

TextView

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.

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. <ul style="list-style-type: none">• 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:maxLength 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", "#rrggb", or "#aarrggb".
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", "#rrggb", or "#aarrggb".
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 ' '. <ul style="list-style-type: none">• 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 ' '. <ul style="list-style-type: none">• normal - 0• sans - 1• serif - 2• monospace - 3

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.

EditText Attributes

Following are the important attributes related to EditText control.

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.

Android Toast Example

Android Toast can be used to display information for the short period of time. A toast contains message to be displayed quickly and disappears after sometime.

The android.widget.Toast class is the subclass of java.lang.Object class.

Toast class

Constants of Toast class

There are only 2 constants of Toast class which are given below.

Constant	Description
public static final int LENGTH_LONG	displays view for the long duration of time.
public static final int LENGTH_SHORT	displays view for the short duration of time.

Methods of Toast class

The widely used methods of Toast class are given below.

Method	Description
--------	-------------

<code>public static Toast makeText(Context context, CharSequence text, int duration)</code>	makes the toast containing text and duration.
<code>public void show()</code>	displays toast.
<code>public void setMargin (float horizontalMargin, float verticalMargin)</code>	changes the horizontal and vertical margin difference.

Full code of activity class displaying Toast

```

package com.example.demo;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity2 extends AppCompatActivity {
    EditText etext2;
    TextView txt1,txt2;
    EditText etext1;
    Button btn;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main2);
        etext1 = (EditText) findViewById(R.id.editText1);
        etext2=(EditText) findViewById(R.id.editText2);
        btn = (Button) findViewById(R.id.button2);
        txt1=(TextView) findViewById(R.id.text1);
        txt2=(TextView) findViewById(R.id.text2);

        sum();
    }

    public void sum(){

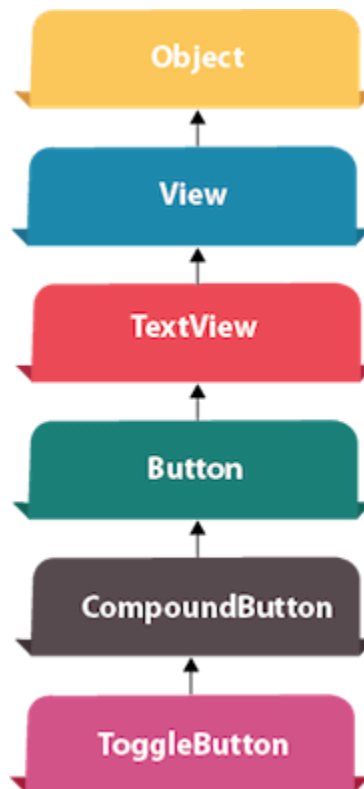
        //convert value into int
        int x=Integer.parseInt(etext1.getText().toString());
        int y=Integer.parseInt(etext2.getText().toString());

        //sum these two numbers
        int z=x+y;
    }
}

```

```
//display this text to TextView  
Toast.makeText(getApplicationContext(),z,Toast.LENGTH_SHORT).show();  
    }  
}
```

Android ToggleButton



Android Toggle Button can be used to display checked/unchecked (On/Off) state on the button.

It is beneficial if user have to change the setting between two states. It can be used to On/Off Sound, Wifi, Bluetooth etc.

Since Android 4.0, there is another type of toggle button called *switch* that provides slider control.

Android ToggleButton and Switch both are the subclasses of CompoundButton class.

XML Attributes of ToggleButton class

The 3 XML attributes of ToggleButton class.

XML Attribute	Description
android:disabledAlpha	The alpha to apply to the indicator when disabled.
android:textOff	The text for the button when it is not checked.
android:textOn	The text for the button when it is checked.

Methods of ToggleButton class

The widely used methods of ToggleButton class are given below.

Method	Description
CharSequence getTextOff()	Returns the text when button is not in the checked state.
CharSequence getTextOn()	Returns the text for when button is in the checked state.
void setChecked(boolean checked)	Changes the checked state of this button.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <ToggleButton
        android:id="@+id/toggleButton1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="ToggleButton"
```



```

        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.145"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.163" />

<ToggleButton
    android:id="@+id/toggleButton2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="ToggleButton"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.746"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.162" />

<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Button"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.463"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.301" />
</androidx.constraintlayout.widget.ConstraintLayout>

```

File: MainActivity.java

```

package com.example.myapplication;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;
import android.widget.ToggleButton;

public class MainActivity extends AppCompatActivity {
    ToggleButton tbutton1, tbutton2;
    Button btn1;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        tbutton1=(ToggleButton) findViewById(R.id.toggleButton1);
        tbutton2=(ToggleButton) findViewById(R.id.toggleButton2);
        btn1=(Button) findViewById(R.id.button);
        btn1.setOnClickListener(new View.OnClickListener() {

```

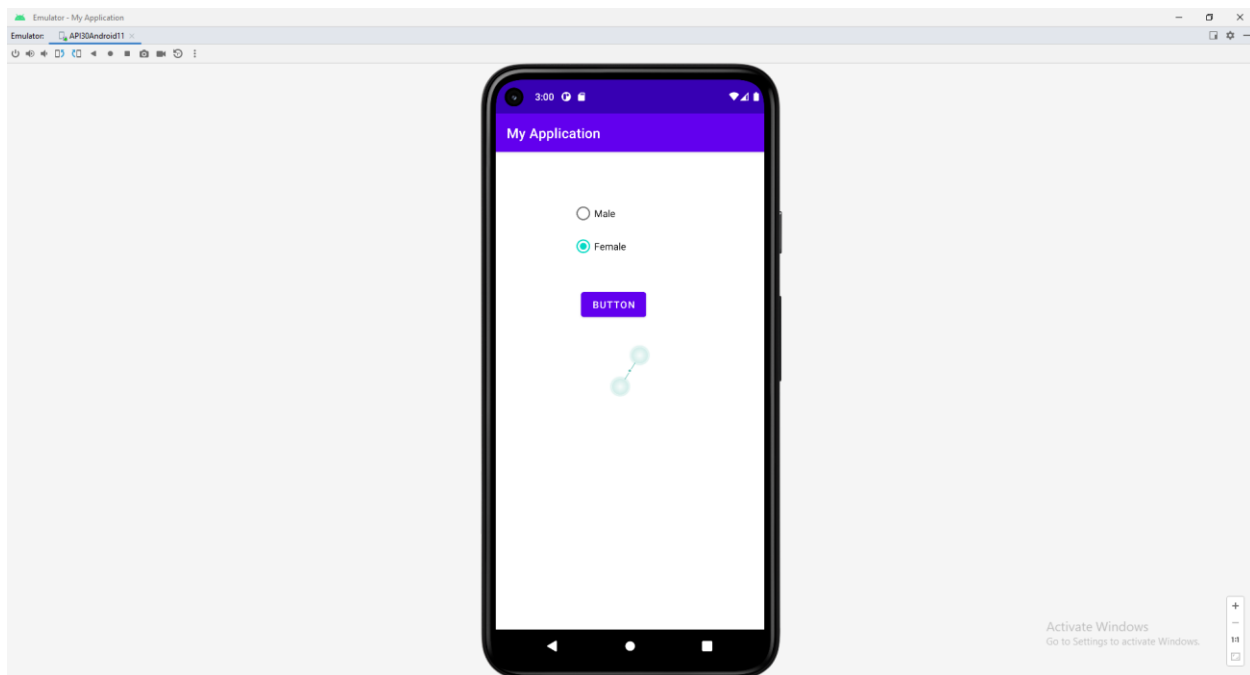
```

@Override
public void onClick(View view) {
    StringBuilder str=new StringBuilder();
    str.append("Toggle Button").append(tbutton1.getText());
    str.append("\nToggleButton2").append(tbutton2.getText());
    Toast.makeText(MainActivity.this, str,
    Toast.LENGTH_SHORT).show();
}
});
}
}

```

Android RadioButton

RadioButton can be understood as a type of two-states button. Here, the two-states means that it can either be checked or unchecked. With a single click, we can convert an unchecked radio button to a checked radio button. The user cannot mark a checked radio button to unchecked. It is generally used with *RadioGroup*. Several radio buttons are present in a RadioGroup. When one radio button is marked as checked, all other radio buttons are made unchecked.



