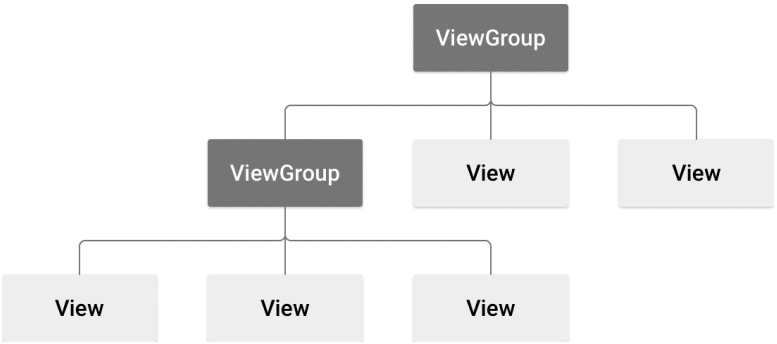
Unit 3

Android View, ViewGroups and UI

Introduction

In android, Layout is used to define the user interface for an app or activity and it will hold the UI elements that will appear to the user.

The user interface in an android app is made with a collection of View and ViewGroup objects. Generally, the android apps will contain one or more activities and each activity is a one screen of app. The activities will contain multiple UI components and those UI components are the instances of View and ViewGroup subclasses.



The user interface in an android app is made with a collection of View and ViewGroup objects. Generally, the android apps will contain one or more activities and each activity is a one screen of the app. The activities will contain multiple UI components and those UI components are the instances of View and ViewGroup subclasses.

Android View

The View is a base class for all UI components in android. For example, the **EditText** class is used to accept the input from users in android apps, which is a subclass of View.

Following are some common View subclasses that will be used in android applications.

- TextView
- EditText
- Button
- CheckBox
- RadioButton
- ImageButton
- Progress Bar
- Spinner

Android ViewGroup

The ViewGroup is a subclass of View and it will act as a base class for layouts and layouts parameters. The ViewGroup will provide an invisible container to hold other Views or ViewGroups and to define the layout properties.

For example, Linear Layout is the ViewGroup that contains a UI controls like button, textview, etc. and other layouts also.

Following are the commonly used ViewGroup subclasses in android applications.

- Linear Layout
- Relative Layout
- Table Layout
- Frame Layout
- Web View
- List View
- Grid View

Both View and ViewGroup subclasses together will play a key role to create a layout in android applications.

Android UI Layouts

In android, **Layout** is used to define the user interface for an app or activity and it will hold the UI elements that will appear to the user.

The user interface in the android app is made with a collection of View and ViewGroup objects. Generally, the android apps will contain one or more activities and each activity is one screen of the app. The activities will contain multiple UI components and those UI components are the instances of View and ViewGroup subclasses.

The View is a base class for all UI components in android and it is used to create an interactive UI component such as TextView, EditText, Checkbox, Radio Button, etc. and it responsible for event handling and drawing.

The ViewGroup is a subclass of View and it will act as a base class for **layouts** and **layouts parameters**. The ViewGroup will provide an invisible container to hold other Views or ViewGroups and to define the layout properties.

In android, we can define a layout in two ways, those are
- Declare UI elements in XML
- Instantiate layout elements at runtime

The android framework will allow us to use either or both methods to define our application's UI.

Declare UI Elements in XML

We can create layouts same like web pages in HTML by using default Views and ViewGroups in the XML file. The layout file must contain only one root element, which must be a View or ViewGroup object. Once we define the root element, then we can add additional layout objects or widgets as child elements to build the View hierarchy that defines our layout.

Example of defining a layout in an XML file (activity_main.xml) using LinearLayout to hold a TextView, EditText, and Button.

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

```
        android:id="@+id/fstTxt"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Enter Name"/>
    <EditText
        android:id="@+id/name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:ems="10" />
    <Button
        android:id="@+id/getName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Get Name" />
</LinearLayout>
```

After creating layout file, we need to load the XML layout resource from our activity onCreate() callback method like

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

Instantiate Layout Elements at Runtime

If we want to instantiate layout elements at runtime, we need to create own custom View and ViewGroup objects programmatically with required layouts.

Following is the example of creating a layout using LinearLayout to hold a TextView, EditText and Button in an activity using custom View and ViewGroup objects programmatically.

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        TextView textView1 = new TextView(this);
        textView1.setText("Name:");
        EditText editText1 = new EditText(this);
        editText1.setText("Enter Name");
        Button button1 = new Button(this);
        button1.setText("Add Name");
        LinearLayout linearLayout = new LinearLayout(this);
        linearLayout.addView(textView1);
        linearLayout.addView(editText1);
        linearLayout.addView(button1);
        setContentView(linearLayout);
    }
}
```

Android Layout Attributes

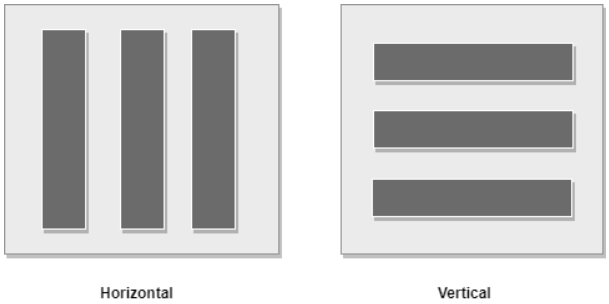
Attribute	Description
id	Uniquely identify the view and ViewGroups
layout_width	Define the width for View and ViewGroup elements in a layout
layout_height	Define the height for View and ViewGroup elements in a layout
layout_marginLeft	Define the extra space in the left side for View and ViewGroup elements in a layout
layout_marginRight	Define the extra space in right side for View and ViewGroup elements in layout
layout_marginTop	Define the extra space on top for View and ViewGroup elements in layout
layout_marginBottom	Define the extra space in the bottom side for View and ViewGroup elements in a layout
paddingLeft	Define the left side padding for View and ViewGroup elements in layout files
paddingRight	Define the right-side padding for View and ViewGroup elements in layout files
paddingTop	Define padding for View and ViewGroup elements in layout files on top side
paddingBottom	Define the bottom side padding for View and ViewGroup elements in layout files
layout_gravity	Define how child Views are positioned

Android LinearLayout

In android, LinearLayout is a ViewGroup subclass which is used to render all child View instances one by one either in Horizontal direction or Vertical direction based on the orientation property.

In android, we can specify the linear layout orientation using `android:orientation=""` attribute.

Following is the pictorial representation of linear layout in android applications.



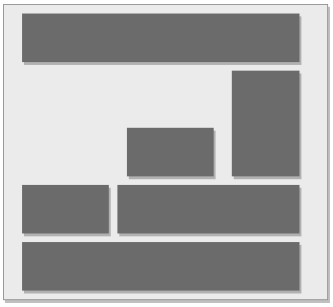
In LinearLayout, the child View instances arranged one by one, so the horizontal list will have only one row of multiple columns and vertical list will have one column of multiple rows.

Android LinearLayout Declaration

We can define the Layout in android applications like,

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:orientation="vertical">
    <EditText
        android:id="@+id/txtTo"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="To"/>
    <EditText
        android:id="@+id/txtSub"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Subject"/>
    <EditText
        android:id="@+id/txtMsg"
        android:layout_width="match_parent"
        android:layout_weight="1"
        android:gravity="top"
        android:hint="Message"
        android:layout_height="5dp"/>
    <Button
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:layout_gravity="right"
        android:text="Send"/>
</LinearLayout>
```

Android RelativeLayout



In android, RelativeLayout is a ViewGroup which is used to specify the position of child View instances relative to each other (Child **A** to the left of Child **B**) or relative to the parent (Aligned to the top of parent).

