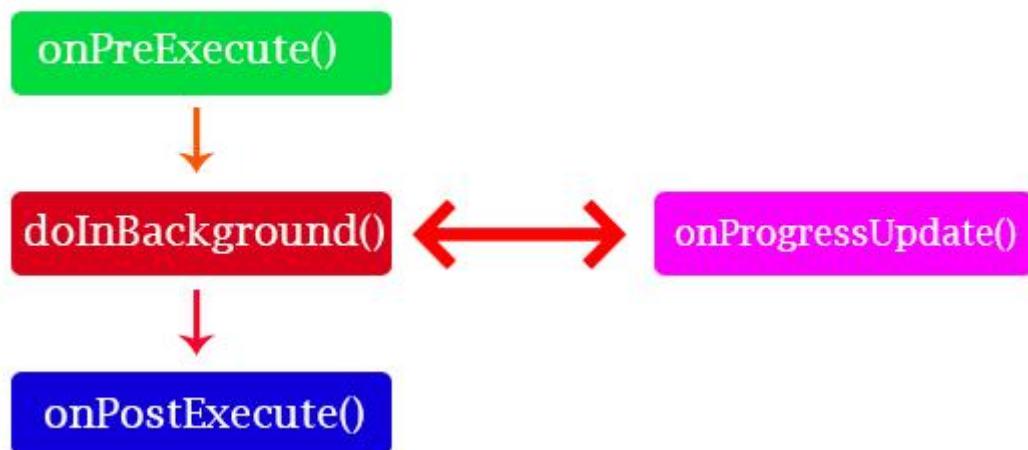
AsyncTask

In Android, AsyncTask (Asynchronous Task) allows us to run the instruction in the background and then synchronize again with our main thread. This class will override at least one method i.e doInBackground(Params) and most often will override second method onPostExecute(Result).

AsyncTask class is used to do background operations that will update the UI(user interface). Mainly we used it for short operations that will not effect on our main thread.

AsyncTask class is firstly executed using execute() method. In the first step AsyncTask is called onPreExecute() then onPreExecute() calls doInBackground() for background processes and then doInBackground() calls onPostExecute() method to update the UI.

AsyncTask Flow



Need of AsyncTask In Android:

By default, our application code runs in our main thread and every statement is therefore execute in a sequence. If we need to perform long tasks/operations then our main thread is blocked until the corresponding operation has finished. For providing a good user experience in our application we need to use AsyncTasks class that runs in a separate thread. This class will executes everything in doInBackground() method inside of other thread which doesn't have access to the GUI where all the views are present. The onPostExecute() method of this class synchronizes itself again with the main UI thread and allows it to make some updating. This method is called automatically after the doInBackground method finished its work.

Syntax of AsyncTask In Android:

To use AsyncTask you must subclass it. The parameters are the following AsyncTask<TypeOfVarArgParams, ProgressValue, ResultValue>. Here is the

Syntax of AsyncTask class:

```
private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {  
    protected Long doInBackground(URL... urls) {  
        // code that will run in the background  
        return ;  
    }  
  
    protected void onProgressUpdate(Integer... progress) {  
        // receive progress updates from doInBackground  
    }  
  
    protected void onPostExecute(Long result) {  
        // update the UI after background processes completes  
    }  
}
```

Executions of AsyncTask class from main thread:

Here is the Syntax for execution of AsyncTasks classss.

```
new DownloadFilesTask().execute(url1, url2, url3);
```

AsyncTask's generic types In Android:

The three types used by Asynchronous task are the following

AsyncTask <TypeOfVarArgParams, ProgressValue, ResultValue>

1. **TypeOfVarArgParams:** Params is the type of the parameters sent to the task upon execution.
2. **ProgressValue:** Progress is the type of the progress units published during the background computation.
3. **ResultValue:** ResultValue is the type of the result of the background computation.

Method of AsyncTask In Android:

In Android, AsyncTask is executed and goes through four different steps or method. Here are these four methods of AsyncTasks.

1. **onPreExecute()** – It invoked on the main UI thread before the task is executed. This method is mainly used to setup the task for instance by showing a ProgressBar or ProgressDialog in the UI(user interface).
2. **doInBackground(Params)** – This method is invoked on the background thread immediately after onPreExecute() finishes its execution. Main purpose of this method is to perform the background operations that can take a long time. The parameters of the Asynchronous task are passed to this step for execution. The result of the operations must be returned by this step and it will be passed back to the last step/method i.e onPostExecute(). This method can also use publishProgress(Progress...) to publish one or more units of progress. These values will be published on the main UI thread in the onProgressUpdate(Progress...) method.

3. onProgressUpdate(Progress...) – This method is invoked on the main UI thread after a call to publishProgress(Progress...). Timing of the execution is undefined. This method is used to display any form of progress in the user interface while the background operations are executing. We can also update our progress status for good user experience.

4. onPostExecute(Result) – This method is invoked on the main UI thread after the background operation finishes in the doInBackground method. The result of the background operation is passed to this step as a parameter and then we can easily update our UI to show the results.

Rules of AsyncTask:

There are a few threading rules that must be followed for this class to work properly:

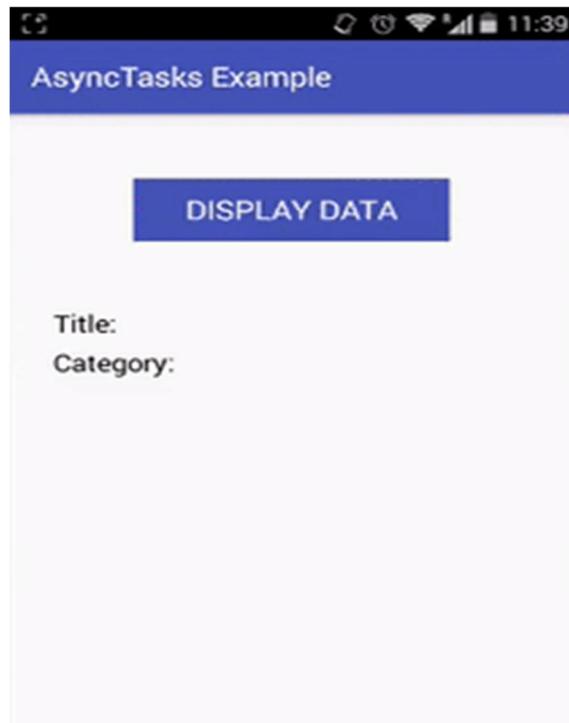
1. This class must be loaded on the UI thread. This is done automatically as from JELLY_BEAN.
2. The task instance must be created on the UI thread.
3. execute(Params...) method that executes it, must be invoked on the UI thread.
4. Do not call onPreExecute(), onPostExecute(Result), doInBackground(Params...), onProgressUpdate(Progress...) manually, just executes the class and then will call automatically for good user experience.

AsyncTask Example In Android Studio:

In this step by step example, we are using AsyncTask class for performing network operations. Here first we are going to fetch some data from API(Web service) and display it in our UI. For that firstly we create a Button and on click of it we are executing our AsyncTasks class that fetches the data in background and after getting response from API in postExecute() method we are displaying the same data in our UI.

Our API url: http://mobileappdatabase.in/demo/smartnews/app_dashboard/jsonUrl/single-article.php?article-id=71

Below you can download code, see final output and follow step by step explanation of the example:



Step 1: Create a new project and name it AsyncTasksExample.

Step 2: Open build.gradle and add Picasso library dependency.

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

android {
    compileSdkVersion 25
    buildToolsVersion "25.0.3"
    defaultConfig {
        applicationId "com.abhiandroid.asynctasksexample"
        minSdkVersion 15
        targetSdkVersion 25
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-
rules.pro'
```

```
        }
    }

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
        exclude group: 'com.android.support', module: 'support-annotations'
    })
    compile 'com.android.support:appcompat-v7:25.3.1'
    compile 'com.android.support.constraint:constraint-layout:1.0.2'
    compile 'com.squareup.picasso:picasso:2.5.2'
    testCompile 'junit:junit:4.12'
}
```

Step 3:

Open res -> layout ->activity_main.xml (or) main.xml and add following code:

In this step firstly we create Button to perform click event and TextView's and ImageView to display the fetched API data.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="30dp"
    tools:context="com.abhiandroid.AsyncTaskExample.MainActivity">

    <Button
        android:id="@+id/displayData"
        android:layout_width="200dp"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginTop="20dp"
        android:background="@color/colorPrimary"
        android:text="Display Data"
        android:textColor="#fff"
        android:textSize="20sp" />

    <TextView
        android:id="@+id/titleTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="50dp"
        android:text="Title: "
        android:textColor="#000"
        android:textSize="18sp" />
```

```
<TextView  
    android:id="@+id/categoryTextView"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_marginTop="5dp"  
    android:text="Category: "  
    android:textColor="#000"  
    android:textSize="18sp" />  
  
<ImageView  
    android:id="@+id/imageView"  
    android:scaleType="centerCrop"  
    android:layout_width="match_parent"  
    android:layout_height="200dp"  
    android:layout_gravity="center"  
    android:layout_marginTop="20dp" />  
  
</LinearLayout>
```

Step 4: Open src -> package -> MainActivity.java

In this step firstly we get the reference of Button, TextView's and ImageView. After that we perform setOnClickListener event on button and execute AsyncTasks class. In AsyncTasks class firstly we display a Progress Dialog in onPreExecute Method then we implement the API in doInBackground method and after getting response in onPostExecute method we simply parse the JSON data and display the title, category and Image in UI. In this, we also used Picasso Library to fetch the image from URL.

```
package com.abhiandroid.AsyncTasksexample;  
  
import android.app.ProgressDialog;  
import android.os.AsyncTask;  
import android.os.Bundle;  
import android.support.v7.app.AppCompatActivity;  
import android.util.Log;  
import android.view.View;  
import android.widget.Button;  
import android.widget.ImageView;  
import android.widget.TextView;  
  
import com.squareup.picasso.Picasso;  
  
import org.json.JSONArray;  
import org.json.JSONException;  
import org.json.JSONObject;
```

```
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;

public class MainActivity extends AppCompatActivity {

    String apiUrl =
    "http://mobileappdatabase.in/demo/smartnews/app_dashboard/jsonUrl/single-
    article.php?article-id=71";
    String title, image, category;
    TextView titleTextView, categoryTextView;
    ProgressDialog progressDialog;
    Button displayData;
    ImageView imageView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // get the reference of View's
        titleTextView = (TextView) findViewById(R.id.titleTextView);
        categoryTextView = (TextView) findViewById(R.id.categoryTextView);
        displayData = (Button) findViewById(R.id.displayData);
        imageView = (ImageView) findViewById(R.id.imageView);
        // implement setOnClickListener event on displayData button
        displayData.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // create object of MyAsyncTasks class and execute it
                MyAsyncTasks myAsyncTasks = new MyAsyncTasks();
                myAsyncTasks.execute();
            }
        });
    }

    public class MyAsyncTasks extends AsyncTask<String, String, String> {

        @Override
        protected void onPreExecute() {
            super.onPreExecute();
            // display a progress dialog for good user experiance
            progressDialog = new ProgressDialog(MainActivity.this);
            progressDialog.setMessage("Please Wait");
            progressDialog.setCancelable(false);
            progressDialog.show();
        }

        @Override
        protected String doInBackground(String... params) {

            // implement API in background and store the response in current variable
            String current = "";
            try {
                URL url;
                HttpURLConnection urlConnection = null;
```

```
try {
    url = new URL(apiUrl);

    urlConnection = (HttpURLConnection) url
        .openConnection();

    InputStream in = urlConnection.getInputStream();

    InputStreamReader isw = new InputStreamReader(in);

    int data = isw.read();
    while (data != -1) {
        current += (char) data;
        data = isw.read();
        System.out.print(current);
    }
    // return the data to onPostExecute method
    return current;
} catch (Exception e) {
    e.printStackTrace();
} finally {
    if (urlConnection != null) {
        urlConnection.disconnect();
    }
}
} catch (Exception e) {
    e.printStackTrace();
    return "Exception: " + e.getMessage();
}
return current;
}

@Override
protected void onPostExecute(String s) {

    Log.d("data", s.toString());
    // dismiss the progress dialog after receiving data from API
    progressDialog.dismiss();
    try {
        // JSON Parsing of data
        JSONArray jsonArray = new JSONArray(s);

        JSONObject oneObject = jsonArray.getJSONObject(0);
        // Pulling items from the array
        title = oneObject.getString("title");
        category = oneObject.getString("category");
        image = oneObject.getString("image");
        // display the data in UI
        titleTextView.setText("Title: "+title);
        categoryTextView.setText("Category: "+category);
        // Picasso library to display the image from URL
        Picasso.with(getApplicationContext())
            .load(image)
            .into(imageView);
    }
}
```

```
        } catch (JSONException e) {
            e.printStackTrace();
        }

    }
}
```

Step 5: Open AndroidManifest.xml file and add Internet Permission.

In this step we define the Internet permission.

