Contents

Topics	Page No.
Unit-2: Working with Android User Interface and handling UI Events	
Chapter-4: Activity and Intents	1
4.0 Structure of Activity and Intents	1
4.1 Learning Outcomes	1
4.2 Overview of Activities	1
4.2.1 Life Cycle of Activity	2
4.2.2 Understanding the life cycle of activity	4
4.3 Toast for displaying messages	6
4.4 Intents	7
4.4.1 Linking Activities using intents	9
4.4.2 Interaction amongst activities using intents	14
4.5 Self-Assessment Questions	15
4.6 Self-Assessment Activities	15
4.7 Multiple-Choice Questions	15
4.8 Key Answers to Multiple-Choice Questions	17
4.9 Summary	17
4.10 Keywords	18
4.11 Recommended Resources for Further Reading	18

Graphic Era University

UNIT 2 - Working with Android User Interface and handling UI Events

CHAPTER 4 - Activity and Intents

Structure of Activity and Intents

4.1 Learning Outcomes
4.2 Overview of Activities
4.3 Toast for displaying messages
4.4 Intents
4.5 Self-Assessment Questions
4.6 Self-Assessment Activities
4.7 Multiple-Choice Questions
4.8 Key answers to multiple-choice questions
4.9 Summary
4.10 Keywords
4.11 Recommended resources for further reading

4.1 Learning Outcomes

After the successful completion of this chapter, the student will be able to:

- Describe an android activity and its life cycle
- Demonstrate the different stages of activity with a suitable example
- Illustrate the usage of Toast in Android
- Illustrate the use of Intents in an android application by creating activities, linking them, passing and receiving data using intent objects

4.2 Overview of Activities

In an android application, activity is used to interact with the users. The activity contains the user interface of the application on a single window screen. An android application may perform various operations like sending SMS, getting information from users, processing data, etc. For all such operations, an activity is defined. A single application usually contains many activities. When an android application is launched, the main activity is displayed to the users. This is an entry point for executing an android application. The user interface of the main activity allows users to invoke other activities depending upon their request. A user can navigate back and forth among the activities. Whenever another activity is invoked, the

Graphic Era University

present activity is paused which can be resumed or stopped later. An activity goes through various stages/states known as the activity's life cycle.

4.2.1 Life Cycle of Activity

An android activity has 7 lifecycle stages/processes.

1. **`onCreate()`** - This method is invoked when an activity is first created. Any business logic to be executed only once during the entire lifetime of the activity can be implemented in this method.
2. **`onStart()`** - This method is invoked when the activity starts and becomes visible to the user. During the lifecycle of an application, this method can be called several times.
3. **`onResume()`** - This method is invoked immediately once **`onStart()`** is called. The activity at this stage comes to the foreground and the user can start interacting.
4. **`onPause()`** - When a user shifts to another activity or a previous activity is being resumed, this method is invoked. If a previous activity comes to the foreground, then this method is followed by the **`onResume()`** method. However, the **`onPause()`** method may be followed by the **`onStop()`** when the activity has lost focus and is running in the background. At the pause stage, the unused resources from memory can be released however they have to be re-initialized at the resume stage.

   In brief **`onPause()`** method is called during the transition of activities so any huge processing required for an application should be avoided at this stage as it may delay transitions leading to poor performance. It may also be called when the activity runs in the background but has not yet been killed.
5. **`onStop()`** – It is called when the activity is no longer visible to the user, has finished running, and is about to be terminated. Releasing of resources and saving data operations can be performed at this stage.
6. **`onRestart()`** – When an activity still exists and the user wants to restart it again then this method is called before the **`onStart()`** method is invoked. This usually happens during the transition of activities.
7. onDestroy() – This method is called when an activity is stopped/finished and is to be destroyed or due to some configuration settings the activity needs to be temporarily destroyed.

Figure 4.1 shows the life cycle of the activity, from the time the activity is launched, started until it ends.

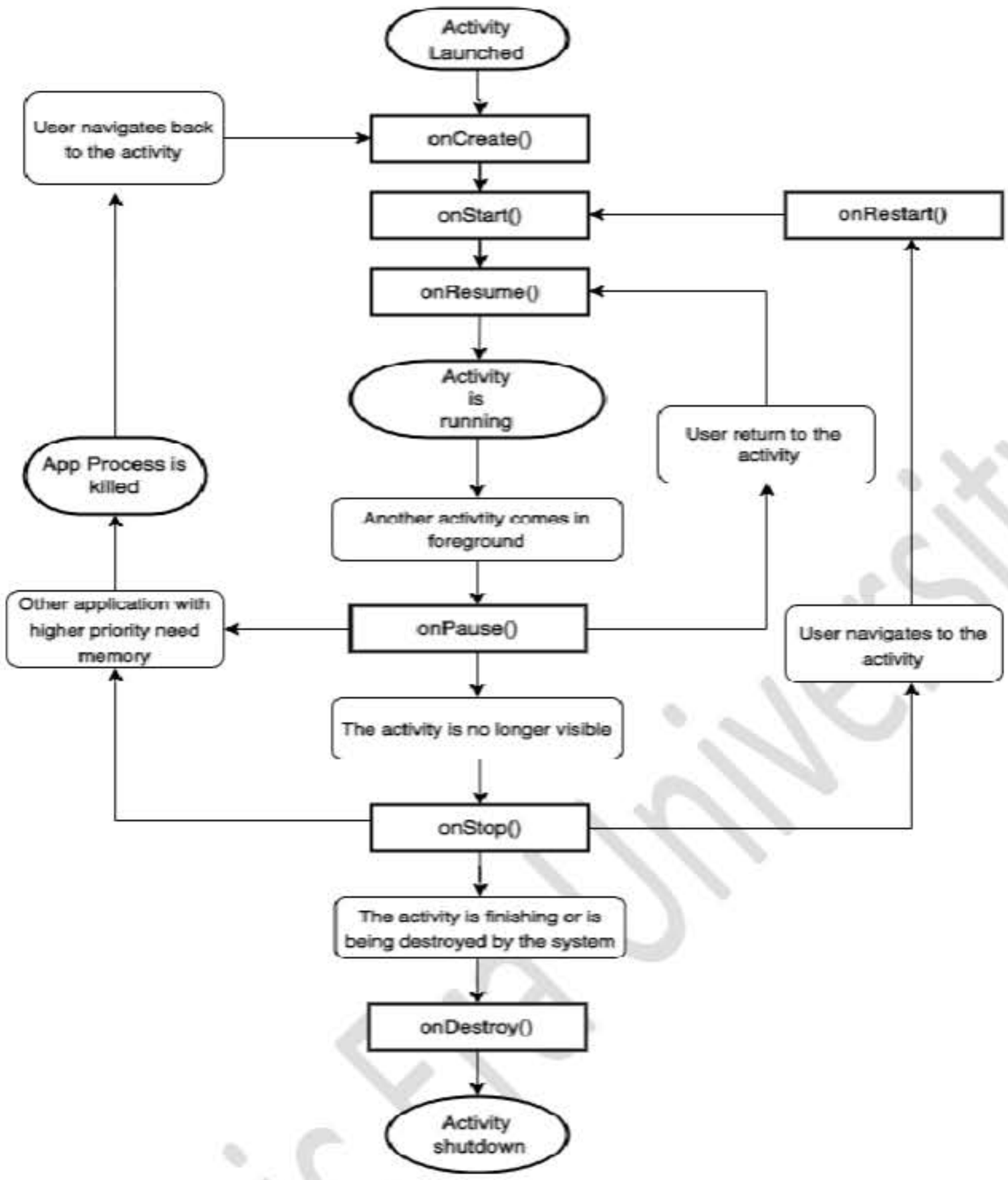


Figure 4.1: Life Cycle of an Activity

When an application is launched the Main Activity, the **`onStart()`** method is executed, the user interface is created using the xml layout file and the **`onResume()`** method is executed. At this stage, the activity is in an active state.

Suppose later a dialog box appears, then at this stage, the user is interacting with the dialog/popup for some confirmation like "OK" or "Cancel" and not with the activity user interface which is now in a paused state. The **`onPause()`** method gets executed as soon as the dialog box appears.

When a back button is pressed, the application is not closed but is minimized and it is in a stop state where the **`onStop()`** method gets executed.

Finally, when the application is completely closed the **`onDestroy()`** method is executed.

4.2.2 Understanding the life cycle of activity

To understand the life cycle of an activity create a new project and implement the methods **`onStart()`**, **`onResume()`**, **`onPause()`**, **`onStop()`**, **`onRestart()`**, and **`onDestroy()`** as shown in figure 4.2. A string TAG is declared to be used in **`Log.d()`** as 1st parameter. **`Log.d()`** prints messages in Logcat.

```
package com.example.activitylifecycle;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
public class MainActivity extends AppCompatActivity {
    private static final String TAG = "MainActivity";
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    @Override
    protected void onStart() {
        super.onStart();
        Log.d(TAG,"MainActivity started");
    }

    @Override
    protected void onStop() {
        super.onStop();
        Log.d(TAG,"MainActivity stopped");
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        Log.d(TAG, "MainActivity destroyed");
    }

    @Override
    protected void onRestart() {
```

Graphic Era University

```
        super.onRestart();
        Log.d(TAG,"MainActivity restarted");
    }

    @Override
    protected void onPause() {
        super.onPause();
        Log.d(TAG,"MainActivity Paused");
    }

    @Override
    protected void onResume() {
        super.onResume();
        Log.d(TAG,"MainActivity Resumed");
    }
}
```

Figure 4.2: Snippet showing overridden methods to demonstrate various stages in the Life Cycle of an Activity

