TextView Control

A **TextView** displays text to the user and optionally allows them to edit it. A TextView is a complete text editor, however the basic class is configured to not allow editing.

TextView Attributes

Following are the important attributes related to TextView control. You can check Android official documentation for complete list of attributes and related methods which you can use to change these attributes are run time.

Sr.No.	Attribute & Description
1	**android:id** This is the ID which uniquely identifies the control.
2	**android:capitalize** If set, specifies that this TextView has a textual input method and should automatically capitalize what the user types. • Don't automatically capitalize anything - 0 • Capitalize the first word of each sentence - 1 • Capitalize the first letter of every word - 2 • Capitalize every character - 3
3	**android:cursorVisible** Makes the cursor visible (the default) or invisible. Default is false.
4	**android:editable** If set to true, specifies that this TextView has an input method.
5	**android:fontFamily** Font family (named by string) for the text.
6	**android:gravity** Specifies how to align the text by the view's x- and/or y-axis when the text is smaller than the view.
7	**android:hint** Hint text to display when the text is empty.
8	**android:inputType** The type of data being placed in a text field. Phone, Date, Time, Number, Password etc.
9	**android:maxHeight**

	Makes the TextView be at most this many pixels tall.
10	**android:maxWidth** Makes the TextView be at most this many pixels wide.
11	**android:minHeight** Makes the TextView be at least this many pixels tall.
12	**android:minWidth** Makes the TextView be at least this many pixels wide.
13	**android:password** Whether the characters of the field are displayed as password dots instead of themselves. Possible value either "true" or "false".
14	**android:phoneNumber** If set, specifies that this TextView has a phone number input method. Possible value either "true" or "false".
15	**android:text** Text to display.
16	**android:textAllCaps** Present the text in ALL CAPS. Possible value either "true" or "false".
17	**android:textColor** Text color. May be a color value, in the form of "#rgb", "#argb", "#rrggbb", or "#aarrggbb".
18	**android:textColorHighlight** Color of the text selection highlight.
19	**android:textColorHint** Color of the hint text. May be a color value, in the form of "#rgb", "#argb", "#rrggbb", or "#aarrggbb".
20	**android:textIsSelectable** Indicates that the content of a non-editable text can be selected. Possible value either "true" or "false".
21	**android:textSize** Size of the text. Recommended dimension type for text is "sp" for scaled-pixels (example: 15sp).

22	**android:textStyle** Style (bold, italic, bolditalic) for the text. You can use or more of the following values separated by '|'. • normal - 0 • bold - 1 • italic - 2
23	**android:typeface** Typeface (normal, sans, serif, monospace) for the text. You can use or more of the following values separated by '|'. • normal - 0 • sans - 1 • serif - 2 • monospace - 3

Example

This example will take you through simple steps to show how to create your own Android application using Linear Layout and TextView.

Step	Description
1	You will use Android studio to create an Android application and name it as *demo* under a package *com.example.demo* as explained in the *Hello World Example* chapter.
2	Modify *src/MainActivity.java* file to add necessary code .
2	Modify the default content of *res/layout/activity_main.xml* file to include Android UI control.
3	No need to change default string constants at string.xml file. Android studio takes care of default string constants.
4	Run the application to launch Android emulator and verify the result of the changes done in the application.

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

```
package com.example.demo;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.View;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends Activity {
```

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

        //--- text view---
        TextView txtView = (TextView) findViewById(R.id.text_id);
    }
}
```

Following will be the content of **res/layout/activity_main.xml** file −

```
<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/text_id"
        android:layout_width="300dp"
        android:layout_height="200dp"
        android:capitalize="characters"
        android:text="hello_world"
        android:textColor="@android:color/holo_blue_dark"

android:textColorHighlight="@android:color/primary_text_dark"
        android:layout_centerVertical="true"
        android:layout_alignParentEnd="true"
        android:textSize="50dp"/>

</RelativeLayout>
```

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

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

Following is the default content of **AndroidManifest.xml** −

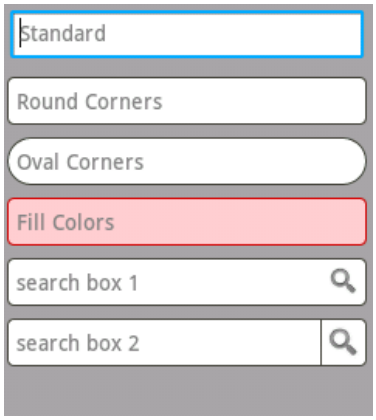
```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.demo" >
```

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

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

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

      </activity>

   </application>
</manifest>
```

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

EditText Control

A EditText is an overlay over TextView that configures itself to be editable. It is the predefined subclass of TextView that includes rich editing capabilities.



Styles of edit text

EditText Attributes

Following are the important attributes related to EditText control. You can check Android official documentation for complete list of attributes and related methods which you can use to change these attributes are run time.

Inherited from **android.widget.TextView** Class −

Sr.No	Attribute & Description
1	**android:autoText** If set, specifies that this TextView has a textual input method and automatically corrects some common spelling errors.
2	**android:drawableBottom** This is the drawable to be drawn below the text.
3	**android:drawableRight** This is the drawable to be drawn to the right of the text.
4	**android:editable** If set, specifies that this TextView has an input method.
5	**android:text** This is the Text to display.

Inherited from **android.view.View** Class −

Sr.No	Attribute & Description
1	**android:background** This is a drawable to use as the background.
2	**android:contentDescription** This defines text that briefly describes content of the view.
3	**android:id** This supplies an identifier name for this view.

4	**android:onClick** This is the name of the method in this View's context to invoke when the view is clicked.
5	**android:visibility** This controls the initial visibility of the view.

Example

This example will take you through simple steps to show how to create your own Android application using Linear Layout and EditText.

Step	Description
1	You will use Android studio IDE to create an Android application and name it as *demo* under a package *com.example.demo* as explained in the *Hello World Example* chapter.
2	Modify *src/MainActivity.java* file to add a click event.
3	Modify the default content of *res/layout/activity_main.xml* file to include Android UI control.
4	Define required necessary string constants in *res/values/strings.xml* file
5	Run the application to launch Android emulator and verify the result of the changes done in the application.

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

```
package com.example.demo;

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

import android.view.View;
import android.view.View.OnClickListener;

import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends Activity {
   EditText eText;
   Button btn;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      eText = (EditText) findViewById(R.id.edittext);
      btn = (Button) findViewById(R.id.button);
```

```
      btn.setOnClickListener(new OnClickListener() {
         public void onClick(View v) {
            String str = eText.getText().toString();
            Toast                          msg                          =
Toast.makeText(getBaseContext(),str,Toast.LENGTH_LONG);
            msg.show();
         }
      });
   }
}
```

Following will be the content of **res/layout/activity_main.xml** file −

```
<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_alignParentLeft="true"
      android:layout_alignParentTop="true"
      android:layout_marginLeft="14dp"
      android:layout_marginTop="18dp"
      android:text="@string/example_edittext" />
   <Button
      android:id="@+id/button"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignLeft="@+id/textView1"
      android:layout_below="@+id/textView1"
      android:layout_marginTop="130dp"
      android:text="@string/show_the_text" />
   <EditText
      android:id="@+id/edittext"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:layout_alignLeft="@+id/button"
      android:layout_below="@+id/textView1"
      android:layout_marginTop="61dp"
      android:ems="10"
      android:text="@string/enter_text"   android:inputType="text"
/>

</RelativeLayout>
```

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

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="app_name">demo</string>
   <string          name="example_edittext">Example          showing
EditText</string>
   <string name="show_the_text">Show the Text</string>
   <string name="enter_text">text changes</string>
</resources>
```

Following is the default content of **AndroidManifest.xml** −

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.example.demo" >
   <application
      android:allowBackup="true"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:theme="@style/AppTheme" >
      <activity
         android:name="com.example.demo.MainActivity"
         android:label="@string/app_name" >
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category
android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
   </application>
</manifest>
```

Let's try to run your **demo** application

Button Control

A Button is a Push-button which can be pressed, or clicked, by the user to perform an action.

Button Attributes

Following are the important attributes related to Button control. You can check Android official documentation for complete list of attributes and related methods which you can use to change these attributes are run time.

Inherited from **android.widget.TextView** Class −

Sr.No	Attribute & Description
1	**android:autoText** If set, specifies that this TextView has a textual input method and automatically corrects some common spelling errors.
2	**android:drawableBottom** This is the drawable to be drawn below the text.
3	**android:drawableRight** This is the drawable to be drawn to the right of the text.
4	**android:editable** If set, specifies that this TextView has an input method.
5	**android:text** This is the Text to display.

Inherited from **android.view.View** Class −

Attribute	Description
1	**android:background** This is a drawable to use as the background.
2	**android:contentDescription** This defines text that briefly describes content of the view.
3	**android:id** This supplies an identifier name for this view.
4	**android:onClick** This is the name of the method in this View's context to invoke when the view is clicked.
5	**android:visibility** This controls the initial visibility of the view.

Example

This example will take you through simple steps to show how to create your own Android application using Linear Layout and Button.

Step	Description
1	You will use Android studio IDE to create an Android application and name it as *myapplication* under a package *com.example.saira_000.myapplication* as explained in the *Hello World Example* chapter.
2	Modify *src/MainActivity.java* file to add a click event.
3	Modify the default content of *res/layout/activity_main.xml* file to include Android UI control.
4	No need to declare default string constants at string.xml, Android studio takes care of default string constants.
5	Run the application to launch Android emulator and verify the result of the changes done in the application.