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

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

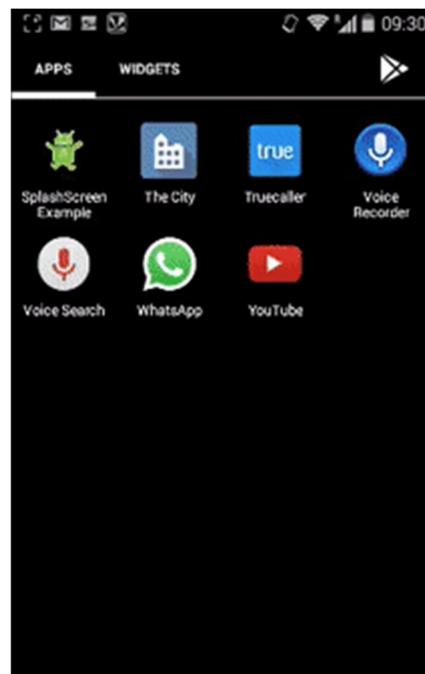
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">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

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

</manifest>
```

Splash Screen

Splash Screen is most commonly the first startup screen which appears when App is opened. In other words, it is a simple constant screen for a fixed amount of time which is used to display the company logo, name, advertising content etc.



Normally it shows when app is first time launched on android device or it may be some kind of process that is used to show screen to user just before the app loads completely.

This tutorial will help you to learn How to create Splash screen in your Android app.

Splash Screen Implementation Method In Android:

Method 1 of implementing Splash Screen:

Create a thread and set time to sleep after that redirect to main app screen.

```
***** Create Thread that will sleep for 5 seconds ****/
Thread background = new Thread() {
    public void run() {
        try {
            // Thread will sleep for 5 seconds
            sleep(5*1000);

            // After 5 seconds redirect to another intent
            Intent i=new Intent(getApplicationContext(),FirstScreen.class);
            startActivity(i);

            //Remove activity
            finish();
        } catch (Exception e) {
            }
        }
    };
    // start thread
background.start();
```

Method 2 of Implementing Splash Screen:

Set time to handler and call Handler().postDelayed, it will call run method of runnable after set time and redirect to main app screen.

Handlers are basically background threads which allows you to communicate with the UI thread (update the UI).

Handlers are subclassed from the Android Handler class and can be used either by specifying a Runnable to be executed when required by the thread, or by overriding the handleMessage() callback method within the Handler subclass which will be called when messages are sent to the handler by a thread.

There are two main uses for a Handler:

- a) To schedule messages and Runnables to be executed at some point in the future
- b) To enqueue an action to be performed on a different thread than your own.

```
new Handler().postDelayed(new Runnable() {  
    // Using handler with postDelayed called runnable run method  
  
    @Override  
  
    public void run() {  
  
        Intent i = new Intent(MainSplashScreen.this, FirstScreen.class);  
  
        startActivity(i);  
  
        // close this activity  
  
        finish();  
  
    }  
  
}, 5*1000); // wait for 5 seconds
```

Splash Screen Image Size For Different Screen Size:

Android provides support for multiple screen sizes and densities, reflecting the many different screen configurations that a device may have. You can prefer below sizes to support Splash Screen on different size smartphones.

Android divides the range of actual screen sizes and densities into:

A set of four generalized sizes: small, normal, large, and xlarge

A set of six generalized densities:

ldpi (low) ~120dpi (240x360px)

mdpi (medium) ~160dpi (320x480px)

hdpi (high) ~240dpi (480x720px)

xhdpi (extra-high) ~320dpi (640x960px)

xxhdpi (extra-extra-high) ~480dpi (960x1440px)

xxxhdpi (extra-extra-extra-high) ~640dpi (1280x1920px)

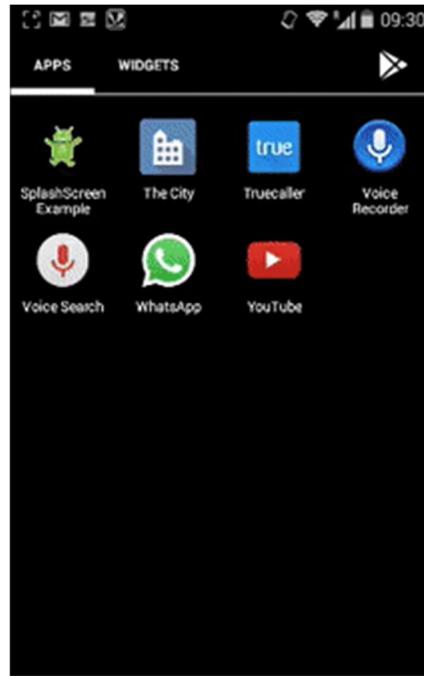
Splash Screen Example in Android Studio

With the help of this tutorial we will cover implementation of splash screen in two scenarios.

First will show splash screen using Handler and second we will not create a layout file for splash screen activity. Instead, specify activity's theme background as splash screen layout.

In the first below example we will see the use of Splash Screen using Handler class.

Below you can download code, see final output and step by step explanation of example:



Step 1 : Create a new project and name it SplashScreen

Step 2: Open res -> layout -> activity_main.xml (or) main.xml and add following code:

In this step we simply added a code to display layout after Splash screen.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="abhiandroid.com.splashscreen.MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World by AbhiAndroid!"
        android:textSize="20sp"
        android:layout_centerInParent="true"/>
</RelativeLayout>
```

Step 3: Create a new XML file splashfile.xml for Splash screen and paste the following code in it.

This layout contains your app logo or other product logo that you want to show on splash screen.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:background="@color/splashBackground">

    <ImageView
        android:id="@+id/logo_id"
        android:layout_width="250dp"
        android:layout_height="250dp"
        android:layout_centerInParent="true"
        android:src="@drawable/abhiandroid"/>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/logo_id"
        android:layout_centerHorizontal="true"
        android:text="Splash Screen"
        android:textSize="30dp"
        android:textColor="@color/blue"/>

</RelativeLayout>
```

Step 4: Now open app -> java -> package -> MainActivity.java and add the below code.

```
package abhiandroid.com.splashscreen;

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

public class MainActivity extends AppCompatActivity {

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

```
}
```

Step 5: For Splash Screen we will create a separate splash activity. Create a new class in your java package and name it as SplashActivity.java.

Step 6: Add this code in SplashActivity.java activity. In this code handler is used to hold the screen for specific time and once the handler is out, our main Activity will be launched. We are going to hold the Splash screen for three second's. We will define the seconds in millisecond's after Post Delayed(){} method.

1 second =1000 milliseconds.

Post Delayed method will delay the time for 3 seconds. After the delay time is complete, then your main activity will be launched.

SplashActivity.java

```
package abhiandroid.com.splashscreen;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.os.Handler;

/**
 * Created by AbhiAndroid
 */

public class SplashActivity extends Activity {

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

```
        handler=new Handler();
        handler.postDelayed(new Runnable() {
            @Override
            public void run() {
                Intent intent=new Intent(SplashActivity.this,MainActivity.class);
                startActivity(intent);
                finish();
            }
        },3000);
    }
}
```

Step 7: Open AndroidManifest.xml file and make your splashactivity.java class as Launcher activity and mention the Main Activity as another activity.

AndroidManifest.xml

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

    <application
        android:allowBackup="true"
        android:icon="@drawable/abhiandroid"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name="abhiandroid.com.splashscreen.SplashActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

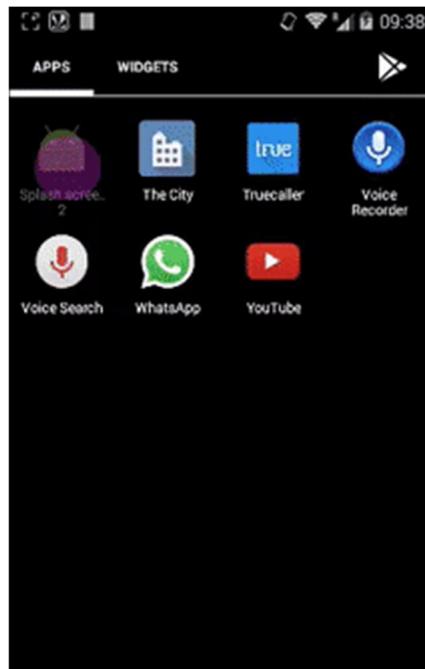
Output:

Now run the App and you will see Splash screen before the main layout loads.

Splash Screen Example 2 In Android Studio:

The second way is little different to implement Splash screen. In this example we are going to specify the Splash screen background as an activity's theme background.

Below you can download code, see final output and step by step explanation of example:



Step 1: Create a new project and name it `Splashscreen 2`

Step 2: Open `res -> layout -> activity_main.xml` (or) `main.xml` and add following code:

In this step the code to display layout after Splash screen is added.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="20dp"
    android:layout_gravity="center"
    tools:context="abhiandroid.com.splashscreen2.SecondActivity">
```

```
<TextView  
    android:id="@+id/hello_id"  
    android:layout_centerInParent="true"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Hello my friend "  
    android:textSize="20sp"/>  
</RelativeLayout>
```

Step 3: Create a new xml layout in res ⇒ drawable ⇒ splash_screenbackground.xml

In this example we will not create a layout file for Splash screen activity. Instead we will use activity theme background for Splash screen layout.

Here, we use a layer list to show the image in the center of splash screen. It is necessary to use a bitmapped image to display the image. (image should be PNG or JPG) and splash_background_color as an background color.

The following is an example code of a drawable resource using layer-list.

Add below code in res ⇒ drawable.xml ⇒ splash_screenbackground.xml

```
<?xml version="1.0" encoding="utf-8"?>  
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">  
    <item>  
        <color android:color="@color/splash_background_color"/>  
    </item>  
    <item>  
        <bitmap  
            android:src="@drawable/mx"  
            android:tileMode="disabled"  
            android:gravity="center"/>  
    </item>  
</layer-list>
```

Step 4: Now open res ⇒ values ⇒ colors.xml

Make sure to add below code in colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#3F51B5</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF4081</color>
    <color name="splash_background_color">#5456e1</color>
</resources>
```

Step 5:

Add the below code in `res/values/styles.xml`

Now we create a theme for the splash screen activity. We create a custom theme for the splash screen Activity, and add to the file **values/styles.xml**

res/values/styles:

SplashTheme – When we declare the window background, it will removes the title bar from the window, and show us the full-screen. The file is shown here with a style named **SplashTheme**.

```
<resources>

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

    <style name="SplashTheme" parent = "Theme.AppCompat.Light.NoActionBar">
        <item name="android:windowBackground">@drawable/splash_screenbackground</item>
        <item name="android:windowNoTitle">true</item>
        <item name="android:windowFullscreen">true</item>
    </style>

</resources>
```

Step 6:

Add the below code in `MainActivity.java` and Create a new `SecondActivity.java` for splash screen

Here we will not define the setcontentview() for the Splash screen activity because we directly apply the Splash Theme on it. Now we just have to launch the Activity(Second Activity) and then finish the activity by calling finish() method.

MainActivity.java

```
package abhiandroid.com.splashscreen2;

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

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        startActivity(new Intent(MainActivity.this,SecondActivity.class));

        // close splash activity
        finish();
    }
}
```

SecondActivity.java

```
package abhiandroid.com.splashscreen2;

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

/**
 * Created by AbhiAndroid
 */

public class SecondActivity extends Activity {

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

Step 7: Theme must be assigned to Main Activity in Android Manifest

Like – (android:name=".MainActivity" android:theme="@style/SplashTheme">)

Here is the full code of AndroidManifest:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="abhiandroid.com.splashscreen2">

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

        <activity android:name="abhiandroid.com.splashscreen2.MainActivity"
            android:theme="@style/SplashTheme">

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

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name="abhiandroid.com.splashscreen2.SecondActivity"/>
    </application>