Execute the application and click the back button. Open the Logcat window and select the verbose or debug from the dropdown, the following information shown in figure 4.3 is displayed. It shows the execution of the activity lifecycle processes.

```
2022-07-08 12:17:27.383 10771-10786/com.example.activitylifecycle I/art: Background partial concurrent mark sweep GC freed 420(
2022-07-08 12:17:27.565 10771-10771/com.example.activitylifecycle D/MainActivity: MainActivity started
2022-07-08 12:17:27.565 10771-10771/com.example.activitylifecycle D/MainActivity: MainActivity Resumed
2022-07-08 12:17:27.575 10771-10789/com.example.activitylifecycle D/OpenGLRenderer: Render dirty regions requested: true
2022-07-08 12:17:27.583 10771-10771/com.example.activitylifecycle D/: HostConnection::get() New Host Connection established 0xa
2022-07-08 12:17:27.584 10771-10771/com.example.activitylifecycle D/Atlas: Validating map...
2022-07-08 12:17:27.638 10771-10789/com.example.activitylifecycle D/: HostConnection::get() New Host Connection established 0xa
2022-07-08 12:17:27.642 10771-10789/com.example.activitylifecycle I/OpenGLRenderer: Initialized EGL, version 1.4
2022-07-08 12:17:27.642 10771-10789/com.example.activitylifecycle W/OpenGLRenderer: Failed to choose config with EGL_SWAP_BEHAV
2022-07-08 12:17:27.729 10771-10789/com.example.activitylifecycle D/EGL_emulation: eglCreateContext: 0xa38240a0: maj 2 min 0 re
2022-07-08 12:17:27.779 10771-10789/com.example.activitylifecycle D/EGL_emulation: eglMakeCurrent: 0xa38240a0: ver 2 0 (tinfo 0
2022-07-08 12:17:27.783 10771-10789/com.example.activitylifecycle D/OpenGLRenderer: Enabling debug mode 0
2022-07-08 12:17:27.792 10771-10789/com.example.activitylifecycle D/EGL_emulation: eglMakeCurrent: 0xa38240a0: ver 2 0 (tinfo 0
2022-07-08 12:17:27.844 10771-10786/com.example.activitylifecycle I/art: Background sticky concurrent mark sweep GC freed 348(
2022-07-08 12:19:09.233 10771-10771/com.example.activitylifecycle D/MainActivity: MainActivity Paused
2022-07-08 12:19:09.832 10771-10771/com.example.activitylifecycle D/MainActivity: MainActivity stopped
2022-07-08 12:19:09.833 10771-10789/com.example.activitylifecycle D/EGL_emulation: eglMakeCurrent: 0xa38240a0: ver 2 0 (tinfo 0
2022-07-08 12:19:09.834 10771-10771/com.example.activitylifecycle D/MainActivity: MainActivity destroyed
```

Figure 4.3: Logcat window output showing the messages printed on the occurrence of the events at various stages

4.3 Toast for displaying messages

Toast is used to display information or show notifications about a particular event that has happened successfully/unsuccessfully in a small popup for a particular interval of time. The message disappears/fades quickly after some time interval. It does not block or need user interaction.

Import the package android.widget.Toast in activity class for using it.

The Toast class is a subclass of the Object class.

The most commonly used methods of Toast class are

1. **`makeText(Context context, CharSequence text, int duration)`** – It is used to instantiate a Toast object.

   It takes 3 parameters.

   - 1st parameter is used to specify the application context
   - 2nd parameter is used to specify the text/message to be displayed
   - 3rd parameter is used to specify the duration of display that should remain on the screen.

   It can be specified by using LENGTH_LONG or LENGTH_SHORT constants
2. **`show()`** - It is used to display toast
3. **`setMargin()`** - It is used to set the horizontal and vertical margin
4. **`setText()`** - It is used to set/change the message to be displayed by the Toast.

For example: To display the **"Hello"** message using Toast, create a project and add the following code to the MainActivity.java

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // 1st method of using Toast
        Toast.makeText(getApplicationContext(),
                        "Hello, this is method 1", Toast.LENGTH_LONG).show();
        // 2nd method using Toast object
        Toast tobj = Toast.makeText(getApplicationContext(),
                      "Hello, this is using Toast object", Toast.LENGTH_LONG);

        tobj.show();
    }
}
```

Figure 4.4: Snippet for displaying messages using Toast

Run the android application, the output appears as shown in figure 4.5

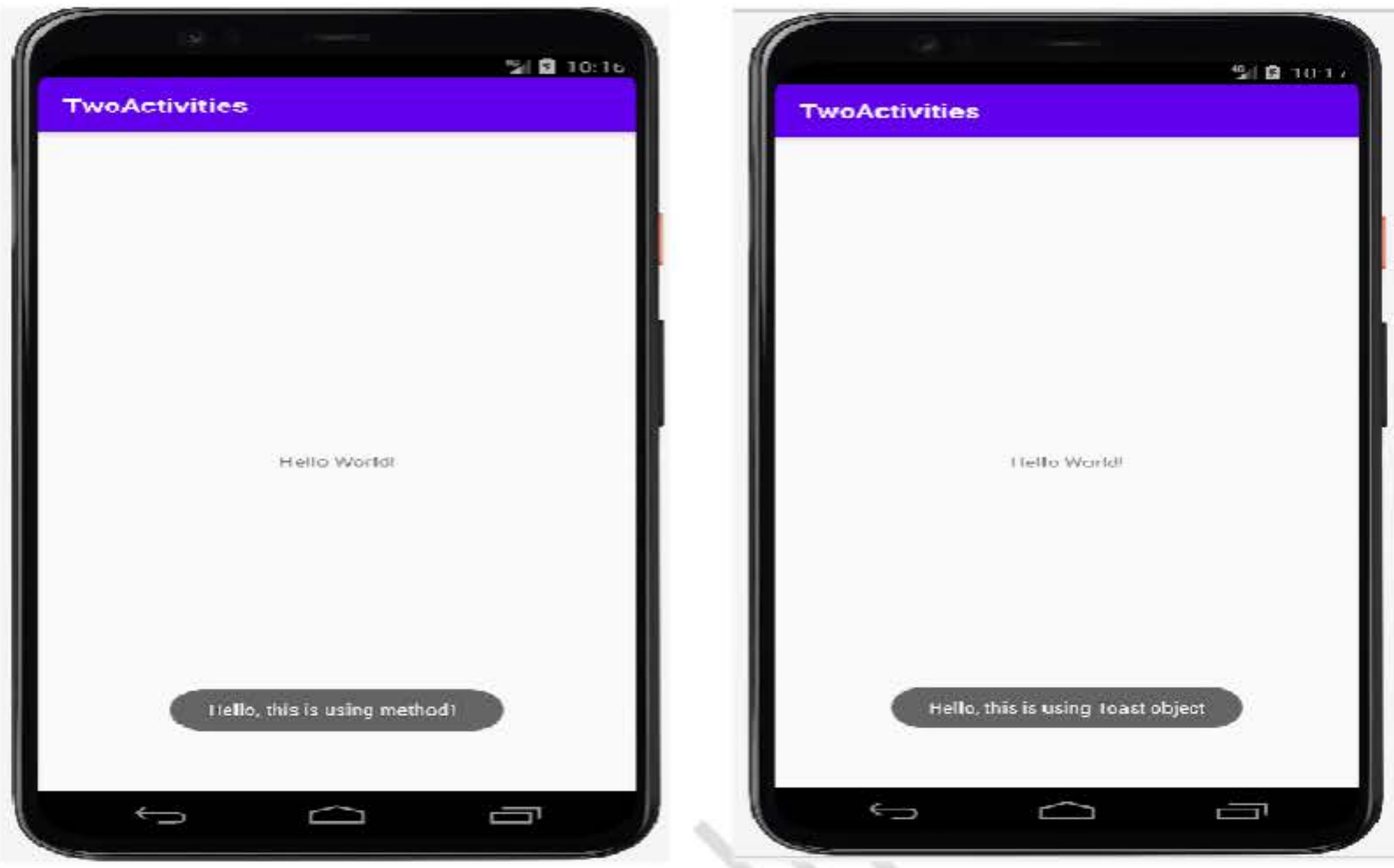


Figure 4.5: Output of Toast methods used in code shown in figure 4.4.

4.4 Intents

Intents are used for communication. Communication can be between components (usually activities) of an application or from one application to another. In other words, intents are used for binding activities in a single application or different applications.

The 3 main purposes of using intents are :

- Starting an activity - another activity can be started from the present activity on the occurrence of an event.

  `startActivity()` method is used to start another activity
- Starting a service - a service can be started to perform a background process without any user interaction (for example – downloading a file).

  `startService()` method is used to start a service
- Broadcast message - any application can send/receive a broadcast message.

  `sendBroadcast()` or `sendOrderedBroadcast()` method is used to send a broadcast message.

Types of Intents :

There are 2 types of Intent

1. Explicit Intent
2. Implicit Intent

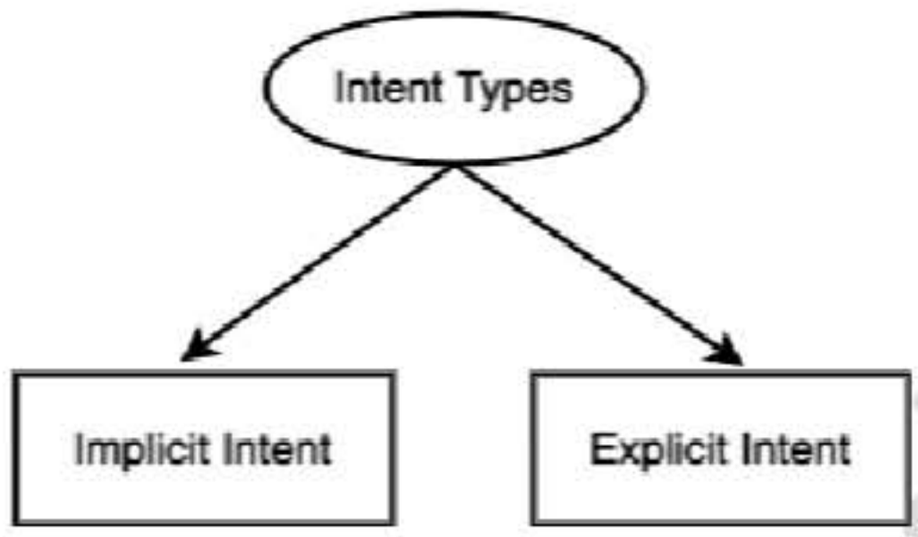


Figure 4.6: Types of Intents

1. Explicit Intent: These are basically used to start an activity or a service within a single application internally.

   To move from one activity to another activity within an application we can write code as follows

```
// create an intent object

Intent intentobj = new Intent(getApplicationContext(),
                                        SecondActivity.class );

startActivity(intentobj);
```

Figure 4.7: Code for starting another activity using Explicit Intent

`SecondActivity` is the name of the second activity used in the application where the navigation can be from the present activity to the second activity.

`startActivity()` is a method used to start another activity by passing an object of intent as a parameter. Here `intentobj` is passed as a parameter.

The name of the component which is affected by the intent for navigation is specified in case of explicit intent. Similarly, a service to download a file can be started in the background by using an intent. While moving from one activity to another, data can also be transferred.

A service for playing music in the background or fetching data from a database over a network without any user interaction can be started by using explicit intent.

2. Implicit Intent: In the case of implicit intent the general action to be performed or handled by the component of another application is specified instead of the name of the component.

Example 1. Suppose we want to show the user a location on a map. This can be done by sending a request to another capable app to show the location using implicit intent.

Example 2. Suppose we want to open a website using the intent object from an android application. The action to be performed i.e. **ACTION_VIEW** is specified in the intent object rather than the name of the component. The following code can be added to the MainActivity.java file

