Chapter 2

2. Android Layout and UI Widgets

2.1. Views and View Groups

What are Views?

Anything which occupies a rectangular area on the screen and is responsible for drawing and event handling is known as a view. Thus every UI element is subclass of view, to be more precise of **android.view.View.**

The android SDK provides a set of pre-built views that can be used to construct a user interface, for example buttons, checkbox, EditText etc. which are also known as widgets.

Widgets: - can be thought of as a pre-built control through which the user interacts with your application. There is nothing such as widget class in android. It simply means that the class is in **android.widgets** package in android.

Android contains the following commonly used View subclasses: TextView, EditText, ImageView, ProgressBar, Button, ImageButton, ChechBox etc. We can also create our widget by extending view or any of its subclass inside android.widgets.

What are ViewGroup?

The ViewGroup is a subclass of View (*android.view.View)* and provides invisible container that hold other views or other ViewGroups and define their layout properties. Thus all the Layouts and Containers inherit from ViewGroup. The view group is the base class for layouts and views containers.

Android contains the following commonly used ViewGroup subclasses: LinearLayout, RelativeLayout, ListView, GridView etc.

2.2. Layout and its types

Layouts are subclasses of ViewGroup which specifies how a view should be arranged inside the viewgroups. It defines the visual structure for a user interface.

You can create layouts in two ways:-

- Declare UI elements in XML
- Instantiate layout elements at runtime

The standard Layouts are:

- ✓ **LinearLayout**: A layout that arranges its children in a single column or a single row. The direction of the row can be set by calling setOrientation( ).
- ✓ **RelativeLayout**: A layout where the positions of the children can be described in relation to each other or to the parent.
- ✓ **FrameLayout**: It is used to block out an area on the screen to display a single item.
- ✓ **TableLayout**: A layout that arranges its children into rows and columns.

2.2.1. Linear Layout

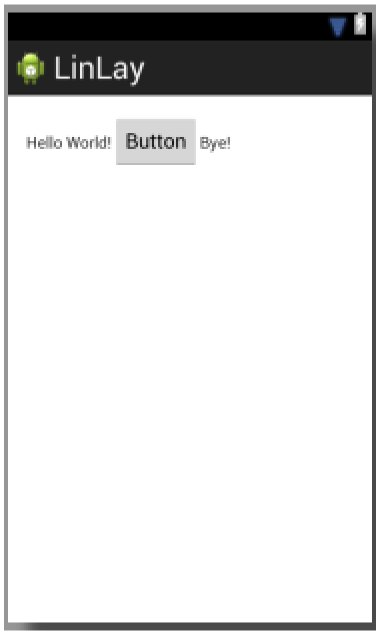
Linear layout is a view group that aligns all children in a single direction, vertically or horizontally. Layout direction is specified by ***android:orientation="horizontal" or android:orientation="vertically"*** attribute.

LinearLayout horizontal arranges the elements horizontally in the LinearLayout. **Example**: In the example below we have arranged a TextView and a Button horizontally.

```
<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: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"
    android:orientation="horizontal">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bye" />

</LinearLayout>
```

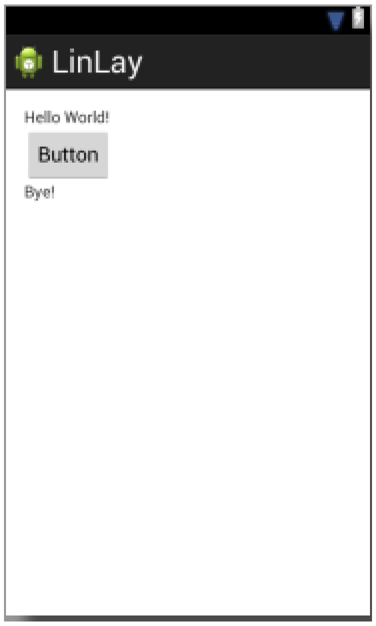


LinearLayout vertical arranges the elements vertically in the LinearLayout. **Example**: In the example below we have arranged a TextView and a Button vertically.

```
<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: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"
    android:orientation="vertical">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bye" />

</LinearLayout>
```



2.2.2. Relative Layout

RelativeLayout lays out elements based on their relationships with one another, and with the parent container. This is arguably the most complicated layout and we need several properties to actually get the layout we want.

Relative to container:

These properties will layout elements relative to the parent container.

- ***android:layout_alignParentBottom***-places the element on the bottom of the container.
- ***android:layout_alignParentLeft***-places the element on the left side of the container.

- ***android:layout_alignParentRight***-places the element on the right side of the container.
- ***android:layout_alignParentTop***-places the element on the top of the container.
- ***android:layout_centerHorizontal***-centers the elements horizontally within its parent container.
- ***android:layout_centerInParent***- centers the elements both horizontally and vertically within its parent container.
- ***android:layout_centerVertical***- centers the elements vertically within its parent container..

Relative to other Elements:

These properties allow you to layout elements relative to other elements on screen. The value for each of these elements is the id of the element you are using to layout the new element.

Each element that is used in this way must have an ID defined using ***android:id="@_id/xxxxx"*** where xxxxx is replaced with the desired id. You use "***@id/xxxxx***" to reference an element by its id. One thing to remember is that referencing an element before it has been declared will produce an error.

- ***android:layout_above***- places an element above the specified element.
- ***android:layout_below***- places an element below the specified element.
- ***android:layout_toLeftOf***- places an element to the left of the specified element.
- ***android:layout_toRightOf***- places an element to the right of the specified element.

Relative with other Elements:

These properties allow you to specify how elements are aligned in relation to other elements.

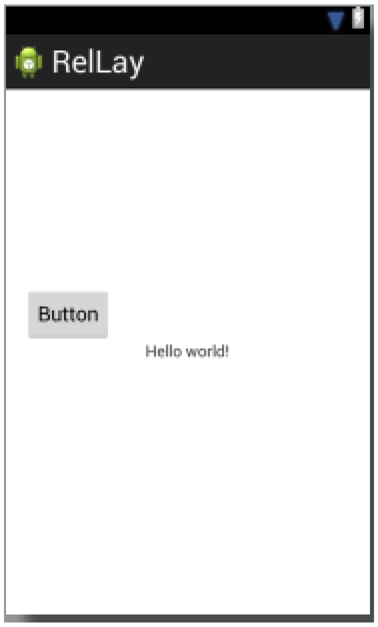
- ***android:layout_alignBaseline***- aligns baseline of the new element with baseline of the specified element.
- ***android:layout_alignBottom***- aligns bottom of the new element in with bottom of the specified element.
- ***android:layout_alignLeft***- aligns left edge of the new element with left edge of the specified element.
- ***android:layout_alignRight***- aligns right edge of the new element with right edge of the specified element.
- ***android:layout_alignTop***- places top of the new element in alignment with the top of the specified element.

