UNIT-2

Designing Application

2.1 Working with activity

In Android, an activity is referred to as one screen in an application. It is very similar to a single window of any desktop application. An Android app consists of one or more screens or activities.

Each activity goes through various stages or a lifecycle and is managed by activity stacks. So when a new activity starts, the previous one always remains below it. There are four stages of an activity.

If an activity is in the foreground of the screen i.e at the top of the stack, then it is said to be active or running. This is usually the activity that the user is currently interacting with.

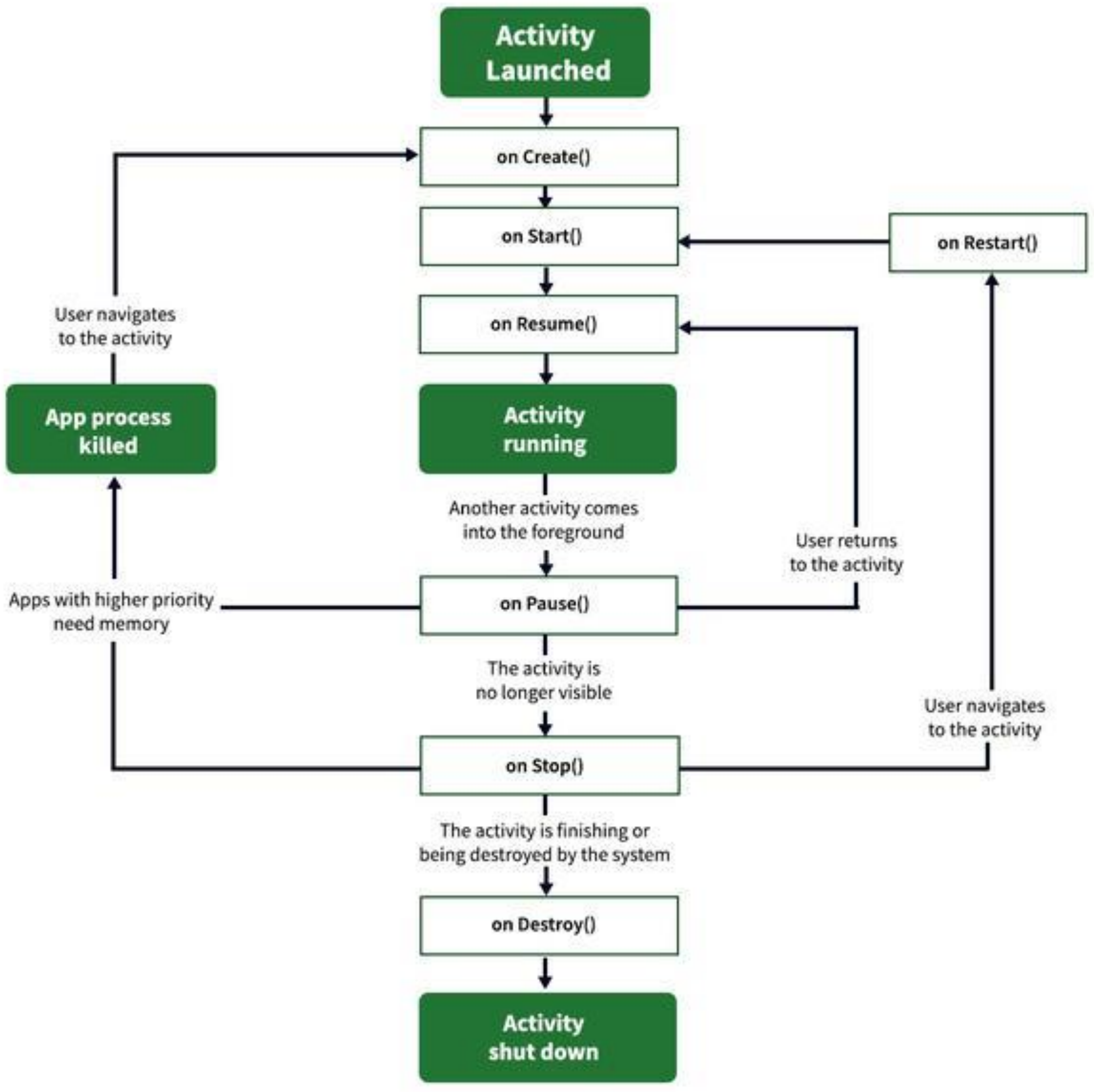
If an activity has lost focus and a non-full-sized or transparent activity has focused on top of your activity. In such a case either another activity has a higher position in multi-window mode or the activity itself is not focusable in the current window mode. Such activity is completely alive.

If an activity is completely hidden by another activity, it is stopped or hidden. It still retains all the information, and as its window is hidden thus it will often be killed by the system when memory is needed elsewhere.

The system can destroy the activity from memory by either asking it to finish or simply killing its process. When it is displayed again to the user, it must be completely restarted and restored to its previous state.

For each stage, android provides us with a set of 7 methods that have their own significance for each stage in the life cycle. The image shows a path of migration whenever an app switches from one state to another.

Activity Lifecycle in Android:


Activity Launched
on Create()
on Start()
on Restart()
on Resume()
User navigates to the activity
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
on Pause()
The activity is no longer visible
User navigates to the activity
on Stop()
The activity is finishing or being destroyed by the system
on Destroy()
Activity shut down

Activity Lifecycle in Android

There are 7 different methods in activity lifecycle: –

1. **onCreate()**: It is called when the activity is first created. This is where all the static work is done like creating views, binding data to lists, etc. This method also provides a Bundle containing its previous frozen state, if there was one.

2. **onStart():** It is invoked when the activity is visible to the user. It is followed by onResume() if the activity is invoked from the background. It is also invoked after onCreate() when the activity is first started.

3. **onRestart():** It is invoked after the activity has been stopped and prior to its starting stage and thus is always followed by onStart() when any activity is revived from background to on-screen.

4. **onResume():** It is invoked when the activity starts interacting with the user. At this point, the activity is at the top of the activity stack, with a user interacting with it. Always followed by onPause() when the activity goes into the background or is closed by the user.

5. **onPause():** It is invoked when an activity is going into the background but has not yet been killed. It is a counterpart to onResume(). When an activity is launched in front of another activity, this callback will be invoked on the top activity (currently on screen). The activity, under the active activity, will not be created until the active activity's onPause() returns, so it is recommended that heavy processing should not be done in this part.
6. **onStop():** It is invoked when the activity is not visible to the user. It is followed by onRestart() when the activity is revoked from the background, followed by onDestroy() when the activity is closed or finished, and nothing when the activity remains on the background only.
7. **onDestroy():**The final call received before the activity is destroyed. This can happen either because the activity is finishing (when finish() is invoked) or because the system is temporarily destroying this instance of the activity to save space.

Example:

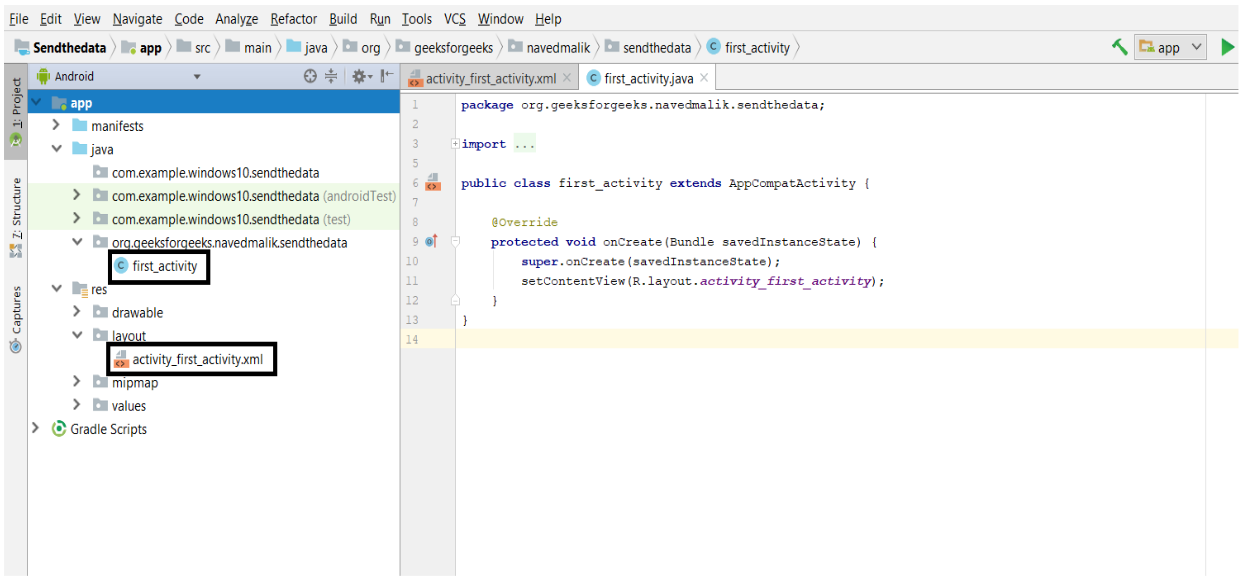
```
import android.os.Bundle
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
class MainActivity : AppCompatActivity() {
     override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
          // Bundle containing previous frozen state
        setContentView(R.layout.activity_main)
         // The content view pointing to the id of layout in the file activity_main.xml
        val toast = Toast.makeText(applicationContext, "onCreate Called",
Toast.LENGTH_LONG).show()
        }
    override fun onStart() {
        super.onStart()
        // It will show a message on the screen then onStart is invoked
        val toast  = Toast.makeText(applicationContext, "onStart Called",
Toast.LENGTH_LONG).show()
    }
    override fun onRestart() {
        super.onRestart()
        // It will show a message on the screen then onRestart is invoked
        val toast = Toast.makeText(applicationContext, "onRestart Called",
Toast.LENGTH_LONG).show()
    }
    override fun onResume() {
        super.onResume()
        // It will show a message on the screen then onResume is invoked
        val toast = Toast.makeText(applicationContext, "onResume Called",
Toast.LENGTH_LONG).show()
    }
    override fun onPause() {
        super.onPause()
        // It will show a message on the screen then onPause is invoked
        val toast = Toast.makeText(applicationContext, "onPause Called",
Toast.LENGTH_LONG).show()
    }
    override fun onStop() {
        super.onStop()
        // It will show a message on the screen then onStop is invoked
        val toast = Toast.makeText(applicationContext, "onStop Called",
Toast.LENGTH_LONG).show()
    }
    override fun onDestroy() {
        super.onDestroy()
        // It will show a message on the screen then onDestroy is invoked
        val toast = Toast.makeText(applicationContext, "onDestroy Called",
Toast.LENGTH_LONG).show()
    }
}
```

2.2 Redirecting to another activity and passing data

How to "**Send the data from one activity to second activity using Intent**". In this example, we have two activities, activity_first which are the source activity, and activity_second which is the destination activity. We can send the data using the putExtra() method from one activity and get the data from the second activity using the getStringExtra() method.