```

package com.example.myapplication;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;

```

```

import android.widget.Button;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
    RadioGroup rg;
    Button btn;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        rg=(RadioGroup) findViewById(R.id.rgroup);
        btn=(Button) findViewById(R.id.button);
        rg.setOnCheckedChangeListener(new
RadioGroup.OnCheckedChangeListener() {
            @Override
            public void onCheckedChanged(RadioGroup radioGroup, int i) {
                RadioButton rb=(RadioButton) rg.findViewById(i);
            }
        });

        btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                int select=rg.getCheckedRadioButtonId();
                if(select!=-1){

                    Toast.makeText(MainActivity.this, "Nothing Selecteced",
Toast.LENGTH_SHORT).show();
                }
                else {

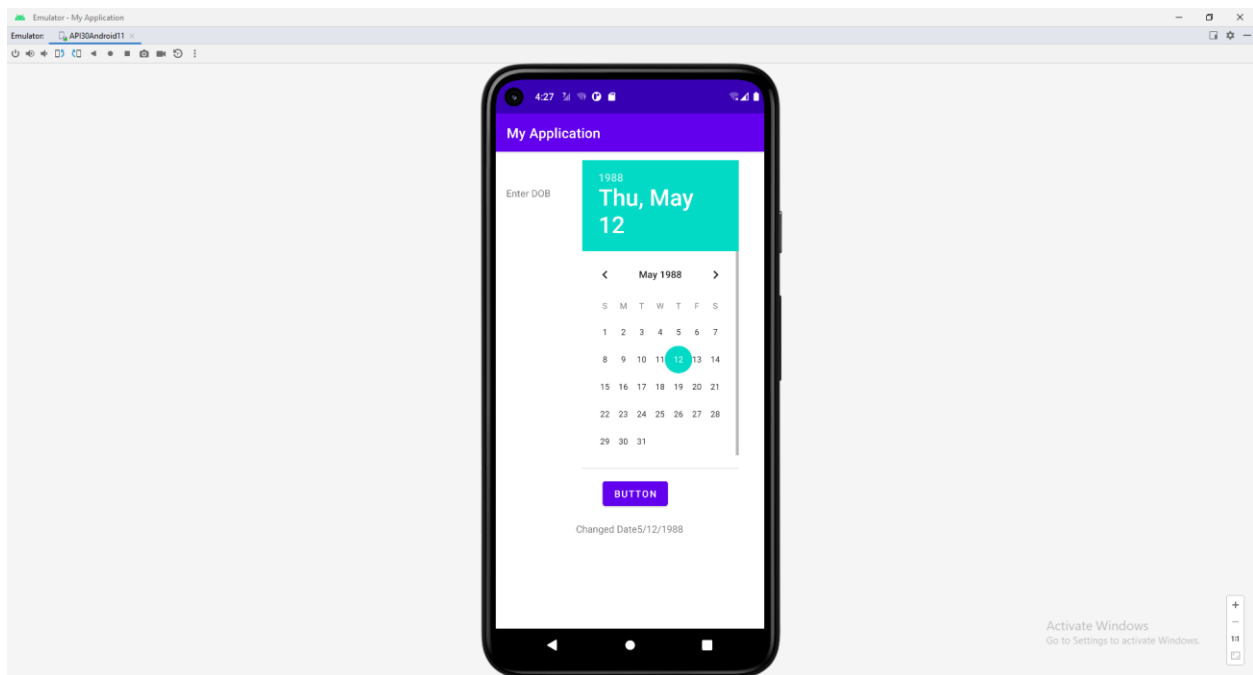
                    RadioButton rb=(RadioButton) rg.findViewById(select);
                    Toast.makeText(MainActivity.this, rb.getText(),
Toast.LENGTH_SHORT).show();
                }
            }
        });
    }
}

```

Android DatePicker –

Android DatePicker is a widget to select date. It allows you to select date by day, month and year. Like DatePicker, android also provides TimePicker to select time.

The `android.widget.DatePicker` is the subclass of `FrameLayout` class.



Activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/txt1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Enter DOB"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.047"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.075" />
```

```

<DatePicker
    android:id="@+id/date1"
    android:layout_width="229dp"
    android:layout_height="451dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.77"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.051"
    tools:ignore="InvalidId" />

<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Button"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.526"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.733" />

<TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="TextView"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.498"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.799" />
</androidx.constraintlayout.widget.ConstraintLayout>

```

Main_activity.java

```

{
    TextView txt1,txt2;
    DatePicker dt;
    Button btn;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        txt1=(TextView) findViewById(R.id.txt1);
        txt2=(TextView) findViewById(R.id.textView);
        dt=(DatePicker) findViewById(R.id.date1);
        btn=(Button) findViewById(R.id.button);
        txt2.setText("Current Date"+getCurrentDate());
    }
}

```

```

        btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                txt2.setText("Changed Date"+getCurrentDate());
            }
        });

    }

    private String getCurrentDate() {
        StringBuilder str=new StringBuilder();

        str.append((dt.getMonth()+1)+"/");
        str.append((dt.getDayOfMonth()+"/");
        str.append((dt.getYear()));
        return str.toString();
    }
}

```

Android TimePicker

Android TimePicker widget is used to select date. It allows you to select time by hour and minute. You cannot select time by seconds.

The android.widget.TimePicker is the subclass of FrameLayout class.

activity_main.xml

```

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/txt1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Check Time"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.047"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.117" />

```

```

<TextView
    android:id="@+id/text2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Current Time"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.437"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.608" />

<TimePicker android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:id="@+id/tp"
    tools:ignore="MissingConstraints" />

<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Button"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.454"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.549" />

</androidx.constraintlayout.widget.ConstraintLayout>

```

Main_Activity.java

```

package com.example.myapplication;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.TimePicker;

public class MainActivity extends AppCompatActivity {
    TextView txt1,txt2;
    TimePicker tp;
    Button btn;

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

        txt1=(TextView) findViewById(R.id.txt1);
    }
}

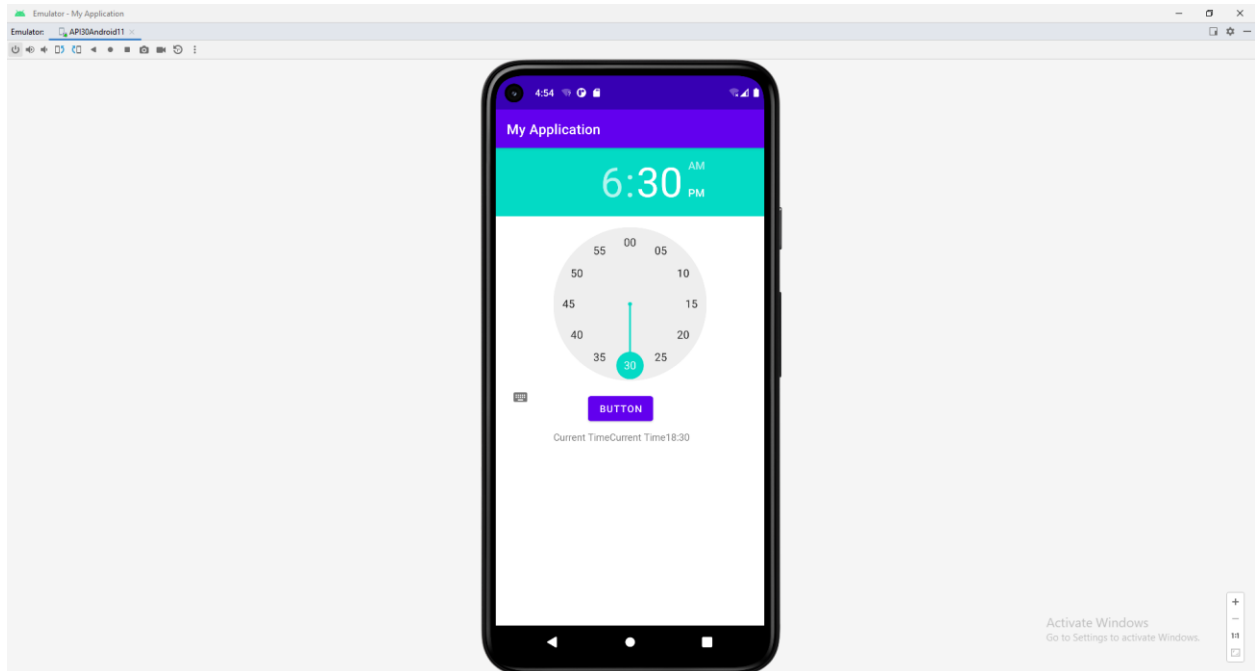
```

```

txt2=(TextView) findViewById(R.id.txt2);
tp=(TimePicker) findViewById(R.id.tp);
btn=(Button) findViewById(R.id.button);
txt2.setText("Current Time"+getCurrentTime());
btn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        txt2.setText("Current Time"+getCurrentTime());
    }
});
}

private String getCurrentTime() {
    String currenttime="Current Time"+tp.getHour()+":"+tp.getMinute();
    return currenttime;
}
}

```



Using Indicators to Display Data to Users

Android ProgressBar Example



We can display the **android progress bar** dialog box to display the status of work being done e.g. downloading file, analyzing status of work etc.