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

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

```
          android:id="@+id/textview"
          android:layout_centerVertical="true"
          android:layout_centerInParent="true"/>
        <Button
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="@string/button"
          android:id="@+id/button1"
         android:layout_above="@+id/textview"
          android:layout_alignParentStart="true"/>

</RelativeLayout>
```



2.2.3. Table Layout

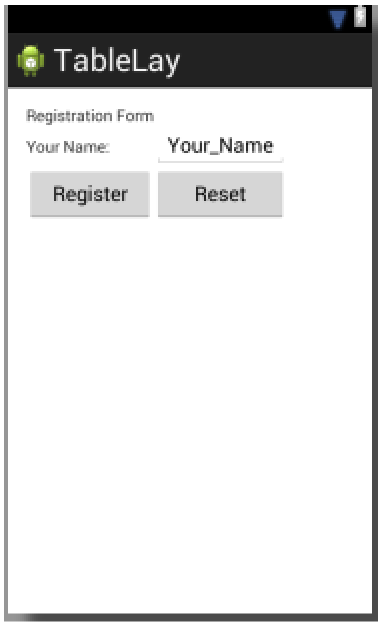
TableLayout organizes content into rows and columns. Table layouts in android works, in the same way HTML table layout work.

Example:

```
<TableLayout 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" >

   <TableRow>
       <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />
   </TableRow>
   <TableRow>
       <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/your_name" />
       <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:inputType="text"
        android:text="@string/nametxt" />
   </TableRow>
   <TableRow>
       <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/submit" />
       <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/cancel" />
   </TableRow>

</TableLayout>
```



2.2.4. Frame Layout

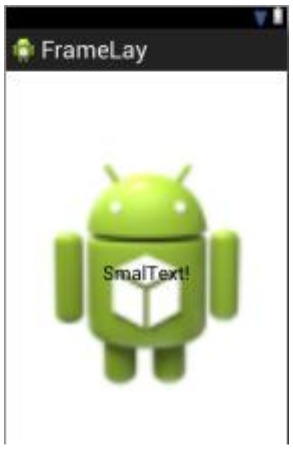
FrameLayout is designed to display a single item at a time. You can have multiple elements within a FrameLayout but each element will be positioned based on the top left of the screen. Elements that overlap will be displayed overlapping. See the following example:

```
<FrameLayout 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:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/image"
        android:src="@drawable/ic_launcher" />"
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/textview2"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="@string/textview3"
        android:layout_gravity="center" />

</FrameLayout>
```



As you can see on the above output the ImageView and TextView fill the parent in both horizontal and vertical layout. Gravity specifies where the text appears within its container, so I set that to center. If the gravity was not set the text would have appeared at the top left of the screen.

FrameLayout can become more useful when elements are hidden and displayed programmatically. You can use the attribute ***android:visibility*** in the XML to hide specific elements. You can even call ***setVisibility*** method from the code to accomplish the same thing. The three available visibility values are **visible, invisible** (does not display but still takes up space in the layout) **and gone** (does not display and does not take space in the layout)**.**

2.2.5. Layout Attributes

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

- ✓ ***android:id*** **-** This is the ID which uniquely identifies the view.
- ✓ ***android:layout_width*** **-** This is the width of the layout.
- ✓ ***android:layout_height*** -This is the height of the layout
- ✓ ***android:layout_marginTop*** **-** This is the extra space on the top side of the layout.
- ✓ ***android:layout_marginBottom*-** This is the extra space on the bottom side of the layout.
- ✓ ***android:layout_marginLeft*-** This is the extra space on the left side of the layout.
- ✓ ***android:layout_marginRight*-** This is the extra space on the right side of the layout.
- ✓ ***android:layout_gravity*-** This specifies how child Views are positioned.

- ✓ ***android:layout_weight-*** This specifies how much of the extra space in the layout should be allocated to the View.
- ✓ ***android:layout_x-*** This specifies the x-coordinate of the layout.
- ✓ ***android:layout_y-*** This specifies the y-coordinate of the layout.
- ✓ ***android:paddingLeft-*** This is the left padding filled for the layout.
- ✓ ***android:paddingRight-*** This is the right padding filled for the layout.
- ✓ ***android:paddingTop-*** This is the top padding filled for the layout.
- ✓ ***android:paddingBottom-***This is the bottom padding filled for the layout.

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

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

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

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

Constant	Value	Description
Top	0x30	Push object to the top of its container, not changing its size.
bottom	0x50	Push object to the bottom of its container, not changing its size.
Left	0x03	Push object to the left of its container, not changing its size.
Right	0x05	Push object to the right of its container, not changing its size.
center_vertical	0x10	Place object in the vertical center of its container, not changing its size.
fill_vertical	0x70	Grow the vertical size of the object if needed so it completely fills its container.
center_horizontal	0x01	Place object in the horizontal center of its container, not changing its size.
fill_horizontal	0x07	Grow the horizontal size of the object if needed so it completely fills its container.
Center	0x11	Place the object in the center of its container in both the vertical and horizontal axis, not changing its size.
Fill	0x77	Grow the horizontal and vertical size of the object if needed so it completely fills its container.

clip_vertical	0x80	Additional option that can be set to have the top and/or bottom edges of the child clipped to its container's bounds. The clip will be based on the vertical gravity: a top gravity will clip the bottom edge, a bottom gravity will clip the top edge, and neither will clip both edges.
clip_horizontal	0x08	Additional option that can be set to have the left and/or right edges of the child clipped to its container's bounds. The clip will be based on the horizontal gravity: a left gravity will clip the right edge, a right gravity will clip the left edge, and neither will clip both edges.
Start	0x00800003	Push object to the beginning of its container, not changing its size.
End	0x00800005	Push object to the end of its container, not changing its size.

View Identification

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

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

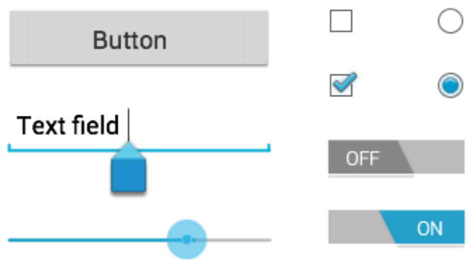
Following is a brief description of @ and + signs −

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

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

2.3. Android User Interface (UI) widgets

Android UI widgets are input controls which are used to build a user interface for your application. Input controls are the interactive components in your app's user interface. A **View** is an object that draws something on the screen that the user can interact with such as Text fields, Buttons etc and a **ViewGroup** is an object that holds other View (and ViewGroup) objects in order to define the layout of the user interface. The following picture shows some of the input controls.



Android provides various UI controls that allow you to build the graphical user interface for your app. These are ***TextView***, ***EditText***, ***AutoCompleteTextView***, ***Button***, ***ImageButton***, ***CheckBox, ToggleButton, RadioButton***, ***RadioGroup, ProgressBar, Spinner, TimePicker, DatePicker etc***

2.3.1. Create UI Controls

Buttons

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

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

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

Example: the following example shows how to create a button via a layout xml file.

```
<Button
    android:id="@+id/btn1 "
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Display"
    android:onClick="btnClick()"
/>
```

- ✓ *android:id*- provides a unique identification to a UI element**.** Thus when there are two or more buttons, it is this id which helps us to distinguish between the two in java code (usually done by ***findViewByID( )*** method)
- ✓ *android:layout_width & android:layout_height*- defines the size that the UI element should occupay.
- ✓ *android:text*- defines the text which the UI element should display. If you want to create an image button use *android:drawableLeft* property and set its value “@mipmap/ic_launcher”. (Where mimap is the directory in which the image is stored and ic_launcher is name of the image.