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

```
package com.example.saira_000.myapplication;

import android.content.Intent;
import android.net.Uri;
import android.support.v7.app.ActionBarActivity;
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 ActionBarActivity {
   Button b1,b2,b3;

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

      b1=(Button)findViewById(R.id.button);
      b1.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View v) {
            Toast.makeText(MainActivity.this,"YOUR
MESSAGE",Toast.LENGTH_LONG).show();
         }
      });
   }
}
```

Following will be the content of **res/layout/activity_main.xml** file −

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   android:paddingBottom="@dimen/activity_vertical_margin"
   tools:context=".MainActivity">

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

   <TextView
      android:id="@+id/textView2"
      android:layout_width="wrap_content"
```

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

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

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

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

</RelativeLayout>
```

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

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

Following is the default content of **AndroidManifest.xml** −

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

   <application
      android:allowBackup="true"
      android:icon="@drawable/ic_launcher"
```

```
      android:label="@string/app_name"
      android:theme="@style/AppTheme" >

      <activity
         android:name="com.example.guidemo4.MainActivity"
         android:label="@string/app_name" >

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

      </activity>

   </application>
</manifest>
```

Let's try to run your **GUIDemo4** application. I assume you had created your **AVD** while doing environment setup. To run the app from Android Studio, open one of your project's activity files and click Run icon from the toolbar.Android Studio installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display following Emulator window −

The following screen will appear by clicking on Button −

ImageButton Control

An ImageButton is an AbsoluteLayout which enables you to specify the exact location of its children. This shows a button with an image (instead of text) that can be pressed or clicked by the user.

Android button style set

ImageButton Attributes

Following are the important attributes related to ImageButton control. You can check Android official documentation for complete list of attributes and related methods which you can use to change these attributes are run time.

Inherited from **android.widget.ImageView** Class −

Sr.No	Attribute & Description
1	**android:adjustViewBounds** Set this to true if you want the ImageView to adjust its bounds to preserve the aspect ratio of its drawable.
2	**android:baseline** This is the offset of the baseline within this view.
3	**android:baselineAlignBottom** If true, the image view will be baseline aligned with based on its bottom edge.
4	**android:cropToPadding** If true, the image will be cropped to fit within its padding.
5	**android:src** This sets a drawable as the content of this ImageView.

Inherited from **android.view.View** Class −

Sr.No	Attribute & Description
1	**android:background** This is a drawable to use as the background.
2	**android:contentDescription** This defines text that briefly describes content of the view.
3	**android:id** This supplies an identifier name for this view
4	**android:onClick** This is the name of the method in this View's context to invoke when the view is clicked.
5	**android:visibility** This controls the initial visibility of the view.

Example

This example will take you through simple steps to show how to create your own Android application using Linear Layout and ImageButton.

Step	Description
1	You will use Android studio IDE to create an Android application and name it as *myapplication* under a package *com.example.myapplication* as explained in the *Hello World Example* chapter.
2	Modify *src/MainActivity.java* file to add a click event.
3	Modify the default content of *res/layout/activity_main.xml* file to include Android UI control.
4	No need to define default constants in android, Android studio takes care of default constants.
5	Run the application to launch Android emulator and verify the result of the changes done in the application.

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

In the below example abc indicates the image of tutorialspoint

```
package com.example.myapplication;

import android.os.Bundle;
import android.app.Activity;
import android.view.View;
import android.view.View.OnClickListener;
```

```
import android.widget.ImageButton;
import android.widget.Toast;

public class MainActivity extends Activity {
   ImageButton imgButton;

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

      imgButton =(ImageButton)findViewById(R.id.imageButton);
      imgButton.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View v) {
            Toast.makeText(getApplicationContext(),"You  download
is
               resumed",Toast.LENGTH_LONG).show();
         }
      });
   }
}
```

Following will be the content of **res/layout/activity_main.xml** file −

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
   android:layout_height="match_parent"
android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   android:paddingBottom="@dimen/activity_vertical_margin"
   tools:context=".MainActivity">

   <TextView android:text="Tutorials Point"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:textSize="30dp"
      android:layout_alignParentTop="true"
      android:layout_alignRight="@+id/imageButton"
      android:layout_alignEnd="@+id/imageButton" />

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

```
</RelativeLayout>
```

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

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

Following is the default content of **AndroidManifest.xml** −

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.example.myapplication" >
   <application
      android:allowBackup="true"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:theme="@style/AppTheme" >
      <activity
         android:name="com.example.myapplication.MainActivity"
         android:label="@string/app_name" >
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category
android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
   </application>
</manifest>
```

Let's try to run your **myapplication** application

The following screen will appear after ImageButton is clicked,It shows a toast message.

ToggleButton Control

A ToggleButton displays checked/unchecked states as a button. It is basically an on/off button with a light indicator.

ToggleButton Attributes

Following are the important attributes related to ToggleButton control. You can check Android official documentation for complete list of attributes and related methods which you can use to change these attributes are run time.

Sr.No.	Attribute & Description
1	**android:disabledAlpha** This is the alpha to apply to the indicator when disabled.
2	**android:textOff** This is the text for the button when it is not checked.
3	**android:textOn** This is the text for the button when it is checked.

Inherited from **android.widget.TextView** Class −

Sr.No.	Attribute & Description
1	**android:autoText** If set, specifies that this TextView has a textual input method and automatically corrects some common spelling errors.
2	**android:drawableBottom** This is the drawable to be drawn below the text.
3	**android:drawableRight** This is the drawable to be drawn to the right of the text.
4	**android:editable** If set, specifies that this TextView has an input method.
5	**android:text** This is the Text to display.

Inherited from **android.view.View** Class −

Sr.No.	Attribute & Description
1	**android:background** This is a drawable to use as the background.
2	**android:contentDescription** This defines text that briefly describes content of the view.
3	**android:id** This supplies an identifier name for this view,
4	**android:onClick** This is the name of the method in this View's context to invoke when the view is clicked.
5	**android:visibility** This controls the initial visibility of the view.

Example

This example will take you through simple steps to show how to create your own Android application using Linear Layout and ToggleButton.

Step	Description
1	You will use Android studio IDE to create an Android application and name it as *My Application* under a package *com.example.saira_000.myapplication* as explained in the *Hello World Example* chapter.
2	Modify *src/MainActivity.java* file to add a click event.
2	Modify the default content of *res/layout/activity_main.xml* file to include Android UI control.
3	No need to declare default constants.Android studio takes care of default constants at string.xml
4	Run the application to launch Android emulator and verify the result of the changes done in the application.

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

In the below example abc indicates the image of tutorialspoint

```
package com.example.saira_000.myapplication;

import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;
import android.widget.ToggleButton;

public class MainActivity extends ActionBarActivity {
   ToggleButton tg1,tg2;
   Button b1;
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      tg1=(ToggleButton)findViewById(R.id.toggleButton);
      tg2=(ToggleButton)findViewById(R.id.toggleButton2);

      b1=(Button)findViewById(R.id.button2);
      b1.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View v) {
            StringBuffer result = new StringBuffer();
            result.append("You have clicked first ON Button-:)
").append(tg1.getText());
            result.append("You have clicked Second ON Button -:)
").append(tg2.getText());
```

```
            Toast.makeText(MainActivity.this,
result.toString(),Toast.LENGTH_SHORT).show();
         }
      });
   }
}
```

Following will be the content of **res/layout/activity_main.xml** file −

```
<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/textView2"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Tutorials point"
      android:textColor="#ff87ff09"
      android:textSize="30dp"
      android:layout_above="@+id/imageButton"
      android:layout_centerHorizontal="true"
      android:layout_marginBottom="40dp" />

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

   <ToggleButton
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="On"
      android:id="@+id/toggleButton"
      android:checked="true"
      android:layout_below="@+id/imageButton"
      android:layout_toEndOf="@+id/button2/>

   <ToggleButton
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Off"
      android:id="@+id/toggleButton2"
```

```
      android:checked="true"
      android:layout_alignTop="@+id/toggleButton" />

   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/button2"
      android:text="ClickMe"
      android:layout_alignParentBottom="true"
      android:layout_centerHorizontal="true" />

</RelativeLayout>
```

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

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

Following is the default content of **AndroidManifest.xml** −

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

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

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

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

      </activity>

   </application>
</manifest>
```

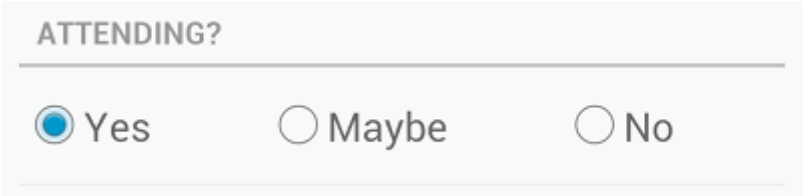
Let's try to run your **My Application** application. I assume you had created your **AVD** while doing environment setup. To run the app from Android studio, open one of your project's activity files and click Run icon from the toolbar. Android studio installs the app on your AVD and starts it and if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

The following screen will appear −

If you have clicked first on Button, you would get a message on Toast as **You have clicked first ON Button-:)** or else if you clicked on second on button, you would get a message on Toast as **You have clicked Second ON Button -:)**

RadioButton Control

A RadioButton has two states: either checked or unchecked.This allows the user to select one option from a set.



Example

This example will take you through simple steps to show how to create your own Android application using Linear Layout and RadioButton.

Step	Description
1	You will use Android studio to create an Android application and name it as *My Application* under a package *com.example.saira_000.myapplication* as explained in the *Hello World Example* chapter.
2	Modify *src/MainActivity.java* file to add a click event.
2	Modify the default content of *res/layout/activity_main.xml* file to include Android UI control.
3	Android studio takes care of default constants so no need to declare default constants at string.xml file
4	Run the application to launch Android emulator and verify the result of the changes done in the application.

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

In the below example abc indicates the image of tutorialspoint

```
package com.example.saira_000.myapplication;

import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.ImageButton;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.Toast;

public class MainActivity extends ActionBarActivity {
   RadioGroup rg1;
   RadioButton rb1;
   Button b1;

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

   private void addListenerRadioButton() {
      rg1 = (RadioGroup) findViewById(R.id.radioGroup);
      b1 = (Button) findViewById(R.id.button2);
      b1.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View v) {
            int selected=rg1.getCheckedRadioButtonId();
            rb1=(RadioButton)findViewById(selected);