In this example, we are displaying the progress dialog for dummy file download operation.

Here we are using **android.app.ProgressDialog** class to show the progress bar. Android ProgressDialog is the subclass of AlertDialog class.

The **ProgressDialog** class provides methods to work on progress bar like `setProgress()`, `setMessage()`, `setProgressStyle()`, `setMax()`, `show()` etc. The progress range of Progress Dialog is 0 to 10000.

Let's see a simple example to display progress bar in android.

1. `ProgressDialog progressBar = new ProgressDialog(this);`
2. `progressBar.setCancelable(true);` //you can cancel it by pressing back button
3. `progressBar.setMessage("File downloading ...");`
4. `progressBar.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);`
5. `progressBar.setProgress(0);` //initially progress is 0
6. `progressBar.setMax(100);` //sets the maximum value 100

7. progressBar.show();//displays the progress bar

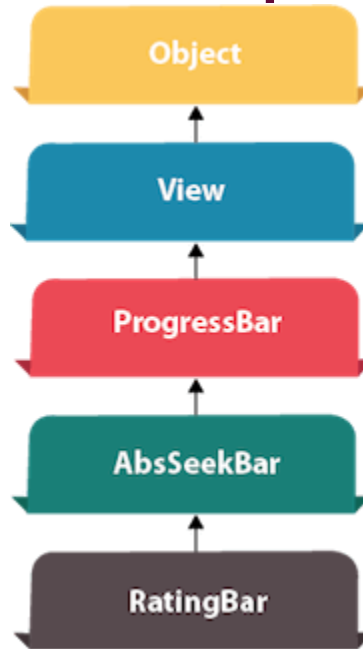
```
<ProgressBar
    android:id="@+id/progressBar1"
    style="?android:attr/progressBarStyleHorizontal"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/textView1"
    android:layout_below="@+id/textView1"
    android:layout_marginTop="20dp"
    android:max="100"
    android:minHeight="50dp"
    android:minWidth="300dp"
    android:progress="1" />
```

Source code in Java

```
final ProgressBar pb1 = (ProgressBar) findViewById(R.id.progressBar1);
pb1.setMax(100);
//pb1.setProgress(70);
Button bPlus = (Button) findViewById(R.id.btnPlus);
Button bMinus = (Button) findViewById(R.id.btnMinus);
bPlus.setOnClickListener(new Button.OnClickListener() {
    @Override
    public void onClick(View view) {
        pb1.setProgress(pb1.getProgress()+10);
    }
});

bMinus.setOnClickListener(new Button.OnClickListener() {
    @Override
    public void onClick(View view) {
        pb1.setProgress(pb1.getProgress()-10);
    }
});
```

Android RatingBar Example



Android RatingBar can be used to get the rating from the user. The Rating returns a floating-point number. It may be 2.0, 3.5, 4.0 etc.

Android RatingBar displays the rating in stars. Android RatingBar is the subclass of AbsSeekBar class.

The **getRating()** method of android RatingBar class returns the rating number.

Android RatingBar Example

Let's see the simple example of rating bar in android.

Drag the RatingBar and Button from the palette, now the activity_main.xml file will like this:

File: activity_main.xml

1. `<?xml version="1.0" encoding="utf-8"?>`
2. `<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"`
3. `xmlns:app="http://schemas.android.com/apk/res-auto"`
4. `xmlns:tools="http://schemas.android.com/tools"`
5. `android:layout_width="match_parent"`

```
6.     android:layout_height="match_parent"
7.     tools:context="example.javatpoint.com.ratingbar.MainActivity">
8.
9.     <Button
10.         android:layout_width="wrap_content"
11.         android:layout_height="wrap_content"
12.         android:text="submit"
13.         android:id="@+id/button"
14.         app:layout_constraintBottom_toBottomOf="parent"
15.         app:layout_constraintLeft_toLeftOf="parent"
16.         app:layout_constraintRight_toRightOf="parent"
17.         app:layout_constraintTop_toTopOf="parent"
18.         app:layout_constraintVertical_bias="0.615" />
19.
20.     <RatingBar
21.         android:id="@+id/ratingBar"
22.         android:layout_width="wrap_content"
23.         android:layout_height="wrap_content"
24.         android:layout_marginLeft="72dp"
25.         android:layout_marginTop="60dp"
26.         app:layout_constraintStart_toStartOf="parent"
27.         app:layout_constraintTop_toTopOf="parent" />
28.
29. </android.support.constraint.ConstraintLayout>
```

Activity class

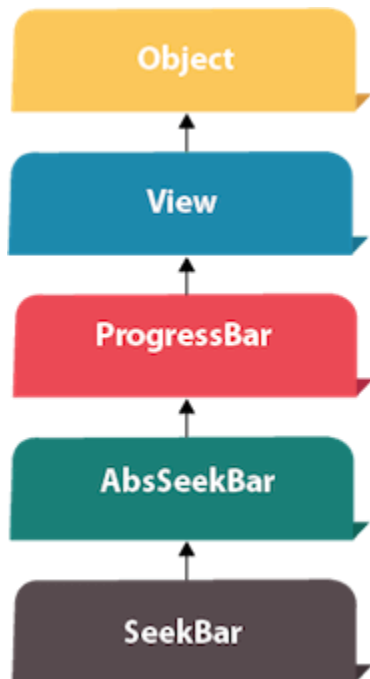
Let's write the code to display the rating of the user.

File: MainActivity.java

```
1. package example.javatpoint.com.ratingbar;
2.
3. import android.support.v7.app.AppCompatActivity;
4. import android.os.Bundle;
```

```
5. import android.view.View;
6. import android.widget.Button;
7. import android.widget.RatingBar;
8. import android.widget.Toast;
9.
10. public class MainActivity extends AppCompatActivity {
11.     RatingBar ratingbar;
12.     Button button;
13.     @Override
14.     protected void onCreate(Bundle savedInstanceState) {
15.         super.onCreate(savedInstanceState);
16.         setContentView(R.layout.activity_main);
17.         addListenerOnButtonClick();
18.     }
19.     public void addListenerOnButtonClick(){
20.         ratingbar=(RatingBar)findViewById(R.id.ratingBar);
21.         button=(Button)findViewById(R.id.button);
22.         //Performing action on Button Click
23.         button.setOnClickListener(new View.OnClickListener(){
24.
25.             @Override
26.             public void onClick(View arg0) {
27.                 //Getting the rating and displaying it on the toast
28.                 String rating=String.valueOf(ratingbar.getRating());
29.                 Toast.makeText(getApplicationContext(), rating, Toast.LENGTH_LONG).show();
30.             }
31.
32.         });
33.     }
34. }
```

Android SeekBar Example



Android SeekBar is a kind of ProgressBar with draggable thumb. The end user can drag the thumb left and right to move the progress of song, file download etc.

The SeekBar.OnSeekBarChangeListener interface provides methods to perform even handling for seek bar.

Android SeekBar and RatingBar classes are the sub classes of AbsSeekBar.