Following is the pictorial representation of relative layout in android applications.

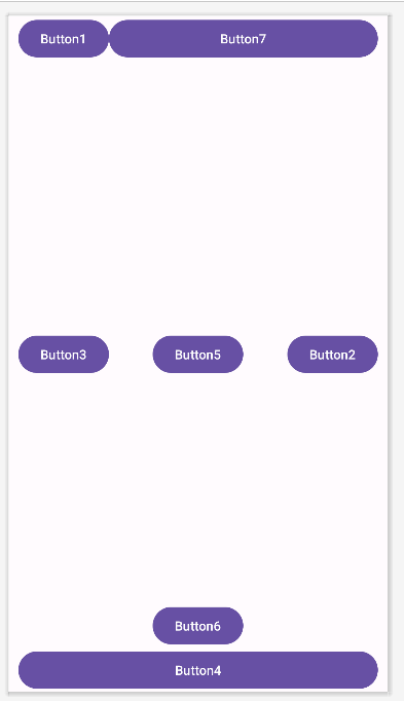
RelativeLayout is very useful to design user interface because by using relative layout we can eliminate the nested view groups and keep our layout hierarchy flat, which improves the performance of application.

Android Positioning Views in Relative Layout

In RelativeLayout we need to specify the position of child views relative to each other or relative to the parent. In case if we did not specify the position of child views, by default all child views are positioned to top-left of the layout.

Some layout properties available to views in RelativeLayout.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="10dp"
    android:paddingRight="10dp">
    <Button
        android:id="@+id/btn1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:text="Button1" />
    <Button
        android:id="@+id/btn2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"
        android:layout_centerVertical="true"
        android:text="Button2" />
    <Button
        android:id="@+id/btn3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_centerVertical="true"
        android:text="Button3" />
    <Button
        android:id="@+id/btn4"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:text="Button4" />
    <Button
        android:id="@+id/btn5"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBottom="@+id/btn2"
        android:layout_centerHorizontal="true"
        android:text="Button5" />
    <Button
        android:id="@+id/btn6"
```

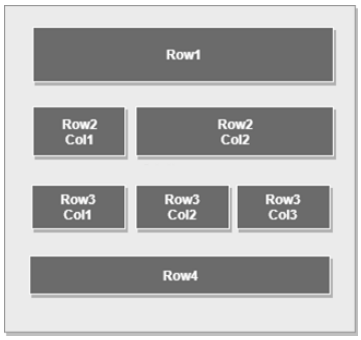


```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_above="@+id/btn4"
        android:layout_centerHorizontal="true"
        android:text="Button6" />
    <Button
        android:id="@+id/btn7"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toEndOf="@+id/btn1"
        android:layout_toRightOf="@+id/btn1"
        android:layout_alignParentRight="true"
        android:text="Button7" />
</RelativeLayout>
```

Here in this code, I have set the relation of button widgets using RelativeLayout.

Android TableLayout

In android, TableLayout is a ViewGroup subclass that is used to display the child View elements in rows and columns.



Following is the pictorial representation of table layout in android applications.

In android, TableLayout will position its children's elements into rows and columns and it will not display any border lines for rows, columns, or cells.

The TableLayout in android will work same as the HTML table and the table will have as many columns as the row with the most cells. The TableLayout can be explained as **<table>** and TableRow is like **<tr>** element.

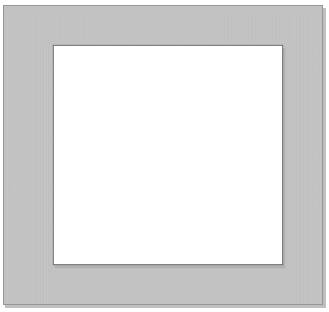
```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_marginTop="100dp"
    android:paddingLeft="10dp"
    android:paddingRight="10dp" >
    <TableRow
        android:background="#0079D6"
        android:padding="5dp">
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Sr. No"
            android:textStyle="bold"
```

```
            android:textColor="@color/white"/>
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="College Name"
            android:textColor="@color/white"
            android:textStyle="bold"/>
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textColor="@color/white"
            android:layout_weight="1"
            android:textStyle="bold"
            android:text="Location" />
    </TableRow>
    <TableRow
        android:background="#DAE8FC"
        android:padding="5dp">
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="1" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="GCET" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Bakrol Road" />
    </TableRow>
    <TableRow
        android:background="#DAE8FC"
        android:padding="5dp">
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="2" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
```

Sr. No	College Name	Location
1	GCET	Bakrol Road
2	ADIT	New V V Nagar

```
            android:text="ADIT" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="New V V Nagar" />
    </TableRow>
</TableLayout>
```

Android FrameLayout



In android, FrameLayout is a ViewGroup subclass that is used to specify the position of View instances it contains on the top of each other to display only single View inside the FrameLayout.

In simple manner, we can say FrameLayout is designed to block out an area on the screen to display a single item.

FrameLayout will act as a placeholder on the screen and it is used to hold a single child view.

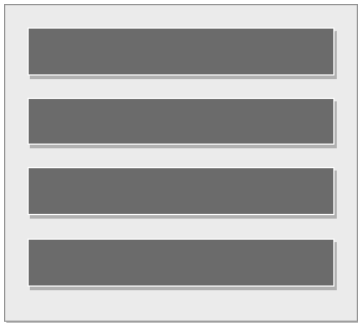
In FrameLayout, the child views are added in a stack and the most recently added child will show on the top. We can add multiple children views to FrameLayout and control their position by using gravity attributes in FrameLayout.

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

    <ImageView
        android:id="@+id/imgvw1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:scaleType="centerCrop"
        android:src="@drawable/gcet" />

    <TextView
        android:id="@+id/txtvw1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="40dp"
        android:background="#4C374A"
        android:padding="10dp"
        android:text="Gcet, Bakrol"
        android:textColor="#FFFFFF"
        android:textSize="20sp" />
    <TextView
```

```
        android:id="@+id/txtvw2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="right|bottom"
        android:background="#AA000000"
        android:padding="10dp"
        android:text="17/Aug/2023"
        android:textColor="#FFFFFF"
        android:textSize="18sp" />
</FrameLayout>
```

Here, you can add your image to drawable folder

Android ListView



ln android, ListView is a ViewGroup that is used to display the list of scrollable of items in multiple rows and the list items are automatically inserted to the list using an adapter.