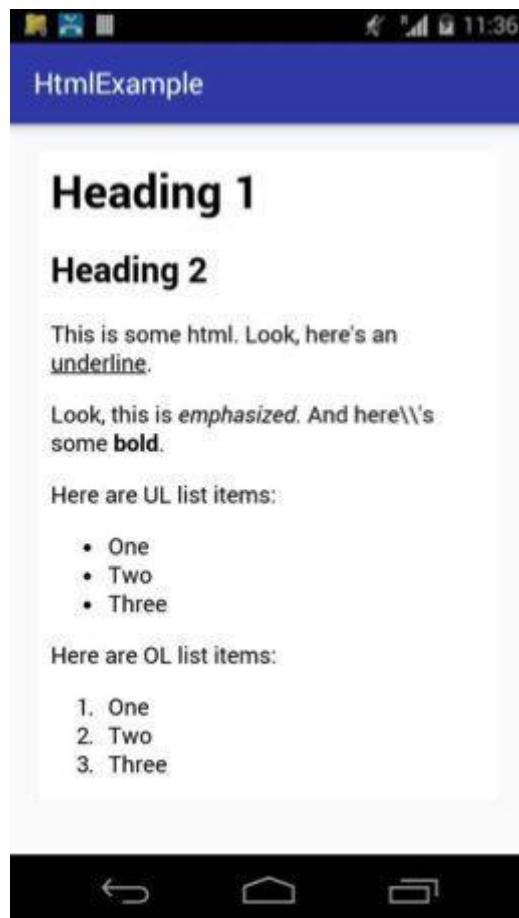
</manifest>
```

Output:

Now run the App and you will Splashscreen launch before the main activity.

HTML In Android

In Android, many times while designing an application we might be in a situation where we would like to use HTML content and display in our App screen. It may be to display some static content like help, support, FAQ and others pages. For this Android provides us a lovely class android.text.HTML that processes HTML strings into displayable styled text and then we can set the text in our TextView. We can also use WebView for displaying HTML content. Currently Android does not support all the HTML tags but it supports all major tags.



HTML Tags Supported in Android Studio:

Android API documentation does not stipulate what HTML tags are supported. After looking into the android Source code and from a quick look at the source code, here's what seems to be supported as of now.

1. b & strong tag for bold
2. i, em, cite & dfn tag for Italics
3. u tag for Underline
4. sub tag for Subtext
5. sup tag for Supertext
6. big tag for Big
7. small tag for Small
8. tt tag for Monospace
9. h1 to h6 tag for Headlines
10. img tag for image
11. font tag for Font face and color
12. blockquote tag for longer quotes
13. a tag for Link
14. div and p tag for Paragraph
15. br tag for Linefeed

How to Display HTML in Android TextView:

In Android, for displaying HTML content we use `HTML.fromHtml()` method. This method can contain `Html.ImageGetter` as argument as well as the text to parse. We can parse null as for the `HTML.TagHandler` but we need to implement `HTML.ImageGetter` as there is not any default implementation for this. The `Html.ImageGetter` needs to run synchronously and if we are downloading images from the web we probably want to do that asynchronously. We can also display the HTML content in `WebView` and its better then displaying it in `TextView`. In this article we will show you both examples with same HTML content.

Methods for Parsing HTML content In Android:

Below we define some common methods used for parsing HTML content.

1. `fromHtml(String source)`: This method is used to display styled text from the provided HTML string. This method was deprecated in API level 24. now please use `fromHtml(String, int)` instead.

2. `fromHtml(String source, int flags)`: This method is replacement of `fromHtml(String source)` method with the legacy flags `FROM_HTML_MODE_LEGACY`.

3. `fromHtml(String source, Html.ImageGetter imageGetter, Html.TagHandler tagHandler)`: This method is used to display styled text from the provided HTML string. Any `` tags in the HTML will use the specified `ImageGetter` to request a representation of the image (use null if you don't want this) and the specified `TagHandler` to handle unknown tags (specify null if you don't want this).

4. `fromHtml(String source, int flags, Html.ImageGetter imageGetter, Html.TagHandler tagHandler)`: This method is replacement of `fromHtml(String source, Html.ImageGetter imageGetter, Html.TagHandler tagHandler)` method with the legacy flags `FROM_HTML_MODE_LEGACY`.

Constant Flags used while Parsing HTML content:

Below we define some common flags that should be used in `fromHtml()` method for parsing HTML content.

1. FROM_HTML_MODE_COMPACT: This flag is used to separate the block level elements with line break means single new line character in between.

2. FROM_HTML_MODE_LEGACY: This flag is used to separate the block level elements with blank lines means two new line characters in between.

3. FROM_HTML_OPTION_USE_CSS_COLORS: This flag is used to indicate that CSS color values should be used instead of those defined in Color.

4. FROM_HTML_SEPARATOR_LINE_BREAK_BLOCKQUOTE: This flag is used to indicate that texts inside `<blockquote>` elements will be separated from other texts with one newline character by default.

5. FROM_HTML_SEPARATOR_LINE_BREAK_DIV: This flag is used to indicate that texts inside `<div>` elements will be separated from other texts with one newline character by default.

6. FROM_HTML_SEPARATOR_LINE_BREAK_HEADING: This flag is used to indicate that texts inside `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>` and `<h6>` elements will be separated from other texts with one newline character by default.

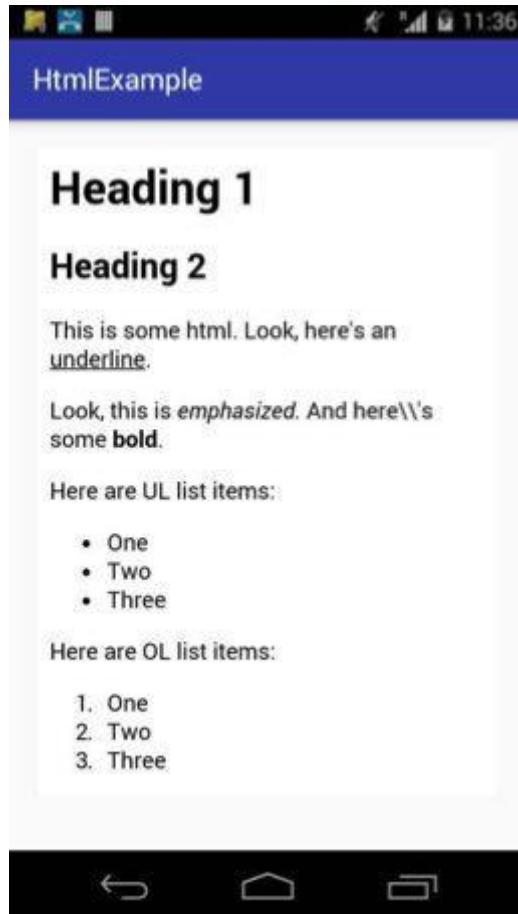
7. FROM_HTML_SEPARATOR_LINE_BREAK_LIST: This flag is used to indicate that texts inside elements will be separated from other texts with one newline character by default.

8. FROM_HTML_SEPARATOR_LINE_BREAK_LIST_ITEM: This flag is used to indicate that texts inside elements will be separated from other texts with one newline character by default.

9. FROM_HTML_SEPARATOR_LINE_BREAK_PARAGRAPH: This flag is used to indicate that inside <p> elements will be separated from other texts with one newline character by default.

Example 1 Of Parse HTML File content Using WebView With Example In Android Studio:

Below is the example of HTML in which we parse the HTML file and display the HTML content in our Android WebView. In this example firstly we create a assets folder and store a HTML file in it. After that we create a WebView in our XML file and then get the reference of WebView in our MainActivity and finally with the help of loadUrl() method we display the content in our webView.



Step 1: Create a new project and name it HtmlExample.

Step 2: Open res -> layout -> activity_main.xml (or) main.xml and add following code:

In this step we open an xml file and add the code for displaying a WebView.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout 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="abhiandroid.com.htmlexample.MainActivity">
```

```
<WebView  
    android:id="@+id/simpleWebView"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_margin="20dp" />  
  
</RelativeLayout>
```

Step 3: Now create assets folder in App. You can read here how to create assets folder in Android Studio

Step 4: Now inside assets folder add a HTML file name myfile.html. You can read here how to add local html file in Android Studio. Also inside myfile.html add the below HTML content or add any content in HTML format.

```
<h1>Heading 1</h1>  
<h2>Heading 2</h2>  
<p>This is some html. Look, here's an <u>underline</u>. </p>  
<p>Look, this is <em>emphasized.</em> And here\\'s some <b>bold</b>. </p>  
<p>Here are UL list items:  
<ul>  
    <li>One</li>  
    <li>Two</li>  
    <li>Three</li>  
</ul>  
<p>Here are OL list items:  
<ol>  
    <li>One</li>  
    <li>Two</li>  
    <li>Three</li>  
</ol>
```

Step 5: Now Open src -> package -> MainActivity.java

In this step we open MainActivity where we add the code to initiate the WebView and then display the HTML content from file stored in assets folder into WebView.

```
package abhiandroid.com.htmlexample;  
  
import android.os.Bundle;  
import android.support.v7.app.AppCompatActivity;
```

```
import android.webkit.WebView;

public class MainActivity extends AppCompatActivity {

    WebView webView;

    public String fileName = "myfile.html";

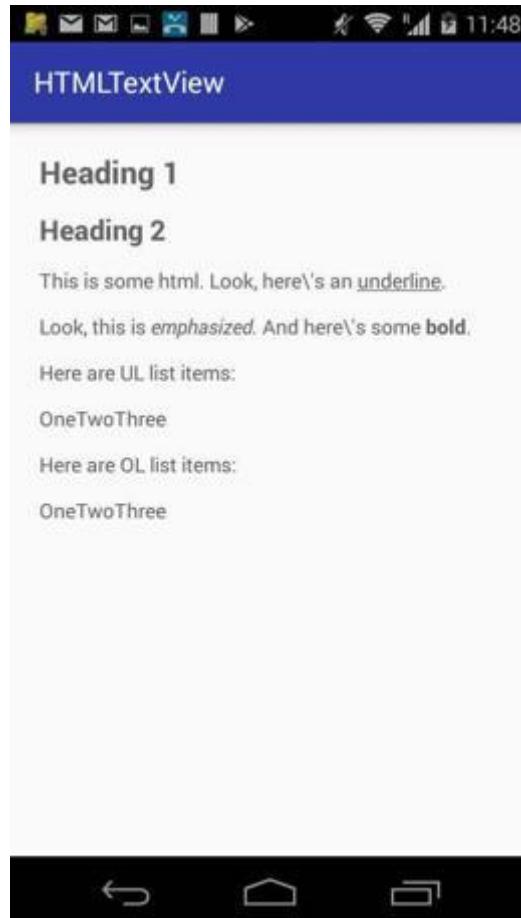
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // init webView
        webView = (WebView) findViewById(R.id.simpleWebView);
        // displaying content in WebView from html file that stored in assets folder
        webView.getSettings().setJavaScriptEnabled(true);
        webView.loadUrl("file:///android_asset/" + fileName);
    }
}
```

Output:

Now run the App and you will see local content added in HTML file is loaded in Webview.

Example 2 Of HTML In Android Studio Using TextView:

Below is the example of HTML in which we display the HTML content in our Android TextView with the help of fromHtml() method. In this example firstly we create a TextView in our XML file and then get the reference of TextView in our MainActivity and finally with the help of fromHtml() method we set the content in our TextView.



Step 1: Create a new project and name it HtmlExample.

Step 2: Open res -> layout -> activity_main.xml (or) main.xml and add following code:

In this step we open an xml file and add the code for displaying a TextView.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout 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="abhiandroid.com.htmlexample.MainActivity">

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

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_margin="20dp" />

</RelativeLayout>
```

Step 3:

Now Open src -> package -> MainActivity.java

In this step we open MainActivity where we add the code to initiate the TextView and then set the HTML content which is stored in a string variable into TextView using fromHtml() method.

```
package abhiandroid.com.htmlexample;

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

public class MainActivity extends AppCompatActivity {

    TextView textView;
    String content = "<h1>Heading 1</h1>\n" +
        "      <h2>Heading 2</h2>\n" +
        "      <p>This is some html. Look, here\\'s an <u>underline</u>. </p>\n" +
        "      <p>Look, this is <em>emphasized.</em> And here\\'s some
<b>bold</b>.</p>\n" +
        "      <p>Here are UL list items:\n" +
        "      <ul>\n" +
        "      <li>One</li>\n" +
        "      <li>Two</li>\n" +
        "      <li>Three</li>\n" +
        "      </ul>\n" +
        "      <p>Here are OL list items:\n" +
        "      <ol>\n" +
        "      <li>One</li>\n" +
        "      <li>Two</li>\n" +
        "      <li>Three</li>\n" +
        "      </ol>";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // init TextView
        textView = (TextView) findViewById(R.id.simpleTextView);
        // set Text in TextView using fromHtml() method with version check
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.N) {
            textView.setText(Html.fromHtml(content, Html.FROM_HTML_MODE_LEGACY));
        } else {
            textView.setText(Html.fromHtml(content));
        }
    }
}
```

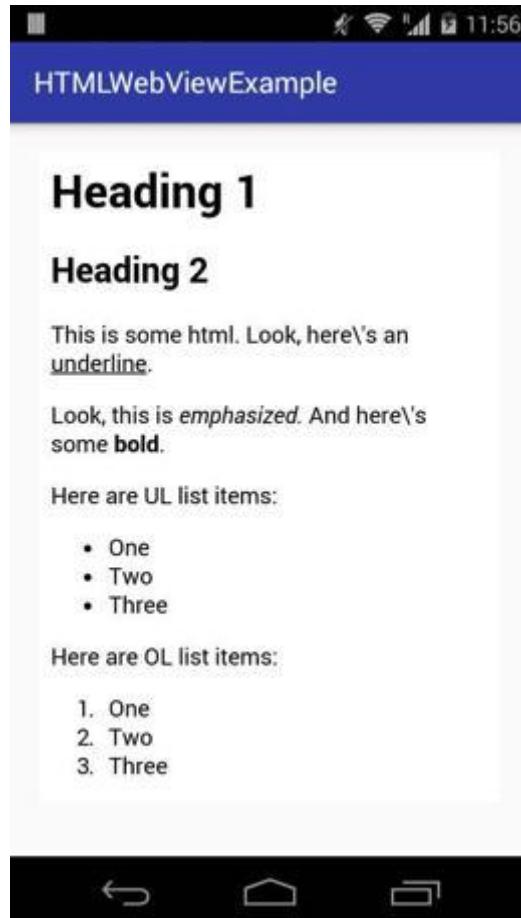
```
        } else
            textView.setText(Html.fromHtml(content));
    }
}
```

Output:

Now run the App and you will see HTML content is shown in TextView.

Example 3 Of HTML content in WebView With Example in Android Studio:

Below is the example of HTML in which we display the HTML content in our Android WebView. In this example firstly we create a WebView in our XML file and then get the reference of WebView in our MainActivity and finally with the help of loadDataWithBaseURL() method we display the content in our webView.



Step 1: Create a new project and name it HtmlWebViewExample.

Step 2: Open res -> layout -> activity_main.xml (or) main.xml and add following code:

In this step we open an xml file and add the code for displaying a WebView.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout 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="abhiandroid.com.htmlexample.MainActivity">

    <WebView
        android:id="@+id/simpleWebView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
    android:layout_margin="20dp" />  
  
  </RelativeLayout>
```

Step 3:

Now Open src -> package -> MainActivity.java

In this step we open MainActivity where we add the code to initiate the WebView and then display the HTML content which is stored in a string variable into WebView.

```
package abhiandroid.com.htmlexample;  
  
import android.os.Bundle;  
import android.support.v7.app.AppCompatActivity;  
import android.webkit.WebView;  
  
public class MainActivity extends AppCompatActivity {  
  
    WebView webView;  
    String content = "<h1>Heading 1</h1>\n" +  
        "      <h2>Heading 2</h2>\n" +  
        "      <p>This is some html. Look, here\\'s an <u>underline</u>. </p>\n" +  
        "      <p>Look, this is <em>emphasized.</em> And here\\'s some  
<b>bold</b>.</p>\n" +  
        "      <p>Here are UL list items:\n" +  
        "      <ul>\n" +  
        "          <li>One</li>\n" +  
        "          <li>Two</li>\n" +  
        "          <li>Three</li>\n" +  
        "      </ul>\n" +  
        "      <p>Here are OL list items:\n" +  
        "      <ol>\n" +  
        "          <li>One</li>\n" +  
        "          <li>Two</li>\n" +  
        "          <li>Three</li>\n" +  
        "      </ol>";  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        // init webView  
        webView = (WebView) findViewById(R.id.simpleWebView);  
        // displaying text in WebView  
        webView.loadDataWithBaseURL(null, content, "text/html", "utf-8", null);  
    }  
}
```