Toast.makeText(MainActivity.this,rb1.getText(),Toast.LENGTH_LONG)
.show();
         }
      });
   }
}
```

Following will be the content of **res/layout/activity_main.xml** file −

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingBottom="@dimen/activity_vertical_margin"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
```

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

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

<TextView
   android:id="@+id/textView2"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Tutorials point"
   android:textColor="#ff87ff09"
   android:textSize="30dp"
   android:layout_above="@+id/imageButton"
   android:layout_centerHorizontal="true"
   android:layout_marginBottom="40dp" />

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

<Button
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:id="@+id/button2"
   android:text="ClickMe"
   android:layout_alignParentBottom="true"
   android:layout_centerHorizontal="true" />

<RadioGroup
   android:id="@+id/radioGroup"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:layout_below="@+id/imageButton"
   android:layout_alignLeft="@+id/textView2"
   android:layout_alignStart="@+id/textView2">

   <RadioButton
      android:layout_width="142dp"
      android:layout_height="wrap_content"
      android:text="JAVA"
      android:id="@+id/radioButton"
      android:textSize="25dp"
```

```
         android:textColor="@android:color/holo_red_light"
         android:checked="false"
         android:layout_gravity="center_horizontal" />

      <RadioButton
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="ANDROID"
         android:id="@+id/radioButton2"
         android:layout_gravity="center_horizontal"
         android:checked="false"
         android:textColor="@android:color/holo_red_dark"
         android:textSize="25dp" />

      <RadioButton
         android:layout_width="136dp"
         android:layout_height="wrap_content"
         android:text="HTML"
         android:id="@+id/radioButton3"
         android:layout_gravity="center_horizontal"
         android:checked="false"
         android:textSize="25dp"
         android:textColor="@android:color/holo_red_dark" />

   </RadioGroup>

</RelativeLayout>
```

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

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

Following is the default content of **AndroidManifest.xml** −

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

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

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

         <intent-filter>
```

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

      </activity>

   </application>
</manifest>
```

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

If User selected any of a Radio Button, It should give same name on Toast message. for suppose, if User selected JAVA, it gives a toast message as JAVA

RadioGroup Control

A RadioGroup class is used for set of radio buttons.

If we check one radio button that belongs to a radio group, it automatically unchecks any previously checked radio button within the same group.

RadioGroup Attributes

Following are the important attributes related to RadioGroup control. You can check Android official documentation for complete list of attributes and related methods which you can use to change these attributes are run time.

Attribute	Description

android:checkedButton	This is the id of child radio button that should be checked by default within this radio group.

Inherited from **android.view.View** Class −

Sr.No.	Attribute & Description
1	**android:background** This is a drawable to use as the background.
2	**android:contentDescription** This defines text that briefly describes content of the view.
3	**android:id** This supplies an identifier name for this view
4	**android:onClick** This is the name of the method in this View's context to invoke when the view is clicked.
5	**android:visibility** This controls the initial visibility of the view.

Example

This example will take you through simple steps to show how to create your own Android application using Linear Layout and RadioGroup.

Step	Description
1	You will use Android studio IDE to create an Android application and name it as *My Application* under a package *com.example.saira_000.myapplication;* as explained in the *Hello World Example* chapter.
2	Modify *src/MainActivity.java* file to add a click event.
2	Modify the default content of *res/layout/activity_main.xml* file to include Android UI control.
3	No need to change default constants at res/values/strings.xml, android studio takes care of default constants.
4	Run the application to launch Android emulator and verify the result of the changes done in the application.

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

In the below example abc indicates the image of tutorialspoint

```
package com.example.saira_000.myapplication;
```

```
import android.app.Activity;
import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.Toast;

public class MainActivity extends Activity {
   private RadioGroup radioSexGroup;
   private RadioButton radioSexButton;
   private Button btnDisplay;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      radioSexGroup=(RadioGroup)findViewById(R.id.radioGroup);

      btnDisplay=(Button)findViewById(R.id.button);

      btnDisplay.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View v) {
            int
selectedId=radioSexGroup.getCheckedRadioButtonId();
            radioSexButton=(RadioButton)findViewById(selectedId);

Toast.makeText(MainActivity.this,radioSexButton.getText(),Toast.L
ENGTH_SHORT).show();
         }
      });
   }
}
```

Following will be the content of **res/layout/activity_main.xml** file −

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   android:paddingBottom="@dimen/activity_vertical_margin"
   tools:context=".MainActivity">

   <TextView
```

```
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Radio button"
      android:id="@+id/textView"
      android:layout_alignParentTop="true"
      android:layout_centerHorizontal="true"
      android:textSize="35dp" />

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Tutorialspoint"
      android:id="@+id/textView2"
      android:layout_below="@+id/textView"
      android:layout_alignRight="@+id/textView"
      android:layout_alignEnd="@+id/textView"
      android:textSize="35dp"
      android:textColor="@android:color/holo_green_dark" />

   <ImageView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/imageView"
      android:src="@drawable/abc"
      android:layout_below="@+id/textView2"
      android:layout_alignLeft="@+id/textView"
      android:layout_alignStart="@+id/textView"
      android:layout_alignRight="@+id/textView"
      android:layout_alignEnd="@+id/textView" />

   <RadioGroup
      android:layout_width="fill_parent"
      android:layout_height="90dp"
      android:layout_below="@+id/imageView"
      android:layout_marginTop="58dp"
      android:weightSum="1"
      android:id="@+id/radioGroup"
      android:layout_alignLeft="@+id/textView2"
      android:layout_alignStart="@+id/textView2"
      android:layout_alignRight="@+id/textView3"
      android:layout_alignEnd="@+id/textView3">

      <RadioButton
         android:layout_width="wrap_content"
         android:layout_height="55dp"
         android:text="Male"
         android:id="@+id/radioButton"
         android:layout_gravity="center_horizontal"
         android:checked="false"
         android:textSize="25dp" />

      <RadioButton
         android:layout_width="wrap_content"
```

```
         android:layout_height="wrap_content"
         android:text="Female"
         android:id="@+id/radioButton2"
         android:layout_gravity="center_horizontal"
         android:checked="false"
         android:textSize="25dp"
         android:layout_weight="0.13" />
   </RadioGroup>

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="    Are you?"
      android:id="@+id/textView3"
      android:textSize="35dp"
      android:layout_below="@+id/imageView"
      android:layout_alignRight="@+id/textView2"
      android:layout_alignEnd="@+id/textView2"
      android:layout_alignLeft="@+id/imageView"
      android:layout_alignStart="@+id/imageView" />

   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="New Button"
      android:id="@+id/button"
      android:layout_gravity="center_horizontal"
      android:layout_below="@+id/radioGroup"
      android:layout_centerHorizontal="true" />

</RelativeLayout>
```

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

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="app_name">My Applicaiton</string>
   <string        name="example_radiogroup">Example        showing
RadioGroup</string>
</resources>
```

Following is the default content of **AndroidManifest.xml** −

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

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

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

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

      </activity>

   </application>
</manifest>
```

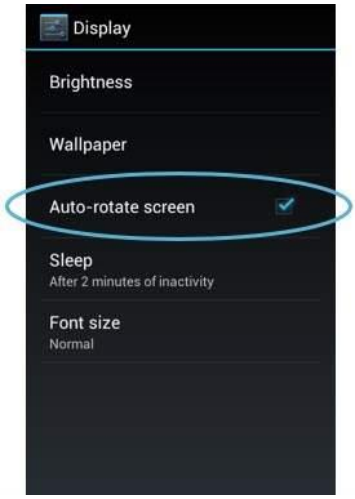
Let's try to run your **My Application** application. I assume you had created your **AVD** while doing environment setup. To run the app from Android studio, open one of your project's activity files and click Run icon from the toolbar. Android studio installs the app on your AVD and starts it and if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

The following screen will appear, here we have a RadioGroup.

Need to select male or female radio button then click on new button. if you do above steps without fail, you would get a toast message after clicked by new button

CheckBox Control

A CheckBox is an on/off switch that can be toggled by the user. You should use check-boxes when presenting users with a group of selectable options that are not mutually exclusive.



CheckBox

CheckBox Attributes

Following are the important attributes related to CheckBox control. You can check Android official documentation for complete list of attributes and related methods which you can use to change these attributes are run time.

Inherited from **android.widget.TextView** Class −

Sr.No	Attribute & Description
1	**android:autoText** If set, specifies that this TextView has a textual input method and automatically corrects some common spelling errors.
2	**android:drawableBottom** This is the drawable to be drawn below the text.
3	**android:drawableRight** This is the drawable to be drawn to the right of the text.
4	**android:editable** If set, specifies that this TextView has an input method.
5	**android:text** This is the Text to display.

Inherited from **android.view.View** Class −

Sr.No	Attribute & Description
1	**android:background** This is a drawable to use as the background.

2	**android:contentDescription** This defines text that briefly describes content of the view.
3	**android:id** This supplies an identifier name for this view.
4	**android:onClick** This is the name of the method in this View's context to invoke when the view is clicked.
5	**android:visibility** This controls the initial visibility of the view.

Example

This example will take you through simple steps to show how to create your own Android application using Linear Layout and CheckBox.

Step	Description
1	You will use Android Studio IDE to create an Android application and name it as *myapplication* under a package *com.example.myapplication* as explained in the *Hello World Example* chapter.
2	Modify *src/MainActivity.java* file to add a click event.
3	Modify the default content of *res/layout/activity_main.xml* file to include Android UI control.
4	No need to declare default string constants. Android studio takes care of default constants at string.xml
5	Run the application to launch Android emulator and verify the result of the changes done in the application.

