Layouts in Android UI Design

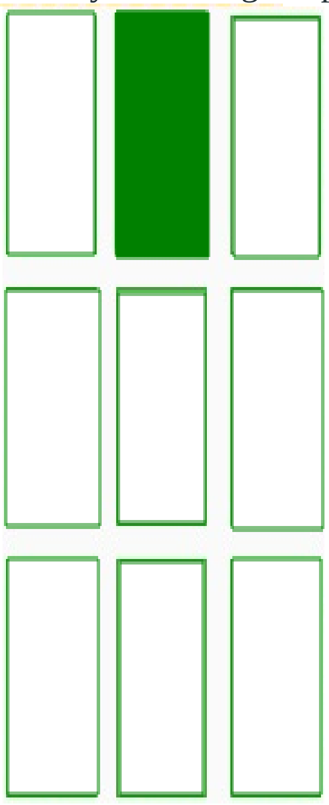
-

Layout Managers (or simply layouts) are said to be extensions of the ViewGroup class. They are used to set the position of child Views within the UI we are building. We can nest the layouts, and therefore we can create arbitrarily complex UIs using a combination of layouts.

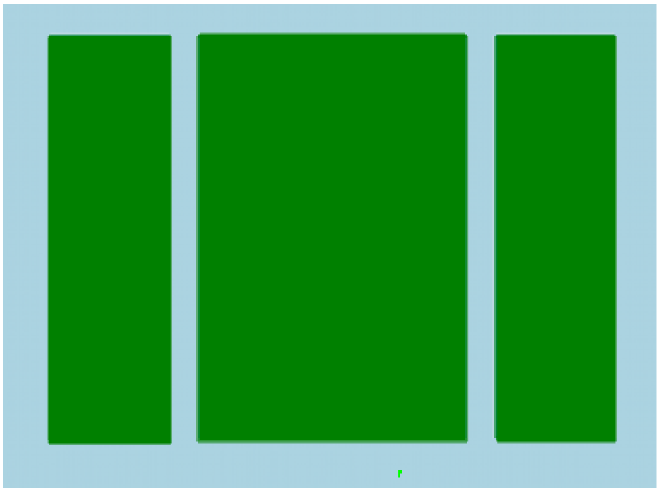
There is a number of layout classes in the Android SDK. They can be used, modified or can create your own to make the UI for your Views, Fragments and Activities. You can display your contents effectively by using the right combination of layouts.

The most commonly used layout classes that are found in Android SDK are:

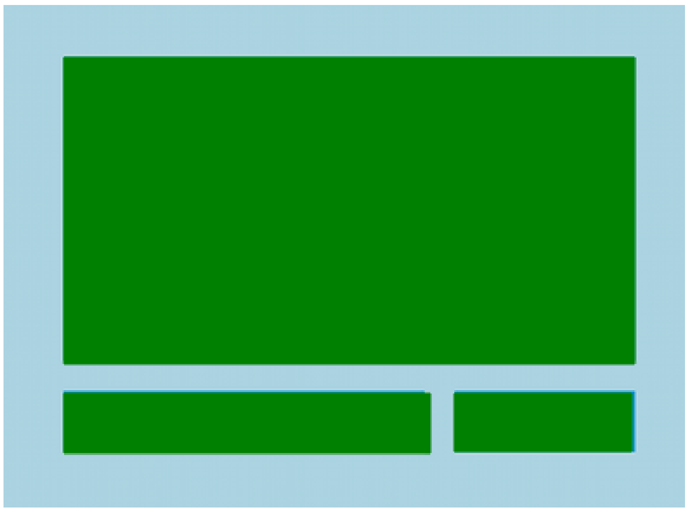
- **FrameLayout-** It is the simplest of the Layout Managers that pins each child view within its frame. By default the position is the top-left corner, though the gravity attribute can be used to alter its locations. You can add multiple children stacks each new child on top of the one before, with each new View potentially obscuring the previous ones.



- **LinearLayout-** A LinearLayout aligns each of the child View in either a vertical or a horizontal line. A vertical layout has a column of Views, whereas in a horizontal layout there is a row of Views. It supports a weight attribute for each child View that can control the relative size of each child View within the available space.