Generally, the adapter pulls data from sources such as an array or database and converts each item into a result view and that is placed into the list.

Android Adapter

In android, Adapter will act as an intermediate between the data sources and adapter views such as ListView, to fill the data into adapter views. The adapter will hold the data and iterates through an item in data set and generate the views for each item in the list.

Generally, in android we have a different type of adapters available to fetch the data from different data sources to fill the data into adapter views, those are

Adapter	Description
ArrayAdapter	It will expect an Array or List as input.
CurosrAdapter	It will accept an instance of a cursor as an input.
SimpleAdapter	It will accept a static data defined in the resources.
BaseAdapter	It is a generic implementation for all three adapter types and it can be used for ListView, Gridview or Spinners based on our requirements

activity_main.xml

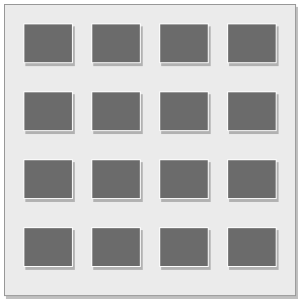
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
```

```
    <ListView
        android:id="@+id/collegelist"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >
    </ListView>
</LinearLayout>
```

MainActivity.java

```
package com.example.layoutexample;

import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {
    private ListView mListView;
    private ArrayAdapter aAdapter;
    private String[] colleges = { "GCET", "ADIT", "MBIT", "SEMCOM",
"ISTAR"};
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mListView = (ListView) findViewById(R.id.collegelist);
        aAdapter = new ArrayAdapter(this,
android.R.layout.simple_list_item_1, colleges);
        mListView.setAdapter(aAdapter);
    }
}
```

Android GridView



In android, GridView is a ViewGroup that is used to display items in a two-dimensional, scrollable grid and grid items are automatically inserted to the GridView layout using a list adapter.

Generally, the adapter pulls data from sources such as an array or database and converts each item into a result view and that is placed into the list.

acticity_main.xml

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

```
    android:columnWidth="110dp"
    android:numColumns="auto_fit"
    android:verticalSpacing="10dp"
    android:horizontalSpacing="10dp"
    android:stretchMode="columnWidth"
    android:gravity="center" />
```

ImageAdapter.java

```
package com.example.layoutexample;

import android.content.Context;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.GridView;
import android.widget.ImageView;

public class ImageAdapter extends BaseAdapter {
    private Context mContext;
    public ImageAdapter(Context c) {
        mContext = c;
    }
    public int getCount() {
        return thumbImages.length;
    }
    public Object getItem(int position) {
        return null;
    }
    public long getItemId(int position) {
        return 0;
    }
    // create a new ImageView for each item referenced by the Adapter
    public View getView(int position, View convertView, ViewGroup parent) {
        ImageView imageView = new ImageView(mContext);
        imageView.setLayoutParams(new GridView.LayoutParams(200, 200));
        imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
        imageView.setPadding(8, 8, 8, 8);
        imageView.setImageResource(thumbImages[position]);
        return imageView;
    }
    // Add all our images to arraylist
    public Integer[] thumbImages = {
            R.drawable.img1, R.drawable.img2,
            R.drawable.img3, R.drawable.img4,
            R.drawable.img5, R.drawable.img6,
            R.drawable.img7, R.drawable.img1,
```

```
            R.drawable.img3, R.drawable.img5,
            R.drawable.img2, R.drawable.img4,
    };
}
```

MainActivity.java

```
package com.example.layoutexample;

import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.GridView;
import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;


public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        GridView gv = (GridView) findViewById(R.id.gridview);
        gv.setAdapter(new ImageAdapter(this));
        gv.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            public void onItemClick(AdapterView<?> parent, View v, int
position, long id) {
                Toast.makeText(MainActivity.this, "Image Position: " +
position, Toast.LENGTH_SHORT).show();
            }
        });
    }
}
```

Android WebView

In android, WebView is an extension of View class and it is used to show the static HTML web pages content or remote web pages content with URL in android applications as a part of our activity layout.

Generally, in android, WebView will act as an embedded browser to include the web pages content in our activity layout and it will not contain any features of normal browsers, such as address bar, navigation controls, etc.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<WebView  xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/webview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" />
```

MainActivity.java

```
package com.example.layoutexample;

import android.os.Bundle;
import android.webkit.WebView;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        WebView wv = (WebView) findViewById(R.id.webview);
        String customHtml = "<html><body><h1>Welcome to GCET</h1>" +
                "<h2>Name: Mobile Application Development</h2><h2>Code:
102046712</h2>" +
                "<p>It's a Static Web HTML Content.</p>" +
                "</body></html>";
        wv.loadData(customHtml, "text/html", "UTF-8");
    }
}
```

Android UI Tools

Android TextView

In android, TextView is a user interface control that is used to set and display the text to the user based on our requirements. The TextView control will act as like label control and it won't allow users to edit the text.

In android, we can create a TextView control in two ways either in XML layout file or create it in Activity file programmatically.

Here is the simple declaration to set the text of TextView control.

```
<TextView
    android:id="@+id/textView"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Welcome to GCET"/>
```

Set the text of TextView control programmatically in activity file using setText() method.

```
TextView tv = (TextView)findViewById(R.id.textView);
tv.setText("Welcome to GCET");
```

layout file using id property and setting the text using setText() method.

TextView Attributes

Attribute	Description
id	Used to uniquely identify the control
autoLink	Automatically convert URLs and email addresses as clickable links.
ems	Used to make the TextView be exactly this many ems wide.
hint	Used to display the hint text when text is empty
width	It makes the TextView be exactly this many pixels wide.
height	It makes the TextView be exactly this many pixels tall.
text	Used to display the text.
textColor	Used to change the color of the text.
gravity	Used to specify how to align the text by the view's x and y-axis.
maxWidth	Used to make the TextView be at most this many pixels wide.
minWidth	Used to make the TextView be at least this many pixels wide.
textSize	Used to specify the size of the text.
textStyle	Used to change the style (bold, italic, bolditalic) of text.
textAllCaps	Used to present the text in all CAPS
typeface	Specify the Typeface (normal, sans, serif, monospace) for the text.
textColor	Used to change the color of the text.
textColorHighlight	Used to change the color of text selection highlight.
textColorLink	Used to change the text color of links.
inputType	Used to specify the type of text being placed in text fields.
fontFamily	Used to specify the fontFamily for the text.
editable	If we set, it specifies that this TextView has an input method.