Step by Step Implementation:

Step1: Create a New Project in Android Studio


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Sendthedata app src main java org geeksforgeeks navedmalik sendthedata first_activity
Android
app
manifests
java
com.example.windows10.sendthedata
com.example.windows10.sendthedata (androidTest)
com.example.windows10.sendthedata (test)
org.geeksforgeeks.navedmalik.sendthedata
first_activity
res
drawable
layout
activity_first_activity.xml
mipmap
values
Gradle Scripts
activity_first_activity.xml
first_activity.java

```
package org.geeksforgeeks.navedmalik.sendthedata;

import ...

public class first_activity extends AppCompatActivity {

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

Step2: 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. Open the "activity_first_activity.xml" file and add the following widgets in a **Relative Layout**.

- An **EditText** to Input the message
- A **Button** to send the data

Also, Assign the **ID** to each component along with other attributes as shown in the image and the code below. The assigned ID on a component helps that component to be easily found and used in the Java/Kotlin files.

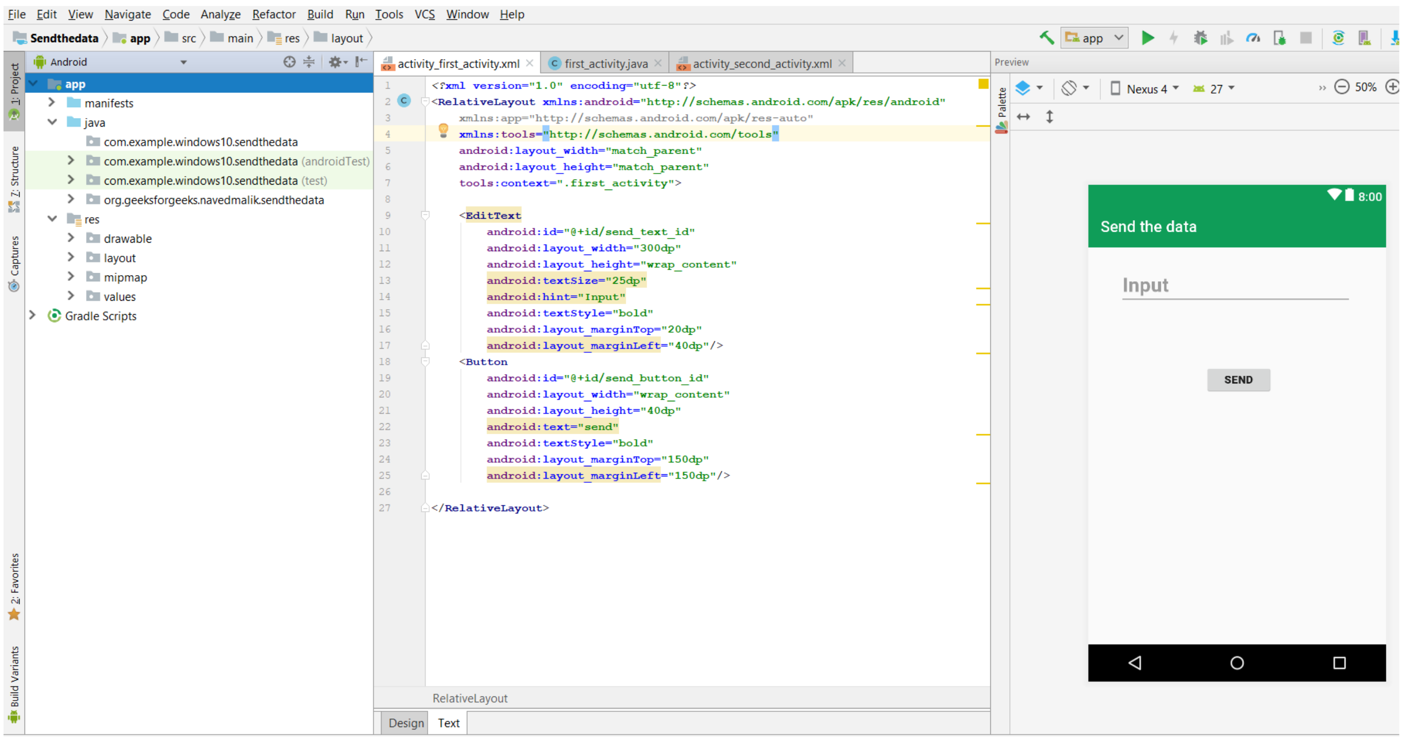
Syntax:

- android:id="@+id/id_name"
- Send Button: send_button_id
- input EditText: send_text_id

XML:

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

    <EditText
        android:id="@+id/send_text_id"
        android:layout_width="300dp"
        android:layout_height="wrap_content"
        android:layout_marginLeft="40dp"
        android:layout_marginTop="20dp"
        android:hint="Input"
        android:textSize="25dp"
        android:textStyle="bold" />

    <Button
        android:id="@+id/send_button_id"
        android:layout_width="wrap_content"
        android:layout_height="40dp"
        android:layout_marginLeft="150dp"
        android:layout_marginTop="150dp"
        android:text="send"
        android:textStyle="bold" />
</RelativeLayout>
```

This will make the UI of the Application



```
<?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=".first_activity">

    <EditText
        android:id="@+id/send_text_id"
        android:layout_width="300dp"
        android:layout_height="wrap_content"
        android:textSize="25dp"
        android:hint="Input"
        android:textStyle="bold"
        android:layout_marginTop="20dp"
        android:layout_marginLeft="40dp"/>
    <Button
        android:id="@+id/send_button_id"
        android:layout_width="wrap_content"
        android:layout_height="40dp"
        android:text="send"
        android:textStyle="bold"
        android:layout_marginTop="150dp"
        android:layout_marginLeft="150dp"/>

</RelativeLayout>
```

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Sendthedata > app > src > main > res > layout
activity_first_activity.xml
first_activity.java
activity_second_activity.xml
Android
app
manifests
java
com.example.windows10.sendthedata
com.example.windows10.sendthedata (androidTest)
com.example.windows10.sendthedata (test)
org.geeksforgeeks.navedmalik.sendthedata
res
drawable
layout
mipmap
values
Gradle Scripts
Preview
Nexus 4
27
50%
8:00
Send the data
Input
SEND
RelativeLayout
Design
Text

Step3: 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. Now, after the UI, this step will create the Backend of the App. For this, open the “first_activity” file and instantiate the components made in the XML file (EditText, send Button) using findViewById() method. This method binds the created object to the UI Components with the help of the assigned ID.

Syntax: General

```
ComponentType object = (ComponentType)findViewById(R.id.IdOfTheComponent);
```

Syntax: for components used is as follows:

Button send_button= findViewById(R.id.send_button_id);

send_text = findViewById(R.id.send_text_id);

Setting up the Operations for the Sending and Receiving of Data.

These Operations are as follows:

Add the listener to the send button and this button will send the data.

This is done as follows:

send_button.setOnClickListener(v -> {}

after clicking this button following operation will be performed. Now create the String type variable to store the value of EditText which is input by the user. Get the value and convert it to a string.

This is done as follows:

String str = send_text.getText().toString();

Now create the Intent object First_activity.java class to Second_activity class.

This is done as follows:

Intent intent = new Intent(getApplicationContext(), Second_activity.class);

where getApplicationContext() will fetch the current activity. Put the value in the putExtra method in the key-value pair then start the activity.

This is done as follows:

intent.putExtra("message_key", str);

startActivity(intent);

where “str” is the string value and the key is “message_key” this key will use to get the str value

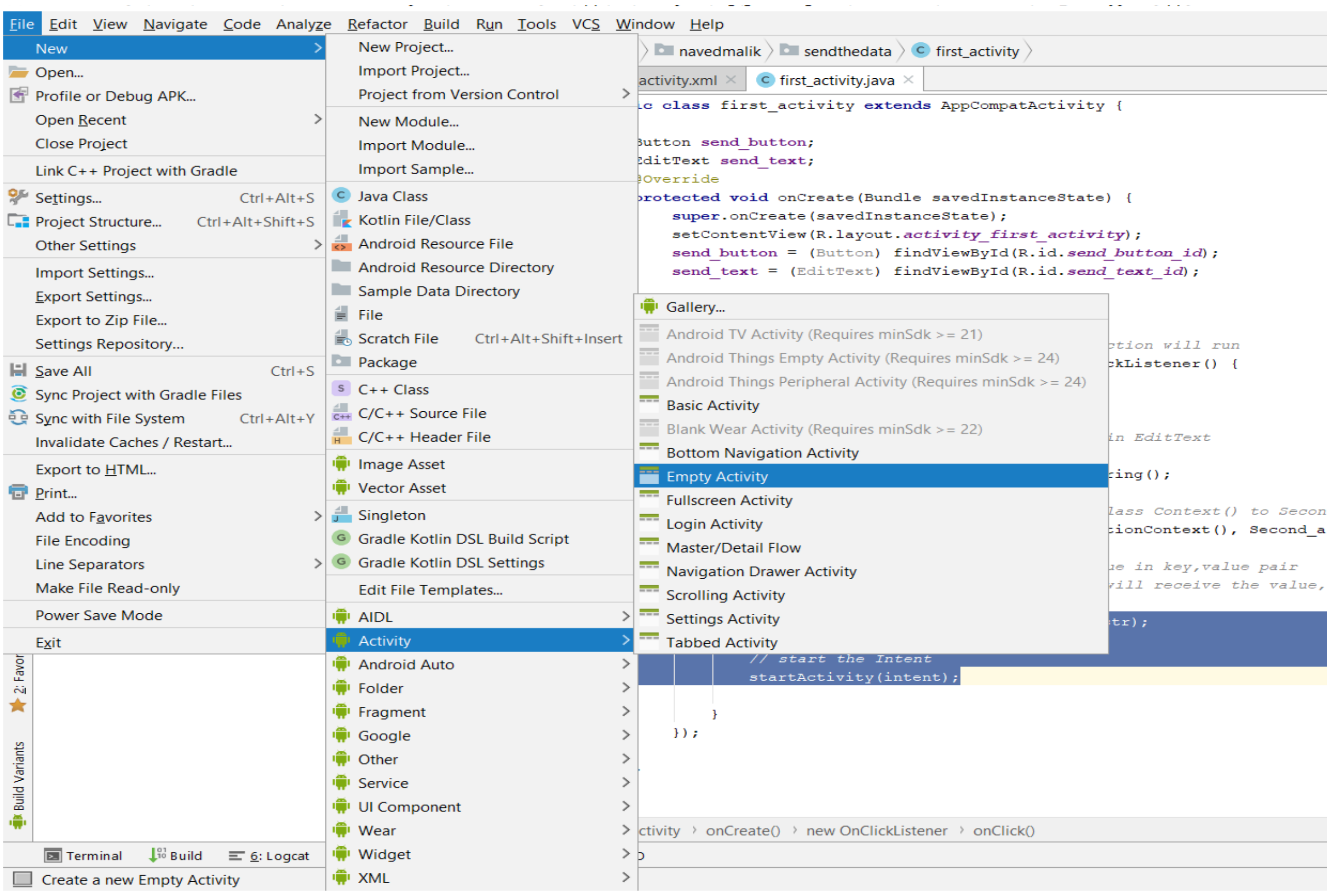
Example:

```
import android.content.Intent
import android.os.Bundle
import android.widget.Button
import android.widget.EditText
import androidx.appcompat.app.AppCompatActivity

class first_activity : AppCompatActivity() {

    // define the variable
    lateinit var send_button: Button
    lateinit var send_text: EditText

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_first_activity)
        send_button = findViewById(R.id.send_button_id)
        send_text = findViewById(R.id.send_text_id)

        // add the OnClickListener in sender button after clicked this button following Instruction will run
        send_button.setOnClickListener {
            // get the value which input by user in EditText and convert it to string
            val str = send_text.text.toString()
            // Create the Intent object of this class Context() to Second_activity class
            val intent = Intent(applicationContext, Second_activity::class.java)
            // now by putExtra method put the value in key, value pair key is
            // message_key by this key we will receive the value, and put the string
            intent.putExtra("message_key", str)
            // start the Intent
            startActivity(intent)
        }
    }
}
```

Step4: Creating Second_Activity to Receive the Data.