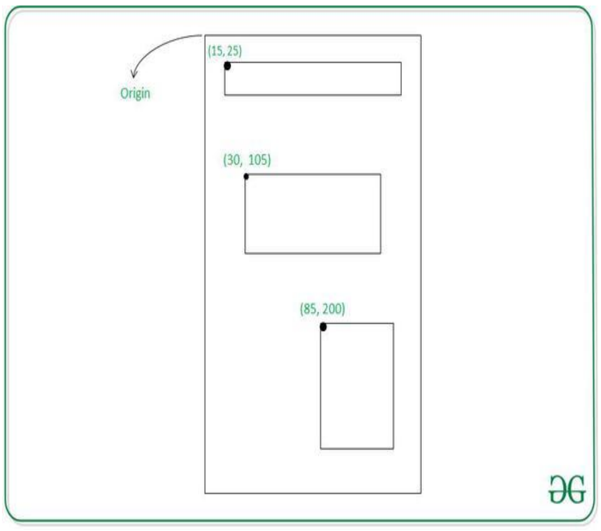
- **RelativeLayout-** It is flexible than other native layouts as it lets us to define the position of each child View relative to the other views and the dimensions of the screen.



An **Absolute Layout** allows you to specify the exact location i.e. X and Y coordinates of its children with respect to the origin at the top left corner of the layout. The absolute layout is less flexible and harder to maintain for varying sizes of screens that’s why it is not recommended. Although Absolute Layout is deprecated now.

Some of the important Absolute Layout attributes are the following:

1. **android:id**: It uniquely specifies the absolute layout
2. **android:layout_x:** It specifies X-Coordinate of the Views (Possible values of this are in density-pixel or pixel)
3. **android:layout_y:** It specifies Y-Coordinate of the Views (Possible values of this are in dp or px)



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.
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.
app:layout_constraintHeight_max	This attribute is used to set the max height of view according to the constraints.
app:layout_constraintHeight_min	This attribute is used to set the height of the view according to the constraints.
app:layout_constraintHorizontal_weight	This attribute is used to set the weight horizontally of a particular view same as linear layouts.
app:layout_constraintVertical_weight	This attribute is used to set the weight vertically of a particular view same as linear layouts.

Dynamic Layouts in Android

Generally, we develop the layout for an Android application by creating the XML file. These are called static layouts.
Static you ask? Yes, because you can not add/delete any View Type in XML on runtime.

Dynamic layouts are developed using Java and can be used to create layouts that you would normally create using an XML file.
Why do we need them?

Let’s say we are fetching some data from our server and we want to create n number of fields/buttons/options in our layout. How do we do that using the only XML? That’s right!
But we can use Java to replicate the exact layout that we would normally create using XML.

It can be used to create LinearLayout, ScrollView, etc.which can further add TextView, EditText, RadioButtons, Checkbox inside it.
Where do Dynamic Layouts excel?

Let’s say we want to show some items in our layout that were frequently bought together(all items are of the same View Type).

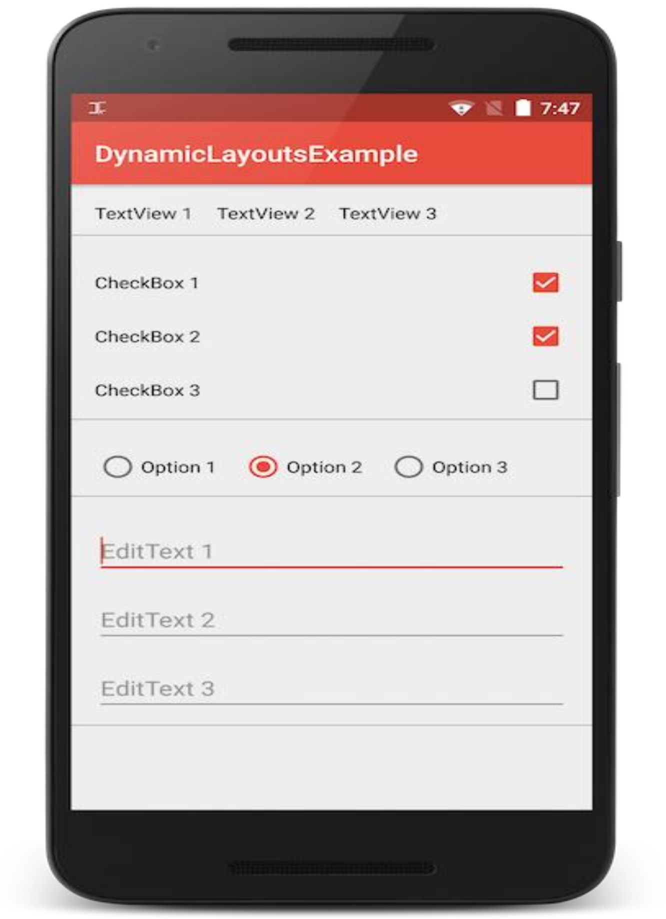
There are quite a few ways to show them in our layout:
1. Adding static views in our XML(if we know the exact amount of items).
2. Using Recycler View to inflate the items in our layout.
3. Creating Dynamic Views using Java.