Example: the following example shows how to create a button programmatically in java file.

```
    public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
       Button button=new Button(this);
   button.setText("Display");
 RelativeLayout relativeLayout=(RelativeLayout) findViewById(R.id.rootlayout);
relativeLayout.addView(button);
  }
}
```

- ✓ Create the instance of the Button class by passing the current context as *this*.
- ✓ Set the text which the button should display.
- ✓ Create the object of the ViewGroup (Layout) in which the button needs to be placed

- ✓ Find the layout by using findViewById( ) method
- ✓ Add the button to the layout by addView function

ImageButton

As the name says, the ImageButton component is a button with an image on it. The ImageButton is represented by the Android class *android.widget.ImageButton*. It is similar with the button component except that its value is an Image than a text.

Example:

- if you want to create an image button use ***android:src*** attribute to define the location and name of the image which you want to use as button.

```
<ImageButton
  ...
  android:src="@drawable/the_image_button_icon"
/>
```

- If you want to create the button programmatically add *button.setImageSource(R.drawable.the_image_button_icon);* to define the location and name of the image which you want to use as button.

```
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
       Button button=new Button(this);
   button.setText("Display");
   button.setImageSource(R.drawable.the_image_button_icon);
RelativeLayout relativeLayout=(RelativeLayout) findViewById(R.id.rootlayout);
relativeLayout.addView(button);
  }
}
```

Button Click Event

When the user clicks on a button, the Button object receives an on-click event. To define the click event handler for a button, add the *android:onClick* attribute to the <Button> element in your xml layout. The value for this attribute must be the name of the method you want to call in response to a click event. For instance *android:onClick="btnOnclick()".* The Activity hosting the layout must then implement the corresponding method. **Example**: the following method will display "Button clicked!" message when a user clicks on the button.

```
  protected void btnClick(View view) {
    Toast.makeText(getBaseContext(), “Button Clicked!”,Toast.LENGTH_SHORT).show();  }
```

Android Toast

Toast is a notification message that pop up for a specific duration. It only fills the amount of space required for the message and the current activity remains visible and interactive. The ***android.widget.Toast*** class is the subclass of ***java.lang.Object*** class.

Constants of Toast Class:

- ✓ *public static final int LENGTH_LONG*- displays view for a long duration of time i.e. 3.5 seconds. (Toast.LENGTH_LONG)
- ✓ *public static final int LENGTH_SHORT*- displays view for a short duration of time i.e. 2 seconds. (Toast.LENGTH_SHORT)

Example:

```
Context context=getApplicationContext();
CharSequence text=”Button Clicked!”;
int duration=Toast.LENGTH_SHORT;
Toast toast=Toast.makeText(context,text,duration);
toast.show();
```

You can also chain methods like this:

```
Toast.makeText(getApplicationContext(), “Button Clicked!”,Toast.LENGTH_SHORT).show();
```

The method takes three parameters:

- ✓ an application context, the text message and the duration for the toast.

Positioning the Toast:

A standard toast notification appears near the bottom of the screen, centered horizontally. You can change this position with the **setGravity(int, int, int)** method.

The method accepts three parameters:

- ✓ a gravity constant, an x-position offset and y-position offset.

Example: this example displays the toast at the top-left corner of your application

```
Toast toast=Toast.makeText(this,”Toast MSG”,Toast.LENGTH_SHORT);
toast.setGravity(Gravity.AXIS_PULL_AFTER,0,0);
toast.show();
```

Android TextView

The TextView view is used to display text to the user. This is the most basic view and one that you will frequently use when you develop Android applications. If you need to allow users to edit the text displayed, you should use the subclass of TextView, EditText. In some other platforms the TextView is commonly known as label view. It’s sole purpose to display a text on the screen. **For example:** the following example will display the text “Hello World” on the screen

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

TextView code in JAVA:

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

```
textView.setText("Hello World"); //set text for text view
```

Attributes of TextView:

Some of the attributes of a TextView are:

- ✓ **id:** id is an attribute used to uniquely identify a text view.
- ✓ **gravity:** The gravity attribute is an optional attribute which is used to control the alignment of the text like left, right, center, top, bottom, center_vertical, center_horizontal etc.
- ✓ **text:** text attribute is used to set the text in a text view. We can set the text in xml as well as in the java class.
- ✓ **textColor:** textColor attribute is used to set the text color of a text view. Color value is in the form of "#argb", "#rgb", "#rrggbb", or "#aarrggbb".
- ✓ **textSize:** textSize attribute is used to set the size of text of a text view. We can set the text size in sp(scale independent pixel) or dp(density pixel).
- ✓ **textStyle:** textStyle attribute is used to set the text style of a text view. The possible text styles are bold, italic and normal. If we need to use two or more styles for a text view then "|" operator is used for that. Example: android:textStyle="bold|italics"
- ✓ **background:** background attribute is used to set the background of a text view. We can set a color or a drawable in the background of a text view.
- ✓ **padding:** padding attribute is used to set the padding from left, right, top or bottom. In above example code of background we also set the 10dp padding from all the side's of text view.

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

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout . . .>
    <TextView
        android:id="@+id/simpleTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
      android:layout_centerHorizontal="true"
        android:text="Before Clicking"
        android:textColor="#f00"
        android:textSize="25sp"
        android:textStyle="bold|italic"
        android:layout_marginTop="50dp"/>
    <Button
        android:id="@+id/btnChangeText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:background="#f00"
        android:padding="10dp"
        android:text="Change Text"
        android:textColor="#fff"
        android:textStyle="bold" />
```

MainActivity.java

```
import android.graphics.Color;
import
android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.*;
import android.widget.*;
public    class    MainActivity    extends
AppCompatActivity {
    @Override
    protected     void     onCreate(Bundle
savedInstanceState) {
      super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
        final  TextView  simpleTextView  =
(TextView)
findViewById(R.id.simpleTextView);
   Button     changeText     =     (Button)
findViewById(R.id.btnChangeText);
changeText.setOnClickListener(new
View.OnClickListener() {
              @Override
   public void onClick(View view) {
```

`</RelativeLayout>`	`simpleTextView.setText("After Clicking");`
	`            } });`
	`    }}`

Android EditText

EditText view is a subclass of the TextView that allows users to edit its text content. It is also known as a text field in other platforms. It can be either single line or multi-line. Touching a text field places the cursor and automatically displays the keyboard. In addition to typing, text fields allow for a variety of activities such as text selection (cut, copy, paste) and data look-up via auto-completion. You can specify the type of keyboard you want for your EditText object with the *android:inputType* attribute. For example, if you want the user to input a phone number, you should use the phone input type.

Example:

```
<EditText
        android:id="@+id/search "
        android:layout_width="fill_parent "
        android:layout_height="wrap_content"
        android:hint="Search Tex"
        android:inputTYpe="text"
        android:imeOptions="actionSend"
    />
```

There are different input types, the most common are:

- ✓ ***text-*** displays a normal text keyboard
- ✓ ***textEmailAddress-*** displays a normal text keyboard with the @ character
- ✓ ***textUri-*** displays a normal text keyboard with the / character
- ✓ ***number-*** displays a basic number keypad
- ✓ ***phone-*** displays a phone style keypad

The *android:inputType* property also allows you to specify certain keyboard behaviors such as whether to capitalize all new words or use features like auto-complete and spelling suggestions.

- ✓ ***textCapSentences***-displays normal text keyboard that capitalizes the first letter for each new sentence.
- ✓ ***textCapWords-*** displays normal text keyboard that capitalizes every word.
- ✓ ***textAutoCorrect-*** displays normal text keyboard that corrects commonly misspelled words
- ✓ ***textPassword-*** displays normal text keyboard but the characters entered turn into dots.
- ✓ ***textMultiLine-*** displays normal text keyboard that allow users to input long strings of text that include line breaks.(carriage return)

Retrieving / Getting the Value From EditText In Java Class:

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

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

```
String editTextValue = simpleEditText.getText().toString();
```

Keyboard Actions

In addition to changing the keyboard's input type android allows you to specify an action to be made when users have completed their input. The action specifies the button that appears in place of the carriage return key and the action to be made, such as Search or Send. You can specify the action by setting the *android:imeOptions* attribute. For example, *android:imeOptions="actionSend"* in the above example specifies a send action.

Listening to Keyboard Actions

If you have specified a keyboard action for the input method using *android:imeOptions* attribute, you can listen for the specific action event using an *TextView.onEditorActionListener*.

The *TextView.onEditorActionListener* interface provides a callback method called *onEditorAction()* that indicates the action type invoked with an action ID such as IME_ACTION_SEND or IME_ACTION_SEARCH. **For example**, here is how you can listen when the user clicks the send button on the keyboard.

```
  EditText editText= (EditText)findViewById(R.id.search);
 editText.setOnEditorActionListener(new OnEditorActionListener(){

    @Override
    Public boolean onEditorAction(TextView v,int actionId, KeyEvent event){
     boolean handled=false;
     if(actionId==EditorInfo.IME_ACTION_SEND){
       sendMessage();
       handled=true;
      }
       return handled;
   }
});
```

Attributes of EditText:

EditText supports the attributes mentioned for TextView , the attribute described below and a lot more:

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

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

activity_main.xml	**MainActivity.java**

`<?xml version="1.0" encoding="utf-8"?>`
`<RelativeLayout ...>`
`<EditText`
`android:id="@+id/editText1"`
`android:layout_width="wrap_content"`
`android:layout_height="wrap_content"`
`android:layout_alignParentLeft="true"`
	`import`
	`android.support.v7.app.AppCompatActivity;`
	`import android.os.Bundle;`
	`import android.view.View;`
	`import android.widget.Button;`
	`import android.widget.EditText;`
	`import android.widget.Toast;`

```
android:layout_alignParentStart="true"
android:layout_alignParentTop="true"
android:layout_marginLeft="50dp"
android:layout_marginStart="50dp"
android:layout_marginTop="24dp"
android:ems="10"
android:hint="@string/name"
android:inputType="textPersonName"
android:selectAllOnFocus="true" />

    <EditText
 android:id="@+id/editText2"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_alignLeft="@+id/editText1"
android:layout_alignStart="@+id/editText1"
android:layout_below="@+id/editText1"
android:layout_marginTop="19dp"
android:ems="10"
android:hint="@string/password_0_9"
android:inputType="numberPassword" />
    <EditText
 android:id="@+id/editText3"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_alignLeft="@+id/editText2"
android:layout_alignStart="@+id/editText2"
android:layout_below="@+id/editText2"
android:layout_marginTop="12dp"
android:ems="10"
android:hint="@string/e_mail"
android:inputType="textEmailAddress" />
    <EditText
        android:id="@+id/editText4"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_alignLeft="@+id/editText3"
android:layout_alignStart="@+id/editText3"
android:layout_below="@+id/editText3"
android:layout_marginTop="18dp"
android:ems="10"
android:hint="@string/date"
android:inputType="date" />
    <EditText
        android:id="@+id/editText5"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_alignLeft="@+id/editText4"
android:layout_alignStart="@+id/editText4"
android:layout_below="@+id/editText4"
android:layout_marginTop="18dp"
android:ems="10"
android:hint="@string/contact_number"
android:inputType="phone" />
    <Button
        android:id="@+id/button"
style="@android:style/Widget.Button"
android:layout_width="match_parent"
android:layout_height="wrap_content"
```

```
public class MainActivity extends
AppCompatActivity {
    Button submit;
    EditText name, password, email,
contact, date;
    @Override
    protected void onCreate(Bundle
savedInstanceState) {

super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
name = (EditText)
findViewById(R.id.editText1);
password = (EditText)
findViewById(R.id.editText2);
 email = (EditText)
findViewById(R.id.editText3);
 date = (EditText)
findViewById(R.id.editText4);
 contact = (EditText)
findViewById(R.id.editText5);
  submit = (Button)
findViewById(R.id.button);
  submit.setOnClickListener(new
View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if
(name.getText().toString().isEmpty() ||
password.getText().toString().isEmpty()
|| email.getText().toString().isEmpty()
|| date.getText().toString().isEmpty()
||contact.getText().toString().isEmpty())
{
Toast.makeText(getApplicationContext(),
"Enter the Data",
Toast.LENGTH_SHORT).show();
                } else {

Toast.makeText(getApplicationContext(),
"Name -  " + name.getText().toString() +
" \n" + "Password -  " +
password.getText().toString()
 + " \n" + "E-Mail -  " +
email.getText().toString() + " \n" +
"Date -  " + date.getText().toString()
 + " \n" + "Contact -  " +
contact.getText().toString(),
Toast.LENGTH_SHORT).show();
                }
            }
        });
    }
}
```

`android:layout_alignParentLeft="true"`
`android:layout_alignParentStart="true"`
`android:layout_below="@+id/editText5"`
`android:layout_marginTop="62dp"`
`android:text="@string/submit"`
`android:textSize="16sp"`
`android:textStyle="normal|bold" />`
`</RelativeLayout>`	

AutoCompleteTextView

An AutoCompleteTextView is a view that is similar to EditText, except that it shows a list of completion suggestions automatically while the user is typing. The list of suggestions is displayed in drop down menu. The user can choose an item from there to replace the content of edit box with. It is the subclass of EditText class.

The list of suggestions is obtained from data adapter and appears only after a given number of characters defined by threshold.

Steps of creating AutoCompleteTextView

1. Create AutoCompleteTextView in the xml

```
<AutoCompleteTextView
     android:layout_width="match_parent "
     android:layout_height="wrap_content"
     android:id="@+id/course_list"
/>
```

2. Create the list in res/values/string.xml

```
<resources>
    <string-array name="course_array">
        <item>OOP in Java</item>
        <item>Data Structures in Java</item>
        <item>Advanced Progarmming using Java</item>
    </string-array>
</resources>
```

3. Get the reference of AutoCompleteTextView in java class

```
AutoCompleteTextView textView=( AutoCompleteTextView)findViewById(R.id.course_list);
String[] courses=getResources().getStringArray(R.array.courses_array);
ArrayAdapter<String> adapter=new
ArrayAdapter<String>(this.android.R.layout.simple_list_item_1,courses);
textView.setAdapter(adapter);
```

Retrieving the Value From AutoCompleteTextView In Java Class:

Below code retrieve the value from a AutoCompleteTextView in Java class.