Android EditText

In android, EditText is a user interface control which is used to allow the user to enter or modify the text. While using EditText control in our android applications, we need to specify the type of data the text field can accept using the inputType attribute.

If it accepts plain text, then we need to specify the inputType as “text”, for password, specify the inputType as “textPassword”.

In android, EditText control is an extended version of TextView control with additional features and it is used to allow users to enter input values.

Same as TextView, we can create EditText control in two ways either in XML layout file or create it in Activity file programmatically.

EditText in Layout File

Following is the sample way to define EditText control in XML layout file in android application.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <EditText
        android:id="@+id/txtSub"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Subject"
        android:inputType="text"/>
</LinearLayout>
```

Create EditText Control in Activity File

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        LinearLayout linearLayout =  (LinearLayout)
findViewById(R.id.linearlayout);
        EditText et = new EditText(this);
        et.setHint("Subject");
        linearLayout.addView(et);
    }
}
```

Set the Text of Android EditText

In android, we can set the text of EditText control either while declaring it in Layout file or by using setText() method in Activity file.

Example to set the text of TextView control while declaring it in XML Layout file.

```
<EditText
    android:id="@+id/editText1"
    android:layout_width="match_parent"
```

```
    android:layout_height="wrap_content"
    android:text="Welcome to GCET" />
```

Similarly we can set Text programmatically in activity file using setText() method.

```
EditText et = (EditText)findViewById(R.id.editText);
et.setText("Welcome to GCET");
```

Attributes gravity, text, hint, textColor, textColorHint, textSize, textStyle, background, ems, width, height, textColorHighlight, fontFamily are like TextView Attributes.

AutoCompleteTextView

In android, AutoCompleteTextView is an editable text view which is used to show the list of suggestions based on the user typing text. The list of suggestions will be shown as a dropdown menu from which the user can choose an item to replace the content of the textbox.

The AutoCompleteTextView is a subclass of EditText class so we can inherit all the properties of EditText in AutoCompleteTextView based on our requirements.

Dropdown list of suggestions can be obtained from the data adapter and those suggestions will be appeared only after giving the number characters defined in the Threshold limit.

The Threshold property of AutoCompleteTextView is used to define the minimum number of characters the user must type to see the list of suggestions.

The dropdown list of suggestions can be closed at any time in case if no item is selected from the list or by pressing the back or enter key.

Attributes gravity, text, hint, textColor, textColorHint, textSize, textStyle, background, ems, width, height, textColorHighlight, fontFamily are like EditText and TextView Attributes.

Following is the example of defining AutoCompleteTextView control in LinearLayout to bind the data to defined control using a data adapter and getting the selected list item value in the android application.

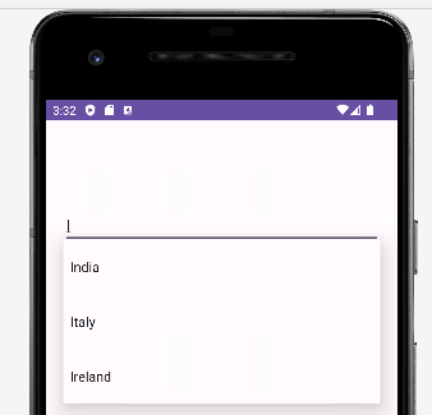
activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:paddingLeft="20dp"
android:paddingRight="20dp"
```

```
android:orientation="vertical"
android:id="@+id/linear_Layout">
<AutoCompleteTextView
    android:id="@+id/ac_Country"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="100dp"
    android:hint="Enter Country Name"/>
</LinearLayout>
```

MainActivity.java

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

public class MainActivity extends AppCompatActivity {
    String[] Countries = { "India", "USA", "Australia", "UK", "Italy",
"Ireland", "Africa" };
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
                android.R.layout.simple_dropdown_item_1line, Countries);
        AutoCompleteTextView actv =
(AutoCompleteTextView)findViewById(R.id.ac_Country);
        actv.setThreshold(1);
        actv.setAdapter(adapter);
        actv.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view, int
position, long id) {
                Toast.makeText(getApplicationContext(), "Selected Item: " +
parent.getSelectedItem(), Toast.LENGTH_SHORT).show();
            }
        });
    }
}
```

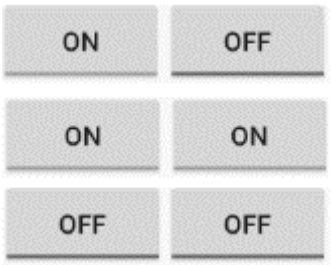


Project: **UITOOLAPPLICATION**

Android ToggleButton

In android, ToggleButton is a user interface control that is used to display ON (Checked) or OFF (Unchecked) states as a button with a light indicator.

The ToggleButton is useful for the users to change the settings between two states either ON or OFF. We can add a ToggleButton to our application layout by using the ToggleButton object.



By default, the android ToggleButton will be in OFF (Unchecked) state. We can change the default state of ToggleButton by using `android:checked` attribute.

In case, if we want to change the state of ToggleButton to ON (Checked), then we need to set `android:checked="true"` in our XML layout file.

We can create ToggleButton control in XML layout file or in the Activity file programmatically.

Create ToggleButton in XML Layout File

Simple way to create ToogleButton

```
<ToggleButton
    android:id="@+id/toggle1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="100dp"
    android:layout_marginTop="120dp"
    android:checked="true"
    android:textOff="OFF"
    android:textOn="ON"/>
```

Create ToggleButton Control in Activity File

```
RelativeLayout layout = (RelativeLayout)findViewById(R.id.layout);
ToggleButton tb = new ToggleButton(this);
tb.setTextOff("OFF");
tb.setTextOn("ON");
tb.setChecked(true);
layout.addView(tb);
```

This is how we can define ToggleButton in XML layout file or programmatically in activity file based on our requirements.