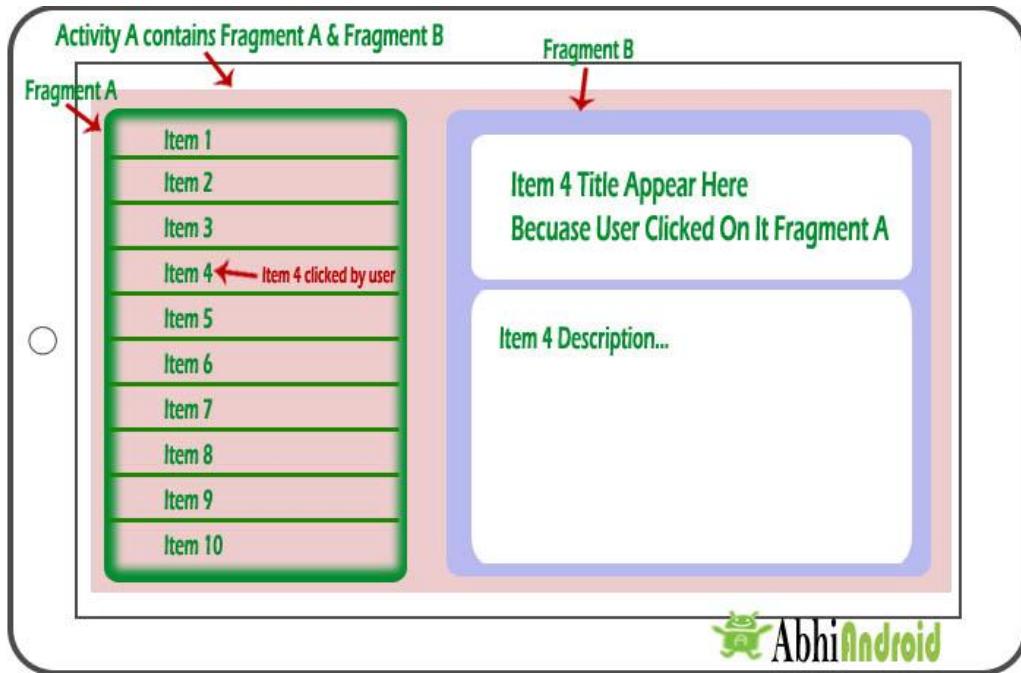
Output:

Now run the App and you will see HTML content is displayed using Webview.

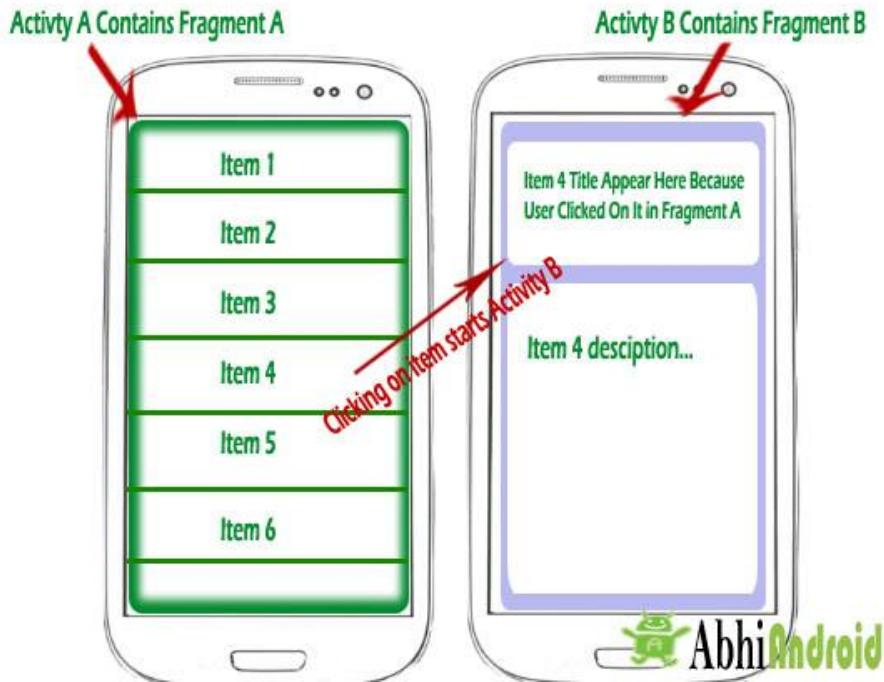
Fragment

In Android, Fragment is a part of an activity which enable more modular activity design. It will not be wrong if we say a fragment is a kind of sub-activity. It represents a behaviour or a portion of user interface in an Activity. We can combine multiple Fragments in Single Activity to build a multi panel UI and reuse a Fragment in multiple Activities. We always need to embed Fragment in an activity and the fragment lifecycle is directly affected by the host activity's lifecycle.

Important Related Read: Fragment Lifecycle



We can create Fragments by extending Fragment class or by inserting a Fragment into our Activity layout by declaring the Fragment in the activity's layout file, as a <fragment> element. We can manipulate each Fragment independently, such as add or remove them.



While performing Fragment Transaction we can add a Fragment into back stack that's managed by the Activity. back stack allow us to reverse a Fragment transaction on pressing Back button of device. For Example if we replace a Fragment and add it in back stack then on pressing the Back button on device it display the previous Fragment.

Some Important Points About Fragment In Android:

1. Fragments were added in Honeycomb version of Android i.e API version 11.
2. We can add, replace or remove Fragment's in an Activity while the activity is running. For performing these operations we need a Layout(Relative Layout, FrameLayout or any other layout) in xml file and then replace that layout with the required Fragment.
3. Fragments has its own layout and its own behaviour with its own life cycle callbacks.
4. Fragment can be used in multiple activities.
5. We can also combine multiple Fragments in a single activity to build a multi-plane UI.

Need Of Fragments In Android:

Before the introduction of Fragment's we can only show a single Activity on the screen at one given point of time so we were not able to divide the screen and control different parts separately. With the help of Fragment's we can divide the screens in different parts and controls different parts separately.

By using Fragments we can comprise multiple Fragments in a single Activity. Fragments have their own events, layouts and complete life cycle. It provide flexibility and also removed the limitation of single Activity on the screen at a time..

Basic Fragment Code In XML:

```
<fragment  
    android:id="@+id/fragments"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent" />
```

Create A Fragment Class In Android Studio:

For creating a Fragment firstly we extend the Fragment class, then override key lifecycle methods to insert our app logic, similar to the way we would with an Activity class. While creating a Fragment we must use onCreateView() callback to define the layout and in order to run a Fragment.

```
import android.os.Bundle;  
import android.support.v4.app.Fragment;  
import android.view.LayoutInflater;  
import android.view.ViewGroup;  
  
public class FirstFragment extends Fragment {  
    @Override  
    public View onCreateView(LayoutInflater inflater, ViewGroup container,  
        Bundle savedInstanceState) {  
        // Inflate the layout for this fragment  
        return inflater.inflate(R.layout.fragment_first, container, false);  
    }  
}
```

Here the inflator parameter is a LayoutInflater used to inflate the layout, container parameter is the parent ViewGroup (from the activity's layout) in which our Fragment layout will be inserted.

The savedInstanceState parameter is a Bundle that provides data about the previous instance of the Fragment. The inflate() method has three arguments first one is the resource layout which we want to inflate, second is the ViewGroup to be the parent of the inflated layout. Passing the container is important in order for the system to apply layout parameters to the root view of the inflated layout, specified by the parent view in which it's going and the third parameter is a boolean value indicating whether the inflated layout should be attached to the ViewGroup (the second parameter) during inflation.

Implementation of Fragment In Android Require Honeycomb (3.0) or Later:

Fragments were added in in Honeycomb version of Android i.e API version 11. There are some primary classes related to Fragment's are:

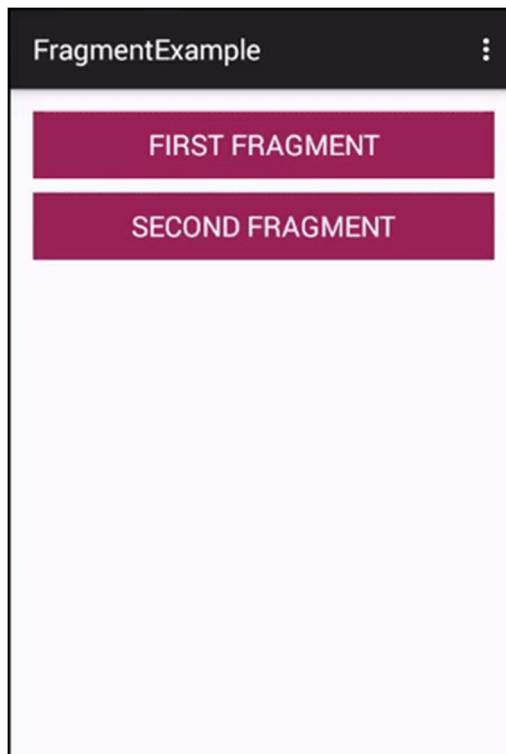
- 1. FragmentActivity:** The base class for all activities using compatibility based Fragment (and loader) features.
- 2. Fragment:** The base class for all Fragment definitions
- 3. FragmentManager:** The class for interacting with Fragment objects inside an activity
- 4. FragmentTransaction:** The class for performing an atomic set of Fragment operations such as Replace or Add a Fragment.

Fragment Example 1 In Android Studio:

Below is the example of Fragment's. In this example we create two Fragments and load them on the click of Button's. We display two Button's and a FrameLayout in our Activity and perform setOnClickListener event on both Button's. On the click of First Button we replace the First Fragment and on click of Second Button we replace the Second Fragment with the

layout(FrameLayout). In the both Fragment's we display a TextView and a Button and onclick of Button we display the name of the Fragment with the help of Toast.

Below you can download code, see final output and read step by step explanation:



Step 1: Create a new project and name it FragmentExample

Step 2: Open res -> layout ->activity_main.xml (or) main.xml and add following code

```
<LinearLayout 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:orientation="vertical"
    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">
    <!-- display two Button's and a FrameLayout to replace the Fragment's -->
    <Button
        android:id="@+id/firstFragment"
        android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content"
        android:background="@color/button_background_color"
        android:text="First Fragment"
        android:textColor="@color/white"
        android:textSize="20sp" />

    <Button
        android:id="@+id/secondFragment"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="10dp"
        android:background="@color/button_background_color"
        android:text="Second Fragment"
        android:textColor="@color/white"
        android:textSize="20sp" />

    <FrameLayout
        android:id="@+id/frameLayout"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_marginTop="10dp" />
</LinearLayout>
```

Step 3: Open src -> package -> MainActivity.java

In this step we open MainActivity and add the code for initiate the Button's. After that we perform setOnClickListener event on both Button's. On the click of First Button we replace the First Fragment and on click of Second Button we replace the Second Fragment with the layout(FrameLayout). For replacing a Fragment with FrameLayout firstly we create a Fragment Manager and then begin the transaction using Fragment Transaction and finally replace the Fragment with the layout i.e FrameLayout.

```
package com.abhiandroid.fragmentexample;

import android.app.Fragment;
import android.app.FragmentManager;
import android.app.FragmentTransaction;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {

    Button firstFragment, secondFragment;
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
// get the reference of Button's
firstFragment = (Button) findViewById(R.id.firstFragment);
secondFragment = (Button) findViewById(R.id.secondFragment);

// perform setOnClickListener event on First Button
firstFragment.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
// load First Fragment
loadFragment(new FirstFragment());
}
});

// perform setOnClickListener event on Second Button
secondFragment.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
// load Second Fragment
loadFragment(new SecondFragment());
}
});
}

private void loadFragment(Fragment fragment) {
// create a FragmentManager
FragmentManager fm = getSupportFragmentManager();
// create a FragmentTransaction to begin the transaction and replace the Fragment
FragmentTransaction fragmentTransaction = fm.beginTransaction();
// replace the FrameLayout with new Fragment
fragmentTransaction.replace(R.id.frameLayout, fragment);
fragmentTransaction.commit(); // save the changes
}
}
```

Step 4: Now we need 2 fragments and 2 xml layouts. So create two fragments by right click on your package folder and create classes and name them as FirstFragment and SecondFragment and add the following code respectively.

FirstFragment.class

In this Fragment firstly we inflate the layout and get the reference of Button. After that we perform setOnClickListener event on Button so whenever a user click on the button a message “First Fragment” is displayed on the screen by using a Toast.

```
package com.abhiandroid.fragmentexample;

import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.Toast;

public class FirstFragment extends Fragment {

    View view;
    Button firstButton;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        view = inflater.inflate(R.layout.fragment_first, container, false);
        // get the reference of Button
        firstButton = (Button) view.findViewById(R.id.firstButton);
        // perform setOnClickListener on first Button
        firstButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // display a message by using a Toast
                Toast.makeText(getActivity(), "First Fragment", Toast.LENGTH_LONG).show();
            }
        });
        return view;
    }
}
```

SecondFragment.class

In this Fragment firstly we inflate the layout and get the reference of Button. After that we perform setOnClickListener event on Button so whenever a user click on the button a message “Second Fragment“ is displayed on the screen by using a Toast.

```
package com.abhiandroid.fragmentexample;

import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
```

```
import android.widget.Toast;

public class SecondFragment extends Fragment {

    View view;
    Button secondButton;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        view = inflater.inflate(R.layout.fragment_second, container, false);
        // get the reference of Button
        secondButton = (Button) view.findViewById(R.id.secondButton);
        // perform setOnClickListener on second Button
        secondButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // display a message by using a Toast
                Toast.makeText(getActivity(), "Second Fragment", Toast.LENGTH_LONG).show();
            }
        });
        return view;
    }
}
```

Step 5: Now create 2 xml layouts by right clicking on res/layout -> New -> Layout Resource File and name them as fragment_first and fragment_second and add the following code in respective files.