```
AutoCompleteTextView simpleAutoCompleteTextView = (AutoCompleteTextView)
findViewById(R.id.course_list);
String AutoCompleteTextViewValue = simpleAutoCompleteTextView.getText().toString();
```

AutoCompleteTextView Attributes

In addition to the attributes described in EditText and TextView section AutoCompleteTextView has the following attributes:

- ✓ ***android:completionHint-*** defines the hint displayed in the drop down menu.
- ✓ ***android:completionHintView-*** defines the hint view displayed in the drop down menu.
- ✓ ***android:completionThreshold-*** defines the number of characters that the user must type before completion suggestions are displayed in a drop down menu.
- ✓ ***android:dropDownAnchor-*** is the View to anchor the auto-complete dropdown to.
- ✓ ***android:dropDownHeight-*** specifies the basic height of the dropdown.
- ✓ ***android:dropDownHorizontalOffset-***The amount of pixels by which the drop down should be offset horizontally.
- ✓ ***android:dropDownSelector-*** is the selector in a drop down list.
- ✓ ***android:dropDownVerticalOffset-***The amount of pixels by which the drop down should be offset vertically.
- ✓ ***android:dropDownWidth-*** specifies the basic width of the dropdown.
- ✓ ***android:popupBackground-*** sets the background.

Example: In the example of AutoCompleteTextView we display a auto complete text view with suggestion list which include country list. To fill the data in country list we implement an Array Adapter.

activity_main.xml

```xml
<?xml version="1.0" encoding="utf-
8"?>
<RelativeLayout ...>
<AutoCompleteTextView
android:id="@+id/sACTV"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:background="#000"
android:hint="Enter Your Name Here"
android:padding="15dp"
android:textColorHint="#fff"
android:textStyle="bold|italic" />


</RelativeLayout>
```

MainActivity.java

```java
import
android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.*;
import android.widget.*;
public class MainActivity extends
AppCompatActivity {
String[] countryNameList = {"Ethiopia",
"Eritrea", "Kenya", "Sudan", "England"};
    @Override
    protected void onCreate(Bundle
savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
 AutoCompleteTextView
simpleAutoCompleteTextView =
(AutoCompleteTextView)
findViewById(R.id.sACTV);
ArrayAdapter adapter = new ArrayAdapter(this,
android.R.layout.simple_list_item_1,
countryNameList);
simpleAutoCompleteTextView.setAdapter(adapter);
simpleAutoCompleteTextView.setThreshold(1);
simpleAutoCompleteTextView.setAdapter(adapter);
}}
```

CheckBox

In Android, CheckBox is a type of two state button either unchecked or checked in Android. Or you can say it is a type of on/off switch that can be toggled by the users. You should use checkbox when presenting a group of selectable options to users that are not mutually exclusive. CompoundButton is the parent class of CheckBoxclass.

CheckBox code in XML:

```
<CheckBox
android:id="@+id/simpleCheckBox"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Simple CheckBox"/>
```

You can check the current state of a check box programmatically by using isChecked() method. This method returns a Boolean value either true or false, if a check box is checked then it returns true otherwise it returns false. Below is an example code in which we checked the current state of a check box.

```
//initiate a check box
CheckBox simpleCheckBox = (CheckBox) findViewById(R.id.simpleCheckBox);
//check current state of a check box (true or false)
Boolean checkBoxState = simpleCheckBox.isChecked();
```

CheckBox Attributes:

In addition to the above mentioned attributes CheckBox has the following attributes:

- ✓ ***android:autoText***- If set, specifies that this TextView has a textual input method and automatically corrects some common spelling errors.
- ✓ ***android:drawableBottom*** - is the drawable to be drawn below the text.
- ✓ ***android:drawableRight***- is the drawable to be drawn to the right of the text.
- ✓ ***android:editable***-If set, specifies that this TextView has an input method.
- ✓ ***android:text***- is the Text to display.
- ✓ ***android:background***- is a drawable to use as the background.
- ✓ ***android:contentDescription***- defines text that briefly describes content of the view.
- ✓ ***android:id***- supplies an identifier name for this view,
- ✓ ***android:onClick***- is the name of the method in this View's context to invoke when the view is clicked.
- ✓ ***android:visibility***- controls the initial visibility of the view.

Example: Below is the example of CheckBox in Android, in which we display five check boxes using background and other attributes we discusses earlier in this post. Every check box represents a different subject name and whenever you click on a check box then text of that checked check box is displayed in a toast.

activity_main.xml	MainActivity.java

`<?xml version="1.0" encoding="utf-8"?>`
`<RelativeLayout ...>`
`<TextView`
	`import android.graphics.Color;`
	`import`
	`android.support.v7.app.AppCompatActivity;`
	`import android.os.Bundle;`

```
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Select   Your   Programming
language:                            "
android:textColor="#f00"
android:textSize="20sp"
android:textStyle="bold" />
    <LinearLayout
android:id="@+id/linearLayout"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:layout_marginTop="30dp"
android:background="#e0e0e0"
android:orientation="vertical">
        <CheckBox
android:id="@+id/androidCheckBox"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_centerHorizontal="true"
android:checked="false"
android:padding="20dp"
android:text="@string/android"
android:textColor="#44f"
android:textSize="20sp"
android:textStyle="bold|italic" />
        <CheckBox
android:id="@+id/javaCheckBox"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_centerHorizontal="true"
android:checked="false"
android:padding="20dp"
android:text="@string/java"
android:textColor="#f44"
android:textSize="20sp"
android:textStyle="bold|italic" />
        <CheckBox
android:id="@+id/phpCheckBox"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_centerHorizontal="true"
android:checked="false"
android:padding="20dp"
android:text="@string/php"
android:textColor="#444"
android:textSize="20sp"
android:textStyle="bold|italic" />
 </LinearLayout>

</RelativeLayout>
```

```
import android.view.*;
import android.widget.*;
public class MainActivity extends
AppCompatActivity implements
View.OnClickListener {
CheckBox android, java, php;
    @Override
    protected void onCreate(Bundle
savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
android = (CheckBox)
findViewById(R.id.androidCheckBox);
android.setOnClickListener(this);
java = (CheckBox)
findViewById(R.id.javaCheckBox);
java.setOnClickListener(this);
php = (CheckBox)
findViewById(R.id.phpCheckBox);
php.setOnClickListener(this);
    }
    @Override
    public void onClick(View view) {
        switch (view.getId()) {
            case R.id.androidCheckBox:
                if (android.isChecked())
Toast.makeText(getApplicationContext(),
"Android", Toast.LENGTH_LONG).show();
                break;
            case R.id.javaCheckBox:
                if (java.isChecked())

Toast.makeText(getApplicationContext(),
"Java", Toast.LENGTH_LONG).show();
                break;
            case R.id.phpCheckBox:
                if (php.isChecked())

Toast.makeText(getApplicationContext(),
"PHP", Toast.LENGTH_LONG).show();
                break;
        }    }}
```

strings.xml

```
<resources>
<string
name="app_name">CheckBoxExample</string>
<string name="hello_world">Hello world!
</string><string
name="action_settings">Settings</string>
<string name="android">Android</string>
<string name="java">Java</string>
<string name="php">PHP</string>
</resources>
```

ToggleButton

ToggleButton allow the users to change the setting between two states like turn on/off your wifi, Bluetooth etc from your phone's setting menu. Since, Android 4.0 version (API level 14) there is an another kind of ToggleButton called Switch which provide the user slider control. You can learn more about it reading Switch tutorial. Android Switch and ToggleButton both are the subclasses of compoundButton class.

ToggleButton code in XML:

```
<ToggleButton

android:id="@+id/simpleToggleButton"

android:layout_width="wrap_content"

android:layout_height="wrap_content"/>
```

How To Check Current State Of ToggleButton:

To check current state of a toggle button programmatically we use isChecked() method. This method returns a Boolean value either true or false. If a toggle button is checked then it returns true otherwise it returns false. Below is the code which checks the current state of a toggle button.