Android SeekBar Example

activity_main.xml

Drag the seek bar from the palette, now activity_main.xml will look like this:

File: activity_main.xml

1. `<?xml version="1.0" encoding="utf-8"?>`
2. `<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"`
3. `xmlns:app="http://schemas.android.com/apk/res-auto"`

```

4.   xmlns:tools="http://schemas.android.com/tools"
5.   android:layout_width="match_parent"
6.   android:layout_height="match_parent"
7.   tools:context="example.javatpoint.com.seekbar.MainActivity">
8.
9.
10.  <SeekBar
11.      android:id="@+id/seekBar"
12.      android:layout_width="match_parent"
13.      android:layout_height="wrap_content"
14.      android:layout_marginEnd="8dp"
15.      android:layout_marginStart="8dp"
16.      android:layout_marginTop="372dp"
17.      app:layout_constraintEnd_toEndOf="parent"
18.      app:layout_constraintStart_toStartOf="parent"
19.      app:layout_constraintTop_toTopOf="parent" />
20. </android.support.constraint.ConstraintLayout>

```

Activity class

Let's see the Activity class displaying seek bar and performing event handling.

File: MainActivity.java

```

1.  package example.javatpoint.com.seekbar;
2.
3.  import android.support.v7.app.AppCompatActivity;
4.  import android.os.Bundle;
5.  import android.widget.SeekBar;
6.  import android.widget.Toast;
7.
8.  public class MainActivity extends AppCompatActivity {
9.      SeekBar seekBar;
10.     @Override
11.     protected void onCreate(Bundle savedInstanceState) {

```

```

12.     super.onCreate(savedInstanceState);
13.     setContentView(R.layout.activity_main);
14.
15.     seekBar=(SeekBar)findViewById(R.id.seekBar);
16.     seekBar.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {
17.         @Override
18.         public void onProgressChanged(SeekBar seekBar, int progress,
19.             boolean fromUser) {
20.             Toast.makeText(getApplicationContext(),"seekbar progress: "+progress, Toast.LENGTH
21.                 _SHORT).show();
22.         }
23.         @Override
24.         public void onStartTrackingTouch(SeekBar seekBar) {
25.             Toast.makeText(getApplicationContext(),"seekbar touch started!", Toast.LENGTH_SHO
26.                 RT).show();
27.         }
28.         @Override
29.         public void onStopTrackingTouch(SeekBar seekBar) {
30.             Toast.makeText(getApplicationContext(),"seekbar touch stopped!", Toast.LENGTH_SH
31.                 ORT).show();
32.         }
33.     });
34. }

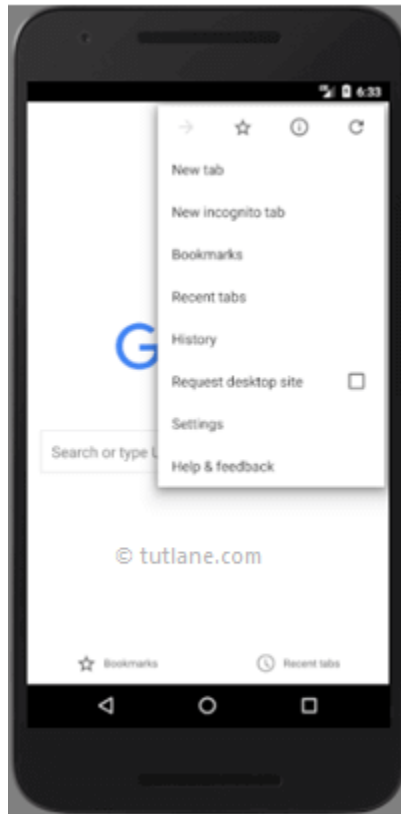
```

Android Menus

In android, **Menu** is a part of the user interface (UI) component which is used to handle some common functionality around the application. By using Menus in our applications, we can provide better and consistent user experience throughout the application.

We can use Menu APIs to represent user actions and other options in our android application activities.

Following is the pictorial representation of using menus in the android application.



In android, we can define a Menu in separate XML file and use that file in our [activities](#) or [fragments](#) based on our requirements.

Define an Android Menu in XML File

For all menu types, Android provides a standard XML format to define menu items. Instead of building a menu in our [activity's](#) code, we should define a menu and all its items in an XML menu resource and load menu resource as a Menu object in our [activity](#) or [fragment](#).

In android, to define menu, we need to create a new folder **menu** inside of our project resource directory (**res/menu/**) and add a new XML file to build the menu with the following elements.

Element	Description
<menu>	It's a root element to define a Menu in XML file and it will hold one or more and elements.
<item>	It is used to create a menu item and it represents a single item on the menu. This element may contain a <menu> element in order to create a submenu.
<group>	It's an optional and invisible for <item> elements. It is used to categorize the menu items so they share as active state and visibility.

Following is the example of defining a menu in an XML file (**menu_example.xml**).

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:id="@+id/mail"
        android:icon="@drawable/ic_mail"
        android:title="@string/mail" />
  <item android:id="@+id/upload"
        android:icon="@drawable/ic_upload"
        android:title="@string/upload"
        android:showAsAction="ifRoom" />
  <item android:id="@+id/share"
        android:icon="@drawable/ic_share"
        android:title="@string/share" />
</menu>
```

The **<item>** element in **menu** supports different type of attributes to define item's behaviour and appearance. Following are the some of commonly used **<item>** attributes in android applications.

Attribute	Description
android: id	It is used to uniquely identify an element in the application.
android:icon	It is used to set the item's icon from drawable folder.
android: title	It is used to set the item's title

Attribute	Description
android:showAsAction	It is used to specify how the item should appear as an action item in the app bar.

In case if we want to add **submenu** in **menu** item, then we need to add a **<menu>** element as the child of an **<item>**. Following is the example of defining a submenu in menu item.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:id="@+id/file"
        android:title="@string/file" >
    <!-- "file" submenu -->
    <menu>
        <item android:id="@+id/create_new"
              android:title="@string/create_new" />
        <item android:id="@+id/open"
              android:title="@string/open" />
    </menu>
  </item>
</menu>
```

Load Android Menu from an Activity

Once we are done with creation of menu, we need to load the menu resource from our [activity](#) using **MenuInflater.inflate()** like as shown below.

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo
menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.menu_example, menu);
}
```

If you observe above code we are calling our menu using MenuInflater.inflate() method in the form of **R.menu.menu_file_name**. Here our xml file name is **menu_example.xml** so we used file name **menu_example**.

Handle Android Menu Click Events

In android, we can handle a menu item click events using **ItemSelected()** event based on the menu type. Following is the example of handling a context menu item click event using **onContextItemSelected()**.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.mail:
            // do something
            return true;
        case R.id.share:
            // do something
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

If you observe above code, the **getItemId()** method will get the id of selected menu item based on that we can perform our actions.

Android Different Types of Menus

In android, we have a three fundamental type of Menus available to define a set of options and actions in our android applications.

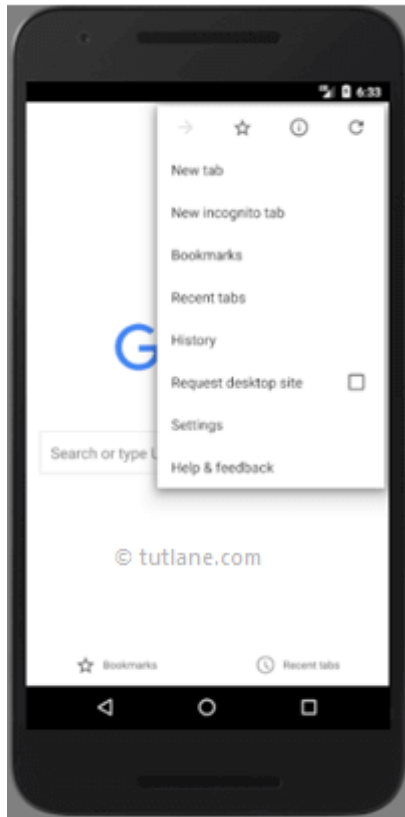
The following are the commonly used Menus in android applications.

- [Options Menu](#)
- [Context Menu](#)
- [Popup Menu](#)

Android Options Menu with Examples

In android, **Options Menu** is a primary collection of menu items for an [activity](#) and it is useful to implement actions that have a global impact on the app, such as Settings, Search, etc.

Following is the pictorial representation of using **Options Menu** in our android applications.



By using Options Menu, we can combine multiple actions and other options that are relevant to our current [activity](#). We can define items for the options menu from either our [Activity](#) or [Fragment](#) class.

In case, if we define items for the options menu in both activity or fragment, then those items will be combined and display in UI.

Create Android Options Menu in XML File

In android, to define **options menu**, we need to create a new folder **menu** inside of our project resource directory (**res/menu/**) and add a new XML (**menu_example**) file to build the menu.

Following is the example of defining a menu in an XML file (**menu_example.xml**).

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

```

        android:title="Search" />
<item android:id="@+id/upload_item"
    android:title="Upload" />
<item android:id="@+id/copy_item"
    android:title="Copy" />
<item android:id="@+id/print_item"
    android:title="Print" />
<item android:id="@+id/share_item"
    android:title="Share" />
<item android:id="@+id/bookmark_item"
    android:title="BookMark" />

</menu>

```

Load Android Options Menu from an Activity

To specify the options menu for an activity, we need to override `onCreateOptionsMenu()` method and load the defined menu resource using `MenuInflater.inflate()` like as shown below.