What if we do not know the exact number of items to be displayed and our layout require different View Types(like 3 TextViews, 2 CheckBoxes, etc.)?

Possible solutions to tackle this problem:

1. Adding static views in our XML won’t work this time because we do not . know the exact number of views we need.
2. Using Recycler View could work but the same View Types should be grouped together in a list. There is not much flexibility in this case.
3. However, this is where Dynamic Layouts take the lead! They are flexible and can add multiple View Types to our layout in any order.

Dynamic Layout containing TextViews, CheckBoxes, RadioButtons, EditTexts.



How to create Dynamic Layouts

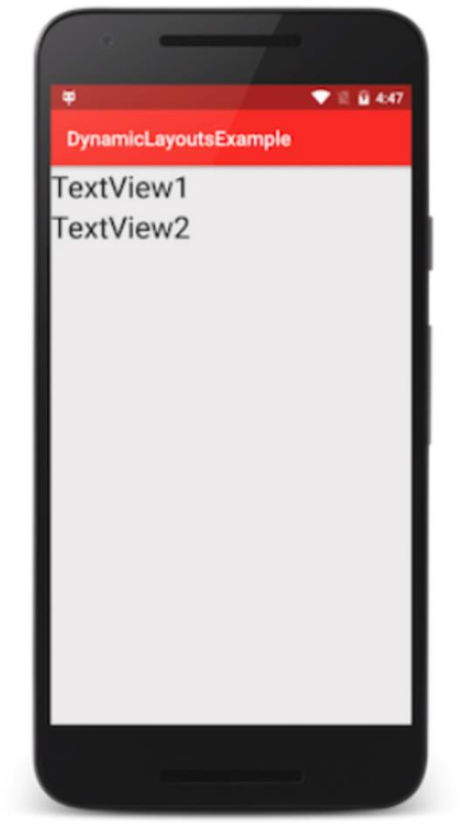
A simple layout in XML containing only TextViews

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

<LinearLayout

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

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

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

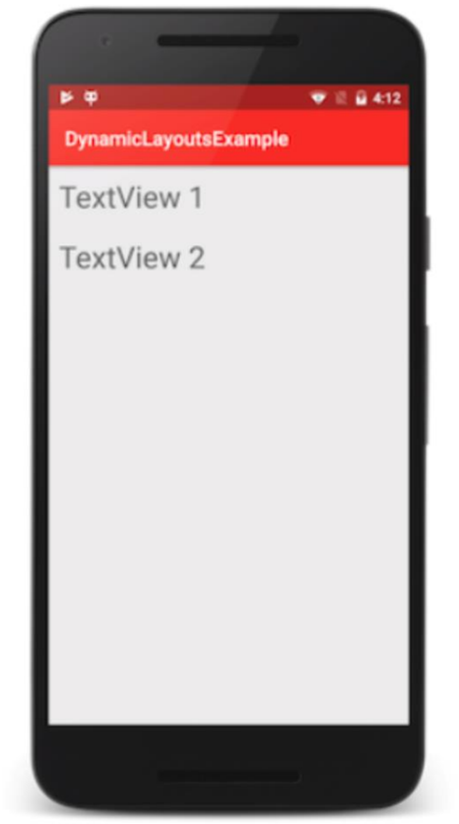
```
android:layout_width="match_parent"

android:layout_height="match_parent"

tools:context=".MainActivity"

android:orientation="vertical">



<TextView

    android:layout_width="wrap_content"

    android:layout_height="wrap_content"

    android:padding="10dp"

    android:text="TextView 1"

    android:textSize="30dp"/>



<TextView

    android:layout_width="wrap_content"

    android:layout_height="wrap_content"

    android:padding="10dp"

    android:text="TextView 2"

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

Same Dynamic Layout created in Java