Example

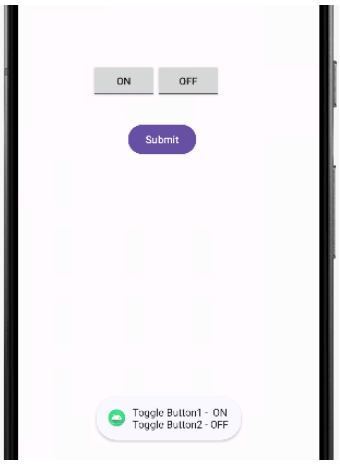
Following is the example of defining two ToggleButton controls and one Button control in RelativeLayout to get the state of ToggleButton controls when we click on Button control in the android application.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
android:layout_height="match_parent">
    <ToggleButton
        android:id="@+id/toggle1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="100dp"
        android:layout_marginTop="120dp"
        android:checked="true"
        android:textOff="OFF"
        android:textOn="ON"/>
    <ToggleButton
        android:id="@+id/toggle2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBottom="@+id/toggle1"
        android:layout_toRightOf="@+id/toggle1"
        android:textOff="OFF"
        android:textOn ="ON"/>
    <Button
        android:id="@+id/getBtn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="150dp"
        android:layout_marginTop="200dp"
        android:text="Submit" />
</RelativeLayout>
```

MainActivity.java

```
package com.example.togglebuttonexample;

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

import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        final ToggleButton tb1 = (ToggleButton)findViewById(R.id.toggle1);
        final ToggleButton tb2 = (ToggleButton)findViewById(R.id.toggle2);
        Button btnGet = (Button)findViewById(R.id.getBtn);
        btnGet.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Toast.makeText(getApplicationContext(), "Toggle Button1 -
" + tb1.getText().toString() + " \n" + "Toggle Button2 - " +
tb2.getText().toString(),Toast.LENGTH_SHORT).show();
            }
        });
    }
}
```



Project: **TOOGLEBUTTON**

Android CheckBox with Examples

In android, CheckBox is a two-states button that can be either checked (ON) or unchecked (OFF) and it will allow users to toggle between the two states (ON / OFF) based on the requirements.

Generally, we can use multiple CheckBox controls in android application to allow users to select one or more options from the set of values.

By default, the android CheckBox will be in the OFF (Unchecked) state. We can change the default state of CheckBox by using `android:checked` attribute.

In case, if we want to change the state of CheckBox to ON (Checked), then we need to set `android:checked="true"` in our XML layout file.

We can create CheckBox control in two ways either in the XML layout file or create it in the Activity file programmatically.

Generally, whenever the user clicks on CheckBox to Select or Deselect the CheckBox object will receive an on-click event.

activity_main.xml

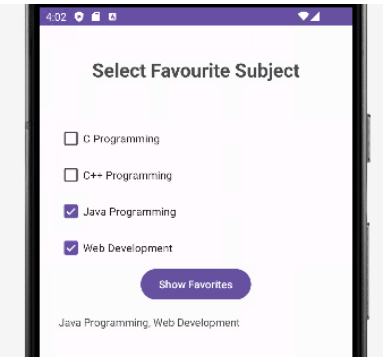
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:padding="25dp"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Select Favourite Subject"
        android:layout_gravity="center"
        android:textStyle="bold"
        android:textSize="25dp"
        android:layout_marginTop="16dp"/>
        <CheckBox
            android:layout_marginTop="50dp"
            android:id="@+id/cCheckbox"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="C Programming"/>
        <CheckBox
            android:id="@+id/cppCheckbox"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="C++ Programming"/>
        <CheckBox
            android:id="@+id/javaCheckbox"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Java Programming"/>
        <CheckBox
            android:id="@+id/webCheckbox"
            android:layout_width="wrap_content"
```

```
            android:layout_height="wrap_content"
            android:text="Web Development"/>
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="Show Favorites"
        android:onClick="showFavorites"/>
    <TextView
        android:id="@+id/favoriteSubjects"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text=""
        android:layout_marginTop="16dp"/>
</LinearLayout>
```

MainActivity.java

```
package com.example.checkboxexample;

import android.os.Bundle;
import android.view.View;
import android.widget.CheckBox;
import android.widget.TextView;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

    private CheckBox cCheckbox;
    private CheckBox cppCheckbox;
    private CheckBox javaCheckbox;
    private CheckBox webCheckbox;
    private TextView favoriteSubjectsTextView;

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

        cCheckbox = findViewById(R.id.cCheckbox);
        cppCheckbox = findViewById(R.id.cppCheckbox);
        javaCheckbox = findViewById(R.id.javaCheckbox);
        webCheckbox = findViewById(R.id.webCheckbox);
        favoriteSubjectsTextView = findViewById(R.id.favoriteSubjects);
    }

    public void showFavorites(View view) {
```

```
        StringBuilder selectedSubjects = new StringBuilder();

        if (cCheckbox.isChecked()) {
            selectedSubjects.append("C Programming, ");
        }
        if (cppCheckbox.isChecked()) {
            selectedSubjects.append("C++ Programming, ");
        }
        if (javaCheckbox.isChecked()) {
            selectedSubjects.append("Java Programming, ");
        }
        if (webCheckbox.isChecked()) {
            selectedSubjects.append("Web Development, ");
        }

        String favorites = selectedSubjects.toString();
        if (!favorites.isEmpty()) {
            favorites = favorites.substring(0, favorites.length() - 2);
        } else {
            favorites = "No subjects selected";
        }

        favoriteSubjectsTextView.setText(favorites);
    }

    public void displayFavoriteSubjects(View view) {
        String favorites = favoriteSubjectsTextView.getText().toString();

        if (favorites.equals("No subjects selected")) {
            Toast.makeText(this, "Select favorite subjects first.",
Toast.LENGTH_SHORT).show();
        } else {
            Toast.makeText(this, "Favorite Subjects: " + favorites,
Toast.LENGTH_LONG).show();
        }
}
```



Project: **CheckBoxExample**

RadioButton

In android, RadioButton is a two-states button that can be either checked or unchecked and it's the same as CheckBox control, except that it will allow only one option to select from the group of options.

Generally, we can use RadioButton controls in an android application to allow users to select only one option from the set of values.

In android, we use radio buttons with in a RadioGroup to combine multiple radio buttons into one group and it will make sure that users can select only one option from the group of multiple options.

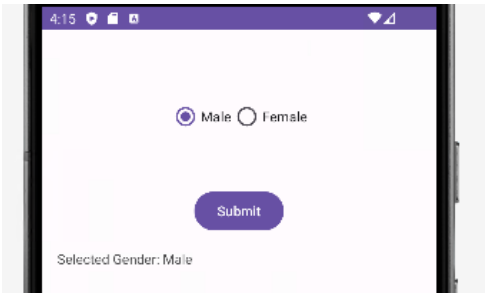
activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp"
    tools:context=".MainActivity">
    <RadioGroup
        android:id="@+id/genderRadioGroup"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:layout_margin="25dp"
        android:layout_gravity="center"
        android:padding="25dp">
        <RadioButton
            android:id="@+id/maleRadioButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Male"/>
        <RadioButton
            android:id="@+id/femaleRadioButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Female"/>
    </RadioGroup>
    <Button
        android:layout_gravity="center"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Submit"
        android:onClick="submitGenderChoice"/>
    <TextView
        android:id="@+id/genderResult"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"/>
</LinearLayout>
```

MainActivity.java