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

```
package com.example.myapplication;

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

import android.view.View;
import android.view.View.OnClickListener;

import android.widget.CheckBox;
import android.widget.Toast;

public class MainActivity extends Activity {
   CheckBox ch1,ch2;
   Button b1,b2;
```

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

      ch1=(CheckBox)findViewById(R.id.checkBox1);
      ch2=(CheckBox)findViewById(R.id.checkBox2);

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

         @Override
         public void onClick(View v) {
            finish();
         }
      });
      b1.setOnClickListener(new View.OnClickListener() {

         @Override
         public void onClick(View v) {
            StringBuffer result = new StringBuffer();
            result.append("Thanks : ").append(ch1.isChecked());
            result.append("\nThanks: ").append(ch2.isChecked());
            Toast.makeText(MainActivity.this, result.toString(),
               Toast.LENGTH_LONG).show();
         }
      });
   }
}
```

Following will be the content of **res/layout/activity_main.xml** file −

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

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

   <TextView
      android:id="@+id/textView1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Example of checkbox"
      android:layout_alignParentTop="true"
      android:layout_centerHorizontal="true"
```

```
      android:textSize="30dp" />

   <CheckBox
      android:id="@+id/checkBox1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Do you like Tutorials Point"
      android:layout_above="@+id/button"
      android:layout_centerHorizontal="true" />

   <CheckBox
      android:id="@+id/checkBox2"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Do you like android "
      android:checked="false"
      android:layout_above="@+id/checkBox1"
      android:layout_alignLeft="@+id/checkBox1"
      android:layout_alignStart="@+id/checkBox1" />

   <TextView
      android:id="@+id/textView2"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignLeft="@+id/checkBox1"
      android:layout_below="@+id/textView1"
      android:layout_marginTop="39dp"
      android:text="Tutorials point"
      android:textColor="#ff87ff09"
      android:textSize="30dp"
      android:layout_alignRight="@+id/textView1"
      android:layout_alignEnd="@+id/textView1" />

   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Ok"
      android:id="@+id/button"
      android:layout_alignParentBottom="true"
      android:layout_alignLeft="@+id/checkBox1"
      android:layout_alignStart="@+id/checkBox1" />

   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Cancel"
      android:id="@+id/button2"
      android:layout_alignParentBottom="true"
      android:layout_alignRight="@+id/textView2"
      android:layout_alignEnd="@+id/textView2" />

   <ImageButton
      android:layout_width="wrap_content"
```

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

</RelativeLayout>
```

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

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

Following is the default content of **AndroidManifest.xml** −

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

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

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

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

      </activity>

   </application>
</manifest>
```

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

User needs you check on either do you like android check box or do you like tutorials point check box. and press ok button, if does all process correctly, it gonna be shown toast message as Thanks. Or else do press on cancel button, if user presses cancel button it going to close the application

Progress Bar using ProgressDialog

Progress bars are used to show progress of a task. For example, when you are uploading or downloading something from the internet, it is better to show the progress of download/upload to the user.

In android there is a class called ProgressDialog that allows you to create progress bar. In order to do this, you need to instantiate an object of this class. Its syntax is.

```
ProgressDialog progress = new ProgressDialog(this);
```

Now you can set some properties of this dialog. Such as, its style, its text etc.

```
progress.setMessage("Downloading Music :) ");
progress.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
progress.setIndeterminate(true);
```

Apart from these methods, there are other methods that are provided by the ProgressDialog class

Sr. No	Title & description
1	**getMax()** This method returns the maximum value of the progress.
2	**incrementProgressBy(int diff)** This method increments the progress bar by the difference of value passed as a parameter.

3	**setIndeterminate(boolean indeterminate)** This method sets the progress indicator as determinate or indeterminate.
4	**setMax(int max)** This method sets the maximum value of the progress dialog.
5	**setProgress(int value)** This method is used to update the progress dialog with some specific value.
6	**show(Context context, CharSequence title, CharSequence message)** This is a static method, used to display progress dialog.

Example

This example demonstrates the horizontal use of the progress dialog which is in fact a progress bar. It display a progress bar on pressing the button.

To experiment with this example, you need to run this on an actual device after developing the application according to the steps below.

Steps	Description
1	You will use Android studio to create an Android application under a package com.example.sairamkrishna.myapplication.
2	Modify src/MainActivity.java file to add progress code to display the progress dialog.
3	Modify res/layout/activity_main.xml file to add respective XML code.
4	Run the application and choose a running android device and install the application on it and verify the results.

Following is the content of the modified main activity file **src/MainActivity.java.**

```
package com.example.sairamkrishna.myapplication;

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

public class MainActivity extends ActionBarActivity {
   Button b1;
   private ProgressDialog progress;

   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      b1 = (Button) findViewById(R.id.button2);
   }
```

```
   public void download(View view){
      progress=new ProgressDialog(this);
      progress.setMessage("Downloading Music");
      progress.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
      progress.setIndeterminate(true);
      progress.setProgress(0);
      progress.show();

      final int totalProgressTime = 100;
      final Thread t = new Thread() {
         @Override
         public void run() {
            int jumpTime = 0;

            while(jumpTime < totalProgressTime) {
               try {
                  sleep(200);
                  jumpTime += 5;
                  progress.setProgress(jumpTime);
               } catch (InterruptedException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
               }
            }
         }
      };
      t.start();
   }
}
```

Modify the content of **res/layout/activity_main.xml** to the following −

```
<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:id="@+id/textView"
      android:layout_alignParentTop="true"
      android:layout_centerHorizontal="true"
      android:textSize="30dp"
      android:text="Progress bar" />
```

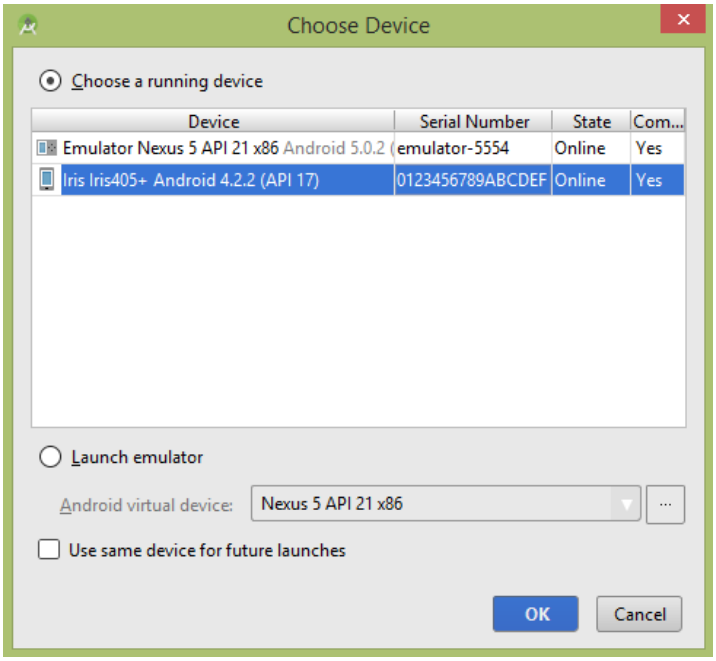
```
   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Tutorials Point"
      android:id="@+id/textView2"
      android:layout_below="@+id/textView"
      android:layout_centerHorizontal="true"
      android:textSize="35dp"
      android:textColor="#ff16ff01" />

   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Download"
      android:onClick="download"
      android:id="@+id/button2"
      android:layout_marginLeft="125dp"
      android:layout_marginStart="125dp"
      android:layout_centerVertical="true" />

</RelativeLayout>
```

This is the default **AndroidManifest.xml** −

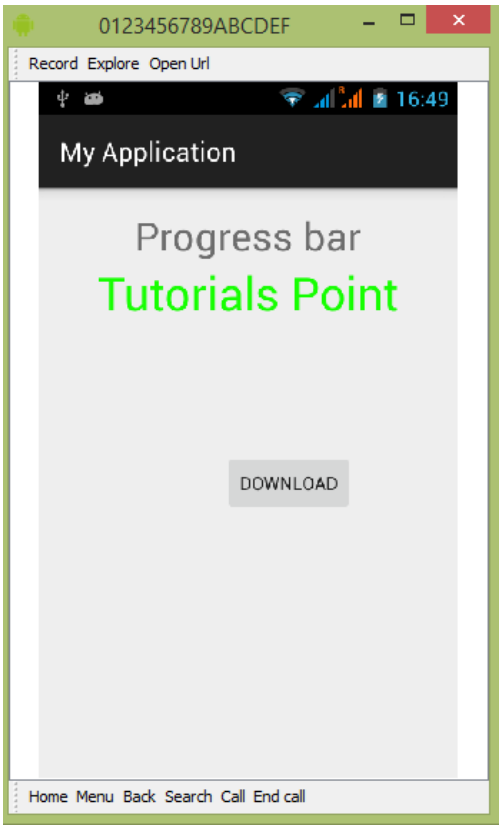
```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.example.sairamkrishna.myapplication" >
   <application
      android:allowBackup="true"
      android:icon="@mipmap/ic_launcher"
      android:label="@string/app_name"
      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>
```

Let's try to run your application. We assume, you have connected your actual Android Mobile device with your computer. To run the app from Android studio, open one of your project's activity files and click Run icon from the toolbar. Before starting your

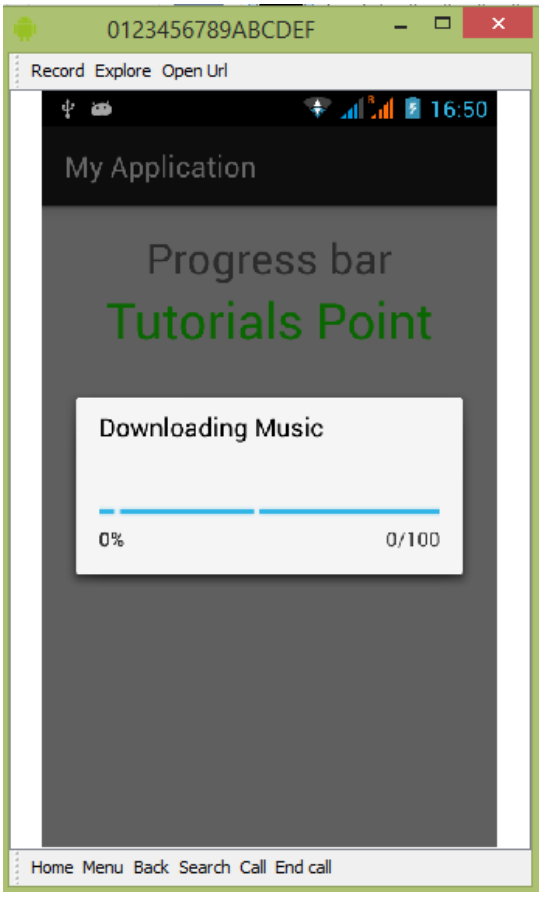
application, Android studio will display following window to select an option where you want to run your Android application.