```
Intent intentobj = new Intent(Intent.ACTION_VIEW);
Intentobj.setData("Uri.parse("https://www.geu.ac.in");
startActivity(intentobj);
```

Figure 4.8: Code for opening a web page using Implicit Intent

4.4.1 Linking Activities using Intent

To move from the present activity to another activity we use intent.

Steps for creating Second Activity and linking using intent

1. Create an Android Project and run it on an emulator as usual. It displays the message "Hello world"

2. In **activity_main.xml** make the following changes
   - Change the default layout to a Linear layout and also add **android:orientation="vertical"**
   - Change the **android:text** to **"This is** the **Main Activity of My Project"**
   - Add a button that can be used to navigate to the second activity on click later in the project.
   - The properties **android:textSize, android:textStyle, android:textColor, android:layout_marginTop,** etc. are added for clarity in the message display

Make the following changes in the code in the **activity_main.xml** file for proper positioning of the controls and clarity of the messages in the display

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

<LinearLayout
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"
    android:orientation="vertical"  >

    <TextView
        android:id="@+id/txtview1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="This is the Main Activity of my project"
        android:textSize="30dp"
        android:textStyle="bold"
        android:textColor="@color/black"
        android:layout_marginTop="200dp"
        android:gravity="center"/>

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Go to Second Activity"
        android:textSize="20dp"
        android:textStyle="bold"
        android:layout_marginTop="300dp"
        android:layout_marginLeft="50dp"
        />
</LinearLayout>
```

Figure 4.9: Snippet of activity_main.xml file with changes

3. To add second activity to the project, select the options from the menus as shown in figure 4.10
   Right-click on java package → new → Activity →Empty Activity

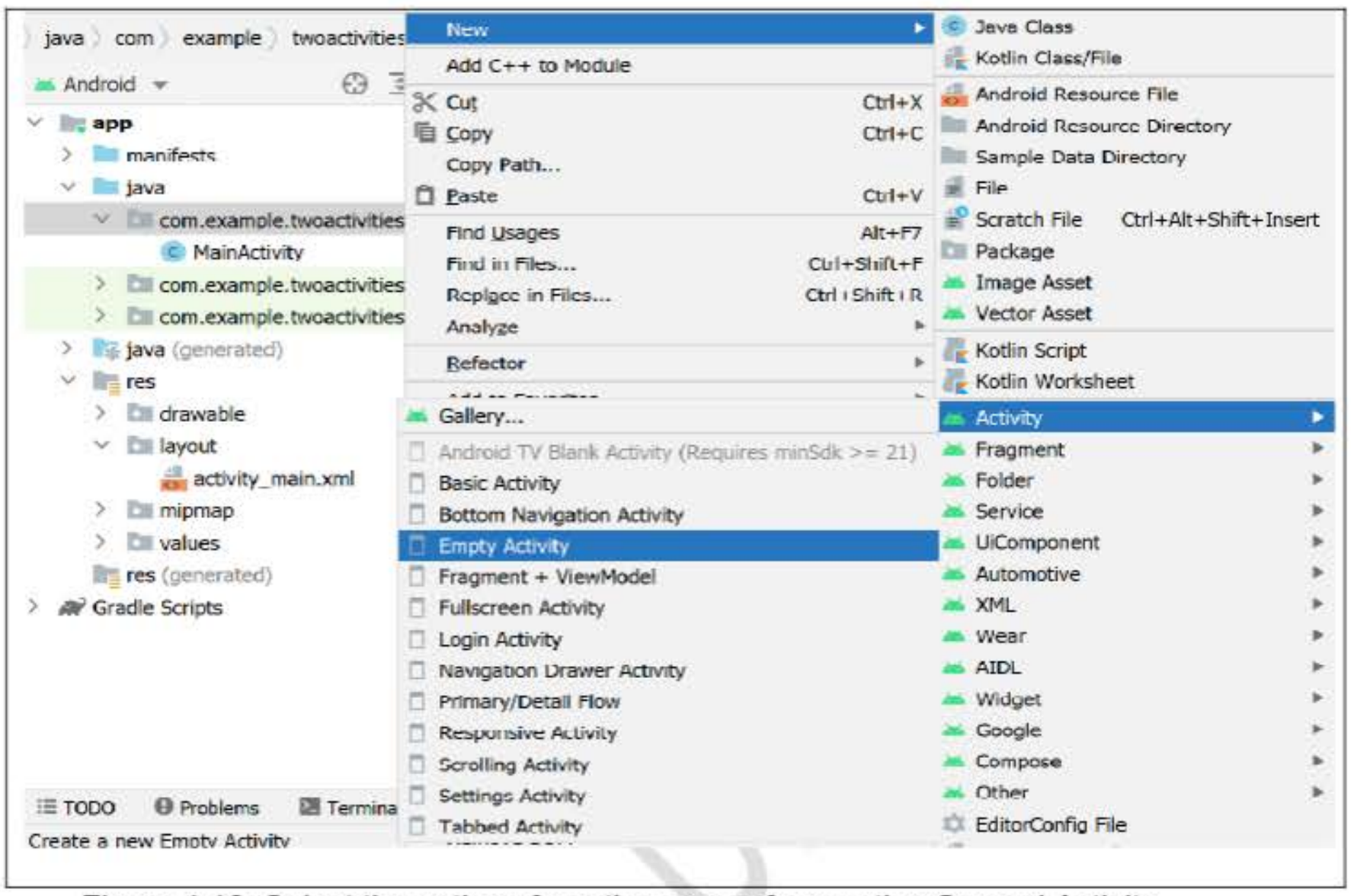


Figure 4.10: Select the options from the menus for creating Second Activity

4. In the Empty Activity window enter the details of the second activity and click finish

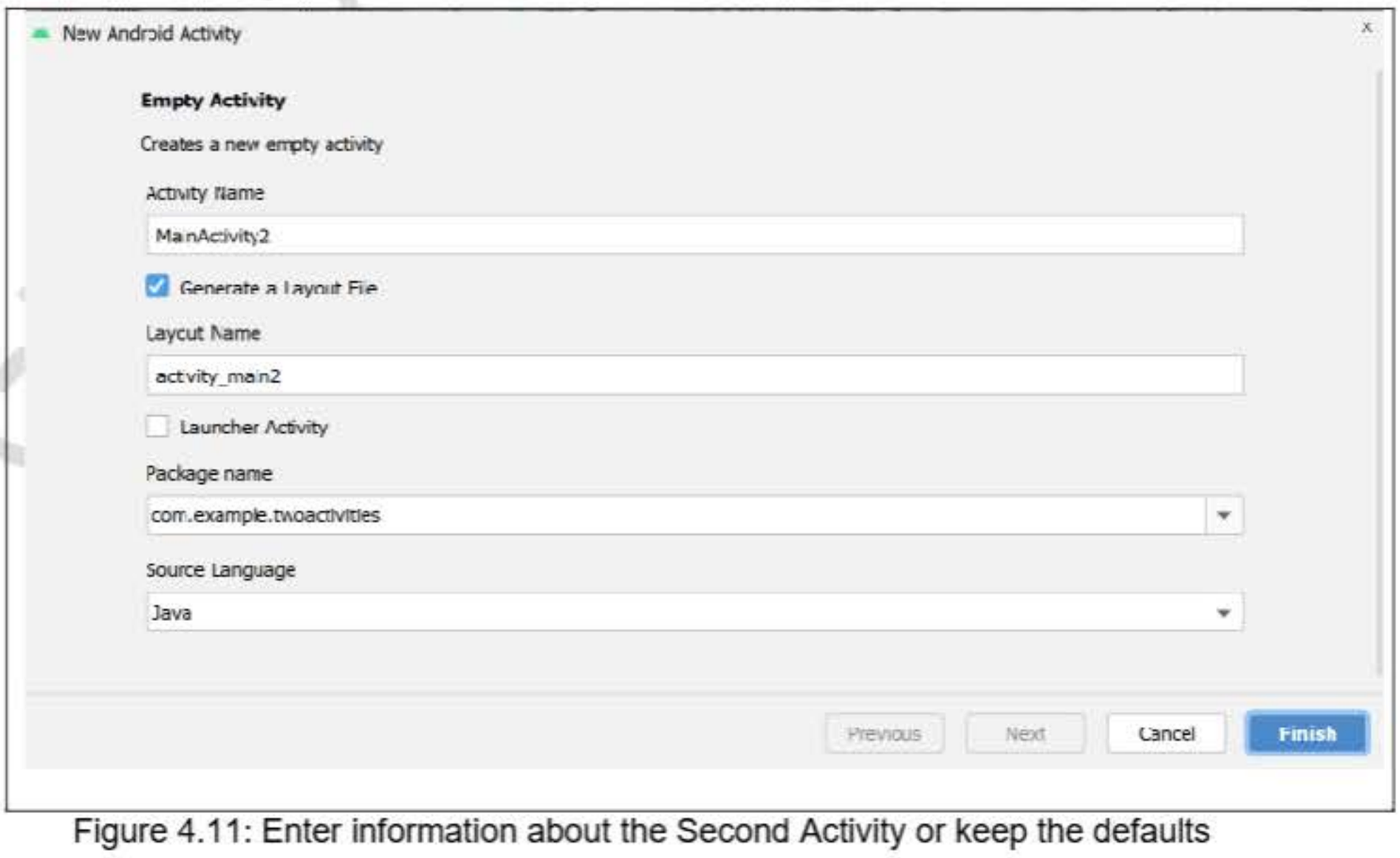


Figure 4.11: Enter information about the Second Activity or keep the defaults

The second activity gets created with the default files **`MainActivity2.java`** and **`activity_main2.xml.`**

5. In **`activity_main2.xml`** make the following changes

- Change the default layout to a Linear layout and also add **`android:orientation="vertical"`**

- Change the **`android:text`** to **"This is** the Second Activity of My **Project"**

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
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"
    android:orientation="vertical">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="200dp"
        android:gravity="center"
        android:text="This is the Second Activity of my project"
        android:textColor="@color/black"
        android:textSize="30dp"
        android:textStyle="bold" />

</LinearLayout>
```

Figure 4.12: Snippet of activity_main2.xml file with changes in code

6. In the **`MainActivity.java`** file add the following lines of codes for creating an intent object and starting the Second Activity i.e. MainActivity2.java