```
package com.example.radiobuttonexample;
```

```
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.*;

public class MainActivity extends AppCompatActivity {

    private RadioGroup genderRadioGroup;
    private TextView genderResult;

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

        genderRadioGroup = findViewById(R.id.genderRadioGroup);
        genderResult = findViewById(R.id.genderResult);
    }

    public void submitGenderChoice(View view) {
        int selectedRadioButtonId =
genderRadioGroup.getCheckedRadioButtonId();
        RadioButton selectedRadioButton =
findViewById(selectedRadioButtonId);

        if (selectedRadioButton != null) {
            String gender = selectedRadioButton.getText().toString();
            genderResult.setText("Selected Gender: " + gender);
        } else {
            genderResult.setText("No gender selected");
        }
    }
}
```



Project: RADIOBUTTONEXAMPLE

ProgressBar

In android, ProgressBar is a user interface control that is used to indicate the progress of an operation. For example, downloading a file, uploading a file.

By default, the ProgressBar will be displayed as a spinning wheel, in case if we want to show it like a horizontal bar then we need to change the style property to horizontal.

Attribute	Description
`id`	It is used to uniquely identify the control
`minHeight`	It is used to set the height of the progress bar.
`minWidth`	It is used to set the width of the progress bar.
`max`	It is used to set the maximum value of the progress bar.
`progress`	It is used to set the default progress value between 0 and max. It must be an integer value.

The ProgressBar supports two types of modes to show the progress, those are Determinate and Indeterminate.

Android ProgressBar with Determinate Mode

Generally, we use the **Determinate** progress mode in progress bar when we want to show the quantity of progress has occurred. For example, the percentage of file downloaded, number of records inserted into a database, etc.

To use Determinate progress, we need to set the style of the progress bar to **Widget_ProgressBar_Horizontal** or **progressBarStyleHorizontal** and set the amount of progress using **android:progress** attribute.

Example which shows a **Determinate** progress bar that is **50%** complete.

```
<ProgressBar
    android:id="@+id/pBar"
    style="?android:attr/progressBarStyleHorizontal"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:max="100"
    android:progress="50" />
```


By using setProgress(int) method, we can update the percentage of progress displayed in app or by calling incrementProgressBy(int) method, we can increase the value of current progress completed based on our requirements.

Generally, when the progress value reaches 100 then the progress bar is full. By using max attribute, we can adjust this default value.

Android ProgressBar with Indeterminate Mode

Generally, we use the Indeterminate progress mode in progress bar when we don't know how long an operation will take or how much work has done.

In indeterminate mode the actual progress will not be shown, only the cyclic animation will be shown to indicate that some progress is happing like as shown in the above progress bar loading images.

Example to set Indeterminate progress mode in an XML layout file.

```
<ProgressBar
    android:id="@+id/progressBar1"
    style="?android:attr/progressBarStyleHorizontal"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:indeterminate="true"/>
```

Example of shows one ProgressBar control, one TextView control and one Button control in RelativeLayout to start showing the progress in the progress bar on Button click in the android application.

activity_main.xml

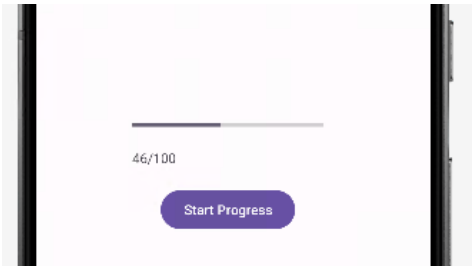
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
android:layout_height="match_parent">
    <ProgressBar
        android:id="@+id/pBar"
        style="?android:attr/progressBarStyleHorizontal"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="100dp"
        android:layout_marginTop="200dp"
        android:minHeight="50dp"
        android:minWidth="200dp"
        android:max="100"
        android:indeterminate="false"
        android:progress="0" />
    <TextView
        android:id="@+id/tView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/pBar"
        android:layout_below="@+id/pBar" />
    <Button
        android:id="@+id/btnShow"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="130dp"
        android:layout_marginTop="20dp"
        android:text="Start Progress"
```

```
        android:layout_below="@+id/tView"/>
</RelativeLayout>
```

MainActivity.java

```
package com.example.progressbarexample;
import android.os.Handler;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.ProgressBar;
import android.widget.TextView;

import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {
    private ProgressBar pgsBar;
    private int i = 0;
    private TextView txtView;
    private Handler hdlr = new Handler();
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        pgsBar = (ProgressBar) findViewById(R.id.pBar);
        txtView = (TextView) findViewById(R.id.tView);
        Button btn = (Button)findViewById(R.id.btnShow);
        btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                i = pgsBar.getProgress();
                new Thread(new Runnable() {
                    public void run() {
                        while (i < 100) {
                            i += 1;
                            // Update the progress bar and display the
current value in text view
                            hdlr.post(new Runnable() {
                                public void run() {
                                    pgsBar.setProgress(i);
                                    txtView.setText(i+"/"+pgsBar.getMax());
                                }
                            });
                            try {
                                // Sleep for 100 milliseconds to show the
progress slowly.
                                Thread.sleep(100);
                            } catch (InterruptedException e) {
```

```
                            e.printStackTrace();
                        }
                    }
                }
            }).start();
        }
    });
  }
}
```



Project: **PROGRESSBARAPPLICATION**

Android Spinner (Dropdown List) with Examples

In android, **Spinner** is a view that allows a user to select one value from the list of values. The spinner in android will behave same as a dropdown list in other programming languages.

Generally, the android spinners will provide a quick way to select one item from the list of values and it will show a dropdown menu with a list of all values when we click or tap on it.

By default, the android spinner will show its currently selected value and by using Adapter we can bind the items to spinner objects.

We can populate our Spinner control with list of choices by defining an ArrayAdapter in our Activity file.