```
public class MainActivity extends AppCompatActivity {


    LinearLayout linearLayout;


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


        linearLayout = findViewById(R.id.linear_layout);        //Adding 2 TextViews
        for (int i = 1; i <= 2; i++) {
            TextView textView = new TextView(this);
            textView.setText("TextView " + String.valueOf(i));
            linearLayout.addView(textView);
        }
   }
}
```



Dynamic Layout using Java
(No Padding for now)



Using XML

Android Widgets

There are given a lot of **android widgets** with simplified examples such as Button, EditText, AutoCompleteTextView, ToggleButton, DatePicker, TimePicker, ProgressBar etc.

Android widgets are easy to learn. The widely used android widgets with examples are given below:

Android Button

Let's learn how to perform event handling on button click.

Android Toast

Displays information for the short duration of time.

Custom Toast

We are able to customize the toast, such as we can display image on the toast

ToggleButton

It has two states ON/OFF.

CheckBox

Let's see the application of simple food ordering.

AlertDialog

AlertDialog displays a alert dialog containing the message with OK and Cancel buttons.

Spinner

Spinner displays the multiple options, but only one can be selected at a time.

AutoCompleteTextView

Let's see the simple example of AutoCompleteTextView.

RatingBar

RatingBar displays the rating bar.

DatePicker

Datepicker displays the datepicker dialog that can be used to pick the date.

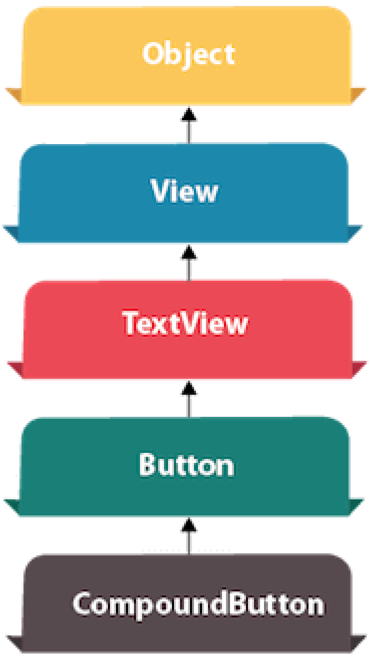
TimePicker

TimePicker displays the timepicker dialog that can be used to pick the time.

ProgressBar

ProgressBar displays progress task.

Android Button Example



Android Button represents a push-button. The android.widget.Button is subclass of TextView class and CompoundButton is the subclass of Button class.

There are different types of buttons in android such as RadioButton, ToggleButton, CompoundButton etc.

Android Button Example with Listener

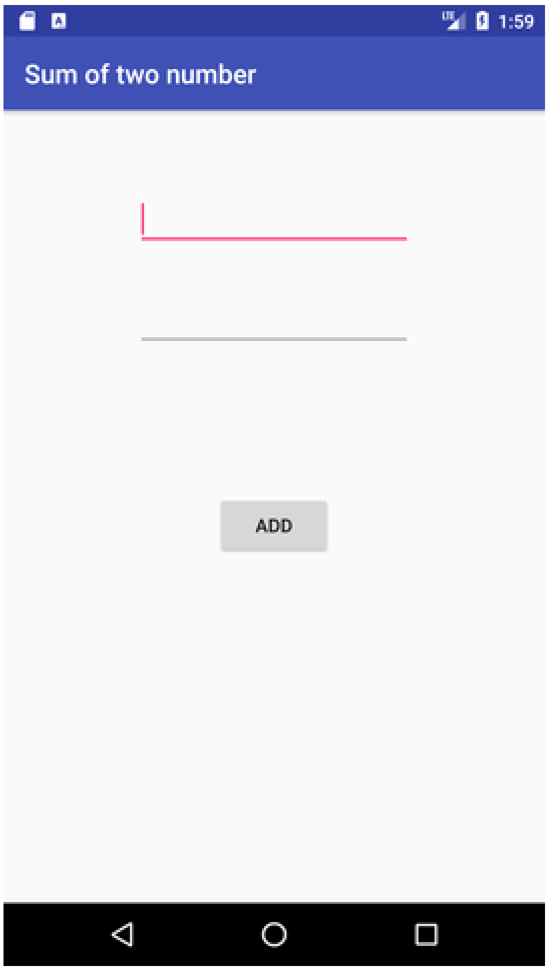
Here, we are going to create two textfields and one button for sum of two numbers. If user clicks button, sum of two input values is displayed on the Toast.

We can perform action on button using different types such as calling listener on button or adding onClick property of button in activity's xml file.