Select your mobile device as an option and then check your mobile device which will display following screen −



Just press the button to start the Progress bar. After pressing, following screen would appear −



It will continuously update itself.

Time Picker

Android Time Picker allows you to select the time of day in either 24 hour or AM/PM mode. The time consists of hours, minutes and clock format. Android provides this functionality through TimePicker class.

In order to use TimePicker class, you have to first define the TimePicker component in your activity.xml. It is define as below −

```
<TimePicker
   android:id="@+id/timePicker1"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content" />
```

After that you have to create an object of TimePicker class and get a reference of the above defined xml component. Its syntax is given below.

```
import android.widget.TimePicker;
private TimePicker timePicker1;
timePicker1 = (TimePicker) findViewById(R.id.timePicker1);
```

In order to get the time selected by the user on the screen, you will use getCurrentHour() and getCurrentMinute() method of the TimePicker Class. Their syntax is given below.

```
int hour = timePicker1.getCurrentHour();
int min = timePicker1.getCurrentMinute();
```

Apart form these methods, there are other methods in the API that gives more control over TimePicker Component. They are listed below.

Sr.No	Method & description
1	**is24HourView()** This method returns true if this is in 24 hour view else false
2	**isEnabled()** This method returns the enabled status for this view
3	**setCurrentHour(Integer currentHour)** This method sets the current hour
4	**setCurrentMinute(Integer currentMinute)** This method sets the current minute
5	**setEnabled(boolean enabled)** This method set the enabled state of this view
6	**setIs24HourView(Boolean is24HourView)** This method set whether in 24 hour or AM/PM mode

7	**setOnTimeChangedListener(TimePicker.OnTimeChangedListener onTimeChangedListener)** This method Set the callback that indicates the time has been adjusted by the user

Example

Here is an example demonstrating the use of TimePicker class. It creates a basic Time Picker application that allows you to set the time using TimePicker Widget

To experiment with this example , you can run this on an actual device or in an emulator.

Steps	Description
1	You will use Android studio to create an Android application and name it as TimePicker under a package com.example.timepicker.
2	Modify src/MainActivity.java file to add necessary code.
3	Modify the res/layout/activity_main to add respective XML components
4	Modify the res/values/string.xml to add necessary string components
5	Run the application and choose a running android device and install the application on it and verify the results

Following is the content of the modified main activity file **src/com.example.timepicker/MainActivity.java**.

```
package com.example.timepicker;

import java.util.Calendar;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.View;
import android.widget.TextView;
import android.widget.TimePicker;

public class MainActivity extends Activity {
   private TimePicker timePicker1;
   private TextView time;
   private Calendar calendar;
   private String format = "";

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

      timePicker1 = (TimePicker) findViewById(R.id.timePicker1);
      time = (TextView) findViewById(R.id.textView1);
```

```
        calendar = Calendar.getInstance();

        int hour = calendar.get(Calendar.HOUR_OF_DAY);
        int min = calendar.get(Calendar.MINUTE);
        showTime(hour, min);
    }

    public void setTime(View view) {
        int hour = timePicker1.getCurrentHour();
        int min = timePicker1.getCurrentMinute();
        showTime(hour, min);
    }

    public void showTime(int hour, int min) {
        if (hour == 0) {
            hour += 12;
            format = "AM";
        } else if (hour == 12) {
            format = "PM";
        } else if (hour > 12) {
            hour -= 12;
            format = "PM";
        } else {
            format = "AM";
        }

        time.setText(new   StringBuilder().append(hour).append("   :
").append(min)
        .append(" ").append(format));
    }


}
```

Following is the modified content of the xml **res/layout/activity_main.xml**.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@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/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
```

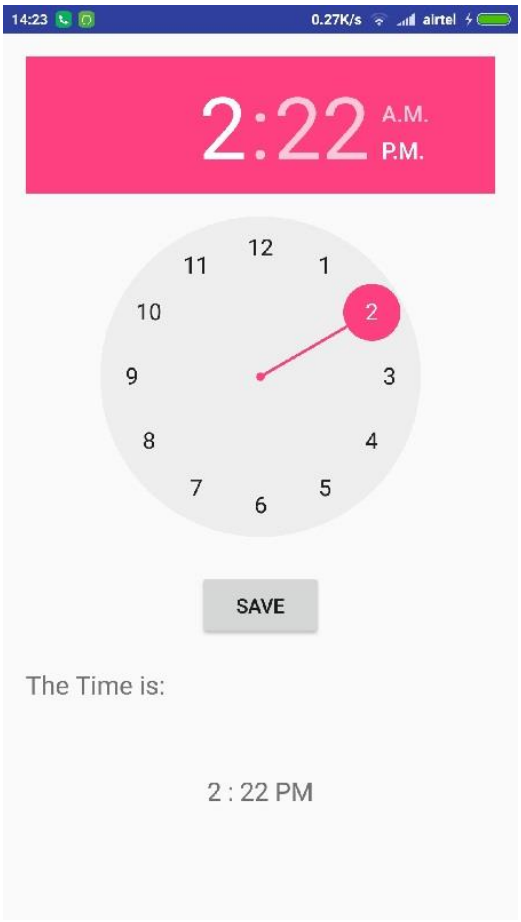
```
      android:layout_centerHorizontal="true"
      android:text="@string/time_pick"
      android:textAppearance="?android:attr/textAppearanceMedium"
/>

   <Button
      android:id="@+id/set_button"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentBottom="true"
      android:layout_centerHorizontal="true"
      android:layout_marginBottom="180dp"
      android:onClick="setTime"
      android:text="@string/time_save" />

   <TimePicker
      android:id="@+id/timePicker1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_above="@+id/set_button"
      android:layout_centerHorizontal="true"
      android:layout_marginBottom="24dp" />

   <TextView
      android:id="@+id/textView3"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignLeft="@+id/timePicker1"
      android:layout_alignTop="@+id/set_button"
      android:layout_marginTop="67dp"
      android:text="@string/time_current"
      android:textAppearance="?android:attr/textAppearanceMedium"
/>

   <TextView
      android:id="@+id/textView1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_below="@+id/textView3"
      android:layout_centerHorizontal="true"
      android:layout_marginTop="50dp"
      android:text="@string/time_selected"
      android:textAppearance="?android:attr/textAppearanceMedium"
/>

</RelativeLayout>
```

Following is the content of the **res/values/string.xml**.

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

```
   <string name="time_picker_example">Time Picker Example</string>
   <string name="time_pick">Pick the time and press save
button</string>
   <string name="time_save">Save</string>
   <string name="time_selected"></string>
   <string name="time_current">The Time is:</string>
</resources>
```

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



Date Picker

Android Date Picker allows you to select the date consisting of day, month and year in your custom user interface. For this functionality android provides DatePicker and DatePickerDialog components.

In this tutorial, we are going to demonstrate the use of Date Picker through DatePickerDialog. DatePickerDialog is a simple dialog containing DatePicker.

In order to show DatePickerDialog , you have to pass the DatePickerDialog id to *showDialog(id_of_dialog)* method. Its syntax is given below −

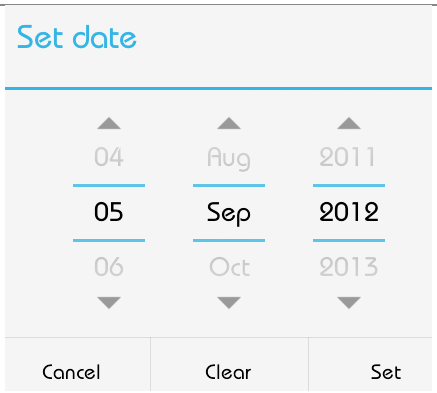
```
showDialog(999);
```

On calling this *showDialog* method, another method called *onCreateDialog* gets automatically called. So we have to override that method too. Its syntax is given below –

```
@Override
protected Dialog onCreateDialog(int id) {
   // TODO Auto-generated method stub
   if (id == 999) {
      return new DatePickerDialog(this, myDateListener, year,
month, day);
   }
   return null;
}
```

In the last step, you have to register the DatePickerDialog listener and override its onDateSet method. This onDateSet method contains the updated day, month and year. Its syntax is given below −

```
private DatePickerDialog.OnDateSetListener myDateListener = new
DatePickerDialog.OnDateSetListener() {
   @Override
   public void onDateSet(DatePicker arg0, int arg1, int arg2, int
arg3) {
      // arg1 = year
      // arg2 = month
      // arg3 = day
   }
};
```



Apart form date attributes, DatePicker object is also passed into this function. You can use the following methods of the DatePicker to perform further operation.

Sr.No	Method & description
1	**getDayOfMonth()** This method gets the selected day of month
2	**getMonth()** This method gets the selected month

3	**getYear()** This method gets the selected year
4	**setMaxDate(long maxDate)** This method sets the maximal date supported by this DatePicker in milliseconds since January 1, 1970 00:00:00 in getDefault() time zone
5	**setMinDate(long minDate)** This method sets the minimal date supported by this NumberPicker in milliseconds since January 1, 1970 00:00:00 in getDefault() time zone
6	**setSpinnersShown(boolean shown)** This method sets whether the spinners are shown
7	**updateDate(int year, int month, int dayOfMonth)** This method updates the current date
8	**getCalendarView()** This method returns calendar view
9	**getFirstDayOfWeek()** This Method returns first day of the week

Example

Here is an example demonstrating the use of DatePickerDialog class. It creates a basic Date Picker application that allows you to set the Date using DatePicker Widget

To experiment with this example , you can run this on an actual device or in an emulator.

Steps	Description
1	You will use Android studio to create an Android application and name it as DatePicker under a package com.example.datepicker.
2	Modify src/MainActivity.java file to add necessary code.
3	Modify the res/layout/activity_main to add respective XML components.
4	Modify the res/values/string.xml to add necessary string components.
5	Run the application and choose a running android device and install the application on it and verify the results.

Following is the content of the modified main activity file **src/com.example.datepicker/MainActivity.java**.