```
public class MainActivity extends AppCompatActivity {
    Button btn1;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    btn1 = (Button) findViewById(R.id.button1);
    btn1.setOnClickListener(new View.OnClickListener() {
        @Override
     public void onClick(View v) {
         Intent intentobj = new Intent(getApplicationContext(),
                                        MainActivity2.class );
          startActivity(intentobj);
        }
     });
    }
}
```

Figure 4.13: Add the required code in MainActivity2.java to start the Second Activity

7. Run the application, the output is as shown in figure 4.14

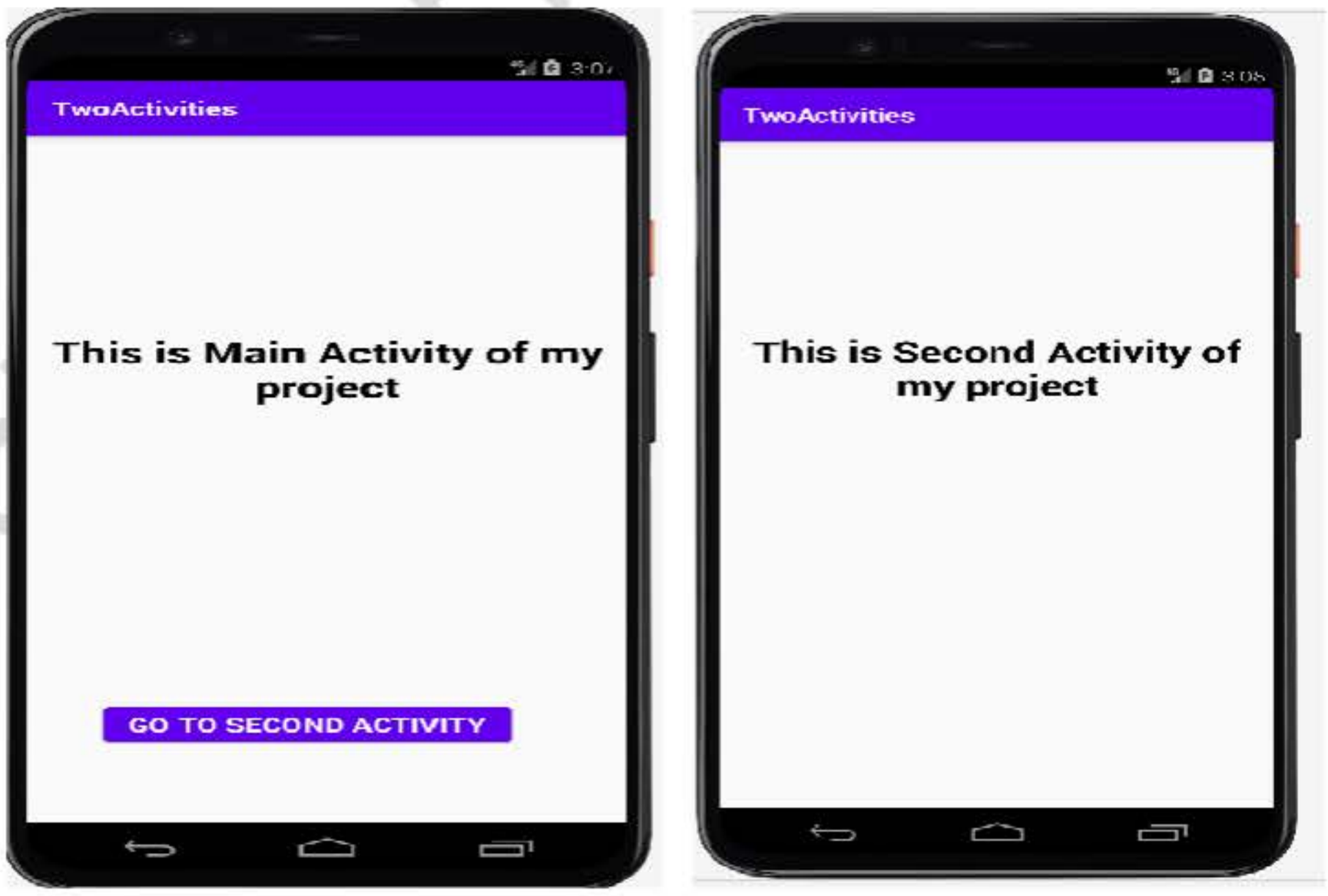


Figure 4.14: Main Activity and on click of the button displays Second Activity

4.4.2 Interaction amongst activities using intents

It's not just that we can move from one activity to another activity using intents, even data can be passed from one activity and retrieved by another

There are various methods that are used for passing and retrieving data using an intent object.

- **putExtra()** method is used when passing data

  1st parameter - is the name of the variable

  2nd parameter – is the value assigned to the variable

- **getExtra()** method is used to retrieve data

  Depending on the type of data to be retrieved appropriate getXXXExtra() method is used.

  - **getCharExtra()** – for retrieving character data
  - **getStringExtra()** - for retrieving String data
  - **getByteExtra()** – for retrieving byte data etc.

Passing data using an intent object: Example

```
Character signeduser = 'N';
…
…
Intent intent=new
Intent(LoginRegdActivity.this,RegdBuynowActivity.class);
intent.putExtra("signeduser", 'Y');     // passing of data
startActivity(intent);
```

Figure 4.15: Snippet showing passing of data from LoginRegdActivity to Buying for Registered customer

Retrieving data using an intent object: Example

```
Character signeduser = 'N';
…
…
Intent intent=getIntent();
signeduser = intent.getCharExtra("signeduser", 'N');
                                                //retrieving data
```

Figure 4.16: Snippet showing retrieving of data using the getCharExtra() method

4.5 Self-Assessment Questions

Q1. What is an Activity in Android? [3 marks, L2]

Q2. With a neat diagram explain the life cycle of an Activity in Android.[8 marks, L4]

Q3. Illustrate with an example the various stages in the life cycle of an Activity in Android [write the appropriate code] [6 marks, L2]

Q4. Explain Toast in android. Illustrate with an example code the use of Toast to display a message in an android application [6 marks, L2]

Q5. Explain the most commonly used methods of Toast class. [5 marks, L2]

Q6. What are Intents in Android? Explain its uses and types. [8 marks, L2]

Q7. Explain explicit and implicit intents. [6 marks, L2]

Q8. Describe the role of Intents in Android.[8 marks, L2]

Q9. Illustrate with a suitable example the use of Intents in linking activities.[8 marks, L2]

Q10. Illustrate with a suitable example the use of Intents in interaction among activities, i.e. passing and retrieving data using intents [8 marks, L2]

4.6 Self-Assessment Activities

Note: While executing the android application use an appropriate emulator.

A1. Create an Android Application and display a message using Toast "Hello using Toast for displaying the message"

A2. Create an Android Application, add a second activity, link the activities, add the required controls and display the appropriate messages. The output should be as shown in figure 4.14

4.7 Multiple-Choice Questions

1. An android application can have _____ activities. [1 mark. L1]
   a) 2
   b) 3
   c) 4
   d) Any number of

2. An activity in the android is _____. [1 mark. L1]
   a) A class
   b) A single screen containing the user interface
   c) A package
   d) None of the above

3. Which of the following method is not part of the activity life cycle? [1 mark. L2]
   a) onStart()
   b) onResume()
   c) onClick()
   d) onPause()

4. Toast is used to display information or show notifications in _____. [1 mark. L1]
   a) a small popup that appears for a small interval of time
   b) a dialog window
   c) a TextView
   d) None of the above

5. To navigate from one activity to another we use _____. [1 mark. L1]
   a) Toast
   b) Dialog
   c) Intent
   d) Adapter

6. An application can send/receive a broadcast message using _____. [1 mark. L1]
   a) sendBroadcast()
   b) sendBroadcastmsg()
   c) startBroadcast()
   d) None of the above

7. The types of intents are _____. [1 mark. L1]
   a) Hybrid and Single
   b) Implicit and Explicit
   c) Start and Close
   d) None of the above

8. To start another activity using the Intent object the method used is _____. [1 mark. L1]
   a) ActivityStart()
   b) startActivity()
   c) beginActivity()
   d) Activityresume()

Graphic Era University

9. To pass data from one activity to another activity using intent the method used is _____. [1 mark. L1]
   a) putdata()
   b) passdata()
   c) senddata()
   d) putExtra()

10. To retrieve data from an Intent in an activity the method used is _____. [1 mark. L1]
   a) getdata()
   b) getExtra()
   c) passdata()
   d) receivedata()

4.8 Key Answers to multiple-choice questions

1. An android application can have any number of activities.[d]
2. An activity in android is a single screen containing the user interface.[b]
3. onClick() method is not part of the activity life cycle.[c]
4. Toast is used to display information or show notifications in a small popup that appears for a small interval of time.[a]
5. To navigate from one activity to another we use intent.[c]
6. An application can send/receive a broadcast message using sendBroadcast().[a]
7. The types of intents are Implicit and Explicit.[b]
8. To start another activity using the Intent object the method used is startActivity().[b]
9. To pass data from one activity to another activity using intent the method used is putExtra().[d]
10. To retrieve data from an Intent in an activity the method used is getExtra().[b]

4.9 Summary

In an android application, activity is used to interact with the users. The activity contains the user interface of the application on a single window screen. An android application may perform various operations like sending SMS, getting information from users, processing data, etc. For all such operations, an activity is defined. A single application usually contains many activities. When an android application is launched, the main activity is displayed to the users. This is an entry point for executing an android application. The user interface of the main activity allows users to invoke other activities depending upon their request. A user can navigate back and forth among the activities. Whenever another activity is invoked, the

present activity is paused which can be resumed or stopped later. An activity goes through various stages/states known as the activity's life cycle.

An android activity has 7 lifecycle stages/processes. onCreate(), onStart, onResume, onPause(), onStop(), onRestart(),onDestroy().

Toast is used to display information or show notifications about a particular event that has happened successfully/unsuccessfully in a small popup for a particular interval of time. The message disappears/fades quickly after some time interval. It does not block or need user interaction.

Intents are used for communication. Communication can be between components (usually activities) of an application or from one application to another. In other words, intents are used for binding activities in a single application or different applications. There are 2 types of Intent - Explicit Intent and Implicit Intent.

Intents are used for linking activities and also for passing data across activities.

4.10 Keywords

- Activity
- Life Cycle
- Intent
- Implicit
- Explicit
- Toast
- Broadcast

4.11 Recommended Resources for Further Reading

Text Book(s)

1. Pradeep Kothari, "Android Application Development" Black-Book, Dreamtech Press,2015
2. Wei-Meng Lee, "Beginning Android Application Development", Wiley 2011.
3. Dawn Griffiths and David Griffiths, "Head First Android Development", O'Reilly, 2017.

References

- https://www.javatpoint.com/android-tutorial
- https://www.tutorialspoint.com/android/index.htm
- https://developer.android.com/guide/components/fundamentals

--*--

Graphic Era University

Contents

Topics	Page No.
Unit-2: Working with Android User Interface and handling UI Events	1
Chapter-5: Layouts and Views	1
5.0 Structure of Layouts and Views	1
5.1 Learning Outcomes	1
5.2 Overview of Layouts and Views	2
5.3 Layouts	5
5.3.1 Linear Layout	9
5.3.2 Relative Layout	15
5.3.3 Absolute Layout	15
5.3.4 Table Layout	19
5.3.5 Frame Layout	20
5.3.6 Scroll View	24
5.4 Views	25
5.4.1 Basic Views	26
5.4.1.1 Text View	27
5.4.1.2 Edit View	27
5.4.1.3 Button and Image View	30
5.4.1.4 Radio Group and Radio Button	35
5.4.1.5 Check Box	38
5.4.1.6 Image View	41
5.4.2 Picker Views	41
5.4.2.1 TimePicker View	45
5.4.2.2 DatePicker View	50
5.5 Adapter and Adapter Views	51
5.5.1 List View	52
5.5.2 Spinner View	52
5.5.3 Grid View	53
5.6 Self-Assessment Questions	53
5.7 Self-Assessment Activities	54
5.8 Multiple-Choice Questions	55
5.9 Key Answers to Multiple-Choice Questions	56
5.10 Summary	57
5.11 Keywords	57
5.12 Recommended Resources for Further Reading	

Graphic Era University

UNIT 2 - Working with Android User Interface and handling UI Events

CHAPTER 5 – Layouts and Views

Structure of Layouts and Views

5.1 Learning Outcomes
5.2 Overview of Layouts and views
5.3 Layouts
5.4 Views
5.5 Adapter and Adapter Views
5.6 Self-Assessment Questions
5.7 Self-Assessment Activities
5.8 Multiple-Choice Questions
5.9 Key answers to multiple-choice questions
5.10 Summary
5.11 Keywords
5.12 Recommended resources for further reading

5.1 Learning Outcomes

After the successful completion of this chapter, the student will be able to:

- Describe views and viewgroups in android
- Differentiate between various layouts used in android and demonstrate the use of layouts for creating a user-friendly interface while creating android applications
- Illustrate the use of adapters and adapter views in android

5.2 Overview of Layouts and Views

Activity in an android application contains the user interface with which the users interact. In other words, an activity's user interface is a collection of **`views`** and **`viewgroups.`** A **`view`** is a control/widget/element that is displayed on the screen. Buttons, textboxes, labels, etc. are examples of views. **`android.view.View`** is the base class for all views.

`Viewgroup` is a collection of one or more views. ViewGroup helps in specifying different types of layouts according to the arrangements and positions of the views/controls on the screen. Linear layout, relative layout, absolute layout, etc. are different types of layouts used by viewgroups. A ViewGroup is a subclass of View. It acts as an invisible container for other views and viewgroups. **`android.view.ViewGroup`** is the base class for all ViewGroups.

Graphic Era University

5.3 Layouts

Layouts are used as containers to define the user interface i.e. the arrangement of elements/controls on the screen in an android application. There are different types of layouts. The most commonly used layouts are :

- Linear Layout
- Relative Layout
- Absolute Layout
- Table Layout
- Frame Layout
- Scroll view

Layouts are declared in 2 ways

1. Declare the elements/views in XML layouts at design time – This helps in separating presentation i.e. user interface of an application from the java code which is placed in the activity class. The layout file has an extension of `.xml` and is placed in the `res/layout` folder.
2. Create elements/views at runtime – The element/view objects are created programmatically in java source code files.

Layouts can be declared in the XML layout file `activity_main.xml` as:

```
<XXXLayout ....>
...
...
</XXXLayout>
```

where **XXX** refers to the name of the layout.

The Layout can either be **"Relative", "Linear", "Absolute",** etc.

```
// Here "XXX" stands for either Relative, Linear, or Absolute Layouts

<XXXLayout 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">

   // Views for example TextView, EditText, and Button are declared here

   <TextView
      android:id="@+id/txtlabel1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Labels go here" />

   <Button
       android:id="@+id/btn_Click"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Click Button" />

</XXXLayout>
```

Figure 5.1: General structure of layouts with TextView and Button widgets placed within it

The XML layout **activity_main.xml** is loaded by the Activity class file when the **onCreate()** method is executed.

A reference to the XML layout file is passed as a parameter to the **setContentView()** method.

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

Figure 5.2 : Snippet showing onCreate() method referencing activity_main.xml layout file

The placement of layouts within the screen is controlled by setting the values of the attributes. Each layout is identified by its **ID** attribute. The width, height, background, spacing, etc. are managed by setting appropriate values of the attributes. The most commonly used layout attributes are shown in table 5.1.

Graphic Era University

Commonly used Layout Attributes:

Table 5.1: Commonly used layout attributes

Sr.No.	Attributes	Description
1	**android:id**	**ID** is used to uniquely identify the view or viewgroup. The value can be an **integer** or **string**. Internally during compilation, the ID is referenced as an integer.
2	**layout_width**	It is used to specify the width of the view of the viewgroup. The value can be given in terms of **units(px, dp, sp, etc.), "wrap_content", "match_parent", "fill_parent".**
3	**layout_height**	It is used to specify the height of the view of the viewgroup. The value can be given in terms of **units(px, dp, sp, etc.), "wrap_content", "match_parent", "fill_parent".**
4	**layout_margin** **layout_marginTop** **layout_marginBottom** **layout_marginLeft** **layout_marginRight**	It specifies the space **around** the view. The space at the top, bottom, left and right margins can be set by using the appropriate value.
5	**layout_padding** **layout_paddingTop** **layout_paddingBottom** **layout_paddingLeft** **layout_paddingRight**	It specifies the space **inside** the view i.e. the space **around the contents** within the view. The padding at the top, bottom, left and right can be set by using the appropriate value.

Graphic Era University

6	**layout_gravity**	It is used to arrange or **position** the view (e.g. A button, EditText, etc.) as a whole. It can take values **Top, Bottom, left, right, center, center_vertical center_horizontal,** etc. Multiple values can be set for a view by using a pipe "|" delimiter. For e.g. bottom|right.
7	**gravity**	It is used for the **alignment** of **content within** the view. It can take values **Top, Bottom, left, right, center, center_vertical center_horizontal,** etc. Multiple values can be set for a view by using a pipe "|" delimiter. For e.g. top|left.
8	**layout_weight weightSum**	**layout_weight** is used to specify the space to be occupied by an element on the screen. An element with a higher value will occupy more space on the screen. **weightSum** value if not set is calculated as the sum of all the layout_weight of the child elements
9	**background**	It is used to specify the **color/image** which can be used as a **background**

5.3.1 Linear Layout

In Linear Layout child elements/views are arranged in a single direction either vertically or horizontally based on the orientation property. The orientation is horizontal by default.

There are two types of orientations

- Vertical - The child elements are arranged one below the other, i.e. A single row has only one child element
- Horizontal - The child elements are arranged next to each other, i.e. A single row has multiple elements

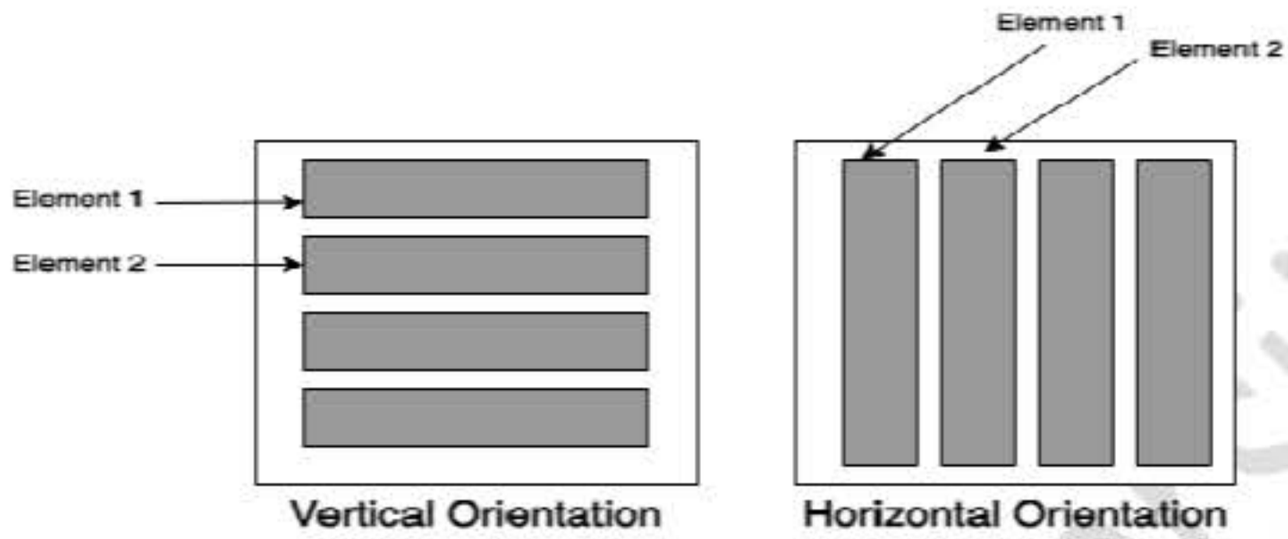


Figure 5.3: Linear Layout Vertical and Horizontal Orientation

Linear Layout Examples :

Example 1: An example of LinearLayout with **orientation** attribute set to "horizontal".

While defining the user interface the default constraint layout has to be changed to LinearLayout.

Here in this example, a linear layout with two elements, one **TextView** and another **EditText** is considered to show the alignment of elements as per the orientation.

The following code is a snippet of the **activity_main.xml** file.

```
<LinearLayout
    …….
    …..
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal" >

  <TextView
    android:id="@+id/TVcustname"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Customer Name"
```

```
    android:textColor="@color/black"
    android:textSize="20sp"
    android:textStyle="bold"
    android:layout_margin="10dp" />

  <EditText
    android:id="@+id/ETcustname"
    android:layout_width="wrap_content"
    android:layout_height="40dp"
    android:text="  Enter Customer Name  "
    android:textSize="20sp"
    android:textStyle="bold"
    android:layout_margin="10dp"
    android:background="@drawable/border_text" />