```
button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
               //code
            }
});
<Button
        android:onClick="methodName"
/>
```

Drag the component or write the code for UI in activity_main.xml

First of all, drag 2 textfields from the Text Fields palette and one button from the Form Widgets palette as shown in the following figure.



The generated code for the ui components will be like this:

File: activity_main.xml

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

    <EditText
        android:id="@+id/editText1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="61dp"
        android:ems="10"
        android:inputType="number"
        tools:layout_editor_absoluteX="84dp"
        tools:layout_editor_absoluteY="53dp" />

    <EditText
```

```
        android:id="@+id/editText2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/editText1"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="32dp"
        android:ems="10"
        android:inputType="number"
        tools:layout_editor_absoluteX="84dp"
        tools:layout_editor_absoluteY="127dp" />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/editText2"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="109dp"
        android:text="ADD"
        tools:layout_editor_absoluteX="148dp"
        tools:layout_editor_absoluteY="266dp" />
</RelativeLayout>
```

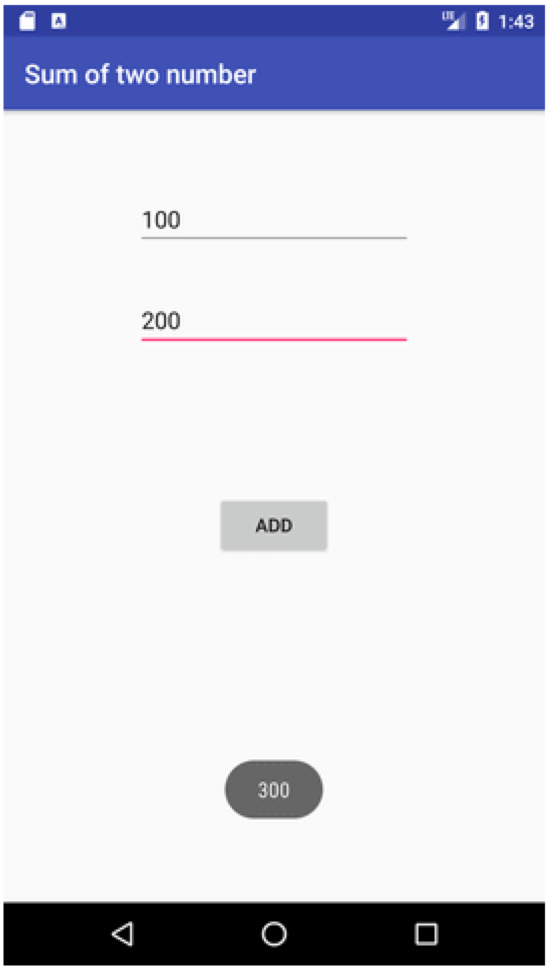
Activity class

Now write the code to display the sum of two numbers.

File: MainActivity.java

```
package example.javatpoint.com.sumoftwonumber;

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

public class MainActivity extends AppCompatActivity {
    private EditText edittext1, edittext2;
    private Button buttonSum;

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

        addListenerOnButton();
    }

    public void addListenerOnButton() {
        edittext1 = (EditText) findViewById(R.id.editText1);
        edittext2 = (EditText) findViewById(R.id.editText2);
        buttonSum = (Button) findViewById(R.id.button);
        buttonSum.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                String value1=edittext1.getText().toString();
                String value2=edittext2.getText().toString();
                int a=Integer.parseInt(value1);
                int b=Integer.parseInt(value2);
                int sum=a+b;
             Toast.makeText(getApplicationContext(),String.valueOf(sum), Toast.LENGTH_LONG).show();
            }
        });
    }
}
```

Output:



How to Create an Alert Dialog Box in Android?