Here we will design the basic simple UI by using TextView and Button in both xml's.

fragment_first.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="com.abhiandroid.fragmentexample.FirstFragment">

    <!--TextView and Button displayed in First Fragment -->
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="100dp"
        android:text="This is First Fragment"
        android:textColor="@color/black"
        android:textSize="25sp" />
```

```
<Button  
    android:id="@+id/firstButton"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:layout_centerInParent="true"  
    android:layout_marginLeft="20dp"  
    android:layout_marginRight="20dp"  
    android:background="@color/green"  
    android:text="First Fragment"  
    android:textColor="@color/white"  
    android:textSize="20sp"  
    android:textStyle="bold" />  
</RelativeLayout>
```

Fragment_second.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="com.abhiandroid.fragmentexample.SecondFragment">  
  
    <!--TextView and Button displayed in Second Fragment -->  
    <TextView  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:layout_centerHorizontal="true"  
        android:layout_marginTop="100dp"  
        android:text="This is Second Fragment"  
        android:textColor="@color/black"  
        android:textSize="25sp" />  
  
    <Button  
        android:id="@+id/secondButton"  
        android:layout_width="fill_parent"  
        android:layout_height="wrap_content"  
        android:layout_centerInParent="true"  
        android:layout_marginLeft="20dp"  
        android:layout_marginRight="20dp"  
        android:background="@color/green"  
        android:text="Second Fragment"  
        android:textColor="@color/white"  
        android:textSize="20sp"  
        android:textStyle="bold" />  
  
</RelativeLayout>
```

Step 6: Open res ->values ->colors.xml

In this step we define the color's that used in our xml file.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<!-- color's used in our project -->
<color name="black">#000</color>
<color name="green">#0f0</color>
<color name="white">#fff</color>
<color name="button_background_color">#925</color>
</resources>
```

Step 7: Open AndroidManifest.xml

In this step we show the Android Manifest file in which do nothing because we need only one Activity i.e MainActivity which is already defined in it. In our project we create two Fragment's but we don't need to define the Fragment's in manifest because Fragment is a part of an Activity.

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

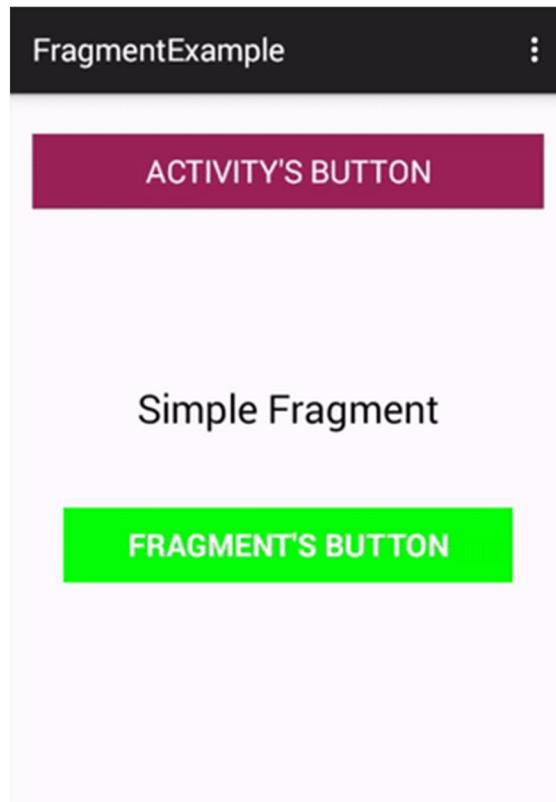
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
        android:name=".MainActivity"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN"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

Step 8: Now run the App and you will two button. Clicking on first button shows First Fragment and on click of Second Button shows the Second Fragment which is actually replacing layout(FrameLayout).

Fragment Example 2 In Android Studio:

Below is the example of Simple Fragment. In this example we create a simple Fragment and display it in our Activity by declaring a <fragment> element in our xml file. We also display a Button in our activity's xml and perform click event so whenever a user click on it a message is displayed on the screen by using a Toast. In the Fragment we display a TextView and a Button and perform click event of Button so whenever a user click on the Button a message is displayed on the screen with the help of Toast.

Below you can download code, see final output and read step by step explanation:



Step 1: Create a new project and name it FragmentExample

Step 2: Open res -> layout ->activity_main.xml (or) main.xml and add following code:

```
        android:layout_height="match_parent"
        android:layout_marginTop="10dp" />
    </LinearLayout><LinearLayout 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:orientation="vertical"
        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">
        <!-- display a Button and a Fragment -->

        <Button
            android:id="@+id/activity_button"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="10dp"
            android:background="@color/button_background_color"
            android:text="Activity's Button"
            android:textColor="@color/white"
            android:textSize="20sp" />

        <fragment
            android:id="@+id/fragments"
            android:name="com.abhiandroid.fragmentexample.SimpleFragment"
            android:layout_width="match_parent"
```

Step 3: Open src -> package -> MainActivity.java

In this step we open MainActivity and add the code for initiate the Button. After that we perform setOnClickListener on Button so whenever a user click on Button a message is displayed on the screen with the help of Toast.

```
package com.abhiandroid.fragmentexample;

import android.app.Fragment;
import android.app.FragmentManager;
import android.app.FragmentTransaction;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

    Button activityButton;
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
// get the reference of Button
activityButton = (Button) findViewById(R.id.activity_button);

// perform setOnClickListener event on Button
activityButton.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
// display a message by using a Toast
Toast.makeText(getApplicationContext(), "Activity's Button", Toast.LENGTH_LONG).show();
}
});
}
```

Step 4: Now we create a fragment by right click on your package folder and create classes and name it SimpleFragment and add the following code.

In this Fragment firstly we inflate the layout and get the reference of Button. After that we perform setOnClickListener event on Button so whenever a user click on the button a message “Fragment’s Button“ is displayed on the screen by using a Toast.

```
package com.abhiandroid.fragmentexample;

import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.Toast;

public class SimpleFragment extends Fragment {

View view;
Button firstButton;

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
Bundle savedInstanceState) {
// Inflate the layout for this fragment
view = inflater.inflate(R.layout.fragment_simple, container, false);
// get the reference of Button
```

```
firstButton = (Button) view.findViewById(R.id.firstButton);
// perform setOnClickListener on first Button
firstButton.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
// display a message by using a Toast
Toast.makeText(getActivity(), "Fragment's Button", Toast.LENGTH_LONG).show();
}
});
return view;
}
```

Step 5: Now create a xml layouts by right clicking on res/layout -> New -> Layout Resource File and name fragment_simple and add the following code in it.

Here we will design the basic simple UI by using TextView and Button.

```
<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="com.abhiandroid.fragmentexample.SimpleFragment">

    <!--TextView and Button displayed in Simple Fragment -->
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="100dp"
        android:text="Simple Fragment"
        android:textColor="@color/black"
        android:textSize="25sp" />

    <Button
        android:id="@+id/firstButton"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:layout_marginLeft="20dp"
        android:layout_marginRight="20dp"
        android:background="@color/green"
        android:text="Fragment's Button"
        android:textColor="@color/white"
        android:textSize="20sp"
        android:textStyle="bold" />
</RelativeLayout>
```

Step 6: Open res ->values ->colors.xml

In this step we define the color's that used in our xml file.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<!-- color's used in our project -->
<color name="black">#000</color>
<color name="green">#0f0</color>
<color name="white">#fff</color>
<color name="button_background_color">#925</color>
</resources>
```

Step 7: Open AndroidManifest.xml

In this step we show the Android Manifest file in which do nothing because we need only one Activity i.e MainActivity which is already defined in it. In our project we a Fragment's but we don't need to define the it in manifest because Fragment is a part of an Activity.

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

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

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

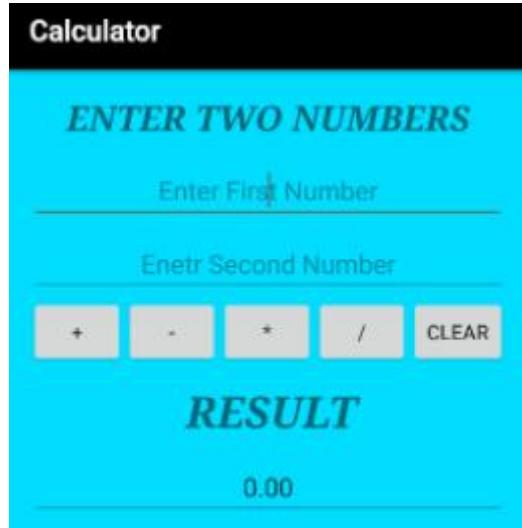
</manifest>
```

Basic Calculator App

Do you use calculator? Of Course you do in your regular life...

So why not create your own basic Calculator Android App in Android Studio and use it for doing those operations.

If you don't know how to built then you are for treat as we are going to share how the App is created in Android. This is the one of simplest App you can create to understand Android basics.

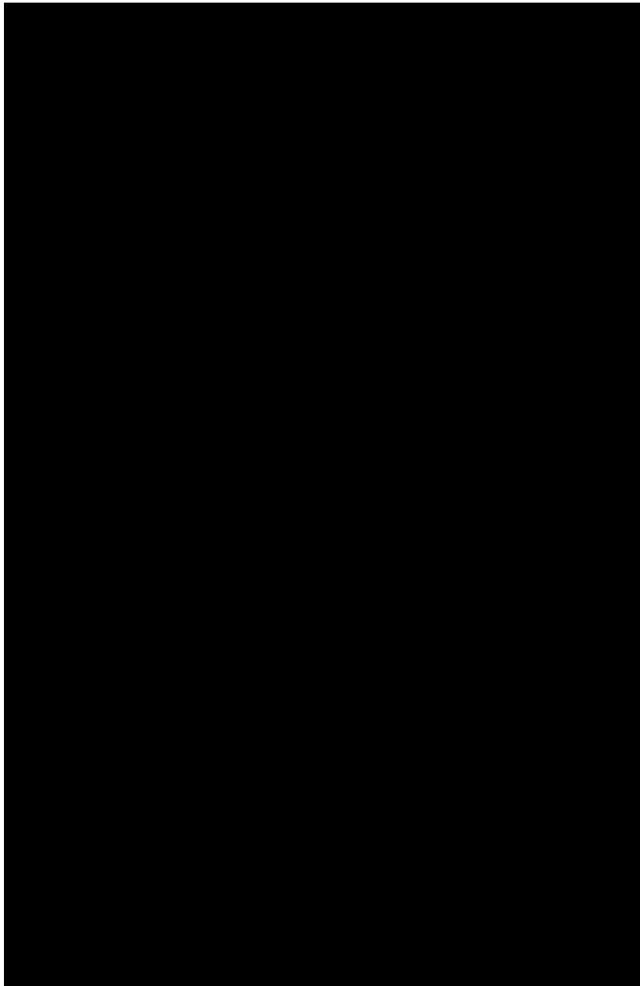


In this Calculator App tutorial we are going use of multiple Android UI components to design and step by step developing a Basic Calculator application in Android Studio.

Topics Used For Creating Calculator App – Before following the below steps it is recommended you check out `EditText`, `TextView`, `Button` & `LinearLayout` topics. Also go through JAVA OOPS concept once.

How To Create Calculator App In Android Studio:

Below you can download code, see final output and step by step explanation of Calculator App in Android Studio.



Step 1: Firstly get the android studio downloaded in your system, then open it.

Step 2: Create a new project and name it Calculator.

Step 3: Open res -> layout -> activity_main.xml (or) main.xml. Here we are going to create the application interface like add layouts, Button , TextView and EditText.

i of Step 3 – Create a LinearLayout vertical, add a textView followed by two textfields Number(decimal) for writing numbers in it. Starting code of activity_main.xml

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

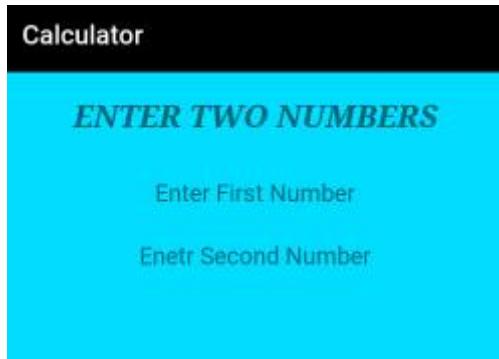
```
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="abhiandroid.com.calculater.MainActivity"
    android:orientation="vertical"
    android:gravity="top"
    android:textAlignment="center"
    android:background="@android:color/holo_blue_bright"
    android:weightSum="1">

    <TextView
        android:text="@string/enter_two_numbers"
        android:layout_width="match_parent"
        android:id="@+id/textView"
        android:layout_height="30dp"
        android:gravity="center_horizontal"
        android:textColorLink="?android:attr/editTextColor"
        tools:textStyle="bold|italic"
        android:textStyle="bold|italic"
        android:fontFamily="serif"
        android:visibility="visible"
        android:textSize="24sp"
        android:layout_weight="0.07" />