```
/*Add in Oncreate() funtion after setContentView()*/

ToggleButton simpleToggleButton = (ToggleButton)
findViewById(R.id.simpleToggleButton); // initiate a toggle button

Boolean ToggleButtonState = simpleToggleButton.isChecked(); // check current state of
a toggle button (true or false).
```

ToggleButton Attributes

In addition to the above mentioned attributes ToggleButton has the following attributes:

- ✓ ***android:disabledAlpha*** - is the alpha to apply to the indicator when disabled.
- ✓ ***android:textOff*** - is the text for the button when it is not checked.
- ✓ ***android:textOn*** - is the text for the button when it is checked.
- ✓ ***android:autoText*** - If set, specifies that this TextView has a textual input method and automatically corrects some common spelling errors.
- ✓ ***android:drawableBottom*** - is the drawable to be drawn below the text.
- ✓ ***android:drawableRight*** - is the drawable to be drawn to the right of the text
- ✓ ***android:editable***-If set, specifies that this TextView has an input method.
- ✓ ***android:text*** - is the Text to display. Inherited from android.view.View Class:
- ✓ ***android:background*** - is a drawable to use as the background.
- ✓ ***android:contentDescription*** - defines text that briefly describes content of the view.
- ✓ ***android:id*** - is supplies an identifier name for this view,
- ✓ ***android:onClick***- is the name of the method in this View's context to invoke when the view is clicked.
- ✓ ***android:visibility*** - controls the initial visibility of the view.

- ✓ ***android:checked*****:** checked is an attribute of toggle button used to set the current state of a toggle button. The value should be true or false where true shows the checked state and false shows unchecked state of a toggle button.
- ✓ ***android:drawableBottom, drawableTop, drawableRight And drawableLeft:*** These attribute draw the drawable below, top, right and left of the text of ToggleButton

Example: Below is the example of ToggleButton in Android Studio. In this example we display two toggle button with background and one "submit" button using attributes discussed earlier in this post. Whenever user click on the submit button, the current state of both toggle button's is displayed in a Toast.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout . . . >
    <LinearLayout
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_gravity="center_horizontal"
android:orientation="horizontal">
        <ToggleButton
android:id="@+id/simpleToggleButton1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_gravity="center_horizontal"
android:checked="false"
android:drawablePadding="20dp"
android:drawableRight="@drawable/ic_launche
r"
android:textColor="#000" />
        <ToggleButton
android:id="@+id/simpleToggleButton2"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_gravity="center_horizontal"
android:layout_marginLeft="50dp"
android:checked="true"
android:drawableLeft="@drawable/ic_launcher
"
android:drawablePadding="20dp"
android:textColor="#000" />
    </LinearLayout>
    <Button
android:id="@+id/submitButton"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_gravity="center"
android:layout_marginTop="50dp"
android:background="#0f0"
android:padding="10dp"
android:text="Submit"
android:textColor="#fff"
android:textSize="20sp"
android:textStyle="bold" />
</LinearLayout>
```

MainActivity.java

```
import
android.support.v7.app.AppCompatActivit
y;
import android.os.Bundle;
import android.view.*;
import android.widget.*;
public   class   MainActivity   extends
AppCompatActivity {
ToggleButton        simpleToggleButton1,
simpleToggleButton2;
    Button submit;
    @Override
    protected    void    onCreate(Bundle
savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
simpleToggleButton1   =   (ToggleButton)
findViewById(R.id.simpleToggleButton1);
simpleToggleButton2   =   (ToggleButton)
findViewById(R.id.simpleToggleButton2);
submit            =            (Button)
findViewById(R.id.submitButton);
submit.setOnClickListener(new
View.OnClickListener() {
            @Override
 public void onClick(View view) {
  String status = "ToggleButton1 : " +
simpleToggleButton1.getText() + "\n" +
"ToggleButton2        :        "      +
simpleToggleButton2.getText();
Toast.makeText(getApplicationContext(),
status, Toast.LENGTH_SHORT).show();
            }
        });
    }
}
```

RadioButton

In Android, RadioButton are mainly used together in a RadioGroup. In RadioGroup checking the one radio button out of several radio button added in it will automatically unchecked all the others. It means at one time we can checked only one radio button from a group of radio buttons which belong to same radio group. The most common use of radio button is in Quiz Android App code. RadioButon is a two state button that can be checked or unchecked. If a radio button is unchecked then a user can check it by simply clicking on it. Once a RadiaButton is checked by user it can't be unchecked by simply pressing on the same button. It will automatically unchecked when you press any other RadioButton within same RadioGroup.

RadioGroup is a widget used in Android for the grouping of radio buttons and provide the feature of selecting only one radio button from the set. When a user try to select any other radio button within same radio group the previously selected radio button will be automatically unchecked.

RadioGroup And RadioButton code in XML:

```
<RadioGroup
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
        <RadioButton
        android:id="@+id/simpleRadioButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
        <RadioButton
        android:id="@+id/simpleRadioButton1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
</RadioGroup>
```

Checking Current State Of Radio Button:

You can check the current state of a radio button programmatically by using isChecked() method. This method returns a Boolean value either true or false. if it is checked then returns true otherwise returns false. Below is an example code with explanation in which we checked the current state of a radio button.

```
/*Add in Oncreate() funtion after setContentView()*/

RadioButton simpleRadioButton = (RadioButton) findViewById(R.id.simpleRadioButton);
// initiate a radio button

Boolean RadioButtonState = simpleRadioButton.isChecked(); // check current state of a
radio button (true or false).
```

RadioButton Attributes: RadioButton supports the attributes mentioned for most UI widgets.