•

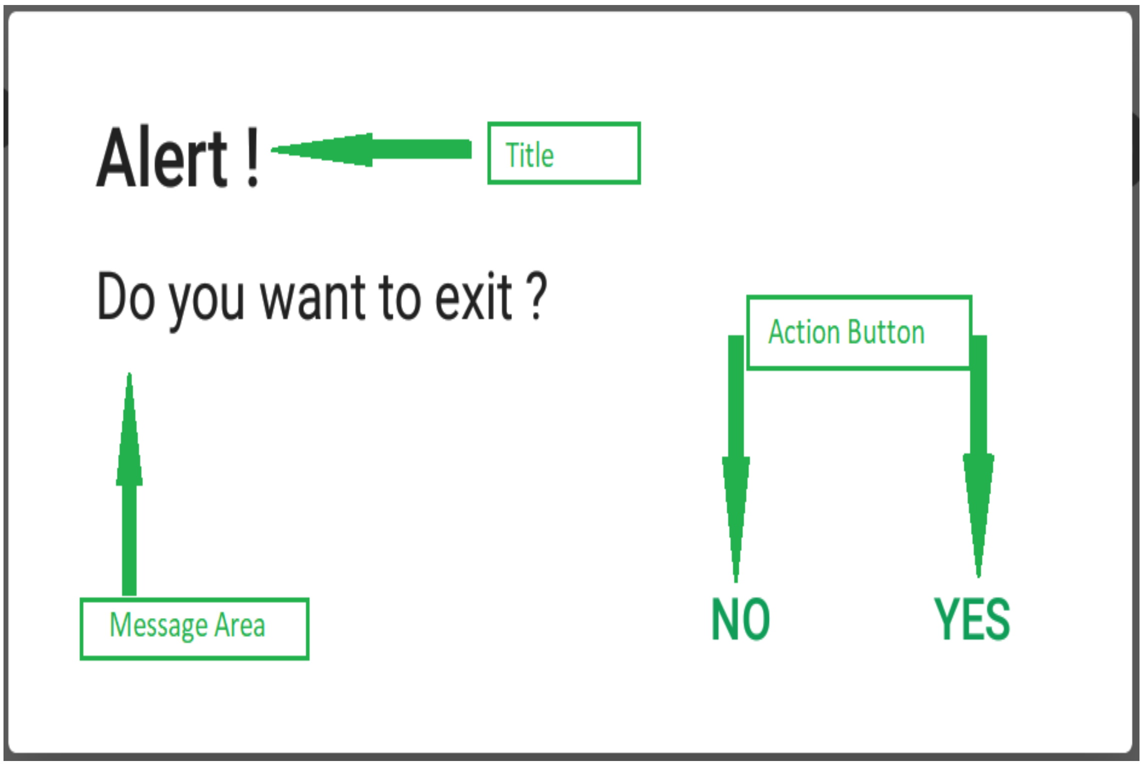
Alert Dialog shows the Alert message and gives the answer in the form of yes or no. Alert Dialog displays the message to warn you and then according to your response, the next step is processed. Android Alert Dialog is built with the use of three fields: **Title, Message area, and Action Button.**

Alert Dialog code has three methods :

- **setTitle()** method for displaying the Alert Dialog box Title
- **setMessage()** method for displaying the message
- **setIcon()** method is used to set the icon on the Alert dialog box.

Then we add the two Buttons, **setPositiveButton** and **setNegativeButton** to our Alert Dialog Box as shown below.

Example:



Looking to become an expert in Android App Development? Whether you're a student or a professional aiming to advance your career in mobile app development, our course, "**Android App Development with Kotlin**," available exclusively on GeeksforGeeks, is the perfect fit for you.

Step By Step Implementation

Step 1: Create a New Project in Android Studio

To create a new project in Android Studio please refer to **How to Create/Start a New Project in Android Studio** . The code for that has been given in both **Java and Kotlin Programming Language for Android.**

Step 2: Working with the XML Files

Next, go to the **activity_main.xml file** , which represents the UI of the project. Below is the code for the **activity_main.xml** file. Comments are added inside the code to understand the code in more detail.