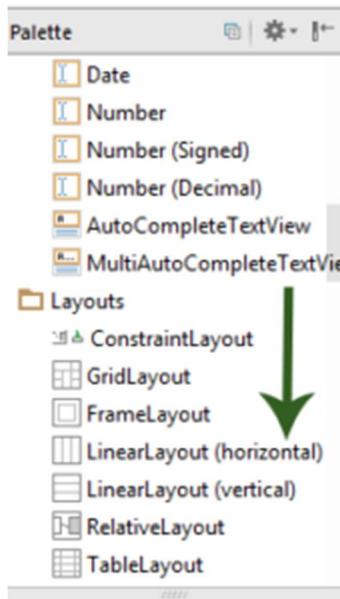
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="number"
        android:ems="10"
        android:id="@+id/editOp1"
        android:textSize="18sp"
        android:gravity="center_horizontal"
        android:layout_marginBottom="5dp"
        android:visibility="visible" />

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="number"
        android:ems="10"
        android:id="@+id/editOp2"
        android:textSize="18sp"
        android:gravity="center_horizontal"
        android:elevation="1dp" />
</LinearLayout>
```

The UI will currently look like this:



ii of Step 3 – Then before closing the above layout define another layout as LinearLayout horizontal, add five button (+, -, *, / and Clear) define their properties like id , width, height etc. in it and close the linearlayout.

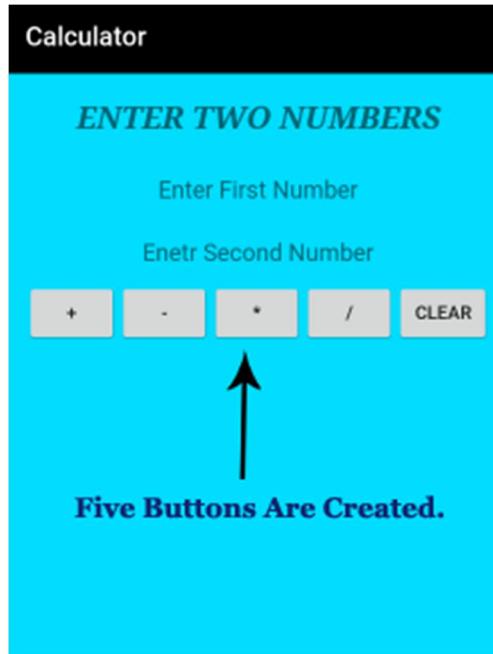


Following code of activity_main.xml . This code will be inserted in main layout:

```
<LinearLayout  
    android:orientation="horizontal"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content">
```

```
<Button  
    android:text="+"  
    android:layout_width="78dp"  
    android:layout_height="wrap_content"  
    android:id="@+id/btnadd"  
    android:layout_weight="0.03" />  
  
<Button  
    android:text="-"  
    android:layout_width="78dp"  
    android:layout_height="wrap_content"  
    android:id="@+id/btnsub"  
    android:layout_weight="0.03" />  
  
<Button  
    android:text="*"  
    android:layout_width="78dp"  
    android:layout_height="wrap_content"  
    android:id="@+id/btnmul"  
    android:layout_weight="0.03"/>  
  
<Button  
    android:text="/"  
    android:layout_height="wrap_content"  
    android:id="@+id/btndiv"  
    android:layout_width="78dp"  
    android:layout_weight="0.03" />  
  
<Button  
    android:text="Clear"  
    android:layout_width="80dp"  
    android:layout_height="wrap_content"  
    android:id="@+id/btnclr"  
    android:layout_weight="0.03" />  
</LinearLayout>
```

The UI will now look like this:



iii of Step 3 – Further in continuation with previous linearlayout add a textView, textField(Number) for displaying result which makes the interface complete.

The complete interface code of activity_main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android: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="abhiandroid.com.calculator.MainActivity"
    android:orientation="vertical"
    android:gravity="top"
    android:textAlignment="center"
    android:background="@android:color/holo_blue_bright"
    android:weightSum="1">

    <TextView
        android:text="@string/enter_two_numbers"
        android:layout_width="match_parent"
        android:id="@+id/textView"
        android:layout_height="30dp"
        android:gravity="center_horizontal"
```

```
        android:textColorLink="?android:attr/editTextColor"
        tools:textStyle="bold|italic"
        android:textStyle="bold|italic"
        android:fontFamily="serif"
        android:visibility="visible"
        android:textSize="24sp"
        android:layout_weight="0.07" />

<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:inputType="number"
    android:ems="10"
    android:id="@+id/editOp1"
    android:textSize="18sp"
    android:gravity="center_horizontal"
    android:layout_marginBottom="5dp"
    android:visibility="visible" />

<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:inputType="number"
    android:ems="10"
    android:id="@+id/editOp2"
    android:textSize="18sp"
    android:gravity="center_horizontal"
    android:elevation="1dp" />

<LinearLayout
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <Button
        android:text="+"
        android:layout_width="78dp"
        android:layout_height="wrap_content"
        android:id="@+id/btnadd"
        android:layout_weight="0.03" />

    <Button
        android:text="-"
        android:layout_width="78dp"
        android:layout_height="wrap_content"
        android:id="@+id/btnsub"
        android:layout_weight="0.03" />

    <Button
        android:text="*"
        android:layout_width="78dp"
        android:layout_height="wrap_content"
        android:id="@+id/btnmul"
        android:layout_weight="0.03" />

    <Button
        android:text="/"
        android:layout_width="78dp"
        android:layout_height="wrap_content"
        android:id="@+id/btndiv"
        android:layout_weight="0.03" />

```

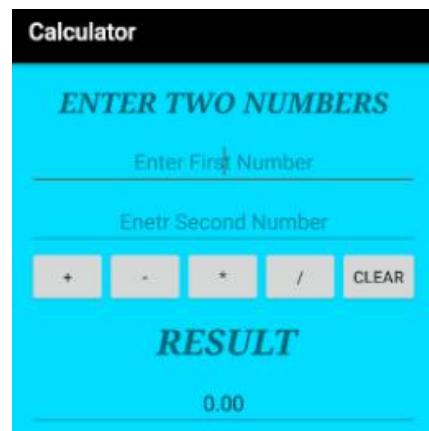
```
        android:layout_height="wrap_content"
        android:id="@+id/btndiv"
        android:layout_width="78dp"
        android:layout_weight="0.03" />

    <Button
        android:text="Clear"
        android:layout_width="80dp"
        android:layout_height="wrap_content"
        android:id="@+id/btnclr"
        android:layout_weight="0.03" />
</LinearLayout>

<TextView
    android:text="@string/result"
    android:layout_width="332dp"
    android:id="@+id/textView1"
    android:layout_marginTop="10dp"
    android:layout_height="50dp"
    android:gravity="center_horizontal"
    android:textColorLink="?android:attr/editTextColor"
    tools:textStyle="bold|italic"
    android:textStyle="bold|italic"
    android:fontFamily="serif"
    android:visibility="visible"
    android:textSize="30sp" />

<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:inputType="number"
    android:ems="10"
    android:id="@+id/result"
    android:textSize="18sp"
    android:text="0.00"
    android:gravity="center_horizontal" />
</LinearLayout>
```

So now we have designed the complete UI of the Calculator App.



Step 4: Open src -> package -> MainActivity.java. The interface part of the application is over, let's focus on adding functionality to the application. This calculator app basically perform five operations i.e addition, subtraction, multiplication, division and reset. So for that we need to define these operation over button click. For that we use setOnClickListener() function.

parseDouble() is used to convert String value to double. By default the value is String and we need to convert it into Double to perform operation over it.

If person doesn't enter the value and directly click on the any button then a Toast message will appear on the screen telling user to enter the required numbers.

```
package abhiandroid.com.calculator;

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

public class MainActivity extends AppCompatActivity {
    private EditText opr1;
    private EditText opr2;
    private Button btnadd;
    private Button btnsub;
    private Button btnmul;
    private Button btndiv;
    private Button btnclr;
    private TextView txtresult;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        opr1 = (EditText) findViewById(R.id.editOp1);
        opr2 = (EditText) findViewById(R.id.editOp2);
        btnadd = (Button) findViewById(R.id.btnadd);
        btnsub = (Button) findViewById(R.id.btnsub);
        btnmul = (Button) findViewById(R.id.btnmul);
        btndiv = (Button) findViewById(R.id.btndiv);
        btnclr = (Button) findViewById(R.id.btnclr);
        txtresult= (TextView) findViewById(R.id.result);
    }
    // Addition
    btnadd.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            if((opr1.getText().length()>0) && (opr2.getText().length()>0))
            {

```

```
        double oper1 = Double.parseDouble(opr1.getText().toString());
        double oper2 = Double.parseDouble(opr2.getText().toString());
        double result = oper1 + oper2;
        txtresult.setText(Double.toString(result));
    }
    else{
        Toast toast= Toast.makeText(MainActivity.this,"Enter The Required
Numbers",Toast.LENGTH_LONG);
        toast.show();
    }
}
});
```

//Subtraction

```
btnsub.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if((opr1.getText().length()>0) && (opr2.getText().length()>0))
        {
            double oper1 = Double.parseDouble(opr1.getText().toString());
            double oper2 = Double.parseDouble(opr2.getText().toString());
            double result = oper1 - oper2;
            txtresult.setText(Double.toString(result));
        }
        else{
            Toast toast= Toast.makeText(MainActivity.this,"Enter The Required
Numbers",Toast.LENGTH_LONG);
            toast.show();
        }
    }
});
```

// Multiplication

```
btncmul.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if((opr1.getText().length()>0) && (opr2.getText().length()>0))
        {
            double oper1 = Double.parseDouble(opr1.getText().toString());
            double oper2 = Double.parseDouble(opr2.getText().toString());
            double result = oper1 * oper2;
            txtresult.setText(Double.toString(result));
        }
        else{
            Toast toast= Toast.makeText(MainActivity.this,"Enter The Required
Numbers",Toast.LENGTH_LONG);
            toast.show();
        }
    }
});
```

// Division

```
btndiv.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if((opr1.getText().length()>0) && (opr2.getText().length()>0))
        {
            double oper1 = Double.parseDouble(opr1.getText().toString());
            double oper2 = Double.parseDouble(opr2.getText().toString());
```

```
        double result = oper1 / oper2;
        txtresult.setText(Double.toString(result));
    }
    else{
        Toast toast= Toast.makeText(MainActivity.this,"Enter The Required
Numbers",Toast.LENGTH_LONG);
        toast.show();
    }
});
// Reset Feilds
btnclr.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        opr1.setText("");
        opr2.setText("");
        txtresult.setText("0.00");
        opr1.requestFocus();
    }
});
}
```

OUTPUT:

Now run the App and you will see the basic calculator App. Enter any number and do the operations.

Youtube Android App

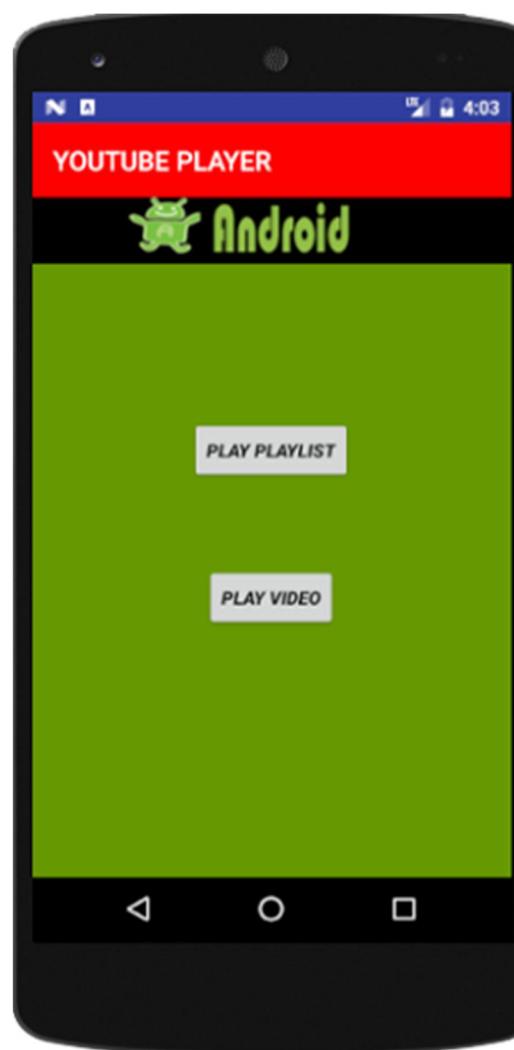
Do you know creating Youtube Android App is so easy as you just need to understand how to use Youtube API for that.

In this application we will share about adding Youtube functionality to your Android application. Further we will also create playlist and run on real device. Will make use of multiple Android UI components to design and step by step developing a Youtube App in Android Studio.

Topics Used For Creating Youtube App – Before following the below steps it is recommended you check out ImageView, Button, Linear Layout & Relative Layout topics. Also go through JAVA OOPS concept once.

Steps To Create a Youtube Application In Android Studio:

Below you can download code, see final output and step by step explanation of Youtube App in Android Studio.



Step 1: Firstly get the Android Studio downloaded in your system, then open it.