Generally, the Adapter pulls data from sources such as an array or database and converts each item into a result view and that is placed into the list.

activity_main.xml

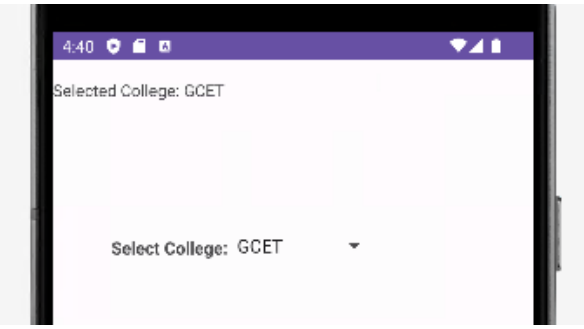
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
android:layout_height="match_parent">
    <TextView
        android:id="@+id/txtVw"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="50dp"
        android:layout_marginTop="150dp"
        android:text="Select College:"
        android:textStyle="bold"
```

```
        android:textSize="15dp" />
    <Spinner
        android:id="@+id/spinner1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBottom="@+id/txtVw"
        android:layout_toRightOf="@+id/txtVw" />

    <TextView
        android:id="@+id/selectedItemTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"/>
</RelativeLayout>
```

MainActivity.java

```
package com.example.spinnerexample;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Spinner;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity implements
AdapterView.OnItemSelectedListener {
    String[] users = { "GCET", "ADIT", "MBIT", "SEMCOM", "ISTAR" };
    private TextView selectedItemTextView;

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

        selectedItemTextView = findViewById(R.id.selectedItemTextView);

        Spinner spin = findViewById(R.id.spinner1);
        ArrayAdapter<String> adapter = new ArrayAdapter<>(this,
android.R.layout.simple_spinner_item, users);

adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_it
em);
        spin.setAdapter(adapter);
        spin.setOnItemSelectedListener(this);
    }
```

```
    @Override
    public void onItemSelected(AdapterView<?> arg0, View arg1, int
position, long id) {
        selectedItemTextView.setText("Selected College: " +
users[position]);
    }

    @Override
    public void onNothingSelected(AdapterView<?> arg0) {
        // TODO - Custom Code
    }
}
```



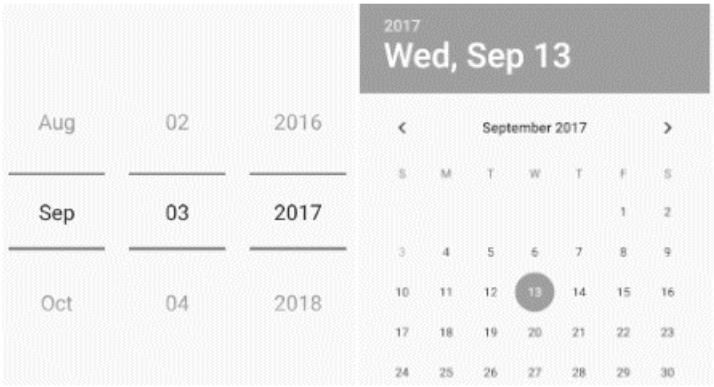
Project: **SPINNEREXAMPLE**

Android DatePicker

In android, DatePicker is a control that will allow users to select the date by a day, month and year in our application user interface.

If we use DatePicker in our application, it will ensure that the users will select a valid date.

Following is the pictorial representation of using a DatePicker control in android applications.



Generally, in android DatePicker available in two modes, one is to show the complete calendar and another one is to show the dates in spinner view.

activity_main.xml

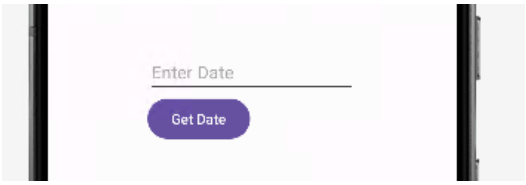
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

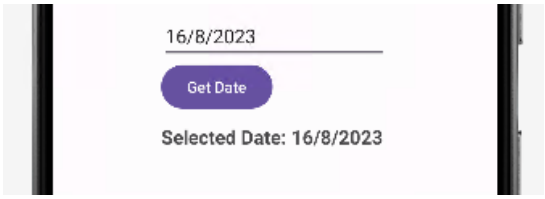
```
    android:layout_width="match_parent"
android:layout_height="match_parent">
    <DatePicker
        android:id="@+id/datePicker1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="20dp" />
    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/datePicker1"
        android:layout_marginLeft="100dp"
        android:text="Get Date" />
    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/button1"
        android:layout_marginLeft="100dp"
        android:layout_marginTop="10dp"
        android:textStyle="bold"
        android:textSize="18dp"/>
</RelativeLayout>
```

MainActivity.java

```
package com.example.datepickerexample;

import android.os.Bundle;
import android.webkit.WebView;
import androidx.appcompat.app.AppCompatActivity;
import android.content.Intent;
import android.view.View;
import android.widget.Button;
import android.widget.DatePicker;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    private Button relButton, tblButton, framButton, listButton,
gridButton, webButton;

    DatePicker picker;
    Button btnGet;
    TextView tvw;
    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        tvw=(TextView)findViewById(R.id.textView1);
        picker=(DatePicker)findViewById(R.id.datePicker1);
        btnGet=(Button)findViewById(R.id.button1);
        btnGet.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                tvw.setText("Selected Date: "+ picker.getDayOfMonth()+"/"+
(picker.getMonth() + 1)+"/"+picker.getYear());
            }
        });
    }
}
```





Project: **DATEPICKEREXAMPLE**

Android RatingBar

In android, RatingBar is a UI control that is used to get the rating from the user. The RatingBar is an extension of SeekBar and ProgressBar that shows a rating in stars and it allows users to set the rating value by touch or click on the stars.

The android RatingBar will always return a rating value as a floating-point number such as 1.0, 2.0, 2.5, 3.0, 3.5, etc.

In android, by using android:numStars attribute we can define the number of stars to display in RatingBar. An example of using RatingBar is in movie sites or product sites to collect the user rating about the movies or products, etc.

In android, by using android.widget.RatingBar component we can display the rating bar with star icons.

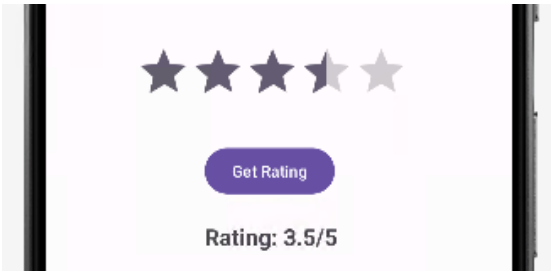
activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
android:layout_height="match_parent">
    <RatingBar
        android:id="@+id/ratingBar1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="80dp"
        android:layout_marginTop="200dp"
        android:numStars="5"
        android:rating="3.5"/>
    <Button
        android:id="@+id/btnGet"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/ratingBar1"
        android:layout_below="@+id/ratingBar1"
        android:layout_marginTop="30dp"
        android:layout_marginLeft="60dp"
        android:text="Get Rating"/>
    <TextView
        android:id="@+id/textview1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/btnGet"
        android:layout_below="@+id/btnGet"
        android:layout_marginTop="20dp"
        android:textSize="20dp"
        android:textStyle="bold"/>
</RelativeLayout>
```

MainActivity.java

```
package com.example.ratingbarexample;

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

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