```

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

```

Handle Android Options Menu Click Events

In android, we can handle options menu item click events using the `onOptionsItemSelected()` event method.

Following is the example of handling a options menu item click event using `onOptionsItemSelected()`.

```

public boolean onOptionsItemSelected(@NonNull MenuItem item) {
    Toast.makeText(this, "Selected Item:"+item.getTitle(),
        Toast.LENGTH_SHORT).show();

    switch (item.getItemId()) {
        case R.id.search_item:
            return true;
        case R.id.upload_item:
            return true;
        case R.id.copy_item:
            return true;
        case R.id.print_item:
            return true;
        case R.id.share_item:
            return true;
    }
}

```

```

        case R.id.bookmark_item:
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}

```

```

package com.example.myapplication;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

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

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

    @Override
    public boolean onOptionsItemSelected(@NonNull MenuItem item) {
        Toast.makeText(this, "Selected Item:"+item.getTitle(),
            Toast.LENGTH_SHORT).show();

        switch (item.getItemId()) {
            case R.id.search_item:
                return true;
            case R.id.upload_item:
                return true;
            case R.id.copy_item:
                return true;
            case R.id.print_item:
                return true;
        }
    }
}

```

```

        case R.id.share_item:
            return true;
        case R.id.bookmark_item:
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
}

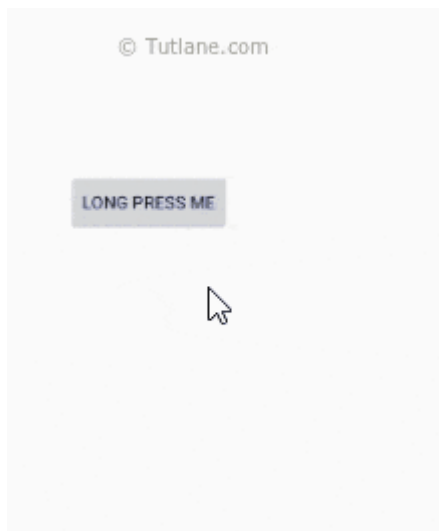
```

Android Context Menu with Examples

In android, **Context Menu** is like a floating menu and that appears when the user performs a long press or click on an element and it is useful to implement actions that affect the selected content or context frame.

The android Context Menu is more like the menu which displayed on right-click in Windows or Linux.

Following is the pictorial representation of using **Context Menu** in our android applications.



In android, the Context Menu offers actions that affect a specific item or context frame in the UI and we can provide a context menu for any [view](#). The context menu won't support any item shortcuts and item icons.

Create Android Context Menu in Activity

The [views](#) which we used to show the context menu on long-press, we need to register that [views](#) using **registerForContextMenu(View)** in our [activity](#) and we need to override **onCreateContextMenu()** in our [activity](#) or [fragment](#).

When the registered [view](#) receives a long-click event, the system calls our **onCreateContextMenu()** method. By using the **onCreateContextMenu()** method, we can create our menu items as shown below.

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

@Override

```
public void onCreateContextMenu(ContextMenu menu, View v,  
ContextMenu.ContextMenuInfo menuInfo) {  
  
    super.onCreateContextMenu(menu, v, menuInfo);  
  
    menu.setHeaderTitle("Context Menu");  
  
    menu.add(0, v.getId(), 0, "Upload");  
  
    menu.add(0, v.getId(), 0, "Search");  
  
    menu.add(0, v.getId(), 0, "Share");  
  
    menu.add(0, v.getId(), 0, "Bookmark");  
  
}
```

If you observe above code, we registered our [Button](#) control using **registerForContextMenu()** to show the context menu on button long-click and binding the Context Menu items using **onCreateContextMenu()** method.

Handle Android Context Menu Click Events

In android, we can handle a context menu item click events using the **onContextItemSelected()** method.

Following is the example of handling a context menu item click event using **onContextItemSelected()** method.

```
public boolean onContextItemSelected(@NonNull MenuItem item) {  
    Toast.makeText(this, "Selected Item"+item.getTitle(),  
        Toast.LENGTH_SHORT).show();  
  
    return true;  
}
```

Android Popup Menu with Examples

In android, **Popup Menu** displays a list of items in a modal popup window that is anchored to the [view](#). The popup menu will appear below the [view](#) if there is a room or above the view in case if there is no space and it will be closed automatically when we touch outside of the popup.

The android Popup Menu provides an overflow style menu for actions that are related to specific content.

Following is the pictorial representation of using **Popup Menu** in our android applications.



In android, the Popup Menu provides an overflow of actions that are related to specific content and the actions in the popup menu won't affect the corresponding content. The popup menu won't support any item shortcuts and item icons.

In android, the Popup menu is available with API level 11 (Android 3.0) and higher versions. If you are using Android 3.0 +, the Popup Menu won't support any item shortcuts and item icons in the menu.

Create Android Popup Menu in XML File

In android, to define the **popup menu**, we need to create a new folder **menu** inside of our project resource directory (**res/menu/**) and add a new XML (menu_example) file to build the menu.

Following is the example of defining a menu in XML file (**menu_example.xml**).

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

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

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

        android:title="Search" />

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

        android:title="Upload" />

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

        android:title="Copy" />

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

        android:title="Print" />

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

        android:title="Share" />

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

        android:title="BookMark" />

</menu>
```

Now open an **activity_main.xml** file from **\res\layout** path and write the code like as shown below

activity_main.xml

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

<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"

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

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

    android:layout_width="match_parent"

    android:layout_height="match_parent"

    tools:context=".MainActivity">

    <TextView

        android:id="@+id/textView"

        android:layout_width="wrap_content"

        android:layout_height="wrap_content"

        android:text="Hello World!"

        app:layout_constraintBottom_toBottomOf="parent"

        app:layout_constraintEnd_toEndOf="parent"

        app:layout_constraintStart_toStartOf="parent"

        app:layout_constraintTop_toTopOf="parent" />

    <Button

        android:id="@+id/button"

        android:layout_width="wrap_content"

        android:layout_height="wrap_content"

        android:layout_marginStart="168dp"

        android:text="Button"
```

```
app:layout_constraintBottom_toBottomOf="parent"

app:layout_constraintStart_toStartOf="parent"

app:layout_constraintTop_toTopOf="parent"

app:layout_constraintVertical_bias="0.377" />
```

```
</androidx.constraintlayout.widget.ConstraintLayout>
```

MainActivity.java

```
package com.example.myapplication;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.PopupMenu;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity implements
PopupMenu.OnMenuItemClickListener {

    @Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_main);

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

        btn.setOnClickListener(new View.OnClickListener() {

            @Override
```

```

    public void onClick(View view) {

        PopupMenu popup = new PopupMenu(MainActivity.this, view);

        popup.setOnMenuItemClickListener(MainActivity.this);

        popup.inflate(R.menu.menu_example);

        popup.show();

    }

});

}

@Override

public boolean onOptionsItemSelected(MenuItem menuItem) {

    Toast.makeText(this, "Selected Item:" + menuItem.getTitle(),
Toast.LENGTH_SHORT).show();

    switch (menuItem.getItemId()) {

        case R.id.search_item:

            // do your code

            return true;

        case R.id.upload_item:

            // do your code

            return true;

        case R.id.copy_item:

            // do your code

            return true;

        case R.id.print_item:

            // do your code

            return true;

        case R.id.share_item:

            // do your code

```

```

        return true;

    case R.id.bookmark_item:

        // do your code

        return true;

    default:

        return false;

    }

}

}

```

ImageView in Android

ImageView class is used to display any kind of image resource in the android application either it can be **android.graphics.Bitmap** or **android.graphics.drawable.Drawable** (it is a general abstraction for anything that can be drawn in Android). **ImageView** class or **android.widget.ImageView** inherits the **android.view.View** class which is the subclass of Kotlin. **AnyClass.Application** of **ImageView** is also in applying tints to an image in order to reuse a drawable resource and create overlays on background images. Moreover, **ImageView** is also used to control the size and movement of an image.

XML Attributes of ImageView

XML Attribute	Description
android:id	To uniquely identify an image view
android:src/app:srcCompat	To add the file path of the inserted image
android:background	To provide a background color to the inserted image

XML Attribute	Description
android:layout_width	To set the width of the image
android:layout_height	To set the height of the image
android:padding	To add padding to the image from the left, right, top, or bottom of the view
android:scaleType	To re-size the image or to move it in order to fix its size

Step by Step Implementation

Step 1: Create a New Project

Step 2: Working with the activity_main.xml File

Go to the **activity_main.xml** File and refer to the following code. Below is the code for the **activity_main.xml** File.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.234" />

    <ImageView
        android:id="@+id/imageView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
```



```
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:srcCompat="@drawable/logo" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Step 4: Working with the MainActivity File

```
package com.example.myapplication;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

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

Android - UI Layouts

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

The **ViewGroup** is a subclass of **View** and provides invisible container that hold other Views or other ViewGroups and define their layout properties.

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

Layout params

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

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

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

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

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

</LinearLayout>
```

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

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

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:

Sr.No	Attribute & Description
1	android:id This is the ID which uniquely identifies the view.
2	android:layout_width This is the width of the layout.
3	android:layout_height This is the height of the layout
4	android:layout_marginTop This is the extra space on the top side of the layout.
5	android:layout_marginBottom This is the extra space on the bottom side of the layout.
6	android:layout_marginLeft This is the extra space on the left side of the layout.
7	android:layout_marginRight This is the extra space on the right side of the layout.
8	android:layout_gravity This specifies how child Views are positioned.
9	android:layout_weight This specifies how much of the extra space in the layout should be allocated to the View.
10	android:layout_x This specifies the x-coordinate of the layout.
11	android:layout_y This specifies the y-coordinate of the layout.
12	android:layout_width This is the width of the layout.

- 13 **android:paddingLeft**
This is the left padding filled for the layout.
- 14 **android:paddingRight**
This is the right padding filled for the layout.
- 15 **android:paddingTop**
This is the top padding filled for the layout.
- 16 **android:paddingBottom**
This is the bottom padding filled for the layout.

Android Linear Layout

Android LinearLayout is a view group that aligns all children in either vertically or horizontally.

Linear Layout

LinearLayout Attributes

Following are the important attributes specific to LinearLayout –

Sr.No	Attribute & Description
1	android:id This is the ID which uniquely identifies the layout.
2	android:baselineAligned This must be a boolean value, either "true" or "false" and prevents the layout from aligning its children's baselines.
3	android:baselineAlignedChildIndex When a linear layout is part of another layout that is baseline aligned, it can specify which of its children to baseline align.
4	android:divider This is drawable to use as a vertical divider between buttons. You use a color value, in the form of "#rgb", "#argb", "#rrggbb", or "#aarrggbb".
5	android:gravity This specifies how an object should position its content, on both the X and Y axes. Possible values are top, bottom, left, right, center, center_vertical, center_horizontal etc.

- 6 **android:orientation**
This specifies the direction of arrangement and you will use "horizontal" for a row, "vertical" for a column. The default is horizontal.
- 7 **android:weightSum**
Sum up of child weight

Example

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

Step	Description
1	You will use Android Studio to create an Android application and name it as <i>Demo</i> under a package <i>com.example.demo</i> as explained in the <i>Hello World Example</i> chapter.
2	Modify the default content of <i>res/layout/activity_main.xml</i> file to include few buttons in linear layout.
3	No need to change string Constants.Android studio takes care of default strings
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;

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


```
        setContentView(R.layout.activity_main);  
    }  
}
```

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

```
<?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/btnStartService"  
        android:layout_width="270dp"  
        android:layout_height="wrap_content"  
        android:text="start_service"/>  
  
    <Button android:id="@+id/btnPauseService"  
        android:layout_width="270dp"  
        android:layout_height="wrap_content"  
        android:text="pause_service"/>  
  
    <Button android:id="@+id/btnStopService"  
        android:layout_width="270dp"  
        android:layout_height="wrap_content"  
        android:text="stop_service"/>  
  
</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">HelloWorld</string>  
    <string name="action_settings">Settings</string>  
</resources>
```

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

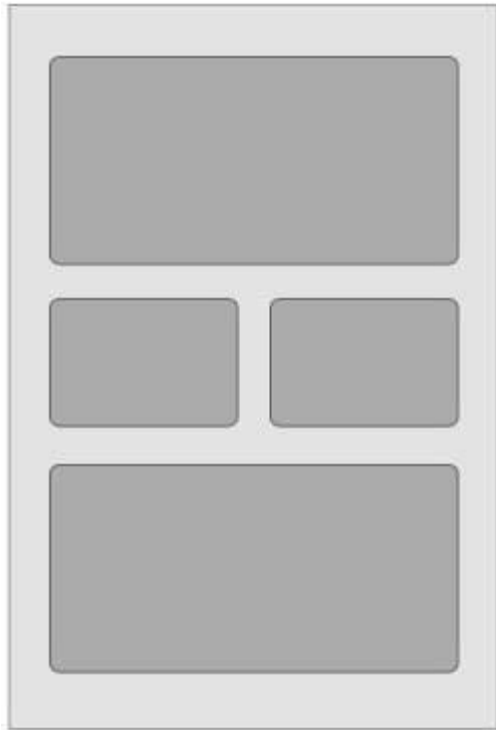


Now let's change the orientation of Layout as **android:orientation="horizontal"** and try to run the same application, it will give following screen –



Android Relative Layout

Android RelativeLayout enables you to specify how child views are positioned relative to each other. The position of each view can be specified as relative to sibling elements or relative to the parent.



Relative Layout

RelativeLayout Attributes

Following are the important attributes specific to RelativeLayout –

Sr.No.	Attribute & Description
1	android:id This is the ID which uniquely identifies the layout.
2	android:gravity This specifies how an object should position its content, on both the X and Y axes. Possible values are top, bottom, left, right, center, center_vertical, center_horizontal etc.
3	android:ignoreGravity This indicates what view should not be affected by gravity.

Using RelativeLayout, you can align two elements by right border, or make one below another, centered in the screen, centered left, and so on. By default, all child views are drawn at the top-left of the

layout, so you must define the position of each view using the various layout properties available from **RelativeLayout.LayoutParams** and few of the important attributes are given below –

Sr.No.	Attribute & Description
1	android:layout_above Positions the bottom edge of this view above the given anchor view ID and must be a reference to another resource, in the form "@+[package:]type:name"
2	android:layout_alignBottom Makes the bottom edge of this view match the bottom edge of the given anchor view ID and must be a reference to another resource, in the form "@+[package:]type:name".
3	android:layout_alignLeft Makes the left edge of this view match the left edge of the given anchor view ID and must be a reference to another resource, in the form "@+[package:]type:name".
4	android:layout_alignParentBottom If true, makes the bottom edge of this view match the bottom edge of the parent. Must be a boolean value, either "true" or "false".
5	android:layout_alignParentEnd If true, makes the end edge of this view match the end edge of the parent. Must be a boolean value, either "true" or "false".
6	android:layout_alignParentLeft If true, makes the left edge of this view match the left edge of the parent. Must be a boolean value, either "true" or "false".
7	android:layout_alignParentRight If true, makes the right edge of this view match the right edge of the parent. Must be a boolean value, either "true" or "false".
8	android:layout_alignParentStart If true, makes the start edge of this view match the start edge of the parent. Must be a boolean value, either "true" or "false".
9	android:layout_alignParentTop If true, makes the top edge of this view match the top edge of the parent. Must be a boolean value, either "true" or "false".

- 10 **android:layout_alignRight**
Makes the right edge of this view match the right edge of the given anchor view ID and must be a reference to another resource, in the form "@+[package:]type:name".
- 11 **android:layout_alignStart**
Makes the start edge of this view match the start edge of the given anchor view ID and must be a reference to another resource, in the form "@+[package:]type:name".
- 12 **android:layout_alignTop**
Makes the top edge of this view match the top edge of the given anchor view ID and must be a reference to another resource, in the form "@+[package:]type:name".
- 13 **android:layout_below**
Positions the top edge of this view below the given anchor view ID and must be a reference to another resource, in the form "@+[package:]type:name".
- 14 **android:layout_centerHorizontal**
If true, centers this child horizontally within its parent. Must be a boolean value, either "true" or "false".
- 15 **android:layout_centerInParent**
If true, centers this child horizontally and vertically within its parent. Must be a boolean value, either "true" or "false".
- 16 **android:layout_centerVertical**
If true, centers this child vertically within its parent. Must be a boolean value, either "true" or "false".
- 17 **android:layout_toEndOf**
Positions the start edge of this view to the end of the given anchor view ID and must be a reference to another resource, in the form "@+[package:]type:name".
- 18 **android:layout_toLeftOf**
Positions the right edge of this view to the left of the given anchor view ID and must be a reference to another resource, in the form "@+[package:]type:name".
- 19 **android:layout_toRightOf**
Positions the left edge of this view to the right of the given anchor view ID and must be a reference to another resource, in the form "@+[package:]type:name".
- 20 **android:layout_toStartOf**
Positions the end edge of this view to the start of the given anchor view ID and must be a reference to another resource, in the form "@+[package:]type:name".

Example

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

Step	Description
1	You will use Android Studio IDE to create an Android application and name it as <i>demo</i> under a package <i>com.example.demo</i> as explained in the <i>Hello World Example</i> chapter.
2	Modify the default content of <i>res/layout/activity_main.xml</i> file to include few widgets in Relative layout.
3	Define required constants in <i>res/values/strings.xml</i> 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/com.example.demo/MainActivity.java**. This file can include each of the fundamental lifecycle methods.