The steps to create the second activity are as follows:

android project > File > new > Activity > Empty Activity



```
c class first_activity extends AppCompatActivity {

utton send_button;
ditText send_text;
Override
rotected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_first_activity);
    send_button = (Button) findViewById(R.id.send_button_id);
    send_text = (EditText) findViewById(R.id.send_text_id);


                                                          tion will run
                                                          kListener() {


                                                          in EditText
                                                          ing();
                                                          lass Context() to Secon
                                                          ionContext(), Second_a
                                                          e in key,value pair
                                                          ill receive the value,
                                                          tr);
            // start the Intent
            startActivity(intent);
        }
    });
```

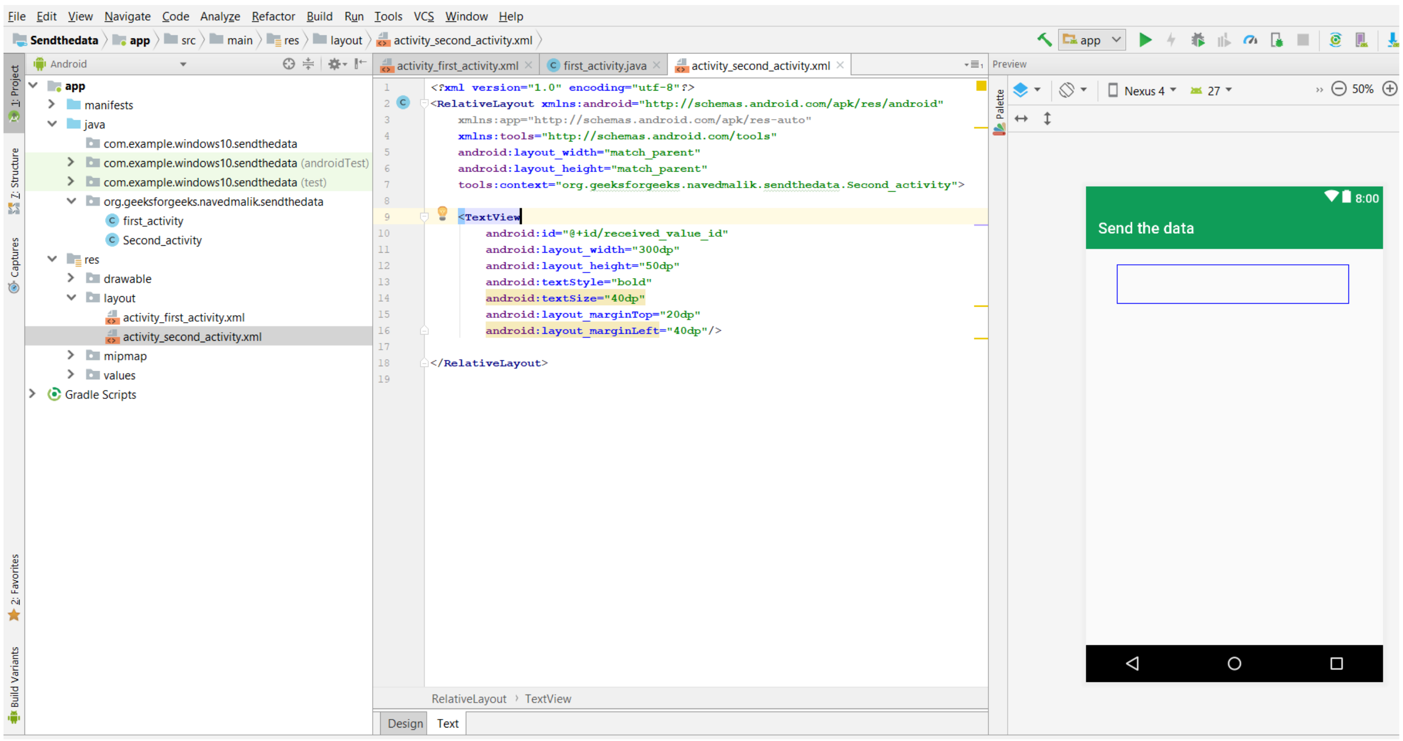
Step5: Working with the Second XML File

Add TextView to display the received messages. assign an ID to Textview.

XML:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".Second_activity">
    <TextView
        android:id="@+id/received_value_id"
        android:layout_width="300dp"
        android:layout_height="50dp"
        android:layout_marginLeft="40dp"
        android:layout_marginTop="20dp"
        android:textSize="40sp"
        android:textStyle="bold"
        android:layout_marginStart="40dp" />
</RelativeLayout>
```

The Second Activity is shown below:



```
<?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="org.geeksforgeeks.navedmalik.sendthedata.Second_activity">

    <TextView
        android:id="@+id/received_value_id"
        android:layout_width="300dp"
        android:layout_height="50dp"
        android:textStyle="bold"
        android:textSize="40dp"
        android:layout_marginTop="20dp"
        android:layout_marginLeft="40dp"/>

</RelativeLayout>
```

Step6: Working with the SecondActivity File

Define the TextView variable, use findViewById() to get the TextView as shown above.

receiver_msg = (TextView) findViewById(R.id.received_value_id);

Now In the second_activity.java file create the object of getIntent to receive the value in String type variable by the getStringExtra method using message_key.

```
Intent intent = getIntent();
String str = intent.getStringExtra("message_key");
```

The received value set in the TextView object of the second activity XML file

```
receiver_msg.setText(str);
```

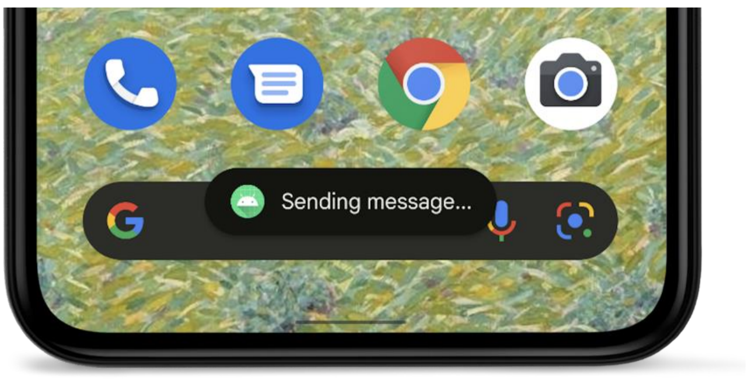
Kotlin:

```
import android.os.Bundle
import android.widget.TextView
import androidx.appcompat.app.AppCompatActivity

class Second_activity : AppCompatActivity() {

    lateinit var receiver_msg: TextView

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_second_activity)

        receiver_msg = findViewById(R.id.received_value_id)
        // create the get Intent object
        val intent = intent
        // receive the value by getStringExtra() method and
        // key must be same which is send by first activity
        val str = intent.getStringExtra("message_key")
        // display the string into textView
        receiver_msg.text = str
    }
}
```

Toast

A toast provides simple feedback about an operation in a small popup. It only fills the amount of space required for the message and the current activity remains visible and interactive. Toasts automatically disappear after a timeout.

For example, clicking Send on an email triggers a "Sending message..." toast, as shown in the following screen capture:



If your app targets Android 12 (API level 31) or higher, its toast is limited to two lines of text and shows the application icon next to the text. Be aware that the line length of this text varies by screen size, so it's good to make the text as short as possible.

Instantiate a Toast object

Use the makeText() method, which takes the following parameters:

1. The activity Context.
2. The text that should appear to the user.
3. The duration that the toast should remain on the screen.

The makeText() method returns a properly initialized Toast object.

Show the toast

To display the toast, call the show() method, as demonstrated in the following example:

```
val text = "Hello toast!"
val duration = Toast.LENGTH_SHORT

val toast = Toast.makeText(this, text, duration) // in Activity
toast.show()
```

2.3 Layouts:

Android **Layout** is used to define the user interface that holds the UI controls or widgets that will appear on the screen of an android application or activity screen. Generally, every application is a combination of View and ViewGroup. As we know, an android application contains a large number of activities and we can say each activity is one page of the application. So, each activity contains multiple user interface components and those components are the instances of the View and ViewGroup. All the elements in a layout are built using a hierarchy of **View** and **ViewGroup** objects.

Size, padding, and margins

The size of a view is expressed with a width and height. A view has two pairs of width and height values.

The first pair is known as *measured width* and *measured height*. Dimensions define how big a view wants to be within its parent. You can obtain measured dimensions by calling getMeasuredWidth() and getMeasuredHeight().

The second pair is known as *width* and *height*, or sometimes *drawing width* and *drawing height*. These dimensions define the actual size of the view on screen, at drawing time and after layout. These values might, but don't have to, differ from the measured width and height. You can obtain the width and height by calling getWidth() and getHeight().

To measure its dimensions, a view takes into account its padding. The padding is expressed in pixels for the left, top, right and bottom parts of the view. You can use padding to offset the content of the view by a specific number of pixels. For instance, a left padding of two pushes the view's content two pixels to the right of the left edge. You can set padding using the setPadding(int, int, int, int) method and query it by calling getPaddingLeft(), getPaddingTop(), getPaddingRight(), and getPaddingBottom().

Although a view can define a padding, it doesn't support margins. However, view groups do support margins. See ViewGroup and ViewGroup.MarginLayoutParams for more information.

Types of Android Layout

- **Android Linear Layout:** LinearLayout is a ViewGroup subclass, used to provide child View elements one by one either in a particular direction either horizontally or vertically based on the orientation property.
- **Android Relative Layout:** RelativeLayout is a ViewGroup subclass, used to specify the position of child View elements relative to each other like (A to the right of B) or relative to the parent (fix to the top of the parent).
- **Android Constraint Layout:** ConstraintLayout is a ViewGroup subclass, used to specify the position of layout constraints for every child View relative to other views present. A ConstraintLayout is similar to a RelativeLayout, but having more power.
- **Android Frame Layout:** FrameLayout is a ViewGroup subclass, used to specify the position of View elements it contains on the top of each other to display only a single View inside the FrameLayout.
- **Android Table Layout:** TableLayout is a ViewGroup subclass, used to display the child View elements in rows and columns.

1.LinearLayout:

Android **LinearLayout** is a ViewGroup subclass, used to provide child View elements one by one either in a particular direction either horizontally or vertically based on the orientation property. We can specify the linear layout orientation using **android:orientation** attribute.

All the child elements arranged one by one in multiple rows and multiple columns.

1. **Horizontal list:** One row, multiple columns.
2. **Vertical list:** One column, multiple rows.

LinearLayout in activity_main.xml file:

```
<?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"
        tools:context=".MainActivity">

        <TextView
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_margin="16dp"
                android:text="Enter your name here:"
                android:textSize="24dp"
                android:id="@+id/txtVw"/>

        <EditText
                android:id="@+id/editText"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_margin="16dp"
                android:hint="Name"
                android:inputType="text"/>

        <Button
                android:id="@+id/showInput"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_gravity="center_horizontal"
                android:text="show"
                android:backgroundTint="@color/colorPrimary"
                android:textColor="@android:color/white"/>
</LinearLayout>
```

2.RelativeLayout:

Android **RelativeLayout** is a ViewGroup subclass, used to specify the position of child View elements relative to each other like (A to the right of B) or relative to the parent (fix to the top of the parent).

Instead of using LinearLayout, we have to use RelativeLayout to design the user interface and keep our hierarchy flat because it improves the performance of the application.

Attributes	Description
layout_alignParentLeft	It is set “true” to match the left edge of view to the left edge of parent.
layout_alignParentRight	It is set “true” to match the right edge of view to the right edge of parent.
layout_alignParentTop	It is set “true” to match the top edge of view to the top edge of parent.
layout_alignParentBottom	It is set “true” to match the bottom edge of view to the bottom edge of parent.

layout_alignLeft	It accepts another sibling view id and align the view to the left of the specified view id
layout_alignRight	It accepts another sibling view id and align the view to the right of the specified view id.
layout_alignStart	It accepts another sibling view id and align the view to start of the specified view id.
layout_alignEnd	It accepts another sibling view id and align the view to end of specified view id.
layout_centerInParent	When it is set “true”, the view will be aligned to the center of parent.
layout_centerHorizontal	When it is set “true”, the view will be horizontally centre aligned within its parent.
layout_centerVertical	When it is set “true”, the view will be vertically centre aligned within its parent.
layout_toLeftOf	It accepts another sibling view id and places the view left of the specified view id.
layout_toRightOf	It accepts another sibling view id and places the view right of the specified view id.
layout_toStartOf	It accepts another sibling view id and places the view to start of the specified view id.
layout_toEndOf	It accepts another sibling view id and places the view to end of the specified view id.
layout_above	It accepts another sibling view id and places the view above the specified view id.
layout_below	It accepts another sibling view id and places the view below the specified view id.

RelativeLayout in activity_main.xml file:

```
<?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">
        <TextView
                android:id="@+id/textView1"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_alignParentLeft="true"
                android:text="First name:"
                android:layout_marginTop="20dp"
                android:textSize="20dp"/>
        <EditText
                android:id="@+id/editText1"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_toRightOf="@id/textView1"
                android:layout_marginTop="8dp"/>
        <TextView
                android:id="@+id/textView2"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_alignParentLeft="true"
                android:layout_below="@+id/textView1"
                android:layout_marginTop="10dp"
                android:text="Last name:"
                android:textSize="20dp"/>
        <EditText
                android:id="@+id/editText2"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_centerHorizontal="true"
                android:layout_toRightOf="@id/textView2"
                android:layout_marginTop="45dp"/>
        <Button
                android:id="@+id/btn4"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_centerHorizontal="true"
                android:layout_below="@id/textView2"
                android:layout_marginTop="20dp"
                android:text="Submit" />
</RelativeLayout>
```

3.ConstraintLayout:

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.

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.

Advantages of using ConstraintLayout in Android:

- ConstraintLayout provides you the ability to completely design your UI with the drag and drop feature provided by the Android Studio design editor.
- It helps to improve the UI performance over other layouts.
- With the help of ConstraintLayout, we can control the group of widgets through a single line of code.
- With the help of ConstraintLayout, we can easily add animations to the UI components which we used in our app.

Disadvantages of using ConstraintLayout:

- When we use the Constraint Layout in our app, the XML code generated becomes a bit difficult to understand.
- In most of the cases, the result obtain will not be the same as we got to see in the design editor.
- Sometimes we have to create a separate layout file for handling the UI for the landscape mode.

Working with the activity_main.xml file:

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

        <TextView
                android:id="@+id/textView"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginLeft="16dp"
                android:layout_marginRight="16dp"
                android:gravity="center"
                android:padding="10dp"
                android:text="Geeks for Geeks"
                android:textColor="@color/black"
                android:textSize="20sp"
                app:layout_constraintBottom_toBottomOf="parent"
                app:layout_constraintLeft_toLeftOf="parent"
                app:layout_constraintRight_toRightOf="parent"
                app:layout_constraintTop_toTopOf="parent" />

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

4.Table Layout:

Android **TableLayout** is a ViewGroup subclass which is used to display the child View elements in rows and columns. It will arrange all the children elements into rows and columns and does not display any border lines in between rows, columns or cells.

The working of TableLayout is almost similar to HTML table and it contains as many columns as row with the most cells.

Attributes	Description
android:id	This is the ID which uniquely identifies the layout.
android:collapseColumns	This specifies the zero-based index of the columns to collapse. The column indices must be separated by a comma: 1, 2, 5.

android:shrinkColumns	The zero-based index of the columns to shrink. The column indices must be separated by a comma: 1, 2, 5.
android:stretchColumns	The zero-based index of the columns to stretch. The column indices must be separated by a comma: 1, 2, 5.

activity_main.xml file:

```
<?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="10dp"
        android:paddingLeft="5dp"
        android:paddingRight="5dp">
        <TextView
                android:id="@+id/txt"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="ICC Ranking of Players:"
                android:textSize = "20dp"
                android:textStyle="bold">
        </TextView>
        <TableRow android:background="#51B435" android:padding="10dp">
                <TextView
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:layout_weight="1"
                        android:text="Rank" />
                <TextView
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:layout_weight="1"
                        android:text="Player" />
                <TextView
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:layout_weight="1"
                        android:text="Team" />
                <TextView
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:layout_weight="1"
                        android:text="Points" />
        </TableRow>
        <TableRow android:background="#F0F7F7" 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="Virat Kohli" />
                <TextView
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:layout_weight="1"
                        android:text="IND" />
                <TextView
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:layout_weight="1"
                        android:text="895" />
        </TableRow>
        <TableRow android:background="#F0F7F7" 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"
                        android:text="Rohit Sharma" />
                <TextView
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:layout_weight="1"
                        android:text="IND" />
                <TextView
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:layout_weight="1"
                        android:text="863" />
        </TableRow>
        <TableRow android:background="#F0F7F7" android:padding="5dp">
                <TextView
                        android:layout_width="wrap_content"
```

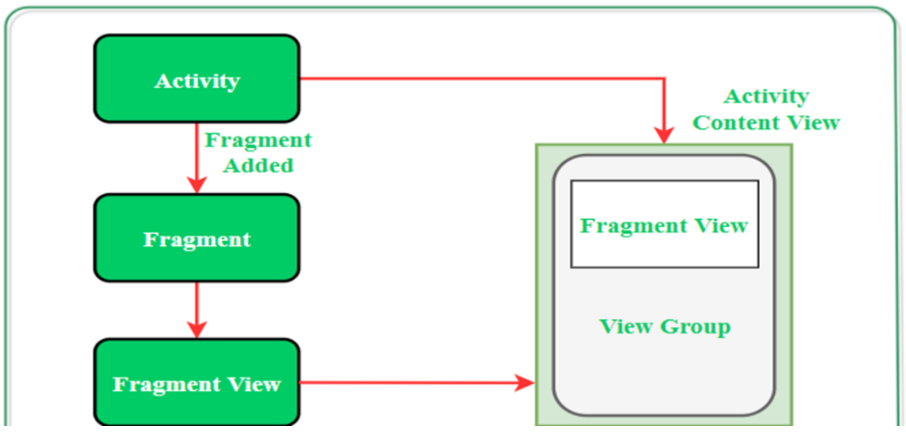
```
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:text="3" />
        <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:text="Faf du Plessis" />
        <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:text="PAK" />
        <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:text="834" />
    </TableRow>
</TableLayout>
```

2.4 Fragments

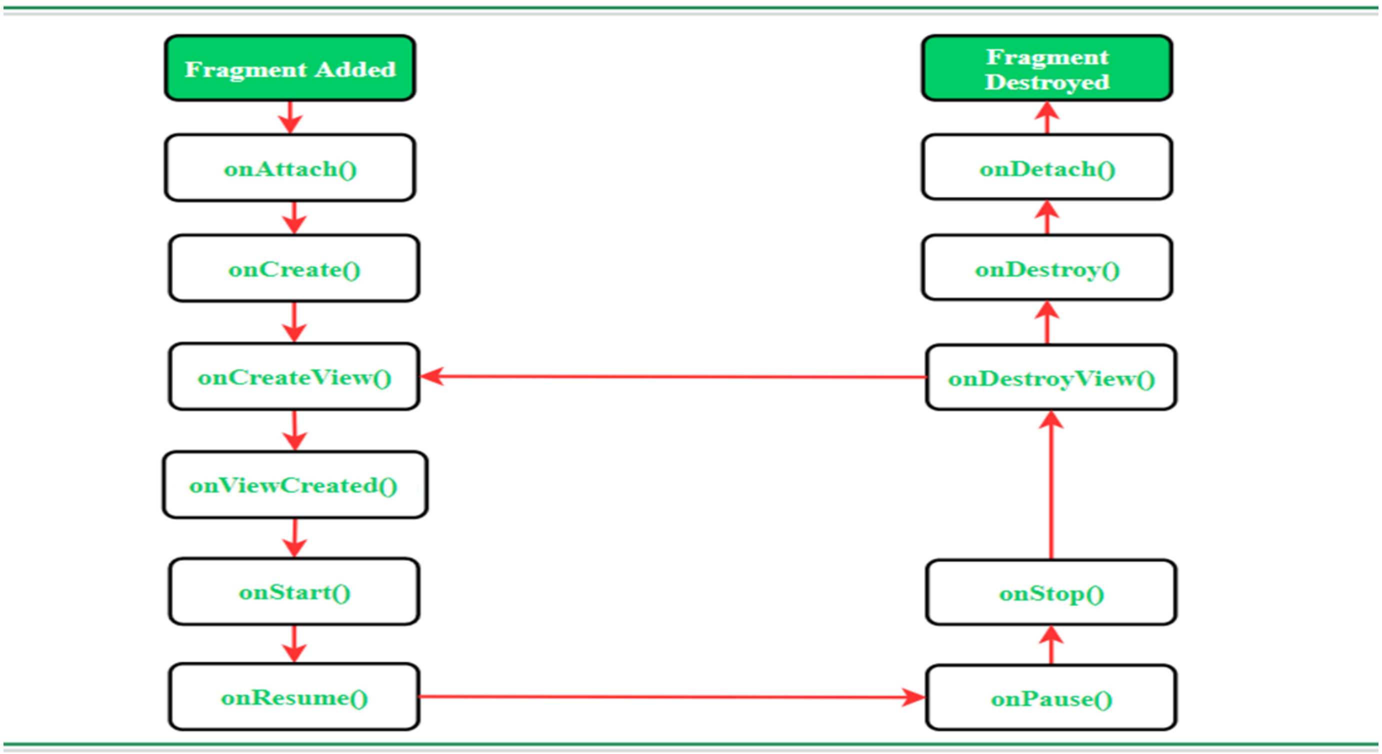
In Android, the fragment is the part of Activity which represents a portion of User Interface(UI) on the screen. It is the modular section of the android activity that is very helpful in creating UI designs that are flexible in nature and auto-adjustable based on the device screen size. The UI flexibility on all devices improves the user experience and adaptability of the application. Fragments can exist only inside an activity as its lifecycle is dependent on the lifecycle of host activity. For example, if the host activity is paused, then all the methods and operations of the fragment related to that activity will stop functioning, thus fragment is also termed as **sub-activity**. Fragments can be added, removed, or replaced dynamically i.e., while activity is running.

<fragment> tag is used to insert the fragment in an android activity layout. By dividing the activity's layout multiple fragments can be added in it.

Below is the pictorial representation of fragment interaction with the activity:



Fragment Lifecycle:



Each fragment has it's own lifecycle but due to the connection with the Activity it belongs to, the fragment lifecycle is influenced by the activity's lifecycle.

Methods of the Android Fragment

1. **onAttach():** once during the lifetime of a fragment.. When we attach fragment(child) to Main(parent) activity then it call first and then not call this method any time(like you run an app and close and reopen) simple means that this method call only one time.
2. **onCreate():** This method initializes the fragment by adding all the required attributes and components.
3. **onCreateView():** System calls this method to create the user interface of the fragment. The root of the fragment's layout is returned as the View component by this method to draw the UI. You should inflate your layout in onCreateView but shouldn't initialize other views using findViewById in onCreateView.
4. **onViewCreated():** It indicates that the activity has been created in which the fragment exists. View hierarchy of the fragment also instantiated before this function call.
5. **onStart():** The system invokes this method to make the fragment visible on the user's device.
6. **onResume():** This method is called to make the visible fragment interactive.
7. **onPause():** It indicates that the user is leaving the fragment. System call this method to commit the changes made to the fragment.
8. **onStop():** Method to terminate the functioning and visibility of fragment from the user's screen.
9. **onDestroyView():** System calls this method to clean up all kinds of resources as well as view hierarchy associated with the fragment. It will call when you can attach new fragment and destroy existing fragment Resoruce
10. **onDestroy():** It is called to perform the final clean up of fragment's state and its lifecycle.

11. **onDetach():** The system executes this method to disassociate the fragment from its host activity. It will call when your fragment Destroy(app crash or attach new fragment with existing fragment)

Example:

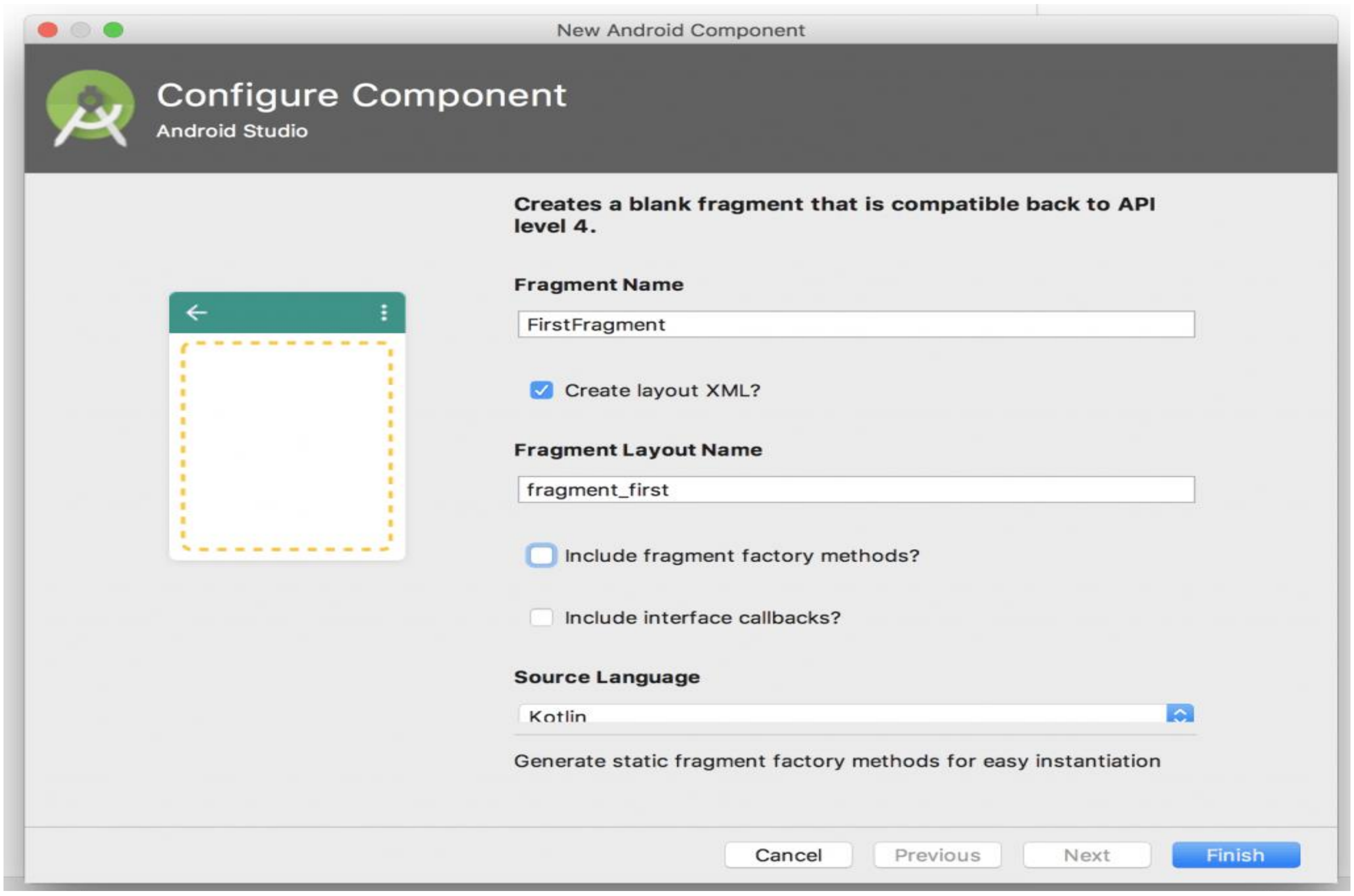
To create a fragment, you must create a subclass of Fragment (or an existing subclass of it).

You can do it with 2 ways, First Create a class and extended with Fragment class and create resource layout file.

OR

Second ways is Right Click on **Package -> New -> Fragment -> Fragments (Blank)**

then give **Fragment Name , check Fragment Layout Name – > Finish**



Creates a blank fragment that is compatible back to API level 4.

Fragment Name: FirstFragment

[✓] Create layout XML?

Fragment Layout Name: fragment_first

[ ] Include fragment factory methods?

[ ] Include interface callbacks?

Source Language: Kotlin

Generate static fragment factory methods for easy instantiation

Step 1. Create an android project in the android studio

Step 2. Create 2 Fragments and its resource layout. Follow upper section #Creating a Fragment

First fragment resource file fragment_first.xml, Outer Layout is FrameLayout, you can use the choice of your layout (LinearLayout, RelativeLayout or ConstraintLayout etc)

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#C0C0C0"
    tools:context="com.example.myapplication.FirstFragment">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="First Fragment"
        android:textSize="16sp" />
</FrameLayout>
```

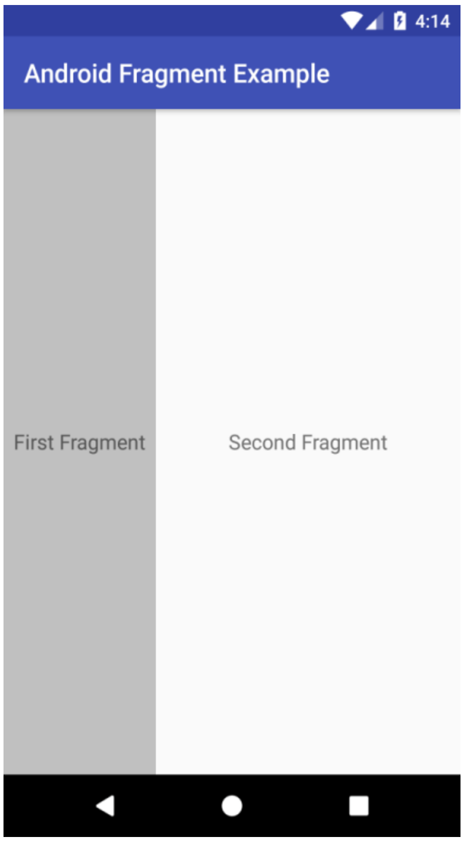
FirstFragment.kt

```
package com.example.myapplication

import android.os.Bundle
import android.support.v4.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup

class FirstFragment : Fragment() {

    override fun onCreateView(inflater: LayoutInflater?, container: ViewGroup?,
                              savedInstanceState: Bundle?): View? {
        // Inflate the layout for this fragment
        return inflater!!.inflate(R.layout.fragment_first, container, false)
    }

}// Required empty public constructor
```

Second fragment resource file fragment_second.xml, Both fragments layout containing TextView widget to show text.