```
public class MainActivity extends AppCompatActivity {
    private RatingBar rBar;
    private TextView tView;
    private Button btn;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        rBar = (RatingBar) findViewById(R.id.ratingBar1);
        tView = (TextView) findViewById(R.id.textview1);
        btn = (Button) findViewById(R.id.btnGet);
        btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                int noofstars = rBar.getNumStars();
                float getrating = rBar.getRating();
                tView.setText("Rating: " + getrating + "/" + noofstars);
            }
        });
    }
}
```



Project: **RATINGBAREXAMPLE**

Android Toast

Toast is a small popup notification that is used to display an information about the operation which we performed in our app. The Toast will show the message for a small period of time and it will disappear automatically after a timeout.

Generally, the size of Toast will be adjusted based on the space required for the message and it will be displayed on the top of the main content of activity for a short period of time.

For example, some of the apps will show a message like "**Press again to exit**" in toast, when we pressed a back button on the home page or showing a message like "**saved successfully**" toast when we click on the button to save the details.

Create a Toast in Android

In android, we can create a Toast by instantiating an android.widget.Toast object using makeText() method. The makeText() method will take three parameters: application context,

text message and the duration for the toast. We can display the Toast notification by using show() method.

Syntax of creating a **Toast** in android applications.

```
Toast.makeText(context, "message", duration).show();
```

Parameter	Description
context	It is our application context.
message	It is our custom message which we want to show in Toast notification.
duration	It is used to define the duration for notification to display on the screen.

We have two ways to define the Toast **duration**, either in **LENGTH_SHORT** or **LENGTH_LONG** to display the toast notification for a short or longer period of time.

Example of defining a Toast

```
Toast.makeText(MainActivity.this, "Details Saved Successfully.",
Toast.LENGTH_SHORT).show();
```

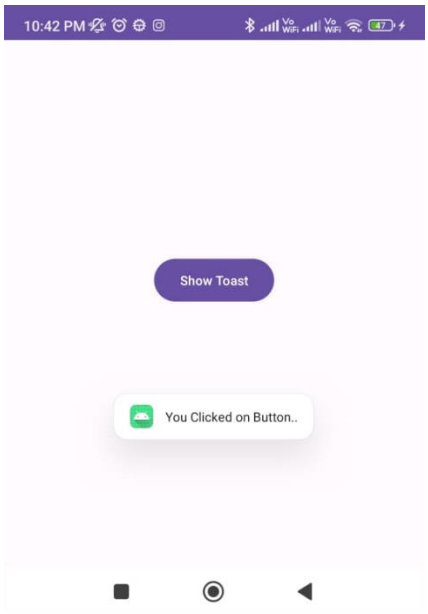
activity_main.xml

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

    <Button
        android:id="@+id/btnShow"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Show Toast"
        android:layout_marginTop="200dp"
android:layout_marginLeft="140dp"/>
</LinearLayout>
```

MainActivty.java

```
package com.example.toastexample;

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

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button btn = (Button)findViewById(R.id.btnShow);
        btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Toast.makeText(MainActivity.this, "You Clicked on
Button..", Toast.LENGTH_SHORT).show();
            }
        });
    }
}
```



Android Menus (Options, Context, Popup)

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.

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 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.

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 nested <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 properties such as active state and visibility.

Here, the simple way how we can define menu in Activity

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

For adding submenu in menu item, 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>
```

Handle Android Menu Click Events

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

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

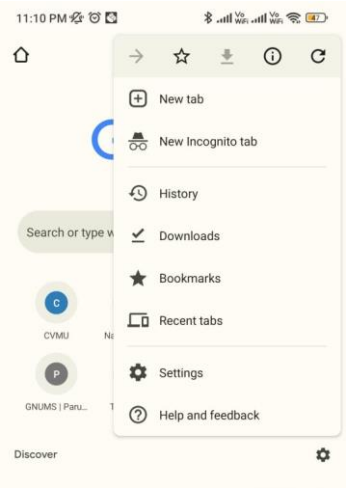
Here, getItemId() method will get the id of selected menu item based on that we can perform our actions.

Types of Menus

- Options Menu
- Context Menu
- Popup Menu

Android Options Menu

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.



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.

Example

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 (options_menu.xml) file to build the menu.

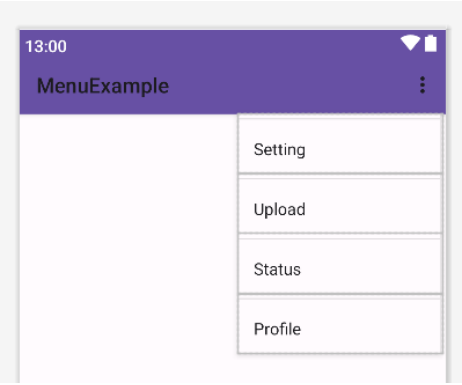
options_menu.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item android:id="@+id/status"
        android:title="Status" />
    <item android:id="@+id/setting"
        android:title="Setting" />
    <item android:id="@+id/profile"
        android:title="Profile" />
    <item android:id="@+id/logout"
        android:title="Logout" />
</menu>
```

MainActivity.java

```
package com.example.menuexample;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Menu;

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.options_menu, menu);
        return true;
    }
}
```



Android Context Menu

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.

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 will not support any item shortcuts and item icons.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <Button
        android:id="@+id/btnShow"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Long press me"
        android:layout_marginTop="200dp"
        android:layout_marginLeft="100dp"/>
</LinearLayout>
```

MainActivity.java

```
package com.example.menuexample;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.view.ContextMenu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button btn = (Button) findViewById(R.id.btnShow);
        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");
    }
    @Override
    public boolean onContextItemSelected(MenuItem item) {
        Toast.makeText(this, "Selected Item: " +item.getTitle(),
Toast.LENGTH_SHORT).show();
        return true;
    }
}
```

Android Popup Menu

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.

Popup Menu provides an overflow of actions that are related to specific content and the actions in the popup menu will not affect the corresponding content. The popup menu will not support any item shortcuts and item icons.

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

activity_main.xml

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

    <Button
        android:id="@+id/btnShow"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Show Popup Menu"
        android:layout_marginTop="200dp"
        android:layout_marginLeft="100dp"/>
</LinearLayout>
```

popup_menu.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item android:id="@+id/status"
        android:title="Status" />
    <item android:id="@+id/setting"
        android:title="Setting" />
    <item android:id="@+id/profile"
        android:title="Profile" />
    <item android:id="@+id/logout"
        android:title="Logout" />
</menu>
```

MainActivity.java

```
package com.example.menuexample;

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;

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.btnShow);
        btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                PopupMenu popup = new PopupMenu(MainActivity.this, v);
                popup.setOnMenuItemClickListener(MainActivity.this);
                popup.inflate(R.menu.popup_menu);
                popup.show();
            }
        });
    }

    @Override
    public boolean onMenuItemClick(MenuItem menuItem) {
        return false;
    }
}
```