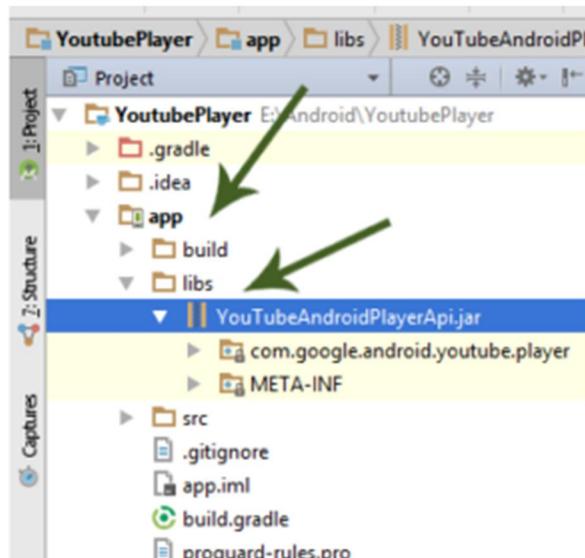
Step 2: Create a new project choose basic activity and name it YoutubePlayer.

Now please read this tutorial How To choose basic activity.

Step 3: Now click here to download the YouTube Android Player API.

Step 4: After downloading extract the downloaded compressed folder, open it and find a executable jar file in libs folder.

Step 5: Copy this library and paste in your YoutubePlayer application **app -> libs**



Step 6: Add dependencies to build.gradle file and sync. Adding this will make our application compatible to add youtube functionality.

Add in Gradle Scripts >> build.gradle (Module: app)

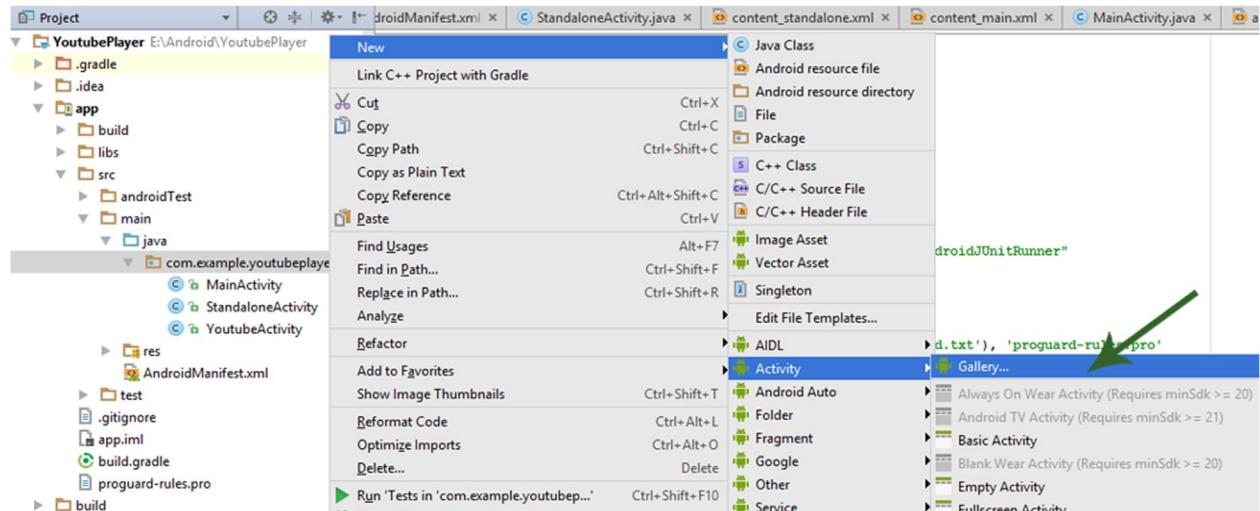
```
compile files('libs/YouTubeAndroidPlayerApi.jar')
```

```

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
        exclude group: 'com.android.support', module: 'support-annotations'
    })
    compile 'com.android.support:appcompat-v7:25.1.0'
    compile 'com.android.support:design:25.1.0'
    compile files('libs/YouTubeAndroidPlayerApi.jar')
    testCompile 'junit:junit:4.12'
}

```

Step 7: Create a new activity “YoutubeActivity” of gallery type and further select it as basic activity.



Step 8: Open YoutubeActivity.java file. Here you need to change the default code.

i of Step 8) Firstly need to change YoutubeActivity extends YouTubeBaseActivity implements YouTubePlayer.OnInitializedListener. This code will give error, to remove it we need to implement the code.

You can see it in below screenshot:

```

public class YoutubeActivity extends YouTubeBaseActivity
    implements YouTubePlayer.OnInitializedListener
{

```

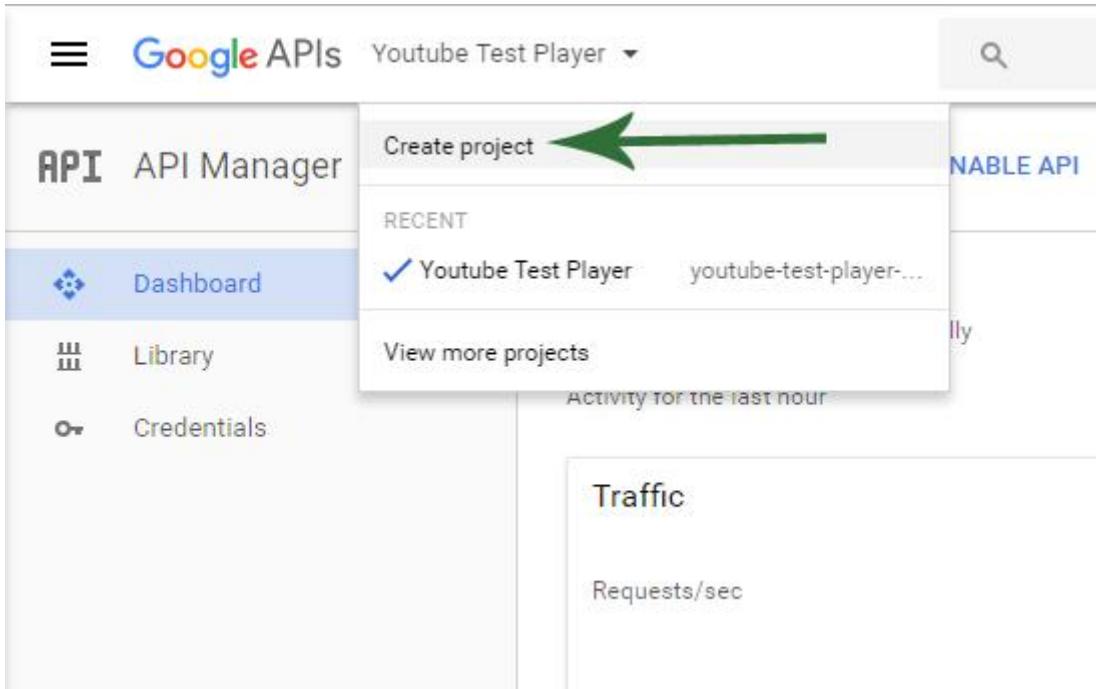
ii of Step 8 – Go to menu bar on the top click Code -> Generate -> Implements method and click ok. This will add a code where we can add toast message when youtube initialization is success and fail.

iii of Step 8) Nextly we gonna add listeners in the code as:

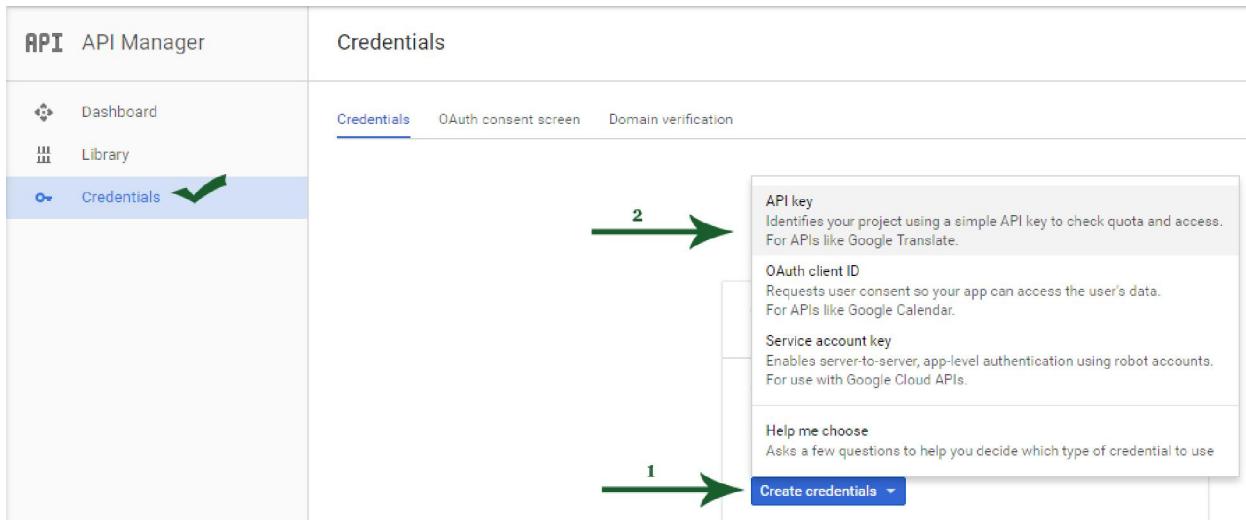
```
youTubePlayer.setPlayerStateChangeListener(playerStateChangeListener);
youTubePlayer.setPlaybackEventListener(playbackEventListener);
```

iv of Step 8) You need to add Google API Key (it's a unique key uses to take advantage of youtube functionality) and Youtube Video ID(it's the id of video we want to play) for that follow following steps:

1. Open this link first.
2. You need to login first to get into this link thought your google ID.
3. Now you need to create a project then name that project.



Click on credentials and further click API key. There will be a pop-up displaying API copy it for its usage in the application.



For Video ID open Youtube.com and play any video you wish to. To get the video ID copy the URL after the equal to sign. Similarly you can get the PlayList ID just open required playlist in Youtube.



Complete CODE of YoutubeActivity.java

```
package com.example.youtubeplayer;

import android.os.Bundle;
import android.widget.Toast;
import com.google.android.youtube.player.YouTubeBaseActivity;
import com.google.android.youtube.player.YouTubeInitializationResult;
import com.google.android.youtube.player.YouTubePlayer;
import com.google.android.youtube.player.YouTubePlayerView;

public class YoutubeActivity extends YouTubeBaseActivity
    implements YouTubePlayer.OnInitializedListener
```

```
{  
    private String GOOGLE_API_KEY = "AIzaSyBZVbNSsdQZCX_yWFCHPQ_fQMcK4xf9hDk";  
    private String YOUTUBE_VIDEO_ID = "EknEIzswvC0";  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_youtube);  
        YouTubePlayerView youTubePlayerView = (YouTubePlayerView)  
findViewById(R.id.youtube_player);  
        youTubePlayerView.initialize(GOOGLE_API_KEY, this);  
    }  
  
    @Override  
    public void onInitializationSuccess(YouTubePlayer.Provider provider, YouTubePlayer  
youTubePlayer, boolean wasRestored) {  
        Toast.makeText(this, "Initialized Youtube Player successfully",  
Toast.LENGTH_LONG).show();  
        youTubePlayer.setPlayerStateChangeListener(playerStateChangeListener);  
        youTubePlayer.setPlaybackEventListener(playbackEventListener);  
  
        if(!wasRestored) {  
            youTubePlayer.cueVideo(YOUTUBE_VIDEO_ID);  
        }  
    }  
  
    private YouTubePlayer.PlaybackEventListener playbackEventListener = new  
YouTubePlayer.PlaybackEventListener() {  
        @Override  
        public void onPlaying() {  
            Toast.makeText(YoutubeActivity.this,"Good, video is playing ok",  
Toast.LENGTH_LONG).show();  
        }  
  
        @Override  
        public void onPaused() {  
            Toast.makeText(YoutubeActivity.this,"Video has paused",  
Toast.LENGTH_LONG).show();  
        }  
  
        @Override  
        public void onStopped() {  
        }  
  
        @Override  
        public void onBuffering(boolean b) {  
        }  
  
        @Override  
        public void onSeekTo(int i) {  
        }  
};
```

```
YouTubePlayer.PlayerStateChangeListener playerStateChangeListener = new
YouTubePlayer.PlayerStateChangeListener() {
    @Override
    public void onLoading() {

    }

    @Override
    public void onLoaded(String s) {

    }

    @Override
    public void onAdStarted() {
        Toast.makeText(YoutubeActivity.this,"Click Ad now, make the video creator
rich!", Toast.LENGTH_LONG).show();
    }

    @Override
    public void onVideoStarted() {
        Toast.makeText(YoutubeActivity.this,"Video has started!",
Toast.LENGTH_LONG).show();
    }

    @Override
    public void onVideoEnded() {
        Toast.makeText(YoutubeActivity.this,"Thanks for watching!",
Toast.LENGTH_LONG).show();
    }

    @Override
    public void onError(YouTubePlayer.ErrorReason errorReason) {
    }
};

@Override
public void onInitializationFailure(YouTubePlayer.Provider provider,
YouTubeInitializationResult youTubeInitializationResult) {
    Toast.makeText(this, "Failed to Initialize Youtube Player",
Toast.LENGTH_LONG).show();
}
```

Step 9: Open content_youtube.xml file, in this we need to extend the layout for youtube activity basically we will add a custom view that enable us to play youtube videos.

Step 10: Firstly change the relative layout to linear layout and add its orientation to vertical also remove the padding in the layout. See the code to be added.