```
package com.example.datepicker;

import java.util.Calendar;

import android.app.Activity;
import android.app.DatePickerDialog;
import android.app.Dialog;

import android.os.Bundle;

import android.view.Menu;
import android.view.View;

import android.widget.DatePicker;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends Activity {
   private DatePicker datePicker;
   private Calendar calendar;
   private TextView dateView;
   private int year, month, day;

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

      dateView = (TextView) findViewById(R.id.textView3);
      calendar = Calendar.getInstance();
      year = calendar.get(Calendar.YEAR);

      month = calendar.get(Calendar.MONTH);
      day = calendar.get(Calendar.DAY_OF_MONTH);
      showDate(year, month+1, day);
   }

   @SuppressWarnings("deprecation")
   public void setDate(View view) {
      showDialog(999);
      Toast.makeText(getApplicationContext(), "ca",
         Toast.LENGTH_SHORT)
      .show();
   }

   @Override
   protected Dialog onCreateDialog(int id) {
      // TODO Auto-generated method stub
      if (id == 999) {
         return new DatePickerDialog(this,
```

```
            myDateListener, year, month, day);
      }
      return null;
   }

   private DatePickerDialog.OnDateSetListener myDateListener = new
      DatePickerDialog.OnDateSetListener() {
      @Override
      public void onDateSet(DatePicker arg0,
         int arg1, int arg2, int arg3) {
         // TODO Auto-generated method stub
         // arg1 = year
         // arg2 = month
         // arg3 = day
         showDate(arg1, arg2+1, arg3);
      }
   };

   private void showDate(int year, int month, int day) {
      dateView.setText(new
StringBuilder().append(day).append("/")
      .append(month).append("/").append(year));
   }
}
```

Following is the modified content of the xml **res/layout/activity_main.xml**.

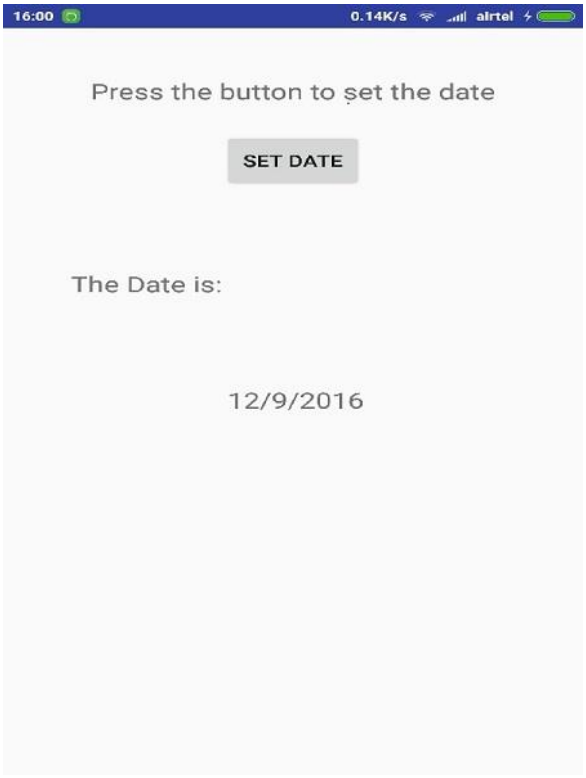
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingBottom="@dimen/activity_vertical_margin"
   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/button1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentTop="true"
      android:layout_centerHorizontal="true"
      android:layout_marginTop="70dp"
      android:onClick="setDate"
      android:text="@string/date_button_set" />

   <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="24dp"
      android:text="@string/date_label_set"
      android:textAppearance="?android:attr/textAppearanceMedium"
/>

   <TextView
      android:id="@+id/textView2"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_below="@+id/button1"
      android:layout_marginTop="66dp"
      android:layout_toLeftOf="@+id/button1"
      android:text="@string/date_view_set"
      android:textAppearance="?android:attr/textAppearanceMedium"
/>

   <TextView
      android:id="@+id/textView3"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignRight="@+id/button1"
      android:layout_below="@+id/textView2"
      android:layout_marginTop="72dp"
      android:text="@string/date_selected"
      android:textAppearance="?android:attr/textAppearanceMedium"
/>

</RelativeLayout>
```

Following is the content of the **res/values/string.xml**.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="app_name">DatePicker</string>
   <string name="action_settings">Settings</string>
   <string name="hello_world">Hello world!</string>
   <string  name="date_label_set">Press  the  button  to  set  the
date</string>
   <string name="date_button_set">Set Date</string>
   <string name="date_view_set">The Date is: </string>
   <string name="date_selected"></string>
</resources>
```

Let's try to run our DatePicker application we just modified. I assume you had created your **AVD** while doing environment set-up. To run the app from Eclipse, open one of your project's activity files and click Run icon from the tool bar. Eclipse installs the app on your AVD and starts it and if everything is fine with your set-up and application, it will display following Emulator window −

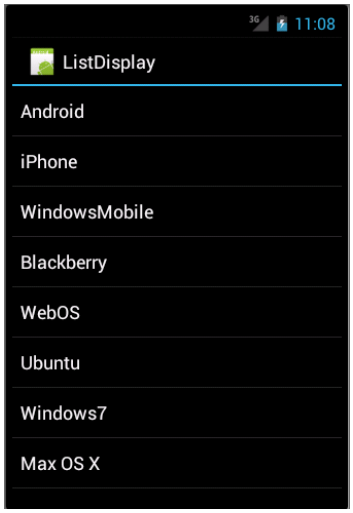


Now you can see that the date has already been set at the bottom label. Now we will change the date through DatePickerDialog by pressing the Set Date button. On pressing the button following screen would appear.

Now set the required date, and after setting the date, press the Done button. This dialog will disappear and your newly setted date will start showing at the screen. This is shown below.

List View

Android **ListView** is a view which groups several items and display them in vertical scrollable list. The list items are automatically inserted to the list using an **Adapter** that pulls content from a source such as an array or database.



List View

An adapter actually bridges between UI components and the data source that fill data into UI Component. Adapter holds the data and send the data to adapter view, the view can takes the data from adapter view and shows the data on different views like as spinner, list view, grid view etc.

The **ListView** and **GridView** are subclasses of **AdapterView** and they can be populated by binding them to an **Adapter**, which retrieves data from an external source and creates a View that represents each data entry.

Android provides several subclasses of Adapter that are useful for retrieving different kinds of data and building views for an AdapterView (i.e. ListView or GridView). The

common adapters are **ArrayAdapter**,**BaseAdapter**, **CursorAdapter**, **SimpleCursorAdapter**,**SpinnerAdapter** and **WrapperListAdapter**. We will see separate examples for both the adapters.

ListView Attributes

Following are the important attributes specific to GridView −

Sr.No	Attribute & Description
1	**android:id** This is the ID which uniquely identifies the layout.
2	**android:divider** This is drawable or color to draw between list items.
3	**android:dividerHeight** This specifies height of the divider. This could be in px, dp, sp, in, or mm.
4	**android:entries** Specifies the reference to an array resource that will populate the ListView.
5	**android:footerDividersEnabled** When set to false, the ListView will not draw the divider before each footer view. The default value is true.
6	**android:headerDividersEnabled** When set to false, the ListView will not draw the divider after each header view. The default value is true.

ArrayAdapter

You can use this adapter when your data source is an array. By default, ArrayAdapter creates a view for each array item by calling toString() on each item and placing the contents in a **TextView**. Consider you have an array of strings you want to display in a ListView, initialize a new **ArrayAdapter** using a constructor to specify the layout for each string and the string array −

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

Here are arguments for this constructor −

- First argument **this** is the application context. Most of the case, keep it **this**.
- Second argument will be layout defined in XML file and having **TextView** for each string in the array.
- Final argument is an array of strings which will be populated in the text view.

Once you have array adapter created, then simply call **setAdapter()** on your **ListView** object as follows −

```
ListView listView = (ListView) findViewById(R.id.listview);
listView.setAdapter(adapter);
```

You will define your list view under res/layout directory in an XML file. For our example we are going to using activity_main.xml file.

Example

Following is the example which will take you through simple steps to show how to create your own Android application using ListView. Follow the following steps to modify the Android application we created in *Hello World Example* chapter −

Step	Description
1	You will use Android Studio IDE to create an Android application and name it as *ListDisplay* under a package *com.example.ListDisplay* as explained in the *Hello World Example* chapter.
2	Modify the default content of *res/layout/activity_main.xml* file to include ListView content with the self explanatory attributes.
3	No need to change string.xml, Android studio takes care of default string constants.
4	Create a Text View file *res/layout/activity_listview.xml*. This file will have setting to display all the list items. So you can customize its fonts, padding, color etc. using this file.
6	Run the application to launch Android emulator and verify the result of the changes done in the application.

Following is the content of the modified main activity file **src/com.example.ListDisplay/ListDisplay.java**. This file can include each of the fundamental life cycle methods.