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.myapplication.SecondFragment">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="Second Fragment"
        android:textSize="16sp" />
</FrameLayout>
```

Output screenshot Android Fragment example app :



2.5 Themes

A *theme* is a collection of attributes that's applied to an entire app, activity, or view hierarchy—not just an individual view. When you apply a theme, every view in the app or activity applies each of the theme's attributes that it supports. Themes can also apply styles to non-view elements, such as the status bar and window background.

Styles and themes are declared in a style resource file in res/values/, usually named styles.xml.

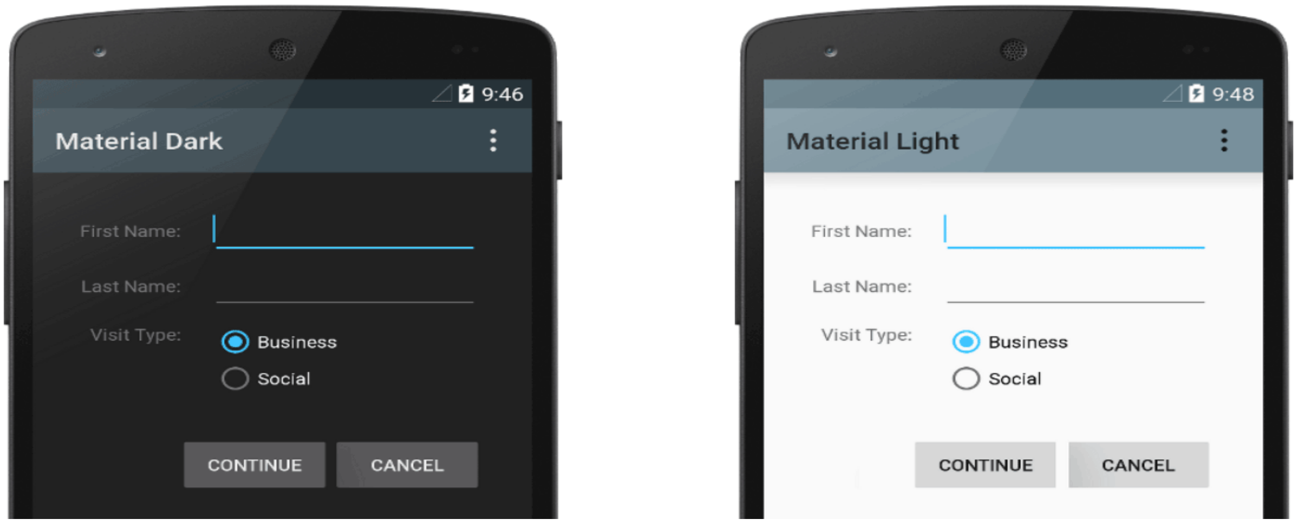


Figure - Two themes applied to the same activity: Theme.AppCompat (left) and Theme.AppCompat.Light (right).

Syntax :
```
<manifest ... >
    <application android:theme="@style/ThemeAppCompat" ... >
    </application>
</manifest>
```

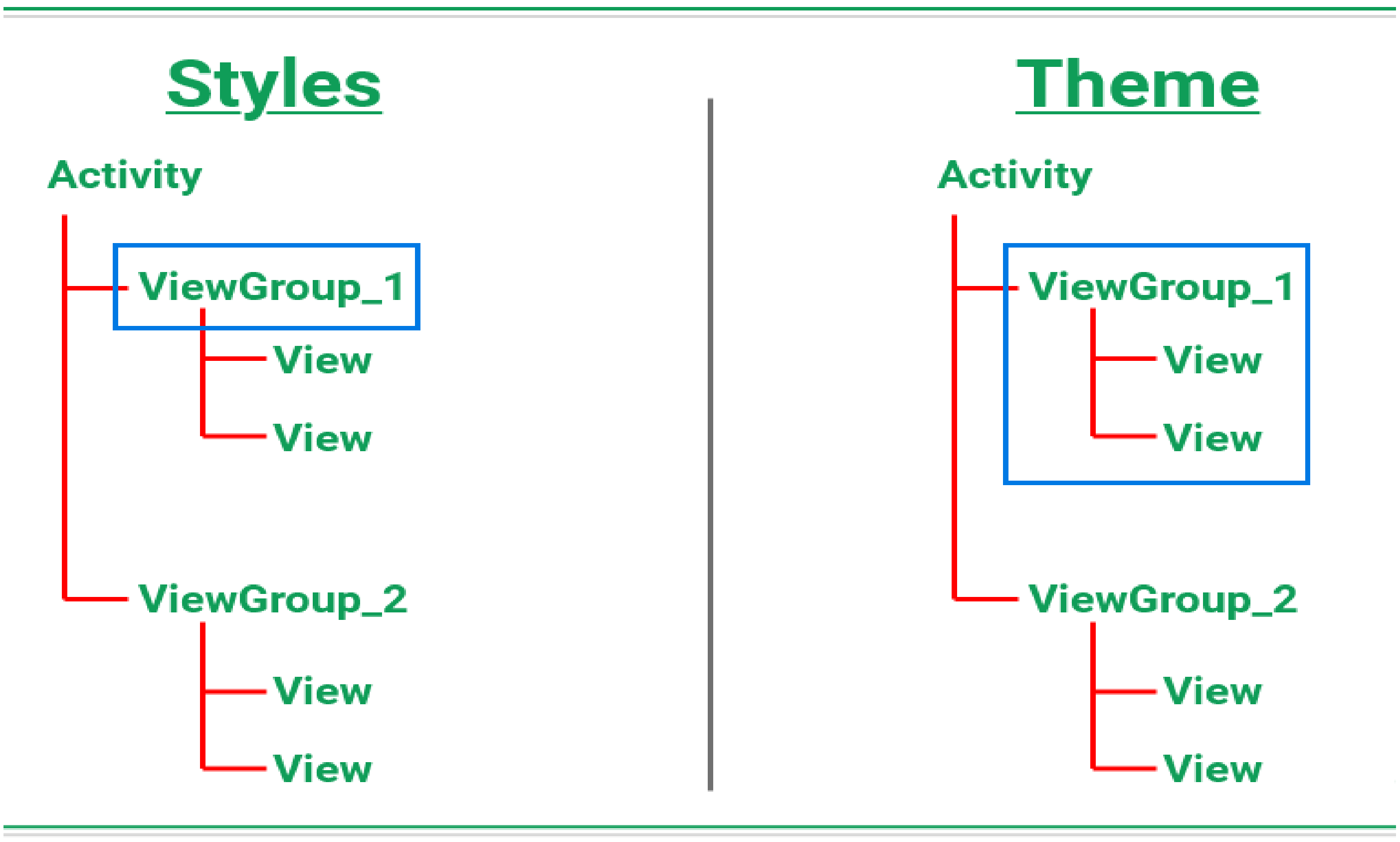
Themes versus styles

Themes and styles have many similarities, but they are used for different purposes. Themes and styles have the same basic structure—a key-value pair that maps *attributes* to *resources*.

A *style* specifies attributes for a particular type of view. For example, one style might specify a button's attributes. Every attribute you specify in a style is an attribute you can set in the layout file. Extracting all the attributes to a style makes it easy to use and maintain them across multiple widgets.

A *theme* defines a collection of named resources that can be referenced by styles, layouts, widgets, and so on. Themes assign semantic names, like colorPrimary, to Android resources.

Styles and themes are meant to work together. For example, you might have a style that specifies that one part of a button is colorPrimary, and another part is colorSecondary. The actual definitions of those colors are provided in the theme. When the device goes into night mode, your app can switch from its "light" theme to its "dark" theme, changing the values for all those resource names. You don't need to change the styles, since the styles are using the semantic names and not specific color definitions.



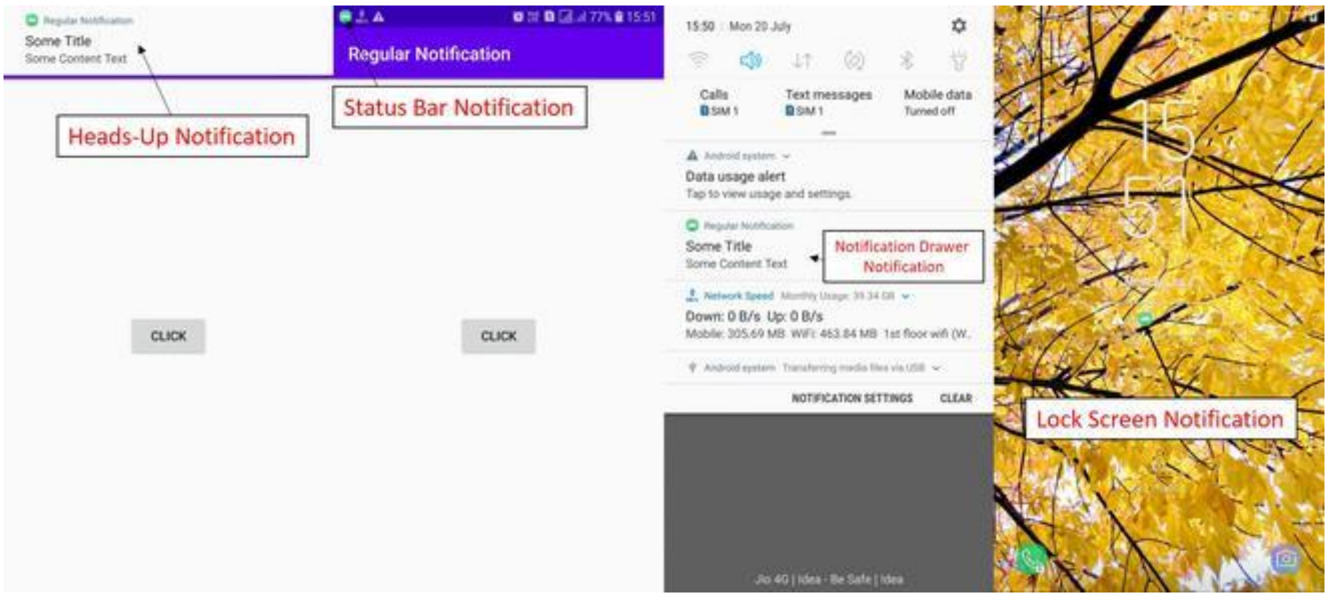
- Android developers witness the ambiguity in styles and themes. The reason is that in Android there are only tag names <style>, there is no <theme> tag available.
- Style is a collection of attributes that define the appearance of a single view. The style attributes are the font, color, size, background color, etc. For example, the font size may be different for heading and body.
- Theme, on the contrary, is applied to the entire app, or activity, or view hierarchy, not just for individual views. Themes can also apply styles to non-view elements such as status bar, window background, etc. For example, the colorPrimary is applied to all the Floating Action Buttons or Normal Buttons of the entire application. One can get the difference in the following image.

2.6 Notifications

Android Notification provides short, timely information about the action happened in the application, even it is not running. The notification displays the icon, title and some amount of the content text.

Notifications could be of various formats and designs depending upon the developer. In General, one must have witnessed these four types of notifications:

1. Status Bar Notification (appears in the same layout as the current time, battery percentage)
2. Notification drawer Notification (appears in the drop-down menu)
3. Heads-Up Notification (appears on the overlay screen, ex: Whatsapp notification, OTP messages)
4. Lock-Screen Notification (I guess you know it)



Set Android Notification Properties:

- **setSmallIcon()**: It sets the icon of notification.
- **setContentTitle()**: It is used to set the title of notification.
- **setContentText()**: It is used to set the text message.
- **setAutoCancel()**: It sets the cancelable property of notification.
- **setPriority()**: It sets the priority of notification.

Example:

Step1: Working with the activity_main.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context=".MainActivity">
   <Button
      android:id="@+id/btn"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_centerInParent="true"
      android:text="Send Notification"
      android:onClick=" onClickSubmit" />
</RelativeLayout>
```

Step2: Create a new empty activity

Name the activity as **afterNotification**. When someone clicks on the notification, this activity will open up in our app that is the user will be redirected to this page. Below is the code for the activity_after_notification.xml file.