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

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="180dp"
        android:gravity="center_horizontal"
        android:text="Press The Back Button of Your Phone."
        android:textSize="30dp"
        android:textStyle="bold" />
</RelativeLayout>
```

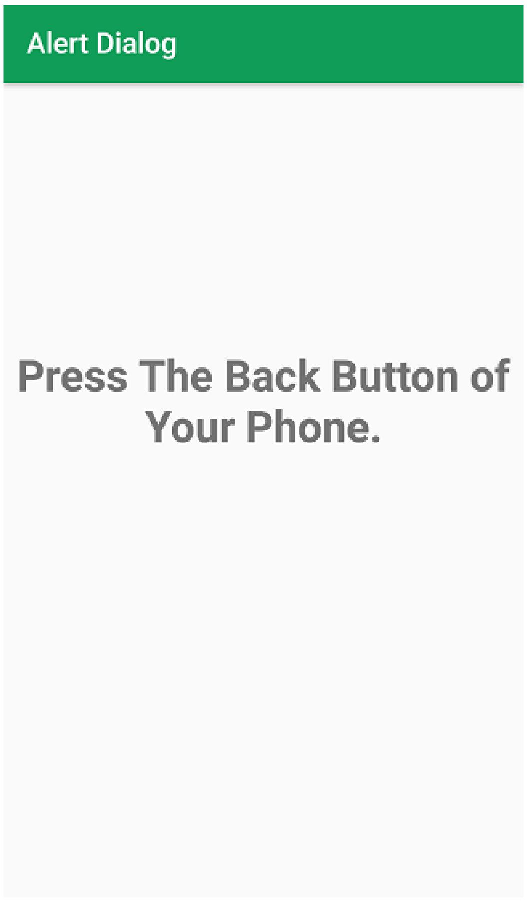
Step 3: Working with the MainActivity File

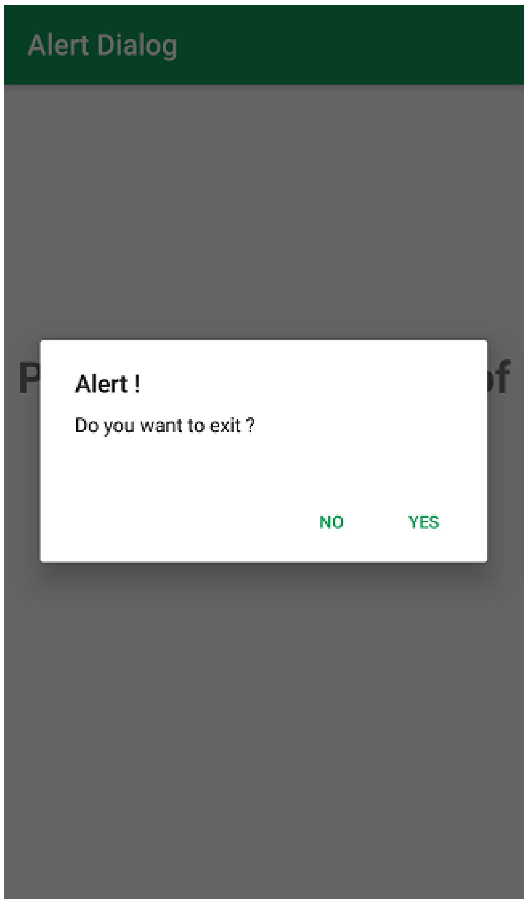
Go to the MainActivity File and refer to the following code. Below is the code for the MainActivity File. Comments are added inside the code to understand the code in more detail.

JavaKotlin

```
import android.content.DialogInterface;
import android.os.Bundle;
import androidx.appcompat.app.AlertDialog;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

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

    // Declare the onBackPressed method when the back button is pressed this
method will call
    @Override
    public void onBackPressed() {
        // Create the object of AlertDialog Builder class
        AlertDialog.Builder builder = new
AlertDialog.Builder(MainActivity.this);

        // Set the message show for the Alert time
        builder.setMessage("Do you want to exit ?");
        // Set Alert Title
        builder.setTitle("Alert !");

        // Set Cancelable false for when the user clicks on the outside the
Dialog Box then it will remain show
        builder.setCancelable(false);

        // Set the positive button with yes name Lambda OnClickListener method
is use of DialogInterface interface.
        builder.setPositiveButton("Yes", (DialogInterface.OnClickListener)
(dialog, which) -> {
            // When the user click yes button then app will close
            finish();
        });