```
package com.example.ListDisplay;

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

public class ListDisplay extends Activity {
   // Array of strings...
   String[]                        mobileArray                                =
{"Android","IPhone","WindowsMobile","Blackberry",
      "WebOS","Ubuntu","Windows7","Max OS X"};

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

      ArrayAdapter adapter = new ArrayAdapter<String>(this,
```

```
         R.layout.activity_listview, mobileArray);

      ListView            listView            =            (ListView)
findViewById(R.id.mobile_list);
      listView.setAdapter(adapter);
   }
}
```

Following will be the content of **res/layout/activity_main.xml** file −

```
<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"
   tools:context=".ListActivity" >

   <ListView
      android:id="@+id/mobile_list"
      android:layout_width="match_parent"
      android:layout_height="wrap_content" >
   </ListView>

</LinearLayout>
```

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

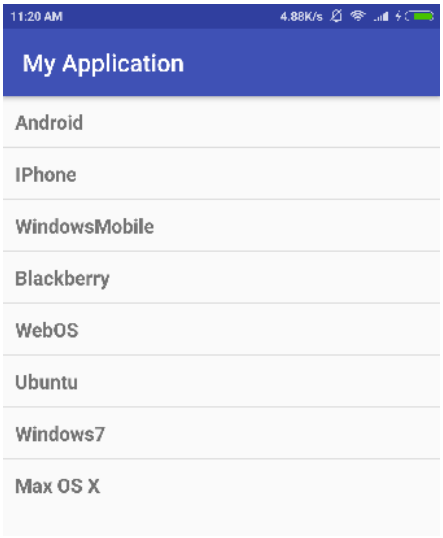
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="app_name">ListDisplay</string>
   <string name="action_settings">Settings</string>
</resources>
```

Following will be the content of **res/layout/activity_listview.xml** file −

```
<?xml version="1.0" encoding="utf-8"?>
<!--  Single List Item Design -->

<TextView
xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/label"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:padding="10dip"
   android:textSize="16dip"
   android:textStyle="bold" >
</TextView>
```

Let's try to run our modified **Hello World!** application we just modified. I assume you had created your **AVD** while doing environment set-up. To run the app from Android studio, open one of your project's activity files and click Run icon from the tool bar.

Android studio installs the app on your AVD and starts it and if everything is fine with your set-up and application, it will display following Emulator window −



SimpleCursorAdapter

You can use this adapter when your data source is a database Cursor. When using *SimpleCursorAdapter*, you must specify a layout to use for each row in the **Cursor** and which columns in the Cursor should be inserted into which views of the layout.

For example, if you want to create a list of people's names and phone numbers, you can perform a query that returns a Cursor containing a row for each person and columns for the names and numbers. You then create a string array specifying which columns from the Cursor you want in the layout for each result and an integer array specifying the corresponding views that each column should be placed −

```
String[] fromColumns = {ContactsContract.Data.DISPLAY_NAME,
   ContactsContract.CommonDataKinds.Phone.NUMBER};
int[] toViews = {R.id.display_name, R.id.phone_number};
```

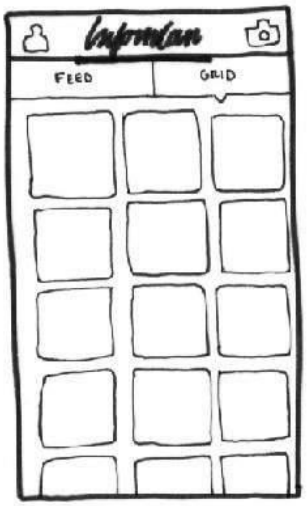
When you instantiate the SimpleCursorAdapter, pass the layout to use for each result, the Cursor containing the results, and these two arrays −

```
SimpleCursorAdapter adapter = new SimpleCursorAdapter(this,
   R.layout.person_name_and_number, cursor, fromColumns, toViews,
0);

ListView listView = getListView();
listView.setAdapter(adapter);
```

The SimpleCursorAdapter then creates a view for each row in the Cursor using the provided layout by inserting each from Columns item into the corresponding **toViews** view.

Grid View

Android **GridView** shows items in two-dimensional scrolling grid (rows & columns) and the grid items are not necessarily predetermined but they automatically inserted to the layout using a **ListAdapter**



Grid view

An adapter actually bridges between UI components and the data source that fill data into UI Component. Adapter can be used to supply the data to like spinner, list view, grid view etc.

The **ListView** and **GridView** are subclasses of **AdapterView** and they can be populated by binding them to an **Adapter**, which retrieves data from an external source and creates a View that represents each data entry.

GridView Attributes

Following are the important attributes specific to GridView −

Sr.No	Attribute & Description
1	**android:id** This is the ID which uniquely identifies the layout.
2	**android:columnWidth** This specifies the fixed width for each column. This could be in px, dp, sp, in, or mm.
3	**android:gravity** Specifies the gravity within each cell. Possible values are top, bottom, left, right, center, center_vertical, center_horizontal etc.
4	**android:horizontalSpacing** Defines the default horizontal spacing between columns. This could be in px, dp, sp, in, or mm.
5	**android:numColumns** Defines how many columns to show. May be an integer value, such as "100" or auto_fit which means display as many columns as possible to fill the available space.

6	**android:stretchMode** Defines how columns should stretch to fill the available empty space, if any. This must be either of the values − • none − Stretching is disabled. • spacingWidth − The spacing between each column is stretched. • columnWidth − Each column is stretched equally. • spacingWidthUniform − The spacing between each column is uniformly stretched..
7	**android:verticalSpacing** Defines the default vertical spacing between rows. This could be in px, dp, sp, in, or mm.

Example

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

Step	Description
1	You will use Android studio IDE to create an Android application and name it as *HelloWorld* under a package *com.example.helloworld* as explained in the *Hello World Example* chapter.
2	Modify the detault content of *res/layout/activity_main.xml* file to include GridView content with the self explanatory attributes.
3	No need to change string.xml, Android studio takes care of defaults strings which are placed at string.xml
4	Let's put few pictures in *res/drawable-hdpi* folder. I have put sample0.jpg, sample1.jpg, sample2.jpg, sample3.jpg, sample4.jpg, sample5.jpg, sample6.jpg and sample7.jpg.
5	Create a new class called **ImageAdapter** under a package com.example.helloworld that extends BaseAdapter. This class will implement functionality of an adapter to be used to fill the view.
6	Run the application to launch Android emulator and verify the result of the changes done in the application.

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

```
package com.example.helloworld;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.widget.GridView;

public class MainActivity extends Activity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        GridView gridview = (GridView) findViewById(R.id.gridview);
        gridview.setAdapter(new ImageAdapter(this));
    }
}
```

Following will be the content of **res/layout/activity_main.xml** file −

```
<?xml version="1.0" encoding="utf-8"?>
<GridView
xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/gridview"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:columnWidth="90dp"
   android:numColumns="auto_fit"
   android:verticalSpacing="10dp"
   android:horizontalSpacing="10dp"
   android:stretchMode="columnWidth"
   android:gravity="center"
/>
```

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

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

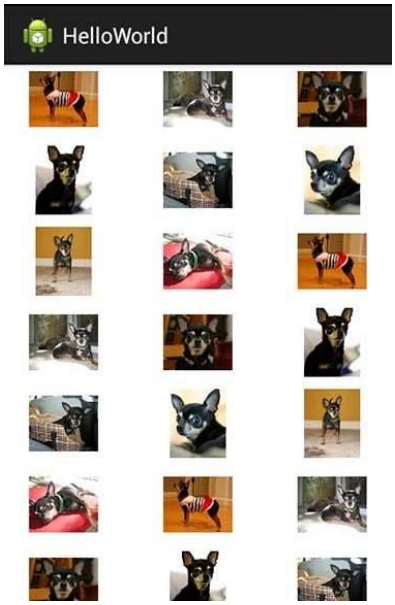
Following will be the content of **src/com.example.helloworld/ImageAdapter.java** file −

```
package com.example.helloworld;

import android.content.Context;

import android.view.View;
import android.view.ViewGroup;

import android.widget.BaseAdapter;
import android.widget.GridView;
import android.widget.ImageView;

public class ImageAdapter extends BaseAdapter {
   private Context mContext;

   // Constructor
   public ImageAdapter(Context c) {
      mContext = c;
   }
```

```
   public int getCount() {
      return mThumbIds.length;
   }

   public Object getItem(int position) {
      return null;
   }

   public long getItemId(int position) {
      return 0;
   }

   // create a new ImageView for each item referenced by the Adapter
   public View getView(int position, View convertView, ViewGroup
parent) {
      ImageView imageView;

      if (convertView == null) {
         imageView = new ImageView(mContext);
         imageView.setLayoutParams(new  GridView.LayoutParams(85,
85));
         imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
         imageView.setPadding(8, 8, 8, 8);
      }
      else
      {
         imageView = (ImageView) convertView;
      }
      imageView.setImageResource(mThumbIds[position]);
      return imageView;
   }

   // Keep all Images in array
   public Integer[] mThumbIds = {
      R.drawable.sample_2, R.drawable.sample_3,
      R.drawable.sample_4, R.drawable.sample_5,
      R.drawable.sample_6, R.drawable.sample_7,
      R.drawable.sample_0, R.drawable.sample_1,
      R.drawable.sample_2, R.drawable.sample_3,
      R.drawable.sample_4, R.drawable.sample_5,
      R.drawable.sample_6, R.drawable.sample_7,
      R.drawable.sample_0, R.drawable.sample_1,
      R.drawable.sample_2, R.drawable.sample_3,
      R.drawable.sample_4, R.drawable.sample_5,
      R.drawable.sample_6, R.drawable.sample_7
   };
}
```

Let's try to run our modified **Hello World!** application we just modified. I assume you had created your **AVD** while doing environment setup. To run the app from Android Studio, open one of your project's activity files and click Run icon from the toolbar.

Android studio installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display following Emulator window −



Sub-Activity Example

Let's extend the functionality of above example where we will show selected grid image in full screen. To achieve this we need to introduce a new activity. Just keep in mind for any activity we need perform all the steps like we have to implement an activity class, define that activity in AndroidManifest.xml file, define related layout and finally link that sub-activity with the main activity by it in the main activity class. So let's follow the steps to modify above example −

Step	Description
1	You will use Android studio IDE to create an Android application and name it as *HelloWorld* under a package *com.example.helloworld* as explained in the *Hello World Example* chapter.
2	Create a new Activity class as *SingleViewActivity.java* under a package *com.example.helloworld* as shown below.
3	Create new layout file for the new activity under **res/layout/** folder. Let's name this XML file as single_view.xml.
4	Define your new activity in *AndroidManifest.xml* file using <activity.../> tag. An application can have one or more activities without any restrictions.
5	Run the application to launch Android emulator and verify the result of the changes done in the application.