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

   <TextView
      android:id="@+id/textView"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_centerInParent="true"
      android:text="Welcome To userdefine Notification"
      android:textSize="15sp"
      android:textStyle="bold" />

</RelativeLayout>
```

Step3: Working with the MainActivity.kt file

```
import android.app.NotificationChannel
import android.app.NotificationManager
import android.app.PendingIntent
import android.content.ActivityNotFoundException
import android.content.Context
import android.content.Intent
import android.graphics.Color
import android.net.Uri
import android.os.Build
import android.os.Bundle
import android.view.View
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import androidx.core.app.NotificationCompat

class MainActivity : AppCompatActivity() {
   lateinit var notificationManager: NotificationManager
   lateinit var notificationChannel: NotificationChannel
   lateinit var builder: NotificationCompat.Builder
   private val channelId = "i.apps.notification"
   private val channelName = "NChannel"
   private val channelDescription = "Test Notification"

   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      setContentView(R.layout.activity_main)
      Toast.makeText(applicationContext,"On Create Called", Toast.LENGTH_LONG).show();
   }

   fun onClickSubmit(view: View) {
         //Because you must create the notification channel before posting any notifications on Android 8.0
         and later, execute this code as soon as your app starts. I
      if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
         val importance = NotificationManager.IMPORTANCE_DEFAULT
         notificationChannel = NotificationChannel(channelId, channelName, importance).apply {
            description = channelDescription
         }
         // Register the channel with the system.
         notificationManager = getSystemService(Context.NOTIFICATION_SERVICE) as
NotificationManager
         notificationManager.createNotificationChannel(notificationChannel)
      }
      notificationManager = getSystemService(Context.NOTIFICATION_SERVICE) as
NotificationManager
      val intent = Intent(this, afterNotification::class.java)
      val pendingIntent = PendingIntent.getActivity(this, 2, intent,
PendingIntent.FLAG_UPDATE_CURRENT)

      builder = NotificationCompat.Builder(this, channelId)
         .setContentTitle("My notification")
```

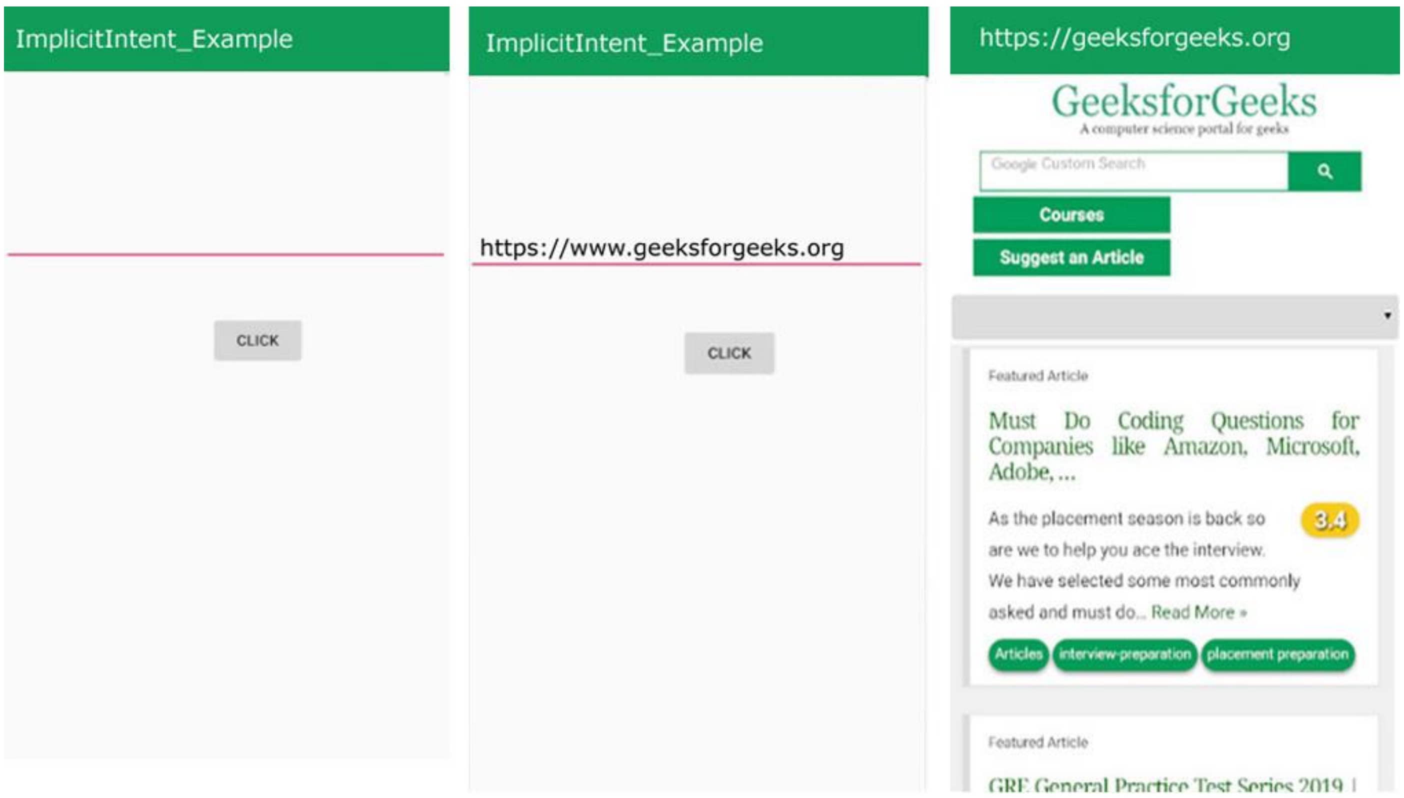
```
            .setContentText("Much longer text that cannot fit one line...")
            .setSmallIcon(R.drawable.ic_launcher_background)
            .setPriority(NotificationCompat.PRIORITY_DEFAULT)
            .setContentIntent(pendingIntent)
            .setAutoCancel(true)

        notificationManager.notify(1234,builder.build())

    }
}
```

2.7 Invoking Built-in Applications

In Android, you can invoke built-in applications using Intents. Intents are a fundamental part of the Android system that allows components (such as activities, services, and broadcast receivers) to request actions or interactions with other components. There are two types of intents: implicit and explicit.

1.Implicit Intent

Using implicit Intent, components can't be specified. An action to be performed is declared by implicit intent. Then android operating system will filter out components that will respond to the action. **For Example,**



Featured Article

Must Do Coding Questions for Companies like Amazon, Microsoft, Adobe, ...

As the placement season is back so are we to help you ace the interview. We have selected some most commonly asked and must do... Read More »

Featured Article

GRE General Practice Test Series 2019 |

In the above example, no component is specified, instead, an action is performed i.e. a webpage is going to be opened. As you type the name of your desired webpage and click on the 'CLICK' button. Your webpage is opened.

Step by Step Implementation

Creating an Android App to Open a Webpage Using Implicit Intent

Step 1: Create a New Project in Android Studio

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.

Syntax:

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

XML

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

        <EditText
                android:id="@+id/editText"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                app:layout_constraintBottom_toBottomOf="parent"
                app:layout_constraintEnd_toEndOf="parent"
                app:layout_constraintStart_toStartOf="parent"
                app:layout_constraintTop_toTopOf="parent" />

        <Button
                android:id="@+id/btn"
                android:text="Search"
                android:onClick="search"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                app:layout_constraintBottom_toBottomOf="parent"
                app:layout_constraintEnd_toEndOf="parent"
                app:layout_constraintStart_toStartOf="parent"
                app:layout_constraintTop_toBottomOf="@+id/editText" />

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

Step 3: Working with the MainActivity File

Now, we will create the Backend of the App. For this, Open the MainActivity file and instantiate the component (Button) created in the XML file using the findViewById() method. This method binds the created object to the UI Components with the help of the assigned ID.

Syntax:

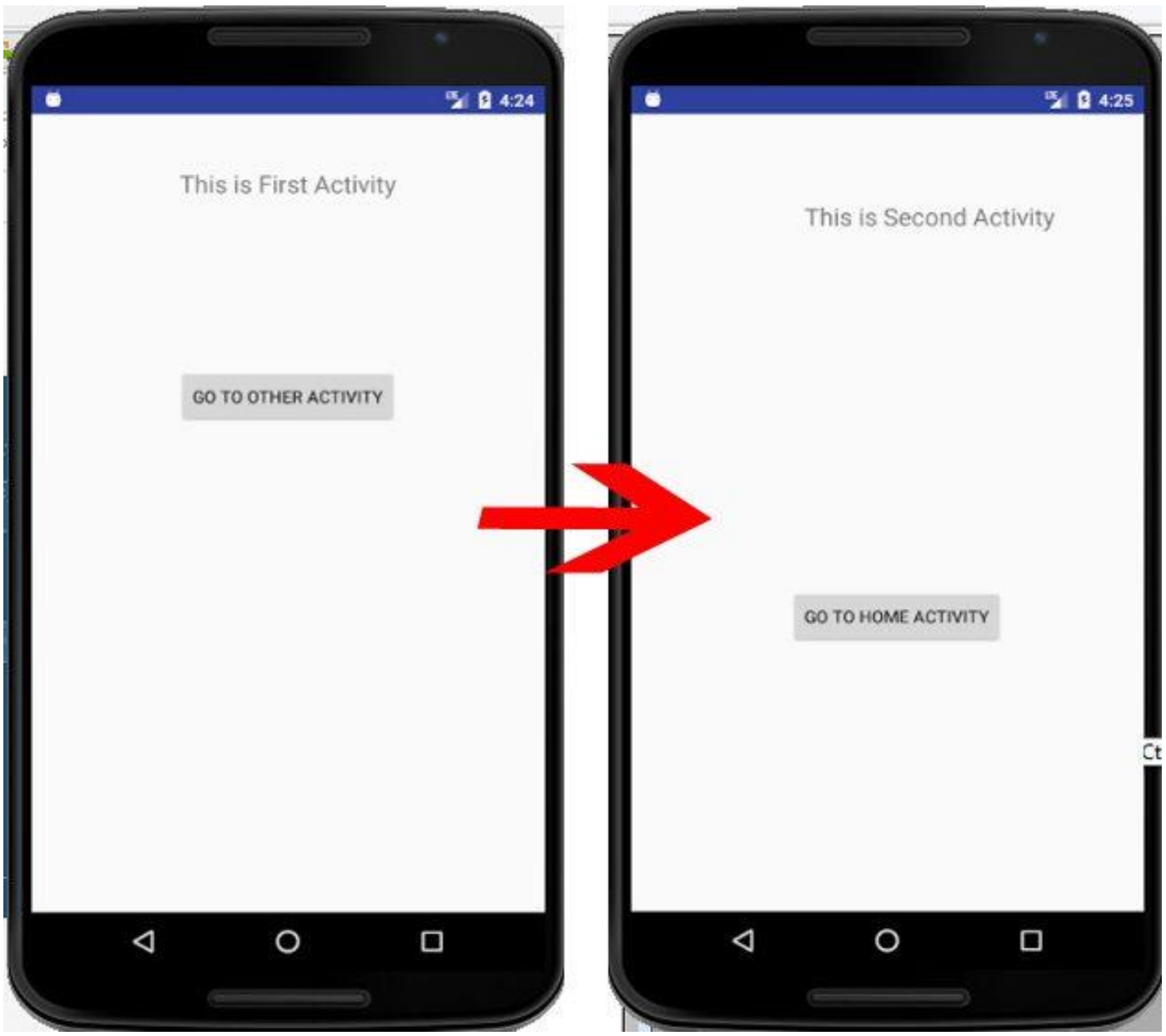
```
ComponentType object = (ComponentType) findViewById(R.id.IdOfTheComponent);
```

Kotlin