Example: Below is the example of Radiobutton in Android where we display true and false radio buttons with background and other attributes. The radio buttons are used to select your answer for the given question with one "submit" button.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout . . . >
    <LinearLayout
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:background="#e0e0e0"
android:orientation="vertical">
        <TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Java is the only
programming language used to develop a
mobile app"
android:textColor="#000"
android:textSize="20sp"
android:textStyle="bold" />
        <RadioGroup
android:layout_width="wrap_content"
android:layout_height="wrap_content">
       <RadioButton
android:id="@+id/rbtrue"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_marginLeft="20dp"
android:layout_marginTop="10dp"
android:checked="true"
android:text="@string/strtrue"
android:textColor="#154"
android:textSize="20sp"
android:textStyle="bold" />
     <RadioButton
android:id="@+id/rbfalse"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_marginLeft="20dp"
android:layout_marginTop="10dp"
android:checked="false"
android:text="@string/strfalse"
android:textColor="#154"
android:textSize="20sp"
android:textStyle="bold" />
          </RadioGroup>
        <Button
android:id="@+id/submitButton"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_gravity="center"
android:layout_margin="20dp"
android:background="#0f0"
android:padding="10dp"
```

MainActivity.java

```
import
android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.*;
import android.widget.*;
public class MainActivity extends
AppCompatActivity {
RadioButton bttrue, btfalse;
    String selectedanswer;
    Button submit;
    @Override
protected void onCreate(Bundle
savedInstanceState) {
 super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
 bttrue = (RadioButton)
findViewById(R.id.rbtrue);
 btfalse = (RadioButton)
findViewById(R.id.rbfalse);
findViewById(R.id.sheamus);
 submit = (Button)
findViewById(R.id.submitButton);
 submit.setOnClickListener(new
View.OnClickListener() {
            @Override
 public void onClick(View v) {
 if (bttrue.isChecked()) {
   selectedanswer =
bttrue.getText().toString(); }
 else if (btfalse.isChecked()) {
 selectedanswer =
btfalse.getText().toString();           }
Toast.makeText(getApplicationContext(),
selectedanswer, Toast.LENGTH_LONG).show();
}        });
    }}
```

strings.xml

```
<resources>
    <string
name="app_name">RadioButtonExample</string>
<string name="hello_world">Hello
world!</string><string
name="action_settings">Settings</string>
<string name="strtrue">True</string>
<string name="strfalse">False</string>
</resources>
```

`android:text="Submit"`
`android:textColor="#fff"`
`android:textSize="20sp"`
`android:textStyle="bold" />`
`    </LinearLayout>`
 	
`</LinearLayout>`	

Assignment1

2.4. Android - Event Handling

Events are a useful way to collect data about a user's interaction with interactive components of Applications. Like button presses or screen touch etc. The Android framework maintains an event queue as first-in, first-out (FIFO) basis. You can capture these events in your program and take appropriate action as per requirements.

There are three concepts related to Android Event Management −

- ✓ **Event Listeners** − An event listener is an interface in the View class that contains a single callback method. These methods will be called by the Android framework when the View to which the listener has been registered is triggered by user interaction with the item in the UI.
- ✓ **Event Listeners Registration** − Event Registration is the process by which an Event Handler gets registered with an Event Listener so that the handler is called when the Event Listener fires the event.
- ✓ **Event Handlers** − When an event happens and we have registered an event listener for the event, the event listener calls the Event Handlers, which is the method that actually handles the event.

Event Listeners & Event Handlers

Event Handler	Event Listener & Description
onClick()	OnClickListener() This is called when the user either clicks or touches or focuses upon any widget like button, text, image etc. You will use onClick() event handler to handle such event.
onLongClick()	OnLongClickListener() This is called when the user either clicks or touches or focuses upon any widget like button, text, image etc. for one or more seconds. You will use onLongClick() event handler to handle such event.
onFocusChange()	OnFocusChangeListener() This is called when the widget looses its focus ie. user goes away from the view item. You will use onFocusChange() event handler to handle such event.
onKey()	OnFocusChangeListener() This is called when the user is focused on the item and presses or releases a hardware key on the device. You will use onKey() event handler to handle such event.

onTouch()	OnTouchListener() This is called when the user presses the key, releases the key, or any movement gesture on the screen. You will use onTouch() event handler to handle such event.
onMenuItemClick()	OnMenuItemClickListener() This is called when the user selects a menu item. You will use onMenuItemClick() event handler to handle such event.
onCreateContextMenu()	onCreateContextMenuItemListener() This is called when the context menu is being built(as the result of a sustained "long click)

There are many more event listeners available as a part of **View** class like OnHoverListener, OnDragListener etc which may be needed for your application. So refer the official documentation for Android application development in for further information.

Event Listeners Registration

Event Registration is the process by which an Event Handler gets registered with an Event Listener so that the handler is called when the Event Listener fires the event. Though there are several tricky ways to register your event listener for any event, but some of them are explained below.

- ✓ Using an Anonymous Inner Class
- ✓ Activity class implements the Listener interface.
- ✓ Using Layout file activity_main.xml to specify event handler directly.

Event listeners registration using an anonymous inner class

Here you will create an anonymous implementation of the listener and will be useful if each class is applied to a single control only and you have advantage to pass arguments to event handler. In this approach event handler methods can access private data of Activity. No reference is needed to call to Activity. But if you applied the handler to more than one control, you would have to cut and paste the code for the handler and if the code for the handler is long, it makes the code harder to maintain.

Steps

1. Create an Android application and name it as *EventDemo* under a package *com.example.eventdemo* as explained in the *Hello World Example* chapter.
2. Modify *java/MainActivity.java* file to add click event listeners and handlers for a button defined.
3. Modify the default content of *res/layout/activity_main.xml* file to include Android UI controls.
4. Define required 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.eventdemo/MainActivity.java**. This file can include each of the fundamental lifecycle methods.

```
package com.example.eventdemo;
import android.os.Bundle;
import android.app.Activity;
import android.view.*;
import android.widget.*;
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //--- find the button---
        Button sButton = (Button) findViewById(R.id.button_s);
        // -- register click event with the button ---
        sButton.setOnClickListener(new View.OnClickListener() {
           public void onClick(View v) {
               // --- find the text view --
               TextView txtView = (TextView) findViewById(R.id.text_id);
               // -- change text size --
               txtView.setTextSize(14);
           }
        });

       }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

}
```

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

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:orientation="vertical" >
    <Button
    android:id="@+id/button_s"
    android:layout_height="wrap_content"
    android:layout_width="match_parent"
    android:text="@string/button_small"/>
    <TextView
    android:id="@+id/text_id"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:capitalize="characters"
    android:text="@string/hello_world" />
</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">EventDemo</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
   <string name="button_small">Small Font</string>
</resources>
```

Registration using the activity implements listener interface

Here your Activity class implements the Listener interface and you put the handler method in the main Activity and then you call ***setOnClickListener(this).*** This approach is fine if your application has only a single control of that Listener type otherwise you will have to do further programming to check which control has generated event. Second you cannot pass arguments to the Listener so, again, works poorly for multiple controls.

Following are the simple steps to show how we will make use of separate Listener class to register and capture click event. Similar way you can implement your listener for any other required event type.

Following are the simple steps to show how we will implement Listener class to register and capture click event. Similar way you can implement your listener for any other required event type.

Steps

1. We do not need to create this application from scratch, so let's make use of above created Android application *EventDemo*.
2. Modify *java/MainActivity.java* file to add click event listeners and handlers for a button defined.
3. We are not making any change in *res/layout/activity_main.xml*, it will remain as shown above.
4. We are also not making any change in *res/values/strings.xml* file, it will also remain as shown above.
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.eventdemo/MainActivity.java**. This filecan include each of the fundamental lifecycle methods.

```
package com.example.eventdemo;

import android.os.Bundle;
import android.app.Activity;
import android.view.*;
import android.widget.*;
public class MainActivity extends Activity implements OnClickListener {

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

        //--- find the button---
        Button sButton = (Button) findViewById(R.id.button_s);

        // -- register click event with the button ---
        sButton.setOnClickListener(this);
}

    //--- Implement the OnClickListener callback
    public void onClick(View v) {
       if(v.getId() == R.id.button_s)
       {
            // --- find the text view --
            TextView txtView = (TextView) findViewById(R.id.text_id);
            // -- change text size --
            txtView.setTextSize(14);
```

```
                return;
            }
        }

        @Override
        public boolean onCreateOptionsMenu(Menu menu) {
            getMenuInflater().inflate(R.menu.main, menu);
            return true;
        }

}
```

Registration using layout file activity_main.xml

Here you put your event handlers in Activity class without implementing a Listener interface or call to any listener method. Rather you will use the layout file (activity_main.xml) to specify the handler method via the ***android:onClick*** attribute for click event. You can control click events differently for different control by passing different event handler methods.