Complete code of content_youtube.xml

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout 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:id="@+id/content_youtube"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
    tools:context="com.example.youtubeplayer.YoutubeActivity"
    tools:showIn="@layout/activity_youtube">

    //custom view to enable youtube player
    <com.google.android.youtube.player.YouTubePlayerView
        android:id="@+id/youtube_player"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:background="@android:color/white">
    </com.google.android.youtube.player.YouTubePlayerView>
</LinearLayout>

```

Step 11: Add users permission for internet in AndroidManifest.xml.

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

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.youtubeplayer">
    <uses-permission android:name="android.permission.INTERNET" /> ←
    <application>

```

Step 12: Open file content_main.xml, add button in it which will redirect user to youtube player.

```

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout 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:id="@+id/content_standalone"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
    tools:context="com.example.youtubeplayer.MainActivity"

```

```
tools:showIn="@layout/activity_main"
android:background="@android:color/holo_green_dark">

<ImageView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    app:srcCompat="@drawable/pic"
    android:id="@+id/imageView"
    android:background="@android:color/background_dark"
    android:layout_alignParentTop="true" />

<Button
    android:text="Next"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignTop="@+id/btnPlayVideo"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="135dp"
    android:textStyle="bold|italic"
    android:id="@+id/next" />

<Button
    android:id="@+id/btnPlayVideo"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/play_video"
    android:textStyle="bold|italic"
    android:layout_marginTop="93dp"
    android:layout_below="@+id/imageView"
    android:layout_centerHorizontal="true" />

</RelativeLayout>
```

Step 13: Now open MainActivity.java class and paste the following code.

In this code we gonna add the onclickListener over button click i.e if user click on button video will run and a next button which will redirect to next activity.

```
package com.example.youtubeplayer;

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

public class MainActivity extends AppCompatActivity implements View.OnClickListener{
    private Button btnSingle;
    private Button btnNext;

    @Override
```

```

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);
    btnSingle = (Button) findViewById(R.id.btnPlayVideo);
    btnNext= (Button) findViewById(R.id.next);
    btnSingle.setOnClickListener(this);
    btnNext.setOnClickListener(this);
}

@Override
public void onClick(View v) {
    Intent intent= null;

    switch (v.getId()){
        case R.id.btnPlayVideo:
            intent = new Intent(MainActivity.this, YoutubeActivity.class);
            break;
        case R.id.next:
            intent = new Intent(MainActivity.this , StandaloneActivity.class);
            break;
        default:
    }

    if(intent!= null){
        startActivity(intent);
    }
}
}

```

Step 14: Similarly create another basic activity and name it StandaloneActivity to see the Youtube Playlist functionality. In this we will define a PlayList ID that you can get same as we extracted Video ID.

Step 15: Open content_standalone.xml file in this add two button and add functionality over it in java file.

```

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout 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:id="@+id/content_standalone"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
    tools:context="com.example.youtubeplayer.StandaloneActivity"
    tools:showIn="@layout/activity_standalone"

```

```
        android:background="@android:color/holo_green_dark">

    <ImageView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:srcCompat="@drawable/pic"
        android:id="@+id/imageView"
        android:background="@android:color/background_dark"
        android:layout_alignParentTop="true"
        android:contentDescription="@string/pic" />

    <Button
        android:id="@+id	btnVideo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/play_video"
        android:textStyle="bold|italic"
        android:layout_marginBottom="186dp"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true" />

    <Button
        android:id="@+id	btnPlayList"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/play_playlist"
        android:textStyle="bold|italic"
        android:layout_marginBottom="63dp"
        android:layout_above="@+id	btnVideo"
        android:layout_centerHorizontal="true" />

</RelativeLayout>
```

Step 16: Now open src -> package -> StandaloneActivity.java. In this we gonna add the onclickListener over button click i.e if user click on PlayVideo video will play otherwise on clicking Play Playlist playlist will run of defined ID.

```
package com.example.youtubeplayer;

import android.content.Intent;
import android.os.Bundle;
import android.support.design.widget.FloatingActionButton;
import android.support.design.widget.Snackbar;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.View;
import android.widget.Button;

import com.google.android.youtube.player.YouTubeStandalonePlayer;
```

```
public class StandaloneActivity extends AppCompatActivity implements View.OnClickListener {
    private String GOOGLE_API_KEY = "AIzaSyBZVbNSsdQZCX_yWFCHPQ_fQMcK4xf9hDk";
    private String YOUTUBE_VIDEO_ID = "EknEIZswvC0";
    private String YOUTUBE_PLAYLIST_ID= "PLS1Qu1Wo1RIbb1cYyzZpLFCKvdYV_yJ-E";
    private Button btnPlayVideo;
    private Button btnPlayplaylist;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_standalone);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        btnPlayplaylist= (Button) findViewById(R.id.btnPlaylist);
        btnPlayVideo= (Button) findViewById(R.id.btnVideo);
        btnPlayVideo.setOnClickListener(this);
        btnPlayplaylist.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        Intent intent= null;
        switch (v.getId()){
            case R.id.btnVideo:
                intent =
YouTubeStandalonePlayer.createVideoIntent(this,GOOGLE_API_KEY,YOUTUBE_VIDEO_ID);
                break;
            case R.id.btnPlaylist:
                intent =
YouTubeStandalonePlayer.createPlaylistIntent(this,GOOGLE_API_KEY,YOUTUBE_PLAYLIST_ID);

                break;
            default:
        }

        if(intent!= null){
            startActivity(intent);
        }
    }
}
```

OUTPUT:

Now run the App and use the play the Youtube video you added.

Countdown Timer Android App

CountDown Timer App is about setting a time that moves in reverse like it shows the time left in upcoming event. Likewise here we are making an Android App in context to CRICKET WORLD CUP which will start in 2019. So let's begin App creation step by step towards its completion.

In this CountDown Timer App tutorial we are going to use multiple Android UI components to design its interface in Android Studio.

Topics Used For Creating CountDown Timer App – Before following the below steps it is recommended you check out TextView, Relative Layout & Linear Layout topics. Also go through JAVA OOPS concept once.

How To Create CountDown Timer App In Android Studio:

Below you can download code, see final output and step by step explanation of CountDown Timer App in Android Studio.



Step 1: Firstly get the android studio downloaded in your system, then open it.

Step 2: Create a new project and name it CountDownTimer.

Step 3: Open res -> layout -> activity_main.xml (or) main.xml. Here we are going to create the application interface like add layouts(linearlayout & relativelayout), TextView. Carefully analyze the code as it's a bit complicated, we used LinearLayout for displaying each part of CountDown Timer(days, hours, minutes , seconds).

The complete interface code of activity_main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:background="@drawable/backimage"
    tools:context="com.example.countdown.MainActivity">

    <TextView
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/counter"
        android:textSize="30sp"
        android:textColor="#ffeeee"
        android:textStyle="bold|italic"
        android:layout_above="@+id/relativeLayout"
        android:layout_centerHorizontal="true"
        android:id="@+id/textViewheader1" />

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true"
        android:id="@+id/relativeLayout">

        <LinearLayout
            android:id="@+id/LinearLayout"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:gravity="center"
            android:orientation="horizontal">

            <TextView
                android:id="@+id/tveventStart"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:paddingLeft="50sp"
                android:gravity="center"
                android:singleLine="true"
                android:text="@string/worldcup_2k19"
                android:textColor="#fff"
                android:textSize="24sp"
                android:textStyle="bold"
                android:visibility="gone" />
            <LinearLayout
                android:id="@+id/LinearLayout1"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:background="@drawable/counter"
                android:gravity="center"
                android:orientation="vertical" >

                <TextView
                    android:id="@+id/txtDay"
                    android:layout_width="fill_parent"
                    android:layout_height="wrap_content"
                    android:layout_weight="3"
                    android:gravity="center"
                    android:textAppearance="?android:attr/textAppearanceLarge"
                    android:textColor="#4a0000"
                    android:textSize="35sp"
                    android:textStyle="bold" />
        
```

```
<TextView
    android:id="@+id/txt_Day"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:gravity="center_horizontal"
    android:text="@string/days"
    android:textAppearance="?android:attr/textAppearanceSmall"
    android:textColor="#fff"
    android:textStyle="bold" />
</LinearLayout>

<LinearLayout
    android:id="@+id/Layout2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:background="@drawable/counter"
    android:gravity="center"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/txtHour"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="3"
        android:gravity="center"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:textColor="#4a0000"
        android:textSize="35sp"
        android:textStyle="bold" />

    <TextView
        android:id="@+id/txt_Hour"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:gravity="center_horizontal"
        android:text="@string/hours"
        android:textAppearance="?android:attr/textAppearanceSmall"
        android:textColor="#fff"
        android:textStyle="bold" />
</LinearLayout>

<LinearLayout
    android:id="@+id/Layout3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:background="@drawable/counter"
    android:gravity="center"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/txtMinute"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
```

```
        android:layout_weight="3"
        android:gravity="center"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:textColor="#4a0000"
        android:textSize="35sp"
        android:textStyle="bold" />

    <TextView
        android:id="@+id/txt_Minute"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:gravity="center_horizontal"
        android:text="@string/minutes"
        android:textAppearance="?android:attr/textAppearanceSmall"
        android:textColor="#ffff"
        android:textStyle="bold" />
</LinearLayout>

<LinearLayout
    android:id="@+id/LinearLayout4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:background="@drawable/counter"
    android:gravity="center"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/txtSecond"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="3"
        android:gravity="center"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:textColor="#4a0000"
        android:textSize="35sp"
        android:textStyle="bold" />

    <TextView
        android:id="@+id/txt_Second"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:gravity="center_horizontal"
        android:text="@string/seconds"
        android:textAppearance="?android:attr/textAppearanceSmall"
        android:textColor="#ffff"
        android:textStyle="bold" />
</LinearLayout>
</LinearLayout>
</RelativeLayout>

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/countleft"
```

```
        android:layout_below="@+id/relativeLayout"
        android:layout_centerHorizontal="true"
        android:id="@+id/textViewheader2"
        android:textSize="35sp"
        android:textStyle="bold|italic"
        android:textColor="@android:color/background_dark" />

    </RelativeLayout>
```



Step 4: Open src -> package -> MainActivity.java. The interface part of the application is over, let's focus on adding functionality to the application.

In this app we use System date and the upcoming event date in our case it's 30-may-2019(World Cup 2k19). Nextly we will subtract the both dates and get the time in milliseconds further we have some arithmetic operations over it to get the left time till the event arrival.

You also see that we use handler in this code to schedule message and runnable in future at some point and scheduling is accomplished by various methods but we used postDelayed(runnable, long).

```
package com.example.countdown;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import java.text.SimpleDateFormat;
import java.util.Date;
import android.os.Handler;
import android.view.View;
import android.widget.TextView;
```

```
public class MainActivity extends AppCompatActivity {

    private TextView txtDay, txtHour, txtMinute, txtSecond;
    private TextView tvEventStart;
    private Handler handler;
    private Runnable runnable;

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

        txtDay = (TextView) findViewById(R.id.txtDay);
        txtHour = (TextView) findViewById(R.id.txtHour);
        txtMinute = (TextView) findViewById(R.id.txtMinute);
        txtSecond = (TextView) findViewById(R.id.txtSecond);
        tvEventStart = (TextView) findViewById(R.id.tveventStart);

        countDownStart();
    }

    public void countDownStart() {
        handler = new Handler();
        runnable = new Runnable() {
            @Override
            public void run() {
                handler.postDelayed(this, 1000);
                try {
                    SimpleDateFormat dateFormat = new SimpleDateFormat(
                        "yyyy-MM-dd");
                    // Please here set your event date//YYYY-MM-DD
                    Date futureDate = dateFormat.parse("2019-5-30");
                    Date currentDate = new Date();
                    if (!currentDate.after(futureDate)) {
                        long diff = futureDate.getTime()
                            - currentDate.getTime();
                        long days = diff / (24 * 60 * 60 * 1000);
                        diff -= days * (24 * 60 * 60 * 1000);
                        long hours = diff / (60 * 60 * 1000);
                        diff -= hours * (60 * 60 * 1000);
                        long minutes = diff / (60 * 1000);
                        diff -= minutes * (60 * 1000);
                        long seconds = diff / 1000;
                        txtDay.setText("" + String.format("%02d", days));
                        txtHour.setText("" + String.format("%02d", hours));
                        txtMinute.setText("" +
                            + String.format("%02d", minutes));
                        txtSecond.setText("" +
                            + String.format("%02d", seconds));
                    } else {
                        tvEventStart.setVisibility(View.VISIBLE);
                        tvEventStart.setText("The event started!");
                        textViewGone();
                    }
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        };
    }
}
```

```
        }
    };
    handler.postDelayed(runnable, 1 * 1000);
}

public void textViewGone() {
    findViewById(R.id.LinearLayout1).setVisibility(View.GONE);
    findViewById(R.id.LinearLayout2).setVisibility(View.GONE);
    findViewById(R.id.LinearLayout3).setVisibility(View.GONE);
    findViewById(R.id.LinearLayout4).setVisibility(View.GONE);
    findViewById(R.id.textViewheader1).setVisibility(View.GONE);
    findViewById(R.id.textViewheader2).setVisibility(View.GONE);
}
}
```

OUTPUT:

Now run the App and you will see the CountDown for the event.

Premium Android App Source Code

List of premium Android App source code on sale at a very affordable price:

1. [**Convert Website Into Advance Android App Source Code**](#) - Do you have app for your website? If NO, then use my Ultimate WebView App source code and convert your website into an advance Android App in just 15 minutes. The App code comes with 20+ advance features, clean code and build in Android Studio. Some of the key features are Firebase push notification, customized navigation menu, ActionBar, Admob, progressbar, offline handling, video support and lots more.
2. [**Smart News Android App Source Code**](#) - This is a web-admin based Android App source code. Using this code you can create news, blog or information type App whose content will be dynamically updated directly from the server with ease. This App code have 20+ advance features, clean code and build in Android Studio.

If you have any query or question in mind, please email to info@abhiandroid.com and I will get back to you within 24 hours(mostly ASAP).

Thanks!

Thank you for reading. I hope you like it...

For more Android tutorials [please like our Facebook page](#) and stay in touch with AbhiAndroid.com!