Following is the content of the modified main activity file **src/com.example.helloworld/MainActivity.java**. This file can include each of the fundamental life cycle methods.

```
package com.example.helloworld;

import android.app.Activity;
```

```
import android.content.Intent;
import android.os.Bundle;

import android.view.Menu;
import android.view.View;

import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.GridView;

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

      GridView gridview = (GridView) findViewById(R.id.gridview);
      gridview.setAdapter(new ImageAdapter(this));

      gridview.setOnItemClickListener(new OnItemClickListener() {
         public void onItemClick(AdapterView<?> parent,
            View v, int position, long id){
            // Send intent to SingleViewActivity
            Intent   i   =   new   Intent(getApplicationContext(),
SingleViewActivity.class);
            // Pass image index
            i.putExtra("id", position);
            startActivity(i);
         }
      });
   }
}
```

Following will be the content of new activity file **src/com.example.helloworld/SingleViewActivity.java** file −

```
package com.example.helloworld;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.widget.ImageView;

public class SingleViewActivity extends Activity {
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.single_view);

      // Get intent data
      Intent i = getIntent();

      // Selected image id
```

```
      int position = i.getExtras().getInt("id");
      ImageAdapter imageAdapter = new ImageAdapter(this);

      ImageView           imageView           =           (ImageView)
findViewById(R.id.SingleView);

imageView.setImageResource(imageAdapter.mThumbIds[position]);
   }
}
```

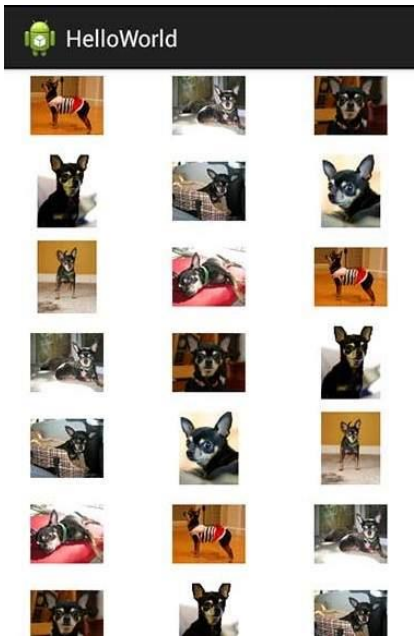
Following will be the content of **res/layout/single_view.xml** file −

```
<?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" >

<ImageView android:id="@+id/SingleView"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"/>

</LinearLayout>
```

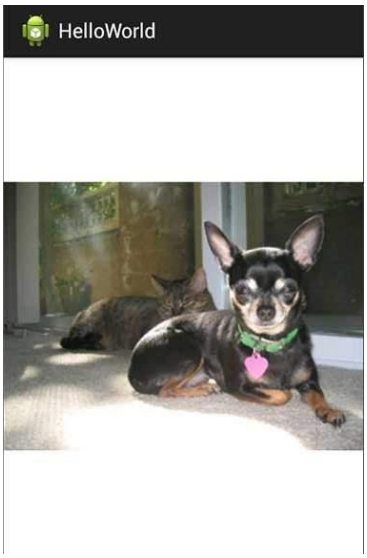
Following will be the content of **AndroidManifest.xml** to define two new constants −

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

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

      <activity
         android:name="com.example.helloworld.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>

      <activity android:name=".SingleViewActivity"></activity>

   </application>
</manifest>
```

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



Now if you click on either of the images it will be displayed as a single image, for example−



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. You can read our ScaleType tutorial to learn all details on it.

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() funtion 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.

```
<ImageView
    android:id="@+id/simpleImageView"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:src="@drawable/lion"
    android:background="#000"/><!--black color in background of a image view-->
```

In Java:

We can also set the background at run time programmatically in java class. In below example code we set the black color in the background of a image view.

```
/*Add in Oncreate() funtion 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-->
```

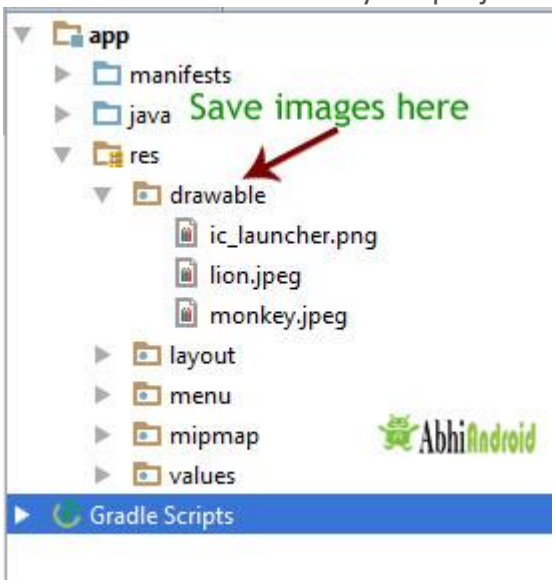
Example of ImageView:

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) findViewById(R.id.simpleImageV
iewLion);//get the id of first image view
        ImageView simpleImageViewMonkey = (ImageView) findViewById(R.id.simpleImag
eViewMonkey);//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_LON
G).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.

ScrollView (Horizontal, Vertical)

In android, **ScrollView** is a kind of layout that is useful to add vertical or horizontal scroll bars to the content which is larger than the actual size of layouts such as linearlayout, relativelayout, framelayout, etc.

Generally, the android **ScrollView** is useful when we have content that doesn't fit our android app layout screen. The ScrollView will enable a scroll to the content which is exceeding the screen layout and allow users to see the complete content by scrolling.

The android **ScrollView** can hold only one direct child. In case, if we want to add multiple views within the scroll view, then we need to include them in another standard layout like linearlayout, relativelayout, framelayout, etc.

To enable scrolling for our android applications, **ScrollView** is the best option but we should not use ScrollView along with ListView or Gridview because they both will take care of their own vertical scrolling.

In android, **ScrollView** supports only **vertical** scrolling. In case, if we want to implement **horizontal** scrolling, then we need to use a **HorizontalScrollView** component.

The android ScrollView is having a property called android:fillViewport, which is used to define whether the ScrollView should stretch it's content to fill the viewport or not.

Now we will see how to use **ScrollView** with linearlayout to enable scroll view to the content which is larger than screen layout in android application with examples.

Android ScrollView Example

Following is the example of enabling vertical scrolling to the content which is larger than the layout screen using an android **ScrollView** object.

Create a new android application using android studio and give names as **ScrollViewExample**. In case if you are not aware of creating an app in android studio check this article Android Hello World App.

Once we create an application, open **activity_main.xml** file from **\res\layout** folder path and write the code like as shown below.

activity_main.xml

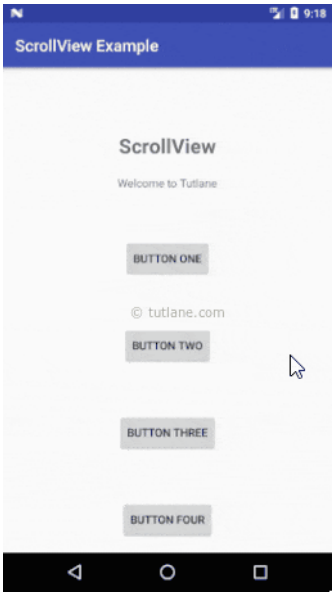
```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:fillViewport="false">
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android
"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView android:id="@+id/loginscrn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="80dp"
        android:text="ScrollView"
        android:textSize="25dp"
        android:textStyle="bold"
        android:layout_gravity="center"/>
    <TextView android:id="@+id/fstTxt"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:text="Welcome to Tutlane"
        android:layout_gravity="center"/>
    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginTop="60dp"
        android:text="Button One" />
    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginTop="60dp"
        android:text="Button Two" />
```

```
    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginTop="60dp"
        android:text="Button Three" />
    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginTop="60dp"
        android:text="Button Four" />
    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginTop="60dp"
        android:text="Button Five" />
    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginTop="60dp"
        android:text="Button Six" />
    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginTop="60dp"
        android:text="Button Seven" />
    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginTop="60dp"
        android:text="Button Eight" />
    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginTop="60dp"
        android:text="Button Nine" />
</LinearLayout>
</ScrollView>
```

If you observe above code, we used a **ScrollView** to enable the scrolling for linearlayout whenever the content exceeds layout screen.

Output of Android ScrollView Example

When we run the above example in android emulator we will get a result as shown below.



If you observe the above result, **ScrollView** provided a vertical scrolling for linearlayout whenever the content exceeds the layout screen.

As we discussed, **ScrollView** can provide only vertical scrolling for the layout. In case, if we want to enable horizontal scrolling, then we need to use **HorizontalScrollView** in our application.

We will see how to enable horizontal scrolling for the content which is exceeding the layout screen in the android application.

Android HorizontalScrollView Example

Now open **activity_main.xml** file in your android application and write the code like as shown below.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<HorizontalScrollView xmlns:android="http://schemas.android.com/apk/res
/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:fillViewport="true">
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android
"
    android:orientation="horizontal" android:layout_width="match_parent
"
    android:layout_height="match_parent"
    android:layout_marginTop="150dp">
    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button One" />
    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button Two" />
    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:text="Button Three" />
    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button Four" />
    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button Five" />
    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button Six" />
    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button Seven" />
    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button Eight" />
    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button Nine" />
</LinearLayout>
</HorizontalScrollView>
```

If you observe above code, we used a **HorizontalScrollView** to enable horizontal scrolling for linearlayout whenever the content exceeds layout screen.

Output of Android HorizontalScrollView Example

When we run the above example in the android emulator we will get a result like as shown below.

If you observe above result, **HorizontalScrollView** provided a horizontal scrolling for linearlayout whenever the content exceeds the layout screen.

This is how we can enable scrolling for the content which exceeds layout screen using **ScrollView** and **HorizontalScrollView** object based on our requirements.