```
package com.example.demo;

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

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

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

```

<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp" >

    <EditText
        android:id="@+id/name"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="@string/reminder" />

    <LinearLayout
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:layout_alignParentStart="true"
        android:layout_below="@+id/name">

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

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="New Button"
            android:id="@+id/button2" />

    </LinearLayout>

</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="action_settings">Settings</string>
    <string name="reminder">Enter your name</string>
</resources>
```

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



Android Frame Layout

Frame Layout is designed to block out an area on the screen to display a single item. Generally, FrameLayout should be used to

hold a single child view, because it can be difficult to organize child views in a way that's scalable to different screen sizes without the children overlapping each other.

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

FrameLayout Attributes

Following are the important attributes specific to `FrameLayout` –

Sr.No	Attribute & Description
1	<code>android:id</code> This is the ID which uniquely identifies the layout.
2	<code>android:foreground</code> This defines the drawable to draw over the content and possible values may be a color value, in the form of " <code>#rgb</code> ", " <code>#argb</code> ", " <code>#rrggb</code> ", or " <code>#aarrggb</code> ".
3	<code>android:foregroundGravity</code> Defines the gravity to apply to the foreground drawable. The gravity defaults to fill. Possible values are top, bottom, left, right, center, center_vertical, center_horizontal etc.
4	<code>android:measureAllChildren</code> Determines whether to measure all children or just those in the <code>VISIBLE</code> or <code>INVISIBLE</code> state when measuring. Defaults to false.

Example

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

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

- 1 You will use Android studio IDE to create an Android application and name it as *demo* under a package *com.example.demo* as explained in the *Hello World Example* chapter.
- 2 Modify the default content of *res/layout/activity_main.xml* file to include few widgets in frame layout.
- 3 No need to change string.xml, android takes care default constants
- 4 Run the application to launch 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.myapplication;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

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

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

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Best college in Maharashtra"
        android:layout_marginLeft="100dp"
        android:layout_marginTop="30dp"
        android:gravity="center"/>
```



```
<TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="President-Baburaoji Gaikwad"
    android:layout_gravity="center_horizontal"
    android:layout_marginTop="100dp"/>

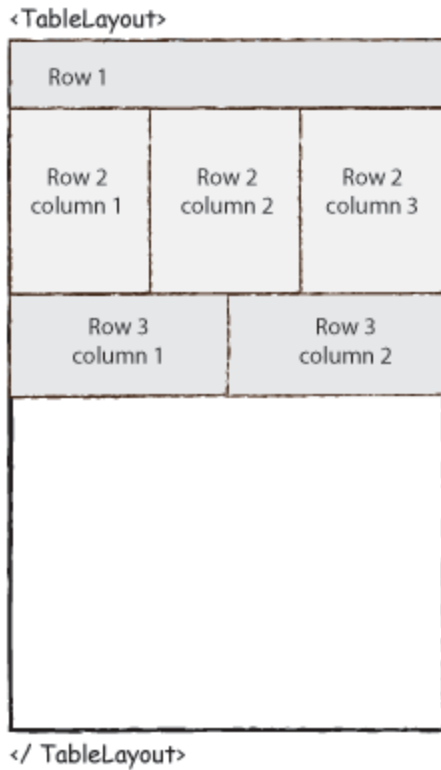
<ImageView
    android:id="@+id/imageView3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:srcCompat="@drawable/collegenamechange" />

</FrameLayout>
```

Android Table Layout

Android TableLayout going to be arranged groups of views into rows and columns. You will use the <TableRow> element to build a row in the table. Each row has zero or more cells; each cell can hold one View object.

TableLayout containers do not display border lines for their rows, columns, or cells.



TableLayout Attributes

Following are the important attributes specific to TableLayout –

Sr.No.	Attribute & Description
1	android:id This is the ID which uniquely identifies the layout.
2	android:collapseColumns This specifies the zero-based index of the columns to collapse. The column indices must be separated by a comma: 1, 2, 5.
3	android:shrinkColumns The zero-based index of the columns to shrink. The column indices must be separated by a comma: 1, 2, 5.
4	android:stretchColumns The zero-based index of the columns to stretch. The column indices must be separated by a comma: 1, 2, 5.

Example

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

Step	Description
1	You will use Android Studio IDE to create an Android application and name it as <i>demo</i> under a package <i>com.example.demo</i> as explained in the <i>Hello World Example</i> chapter.
2	Modify the default content of <i>res/layout/activity_main.xml</i> file to include few widgets in table layout.
3	No need to modify <i>string.xml</i> , 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/com.example.demo/MainActivity.java**. This file can include each of the fundamental lifecycle methods.

CoordinatorLayout

`CoordinatorLayout` is a super-powered [FrameLayout](#).

`CoordinatorLayout` is a general-purpose container that allows for coordinating interactive behaviors between its children.

`CoordinatorLayout` manages interactions between its children, and as such needs to contain all the `Views` that interact with each other. The two general cases supported by `CoordinatorLayout` are:

- As a top-level content layout (meaning `CoordinatorLayout` is at the root of all views within an activity or fragment).
- As a container for a specific interaction with one or more child views.

By specifying [Behaviors](#) for child views of a `CoordinatorLayout` you can provide many different interactions within a single parent and those views can also interact with one another.

Important XML Attributes

- **android:layout_gravity**: Specifies the gravity of the child relative to the parent. If you specify an Anchor using **app:layout_anchor**, then this attribute would be ignored. And you have to use **app:layout_anchorGravity** to position the child. Do not use both of these together in any view. It may cause of unexpected result.
- **app:layout_anchor**: This attribute can be set on children of the `CoordinatorLayout` to attach them to another view. The value would be the **id** of an anchor view that this view should position relative to. Note that, the anchor view can be **any** child View (a child of a child of a child of a *CoordinatorLayout*, for example).
- **app:layout_anchorGravity**: Specifies how an object should position relative to an anchor, on both the X and Y axes, within its parent's bounds.

- **app:layout_insetEdge:** Specifies how this view insets the `CoordinatorLayout` and make some other views dodge it. The child consumes the area of the screen it occupies and other children should not be placed in that area. Bottom is the default insetEdge of **SnackBar**

app:layout_dodgeInsetEdges: Specifies which edges of the child should be offset by insets. Bottom is the default `dodgeInsetEdges` of **FAB** button.

app:layout_behavior: The class name of a Behavior class defining special runtime behavior for this child view. **AppBarLayout** and **FAB** button have default behavior attached to them. This is most important attributes of `CoordinatorLayout`.

ConstraintLayout in Android

ConstraintLayout is similar to that of other View Groups which we have seen in Android such as [RelativeLayout](#), [LinearLayout](#), and many more. In this article, we will take a look at using ConstraintLayout in Android.

Important Attributes of ConstraintLayout

Attributes	Description
android:id	This is used to give a unique id to the layout.
app:layout_constraintBottom_toBottomOf	This is used to constrain the view with respect to the bottom position.

Attributes	Description
app:layout_constraintLeft_toLeftOf	This attribute is used to constrain the view with respect to the left position.
app:layout_constraintRight_toRightOf	This attribute is used to constrain the view with respect to the right position.
app:layout_constraintTop_toTopOf	This attribute is used to constrain the view with respect to the top position.

ImageSwitcher

In Android, [ImageSwitcher](#) is a specialized [ViewSwitcher](#) that contain [ImageView](#) type children. [ImageSwitcher](#) is available in Android from v1.6+.

It is an element of transition widget which helps us to add transitions on the images. It is mainly useful to animate an image on screen. [ImageSwitcher](#) switches smoothly between two images and thus provides a way of transitioning from one Image to another through appropriate animations.

ImageSwitcher code in XML:

```
<ImageSwitcher
    android:id="@+id/img1"
    android:layout_width="345dp"
    android:layout_height="430dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.432"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.318" />
```

Important Methods Of ImageSwitcher:

Let's we discuss some important methods of ImageSwitcher that may be called in order to manage the ImageSwitcher.

1. setFactory(ViewFactory factory): This method is used to create a new view for ImageSwitcher. By using this method we create a new [ImageView](#) and replace the old view with that.

Below code create a new view using setFactory method.

```
ImageSwitcher img=(ImageSwitcher) findViewById(R.id.img1);
img.setFactory(new ViewSwitcher.ViewFactory() {
    @Override
    public View makeView() {
        ImageView img=new ImageView(getApplicationContext());
        img.setScaleType(ImageView.ScaleType.FIT_XY);
        return img;
    }
});
```

2. setImageDrawable(Drawable drawable): This method is used to set an image in ImageSwitcher. In this we pass an image for Drawable.

Below we set image in ImageSwitcher by using setImageDrawable method.

```
ImageSwitcher img=(ImageSwitcher) findViewById(R.id.img1);
img.setImageDrawable(getResources().getDrawable(R.drawable.ic_launcher));
```

3. setImageResource(int resid): This method is also used to set an image in [image switcher](#). The image is passed in the form of integer id.

Below we set image in ImageSwitcher by using setImageDrawable method.

```
ImageSwitcher img=(ImageSwitcher) findViewById(R.id.img1);
img.setImageResource(R.drawable.ic_launcher);
```

4. setImageURI(Uri uri): This method is also used to set an image in [image switcher](#). The image is passed in the form of URI.

Below we set Image in ImageSwitcher from raw folder using Uri:

Before using below code first create a raw name folder in res directory and save an image name image1 in that folder.

```
ImageSwitcher img=(ImageSwitcher) findViewById(R.id.img1);
img.setImageURI(Uri.parse("android.resource://" + getPackageName() + "/" + R.raw.image1));
```

5. loadAnimation(Context context, int id): This method is used whenever we need to define an object of [Animation](#) class through AnimationUtils class by calling a static method loadAnimation.

Below we create an object of Animation class and load an animation by using AnimationUtils class.

```
Animation in= AnimationUtils.loadAnimation(this,
android.R.anim.slide_in_left);
Animation out=AnimationUtils.loadAnimation(this,
android.R.anim.slide_out_right);
```

6. setInAnimation(Animation inAnimation): This method is used to set the animation of the appearance of the object on the screen.

Below we create an object of Animation class and load an animation by using AnimationUtils class and then set the Animation on ImageSwitcher.

```
img.setInAnimation(in);
```

7. setOutAnimation(Animation outAnimation): This method does the opposite of setInAnimation().

Whenever we use set a new Image in [ImageView](#), it first removes the old view using an animation set with the setOutAnimation() method, and then places the new one using the animation set by the setInAnimation() method.

Below we create an object of Animation class and load an animation by using AnimationUtils class and then set the out Animation on ImageSwitcher

```
img.setOutAnimation(out);
```

Attributes of ImageSwitcher:

Now let's we discuss some common attributes of a ImageSwitcher that helps us to configure it in our layout ([xml](#)).

1. Id: id attribute is used to uniquely identify a ImageSwitcher.

Below we set the id of the ImageSwitcher that is used to uniquely identify it.

```
<ImageSwitcher
android:id="@+id/simpleImageSwitcher"
android:layout_width="wrap_content"
android:layout_height="wrap_content" /> <!-- id of ImageSwitcher that is used to uni
quely identify it -->
```

2. padding: This attribute is used to set the padding from left, right, top or bottom side of a ImageSwitcher.

- **paddingRight:** This attribute is used to set the padding from the right side of a ImageSwitcher.
- **paddingLeft:** This attribute is used to set the padding from the left side of a ImageSwitcher.
- **paddingTop:** This attribute is used to set the padding from the top side of a ImageSwitcher.
- **paddingBottom:** This attribute is used to set the padding from the bottom side of a ImageSwitcher.
- **Padding:** This attribute is used to set the padding from the all the side's of a ImageSwitcher.

Below we set the 10dp padding from all the sides of a ImageSwitcher

```
<ImageSwitcher
    android:id="@+id/imageSwitcher"
    android:layout_width="match_parent"
    android:layout_height="200dp"
    android:layout_gravity="center_horizontal"
    android:padding="10dp" /> <!-- set 10 dp padding from all the sides of TextSwitcher -
->
```

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

Below is the example code with explanation included in which we set the black color for the background of a ImageSwitcher.

```
<ImageSwitcher
    android:id="@+id/imageSwitcher"
    android:layout_width="match_parent"
    android:layout_height="200dp"
    android:layout_gravity="center_horizontal"
    android:background="#000" /> <!-- set black color in the background of TextSwitcher -
->
```

ImageSwitcher Example In Android Studio:

Step 1: [Create a new project](#) and name it ImageSwitcherExample

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

In this step we open an [xml](#) file and add the code for displaying a Button and a ImageSwitcher by using its different attributes.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/txt1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Sangola College"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.422"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.073" />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.435"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.815" />

    <ImageSwitcher
        android:id="@+id/img1"
        android:layout_width="345dp"
        android:layout_height="430dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.432"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.318" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

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

```
package com.example.myapplication;

import androidx.appcompat.app.AppCompatActivity;
```

```

import android.os.Bundle;
import android.view.View;
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
import android.widget.Button;
import android.widget.ImageSwitcher;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.ViewSwitcher;

public class MainActivity extends AppCompatActivity {
    Button btn;
    TextView txt;
    ImageSwitcher img;
    int
    imageId[]={R.drawable.collegenamechange,R.drawable.hami,R.drawable.logo};
    int count=imageId.length;
    int currentindex=-1;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        btn=(Button) findViewById(R.id.button);
        txt=(TextView) findViewById(R.id.txt1);
        img=(ImageSwitcher) findViewById(R.id.img1);

        img.setFactory(new ViewSwitcher.ViewFactory() {
            @Override
            public View makeView() {
                ImageView img=new ImageView(getApplicationContext());
                img.setScaleType(ImageView.ScaleType.FIT_XY);
                return img;
            }
        });

        Animation in= AnimationUtils.loadAnimation(this,
        android.R.anim.slide_in_left);
        Animation out=AnimationUtils.loadAnimation(this,
        android.R.anim.slide_out_right);

        img.setInAnimation(in);
        img.setOutAnimation(out);

        btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                currentindex++;
                if(currentindex==count)
                    currentindex=0;
                img.setImageResource(imageId[currentindex]);
            }
        });
    }
}

```