</LinearLayout>
```

Figure 5.4: Snippet for Linear Layout orientation set to Horizontal with TextView and EditText controls placed within it

A **TextView** and **EditText** controls appear next to each other on the screen during execution as shown in figure 5.5.

TextView attribute text is set as - **android:text**="Customer Name"

EditText attribute text is set as - **android:text**="Enter Customer Name"

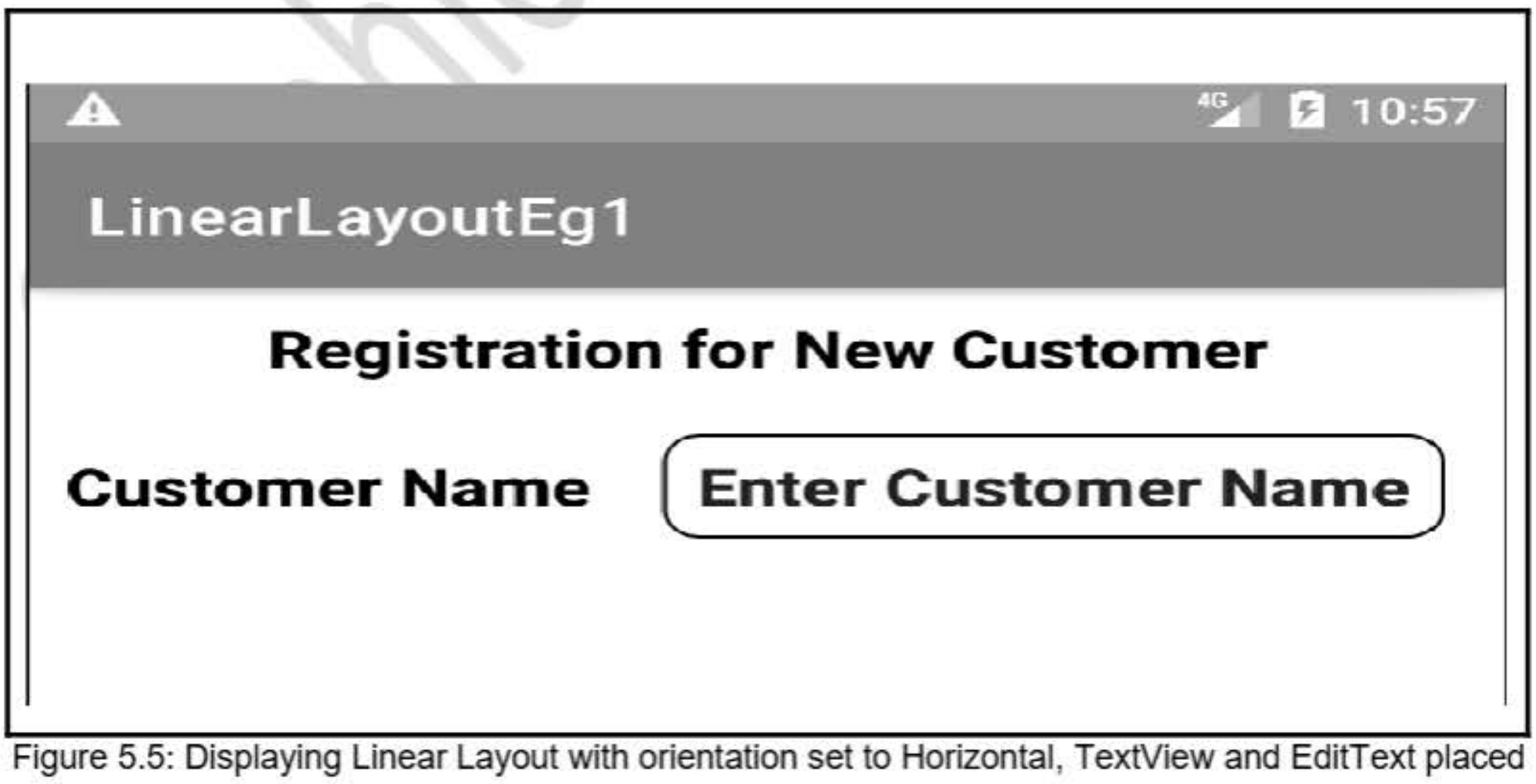


Figure 5.5: Displaying Linear Layout with orientation set to Horizontal, TextView and EditText placed next to each other

Example 2: An example of LinearLayout with orientation attribute set to "vertical".

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
  <TextView
     android:id="@+id/TVcustname"
     android:text="Customer Name"
     android:textColor="@color/black"
     android:textSize="20sp"
     android:textStyle="bold"
     android:layout_margin="10dp"      />
  <EditText
    android:id="@+id/ETcustname"
    android:layout_width="wrap_content"
    android:layout_height="40dp"
    android:text="  Enter Customer Name  "
    android:textSize="20sp"
    android:textStyle="bold"
    android:layout_margin="10dp"
    android:background="@drawable/border_text" />
</LinearLayout>
```

Figure 5.6: Snippet for Linear Layout orientation set to vertical with TextView and EditText placed within it

A **TextView** and **EditText** controls appear one below the other on the screen during execution as shown in Figure 5.7.

TextView attribute **text** is set as follows - **android:text**="Customer Name"

EditText attribute **text** is set as follows - **android:text**="Enter Customer Name"

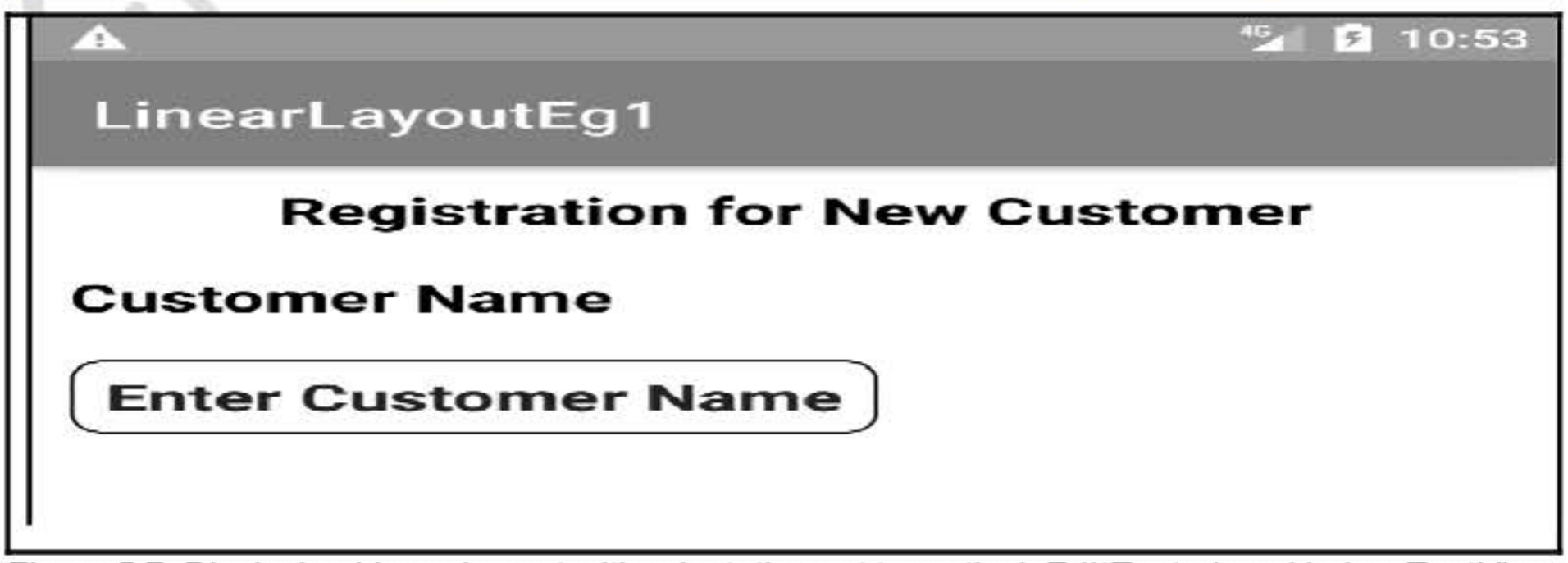


Figure 5.7: Displaying Linear Layout with orientation set to vertical, EditText placed below TextView

5.3.2 Relative Layout

- In a Relative layout, the elements or views are placed/positioned relative to each other.
- The position of each element is specified relatively with respect to its sibling or its parent container.
- The relative layout is widely used as it is more flexible.
- By default, the elements are placed at the top-left of the layout.
- The **ID** of the placed element "@id/XXXXX" is used as a reference to specify the value of an attribute while placing an element. If the referenced element **ID** is not declared/available it will produce an error.

The attributes used by the widgets placed in the relative layout are categorized based on the

- Placement of new view relative to siblings
- Alignment of new view with respect to existing element/view
- Alignment of new view with respect to the parent view

Commonly used Relative Layout attributes - Relative to Siblings

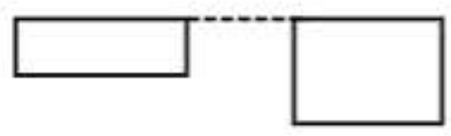
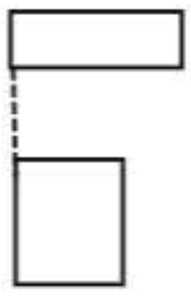
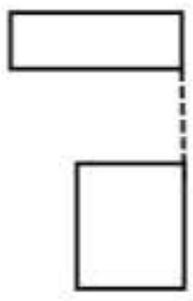
Table 5.2: Relative layout attributes - Relative to Siblings

Sr.No.	Attribute	Description
1	**layout_above**	It is used to place a view above another view that has been placed already in the layout.
2	**layout_below**	It is used to place a view below another view that has been placed already in the layout.
3	**layout_toLeftof**	It is used to place a view on the left side of another view which has been placed already in the layout.
4	**layout_toRightof**	It is used to place a view on the right side of another view which has been placed already in the layout.

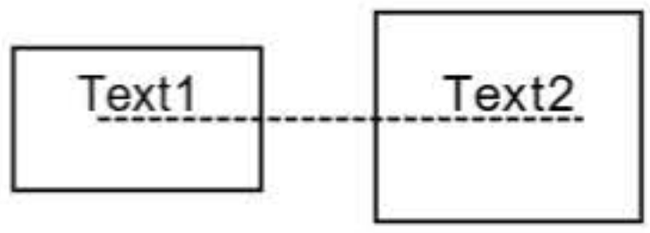
Graphic Era University

Commonly used Relative Layout attributes - Alignment of new view with an existing element

Table 5.3: Relative layout attributes - Alignment of new views w.r.t existing view

Sr.No.	Attribute	Description
1	layout_alignTop	The new element/view top margin is aligned with the top margin of another view which has been placed already in the layout 
2	layout_alignBottom	The new element/view bottom margin is aligned with the bottom margin of another view which has been placed already in the layout
3	layout_alignLeft	The new element/view left margin is aligned with the left margin of another view which has been placed already in the layout 
4	layout_alignRight	The new element/view right margin is aligned with the right margin of another view which has been placed already in the layout 

Graphic Era University

5	**layout_alignBaseline**	The contents(text) of the new view are aligned with the contents(text) of another view which has been placed already in the layout 

Commonly used Relative Layout attributes - Alignment of new view with parent view.