        // Set the Negative button with No name Lambda OnClickListener method
is use of DialogInterface interface.
        builder.setNegativeButton("No", (DialogInterface.OnClickListener)
(dialog, which) -> {
            // If user click no then dialog box is canceled.
            dialog.cancel();
        });

        // Create the Alert dialog
        AlertDialog alertDialog = builder.create();
        // Show the Alert Dialog box
        alertDialog.show();
    }
}
```

Output:





Context Menu in Android with Example

In Android, there are three types of **menus** available to define a set of options and actions in the Android apps. The lists of menus in Android applications are the following:

- Android options menu
- Android context menu
- Android popup menu

Here in this article let’s discuss the detail of the **Context Menu**. In Android, the context menu is like a floating menu and arises when the user has long-pressed or clicked on an item and is beneficial for implementing functions that define the specific content or reference frame effect. The Android context menu is alike to the right-click menu in Windows or Linux. In the Android system, the context menu provides actions that change a specific element or context frame in the user interface and one can provide a context menu for any view. The context menu will not support any object shortcuts and object icons. A sample GIF is given below to get an idea about what we are going to do in this article.

Step By Step Implementation

Step 1: Create a New Project in Android Studio

To create a new project in Android Studio please refer to **How to Create/Start a New Project in Android Studio**. The code for that has been given in both **Java and Kotlin Programming Language for Android.**

Looking to become an expert in Android App Development? Whether you're a student or a professional aiming to advance your career in mobile app development, our course, "**Android App Development with Kotlin**," available exclusively on GeeksforGeeks, is the perfect fit for you.

Step 2: Working with the XML Files

Open **res -> Layout -> activity_main.xml** and write the following code. In this file add only a **TextView** to display a simple text.

```xml
<?xml version="1.0" encoding="utf-8"?>
<!-- Relative Layout to display all the details -->
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/relLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#fff"
    android:padding="16dp"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="20dp"
        android:text="Long press me!"
        android:textColor="#000"
        android:textSize="20sp"
        android:textStyle="bold" />
</RelativeLayout>
```

Step 3: Working with the MainActivity file

Open the **app -> Java -> Package -> Mainactivity.java** file. In this step, add the code to show the ContextMenu. Whenever the app will start make a long click on a text and display the number of options to select of them for specific purposes. Comments are added inside the code to understand the code in more detail.

JavaKotlin

```java
import android.graphics.Color;
import android.os.Bundle;
import android.view.ContextMenu;
import android.view.MenuItem;
import android.view.View;
import android.widget.RelativeLayout;
import android.widget.TextView;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {
    TextView textView;
    RelativeLayout relativeLayout;

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

        // Link those objects with their respective id's that we have given in
.XML file
        textView = (TextView) findViewById(R.id.textView);
        relativeLayout = (RelativeLayout) findViewById(R.id.relLayout);

        // here you have to register a view for context menu you can register
any view
          // like listview, image view, textview, button etc
        registerForContextMenu(textView);

    }

    @Override
    public void onCreateContextMenu(ContextMenu menu, View v,
ContextMenu.ContextMenuInfo menuInfo) {
        super.onCreateContextMenu(menu, v, menuInfo);
        // you can set menu header with title icon etc
        menu.setHeaderTitle("Choose a color");
        // add menu items
        menu.add(0, v.getId(), 0, "Yellow");
        menu.add(0, v.getId(), 0, "Gray");
        menu.add(0, v.getId(), 0, "Cyan");
    }

    // menu item select listener
    @Override
    public boolean onContextItemSelected(MenuItem item) {
        if (item.getTitle() == "Yellow") {
            relativeLayout.setBackgroundColor(Color.YELLOW);
        } else if (item.getTitle() == "Gray") {
            relativeLayout.setBackgroundColor(Color.GRAY);
        } else if (item.getTitle() == "Cyan") {
            relativeLayout.setBackgroundColor(Color.CYAN);
        }
        return true;
    }
}
```
Output: Run on Emulator

Now connect the device with a USB cable or in an Emulator and launch the application. The user will see a text. Now long pressing on the text will generate menu options and select one of them to perform specific functionality.

Video Player