The event handler method must have a void return type and take a View as an argument. However, the method name is arbitrary, and the main class need not implement any particular interface.

This approach does not allow you to pass arguments to Listener and for the Android developers it will be difficult to know which method is the handler for which control until they look into activity_main.xml file. Second, you cannot handle any other event except click event using this approach.

Following are the simple steps to show how we can make use of layout file Main.xml to register and capture click event.

Step

1. We do not need to create this application from scratch, so let's make use of above created Android application *EventDemo*.
2. Modify *java/MainActivity.java* file to add click event listeners and handlers for the two buttons defined.
3. Modify layout file *res/layout/activity_main.xml*, to specify event handlers for the two buttons.
4. We are also not making any change in *res/values/strings.xml* file, it will also remain as shown above.
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.eventdemo/MainActivity.java**. This file can include each of the fundamental lifecycle methods.

```
package com.example.eventdemo;
import android.os.Bundle;
import android.app.Activity;
import android.view.*;
import android.widget.*;

public class MainActivity extends Activity{

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

    //--- Implement the event handler for the button.
    public void doSmall(View v)  {
       // --- find the text view --
       TextView txtView = (TextView) findViewById(R.id.text_id);
       // -- change text size --
       txtView.setTextSize(14);
       return;
   }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

}
```

Following will be the content of **res/layout/activity_main.xml** file. Here we have to add **android:onClick="methodName"** for the button, which will register given method names as click event handlers.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:orientation="vertical" >
    <Button
    android:id="@+id/button_s"
    android:layout_height="wrap_content"
    android:layout_width="match_parent"
    android:text="@string/button_small"
    android:onClick="doSmall"/>
   <TextView
    android:id="@+id/text_id"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:capitalize="characters"
    android:text="@string/hello_world" />

</LinearLayout>
```

2.5. Styles and Themes

If you already know about Cascading Style Sheet (CSS) in web design then to understand Android Style also works very similar way. There are number of attributes associated with each Android widget which you can set to change your application look and feel. A style can specify properties such as height, padding, font color, font size, background color, and much more.

You can specify these attributes in Layout file as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:orientation="vertical" >
   <TextView
   android:id="@+id/text_id"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:capitalize="characters"
   android:textColor="#00FF00"
   android:typeface="monospace"
   android:text="@string/hello_world" />
</LinearLayout>
```

But this way we need to define style attributes for every attribute separately which is not good for source code maintenance point of view. So we work with styles by defining them in separate file as explained below.

Defining Styles

A style is defined in an XML resource that is separate from the XML that specifies the layout. This XML file resides under res/values/ directory of your project and will have <resources> as the root node which is mandatory for the style file. The name of the XML file is arbitrary, but it must use the .xml extension.

You can define multiple styles per file using <style> tag but each style will have its name that uniquely identifies the style. Android style attributes are set using <item> tag as shown below:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <style name="CustomFontStyle">
      <item name="android:layout_width">fill_parent</item>
      <item name="android:layout_height">wrap_content</item>
      <item name="android:capitalize">characters</item>
      <item name="android:typeface">monospace</item>
      <item name="android:textSize">12pt</item>
      <item name="android:textColor">#00FF00</item>/>
   </style>
</resources>
```

The value for the <item> can be a keyword string, a hex color, a reference to another resource type, or other value depending on the style property.

Using Styles

Once your style is defined, you can use it in your XML Layout file using style attribute as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:orientation="vertical" >
   <TextView
   android:id="@+id/text_id"
   style="@style/CustomFontStyle"
   android:text="@string/hello_world" />

</LinearLayout>
```

Style Inheritance

Android supports style Inheritance in very much similar way as cascading style sheet in web design. You can use this to inherit properties from an existing style and then define only the properties that you want to change or add.

It is simple, to create a new style LargeFont that inherits the CustomFontStyle style defined above, but make the font size big, you can author the new style like this:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <style name="CustomFontStyle.LargeFont">
      <item name="android:textSize">20ps</item>
   </style>
</resources>
```

You can reference this new style as @style/CustomFontStyle.LargeFont in your XML Layout file. You can continue inheriting like this as many times as you'd like, by chaining names with periods. For example, you can extend FontStyle.LargeFont to be Red, with:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <style name="CustomFontStyle.LargeFont.Red">
      <item name="android:textColor">#FF0000</item>/>
</style>
</resources>
```

This technique for inheritance by chaining together names only works for styles defined by your own resources.

You can't inherit Android built-in styles this way. To reference an Android built-in style, such as TextAppearance, you must use the parent attribute as shown below:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <style name="CustomFontStyle" parent="@android:style/TextAppearance">
      <item name="android:layout_width">fill_parent</item>
      <item name="android:layout_height">wrap_content</item>
      <item name="android:capitalize">characters</item>
      <item name="android:typeface">monospace</item>
      <item name="android:textSize">12pt</item>
      <item name="android:textColor">#00FF00</item>/>
   </style>
</resources>
```

Android Themes

Hope you understood the concept of Style, so now let's try to understand what is a Theme. A theme is nothing but an Android style applied to an entire Activity or application, rather than an individual View.

Thus, when a style is applied as a theme, every View in the Activity or application will apply each style property that it supports. For example, you can apply the same CustomFontStyle style as a theme for an Activity and then all text inside that Activity will have green monospace font.

To set a theme for all the activities of your application, open the AndroidManifest.xml file and edit the<application> tag to include the android:theme attribute with the style name. For example:

```
<application android:theme="@style/CustomFontStyle">
```

But if you want a theme applied to just one Activity in your application, then add the android:theme attribute to the <activity> tag only. For example:

```
<activity android:theme="@style/CustomFontStyle">
```

There are number of default themes defined by Android which you can use directly or i nherit them using parent attribute as follows:

```
<style name="CustomTheme" parent="android:Theme.Light">
    ...
</style>
```

Default Styles & Themes

The Android platform provides a large collection of styles and themes that you can use in your applications. You can find a reference of all available styles in the R.style class. To use the styles listed here, replace all underscores in the style name with a period. For example, you can apply the Theme_NoTitleBar theme with "@android:style/Theme.NoTitleBar". You can see the following source code for Android styles and themes:

- Android Styles (styles.xml)
- Android Themes (themes.xml)

Further Reading

- Read about the layouts which are not mentioned in this course.
- Read about the other UI widgets which are not covered in this course such as Spinner, RatingBar, ProgressBar, DatePicker, TimePicker, TextSwitcher, ImageSwitcher, AlertDialog, AdapterViewFlipper etc.
- Read about the various attributes of View and ViewGroups.