Table 5.4: Relative layout attributes - Alignment of views w.r.t parent view

Sr.No.	Attribute	Description
1	**layout_alignParentTop**	The new element's top is aligned with the top of the parent view
2	**layout_alignParentLeft**	The new element's left is aligned with the left of the parent view
3	**layout_alignParentRight**	The new element's right is aligned with the right of the parent view
4	**layout_alignParentBottom**	The new element's bottom is aligned with the bottom of the parent view
5	**layout_centerHorizontal**	The element is centered horizontally within its parent
6	**layout_centerVertical**	The element is centered vertically within its parent
7	**layout_centerInParent**	The element is centered horizontally and vertically within its parent

Graphic Era University

Relative Layout Examples :

Example 1: Placing one view below another using a relative layout

When a Relative layout is used, controls are placed depending on the position of the previous control. Here is a simple example showing one view placed below the other. **ID** is used to reference the above view.

```
<?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=".MainActivity">
    <TextView
        android:id="@+id/TVView1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="View 1"
        android:gravity="center"
        android:textColor="@color/black"
        android:textSize="20sp"
        android:textStyle="bold"
        android:layout_margin="30dp"
        android:background="@drawable/border_text"
        />
    <TextView
        android:id="@+id/TVView2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/TVView1"
        android:gravity="center"
        android:text="View 2 "
        android:textColor="@color/black"
        android:textSize="20sp"
        android:textStyle="bold"
        android:layout_margin="30dp"
        android:background="@drawable/border_text"   />
</RelativeLayout>
```

Figure 5.8: Snippet for Relative Layout with View2 placed below View1

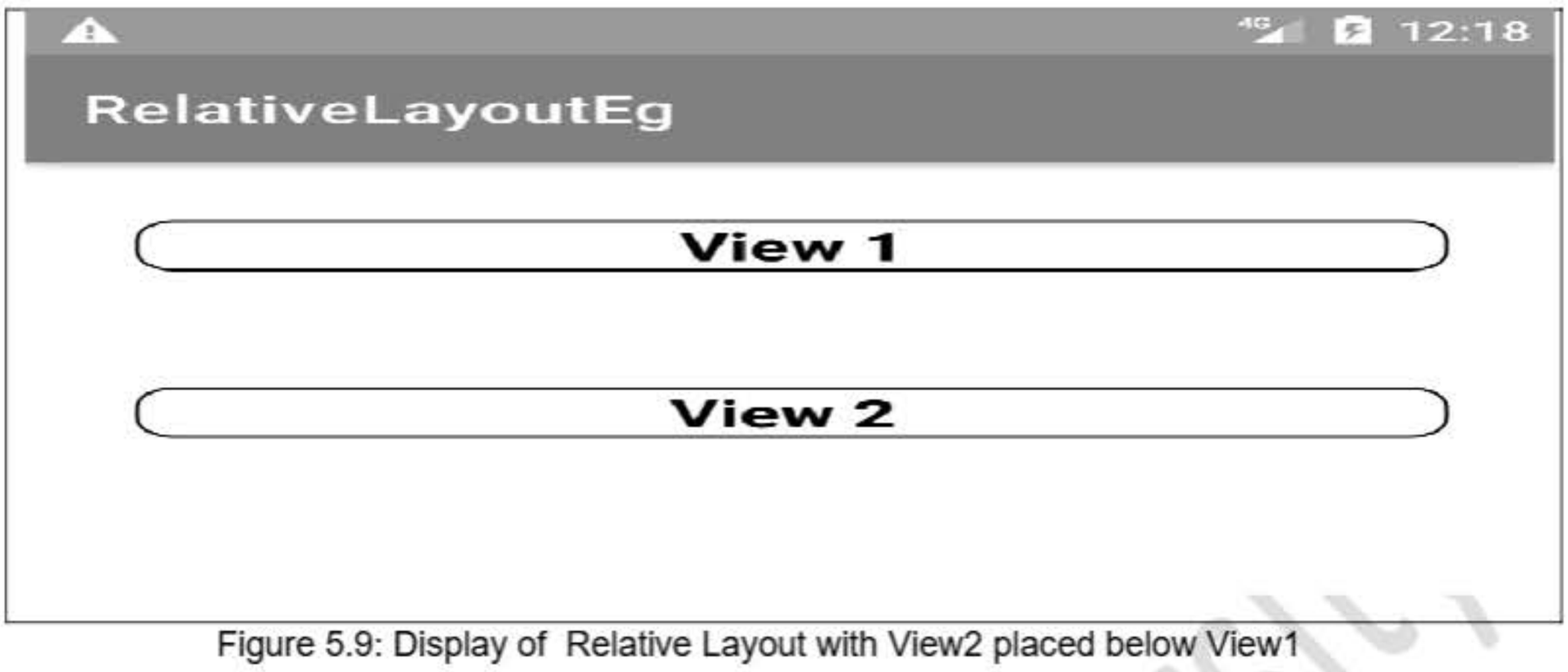


Figure 5.9: Display of Relative Layout with View2 placed below View1

Example 2: Placing views in relative layout using alignment attributes

Here is a simple example showing the first view i.e. View1 aligned with the parent view at the top, left, and right. The second View i.e. View2 is aligned with the parent bottom. **ID** is used to reference the above view.

```
<?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=".MainActivity">

    <TextView
        android:id="@+id/TVView1"
        android:layout_width="wrap_content"
        android:layout_height="40dp"
        android:text="View 1"
        android:background="@color/grey2"
        android:gravity="center"
        android:textColor="@color/black"
        android:textSize="20sp"
        android:textStyle="bold"
        android:layout_margin="20dp"
        android:layout_alignParentTop="true"
```

```
        android:layout_alignParentLeft="true"
        android:layout_alignParentRight="true"  />

    <TextView
        android:id="@+id/TVView2"
        android:layout_width="wrap_content"
        android:layout_height="40dp"
        android:layout_below="@+id/TVView1"
        android:gravity="center"
        android:text="View 2 "
        android:textColor="@color/black"
        android:textSize="20sp"
        android:textStyle="bold"
        android:layout_margin="20dp"
        android:background="@color/grey2"
        android:layout_alignParentBottom="true"  />

</RelativeLayout>
```

Figure 5.10: Snippet for Relative Layout with View2 placed below View1 using alignment with parent attributes

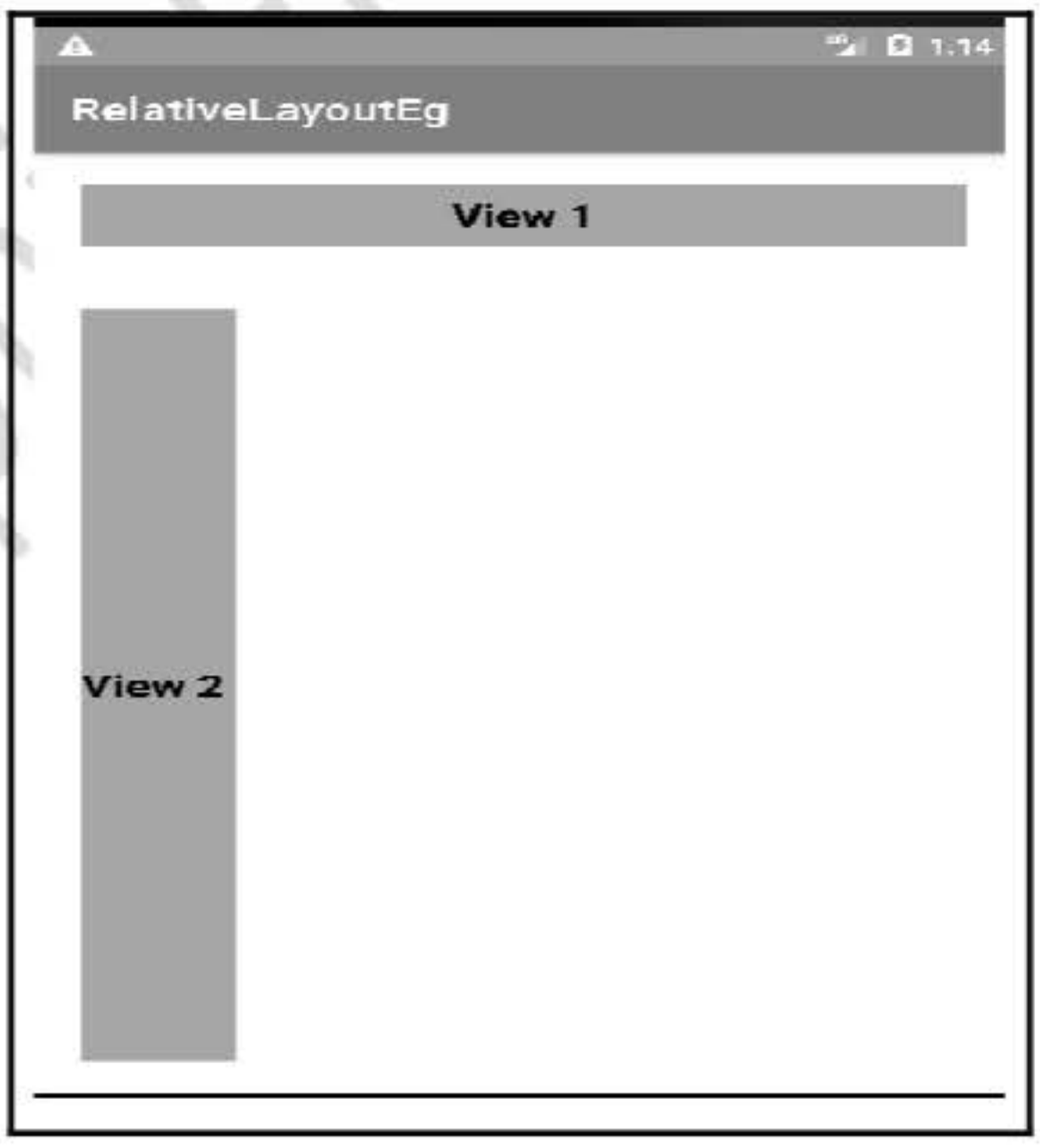


Figure 5.11: Display of Relative Layout with View1 and View2 using alignment with Parent View

5.3.3 Absolute Layout

Absolute layouts are difficult to maintain as the exact location of the element/view is specified by using x and y coordinates. Mobile devices differ in screen sizes so using an absolute layout makes the displays different. Attributes used for Absolute layout are shown here.

Table 5.5: Absolute layout attributes

Sr.No.	Attributes	Description
1	layout_x	It is used to specify the x_coordinate of the view.
2	layout_y	It is used to specify the y_coordinate of the view.

5.3.4 Table Layout

The elements/views in the Table layout are arranged in rows and columns.

- The **<TableRow>** tag is used to define a row within the root element **<TableLayout>.**
- A row can have zero or more cells and each cell can contain only one element/view e.g. **EditText, TextView, ImageView,** etc.
- A cell can even be empty.
- The maximum number of views in a row represents the number of columns in a table.
- In Table, layout borders are not displayed for rows, columns, or cells. By default, the elements are aligned to the left. Columns are internally referenced starting from 0,1,2... and so on.