```
import android.content.Intent
import android.net.Uri
import android.os.Bundle
import android.widget.Button
import android.widget.EditText
import androidx.appcompat.app.AppCompatActivity

class MainActivity : AppCompatActivity() {
    lateinit var editText: EditText
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
        editText = findViewById(R.id.editText)
    }
    fun search() {
        val url = editText.text.toString()
        val urlIntent = Intent(Intent.ACTION_VIEW, Uri.parse(url))
        startActivity(urlIntent)
    }
}
```

2.Explicit Intent

Using explicit intent any other component can be specified. In other words, the targeted component is specified by explicit intent. So only the specified target component will be invoked. **For Example:**



In the above example, There are two activities (FirstActivity, and SecondActivity). When you click on the 'GO TO OTHER ACTIVITY' button in the first activity, then you move to the second activity. When you click on the 'GO TO HOME ACTIVITY' button in the second activity, then you move to the first activity. This is getting done through Explicit Intent.

Step by Step Implementation

How to create an Android App to move to the next activity using Explicit Intent(with Example)

Step 1: Create a New Project in Android Studio

Step 2: Working with the activity_main.xml File

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.

Syntax:

android:id="@+id/id_name"

XML

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

        <TextView
                android:id="@+id/editText"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:text="Welcome to GFG Home Screen"
                android:textAlignment="center"
                android:textSize="28sp"
                app:layout_constraintBottom_toBottomOf="parent"
                app:layout_constraintEnd_toEndOf="parent"
                app:layout_constraintHorizontal_bias="0.0"
                app:layout_constraintStart_toStartOf="parent"
                app:layout_constraintTop_toTopOf="parent" />
        <Button
                android:id="@+id/btn1"
                android:text="Go to News Screen"
                android:onClick="newsScreen"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                app:layout_constraintBottom_toBottomOf="parent"
                app:layout_constraintEnd_toEndOf="parent"
                app:layout_constraintStart_toStartOf="parent"
                app:layout_constraintTop_toBottomOf="@+id/editText" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Step 3: Working with the MainActivity File

Now, we will create the Backend of the App. For this, Open the MainActivity file and instantiate the component (Button, TextView) created in the XML file using the findViewById() method. This method binds the created object to the UI Components with the help of the assigned ID.

Syntax:

```
ComponentType object = (ComponentType) findViewById(R.id.IdOfTheComponent);
Intent i = new Intent(getApplicationContext(), <className>);
```

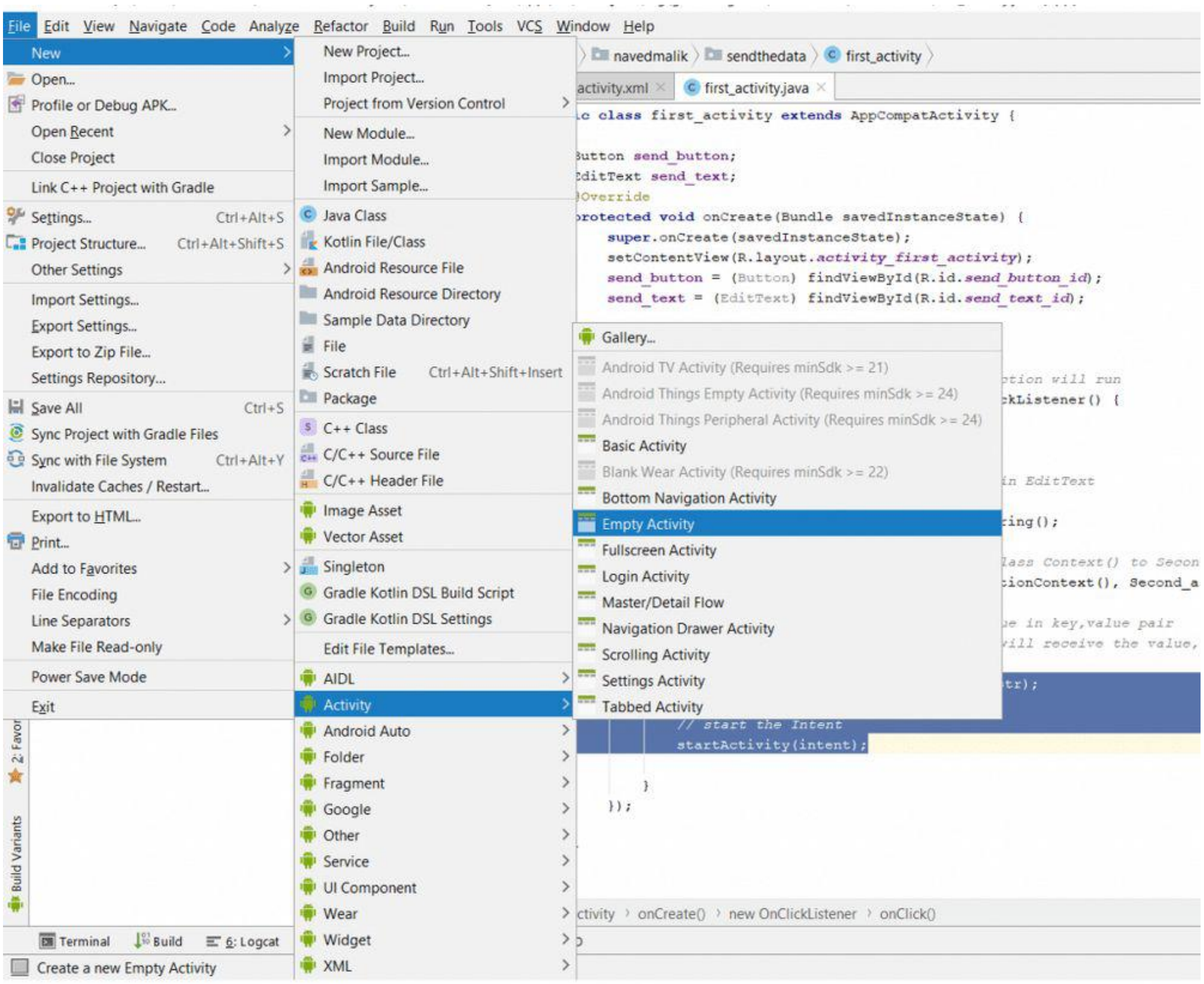
```
startActivity(i);
```

Kotlin

```
import android.content.Intent
import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity
class MainActivity : AppCompatActivity() {
        override fun onCreate(savedInstanceState: Bundle?) {
                super.onCreate(savedInstanceState)
                setContentView(R.layout.activity_main)}
        fun newsScreen() {
                val i = Intent(applicationContext, MainActivity2::class.java)
                startActivity(i);  } }
```

Step 4: Working with the activity_main2.xml File

Now we have to create a second activity as a destination activity. The steps to create the second activity are **File > new > Activity > Empty Activity.**



```
c class first_activity extends AppCompatActivity {

utton send_button;
ditText send_text;
Override
rotected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_first_activity);
    send_button = (Button) findViewById(R.id.send_button_id);
    send_text = (EditText) findViewById(R.id.send_text_id);

                                                     ption will run
                                                     kListener() {

                                                     in EditText

                                                     ing();

                                                     lass Context() to Secon
                                                     ionContext(), Second_a

                                                     e in key,value pair
                                                     ill receive the value,

                                                     tr);
            // start the Intent
            startActivity(intent);

        }
    });
```

Next, go to the **activity_main2.xml file**, which represents the UI of the project. Below is the code for the **activity_main2.xml** file. Comments are added inside the code to understand the code in more detail.

XML

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".MainActivity2">

        <TextView
                android:id="@+id/editText"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:text="Welcome to GFG News Screen"
                android:textAlignment="center"
                android:textSize="28sp"
                app:layout_constraintBottom_toBottomOf="parent"
                app:layout_constraintEnd_toEndOf="parent"
                app:layout_constraintHorizontal_bias="0.0"
                app:layout_constraintStart_toStartOf="parent"
                app:layout_constraintTop_toTopOf="parent" />


        <Button
                android:id="@+id/btn2"
                android:text="Go to Home Screen"
                android:onClick="homeScreen"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                app:layout_constraintBottom_toBottomOf="parent"
                app:layout_constraintEnd_toEndOf="parent"
                app:layout_constraintStart_toStartOf="parent"
                app:layout_constraintTop_toBottomOf="@+id/editText" />
</androidx.constraintlayout.widgext.ConstraintLayout>
```

Step 5: Working with the MainActivity2 File

Now, we will create the Backend of the App. For this, Open the MainActivity file and instantiate the component (Button, TextView) created in the XML file using the findViewById() method. This method binds the created object to the UI Components with the help of the assigned ID.

Syntax:

```
ComponentType object = (ComponentType) findViewById(R.id.IdOfTheComponent);
Intent i = new Intent(getApplicationContext(), <className>);
startActivity(i);
```

Kotlin

```
import android.content.Intent
import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity

class MainActivity2 : AppCompatActivity() {
        override fun onCreate(savedInstanceState: Bundle?) {
                super.onCreate(savedInstanceState)
                setContentView(R.layout.activity_main2)
        }
        fun homeScreen() {
                val i = Intent(applicationContext, MainActivity::class.java)
                startActivity(i)
        }
}
```