Commonly used attributes for table layout are :

Table 5.6: Table layout attributes

Sr.No.	Attributes	Description
1	**stretchColumns**	The columns can be **stretched** to occupy the free space
2	**shrinkColumns**	The columns can be **shrunk** to reduce the width. The contents will be word-wrapped when shrunk.
3	**collapseColumns**	The columns can be made invisible by using this attribute. Later it can be made visible by using the **setColumnCollapsed()** method in the code i.e. programmatically

Table Layout Example :

An example of TableLayout consisting of 4 rows

1st Row contains only one cell with the title - "Customer Registration Form"

2nd Row contains 2 cells: **TextView** with text – **"Customer Name"**

EditText with inputtype set to "textPersonName"

3rd Row contains 2 cells: **TextView** with text – **"Mobile No"**

EditText with inputtype set to "textPhonetic"

4th Row contains 1 cell: Submit Button

The following is the TableLayout code for the specification mentioned

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout 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"
    android:orientation="vertical"
    tools:context=".MainActivity">
    <TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:padding="20dp"
        android:stretchColumns="*" >
        <!-- first row -->
        <TableRow>
            <TextView
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:text="Customer Registration Form"
                android:textStyle="bold"
                android:textSize="20dp"
                android:textColor="@color/black"
                android:padding="20dp"
                android:layout_span="2"
                android:gravity="center"  />
        </TableRow>
        <!-- second row -->
        <TableRow>
```

```
            <TextView
                android:id="@+id/TVView1"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                 android:text="Customer Name"
                android:textStyle="bold"
                android:textSize="20dp"
                android:textColor="@color/black"
                android:padding="5dp"/>
            <EditText
                android:id="@+id/ETCustname"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:hint="Enter Customer Name"
                android:inputType="textPersonName"
                android:textStyle="bold"
                android:textSize="20dp"
                android:textColor="@color/black"
                android:padding="5dp"/>
        </TableRow>
        <!-- Third row -->
         <TableRow>
            <TextView
                android:id="@+id/TVView2"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="Mobile No"
                android:textStyle="bold"
                android:textSize="20dp"
                android:textColor="@color/black"
                android:padding="5dp"/>
            <EditText
                android:id="@+id/ETMobileno"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:hint="Enter Mobile No"
                android:inputType="textPhonetic"
                android:textStyle="bold"
                android:textSize="20dp"
```

Graphic Era University

```
            android:textColor="@color/black"
            android:padding="5dp"/>
        </TableRow>
        <!-- Fourth row -->
        <TableRow
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:gravity="center"
            android:padding="15dp">
            <Button
                android:id="@+id/btnSubmit"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:background="@color/grey2"
                android:text="SUBMIT"
                android:textColor="@color/white"
                android:textStyle="bold"
                android:textSize="20dp"/>
        </TableRow>
    </TableLayout>
</LinearLayout>
```

Figure 5.12: Snippet of Table Layout consisting of 4 rows

On execution following user interface is displayed on the emulator

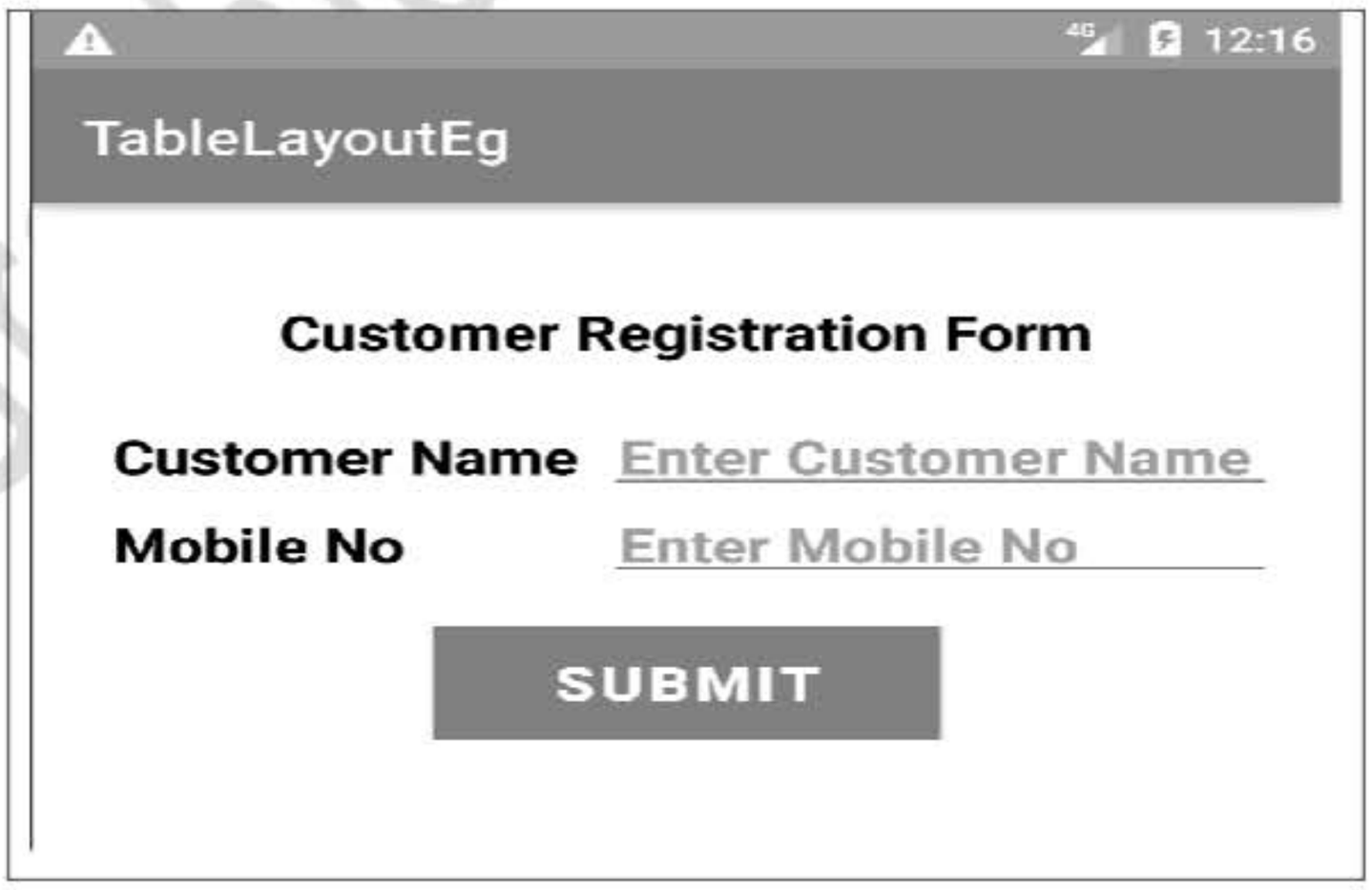


Figure 5.13: Display of Table Layout with 4 rows

5.3.5 Frame Layout

In Frame Layout, only a single element/view can be displayed at a time. Multiple elements/views can be added to the **Frame layout** but they are placed like a stack overlapping each other. The order of adding elements/views to the frame layout is of importance as the most recent element/view is shown at the top. However multiple elements can be made visible in a single frame by setting the position i.e. **layout_gravity**, **foreground** and **background** attribute values appropriately to the elements that are to be displayed. Frame layout acts as a placeholder.

Frame Layout Example:

Two controls **ImageView** and **TextView** are placed in a single-frame layout.

TextView text "Graphic Era Deemed University" is overlapping **ImageView** with consists of the Graphic Era logo.

The following is the FrameLayout code

```
<FrameLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
     <ImageView
        android:id="@+id/GREl ogo"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:scaleType="fitCenter"
        android:src="@drawable/grelogo"
        android:background="#D9E4EC"/>
    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="Graphic Era Deemed University Dehradun"
        android:textSize="30sp"
        android:textStyle="bold"
        android:textColor="@color/black"
        android:gravity="center"/>
</FrameLayout>
```

Figure 5.14: Snippet for Frame Layout with ImageView and TextView controls

On execution, following user interface is displayed on the emulator

Figure 5.15: Display of Frame Layout with TextView overlapping ImageView in single FrameLayout

5.3.6 Scroll View

ScrollView is generally used when the content to be displayed by an android application is larger and does not fit on a single screen.

- **ScrollView** supports vertical scrolling thereby allowing users to see the complete content. Only one direct element/view can be placed in a **ScrollView**.
- If multiple views are to be added within the **ScrollView** then they must be enclosed in another standard layout (e.g. **Linear Layout, Relative Layout, Table Layout,** etc.) which is its direct child element.
- Only vertical scrolling is supported by **ScrollView.**
- If horizontal scroll is required by an android application, then the **HorizontalScrollView** component has to be implemented.
- ScrollView should not be used with **ListView** and **RecyclerView** because ListView and RecyclerView itself are scrollable by default.

Example of ScrollView:

A ScrollView example shows scrolling of 10 buttons that do not fit on a single screen. According to the size of the screen, only a few buttons are displayed while other buttons are made visible on scrolling up and down.

```
<?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">

    <ScrollView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:scrollbars="vertical"   >

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical" >

        <Button
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_margin="10dp"
            android:text="B1"
            android:textSize="40dp"
            android:textStyle="bold"
            android:textColor="@color/black"
            android:background="@color/grey4" />

        <Button
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_margin="10dp"
            android:text="B2"
            android:textSize="40dp"
            android:textStyle="bold"
            android:textColor="@color/black"
            android:background="@color/grey4" />
```

```
        <Button
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_margin="10dp"
            android:text="B3"
            android:textSize="40dp"
            android:textStyle="bold"
            android:textColor="@color/black"
            android:background="@color/grey4" />
        <Button
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_margin="10dp"
            android:text="B4"
            android:textSize="40dp"
            android:textStyle="bold"
            android:textColor="@color/black"
            android:background="@color/grey4" />
        <Button
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_margin="10dp"
            android:text="B5"
            android:textSize="40dp"
            android:textStyle="bold"
            android:textColor="@color/black"
            android:background="@color/grey4" />
        <Button
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_margin="10dp"
            android:text="B6"
            android:textSize="40dp"
            android:textStyle="bold"
            android:textColor="@color/black"
            android:background="@color/grey4" />
        <Button
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
```

```
            android:layout_margin="10dp"
            android:text="B7"
            android:textSize="40dp"
            android:textStyle="bold"
            android:textColor="@color/black"
            android:background="@color/grey4" />
        <Button
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_margin="10dp"
            android:text="B8"
            android:textSize="40dp"
            android:textStyle="bold"
            android:textColor="@color/black"
            android:background="@color/grey4" />
        <Button
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_margin="10dp"
            android:text="B9"
            android:textSize="40dp"
            android:textStyle="bold"
            android:textColor="@color/black"
            android:background="@color/grey4" />
        <Button
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_margin="10dp"
            android:text="B10"
            android:textSize="40dp"
            android:textStyle="bold"
            android:textColor="@color/black"
            android:background="@color/grey4" />
    </LinearLayout>

  </ScrollView>

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

Figure 5.16: Snippet for ScrollView with 10 Button controls

On execution, following user interface is displayed on the emulator

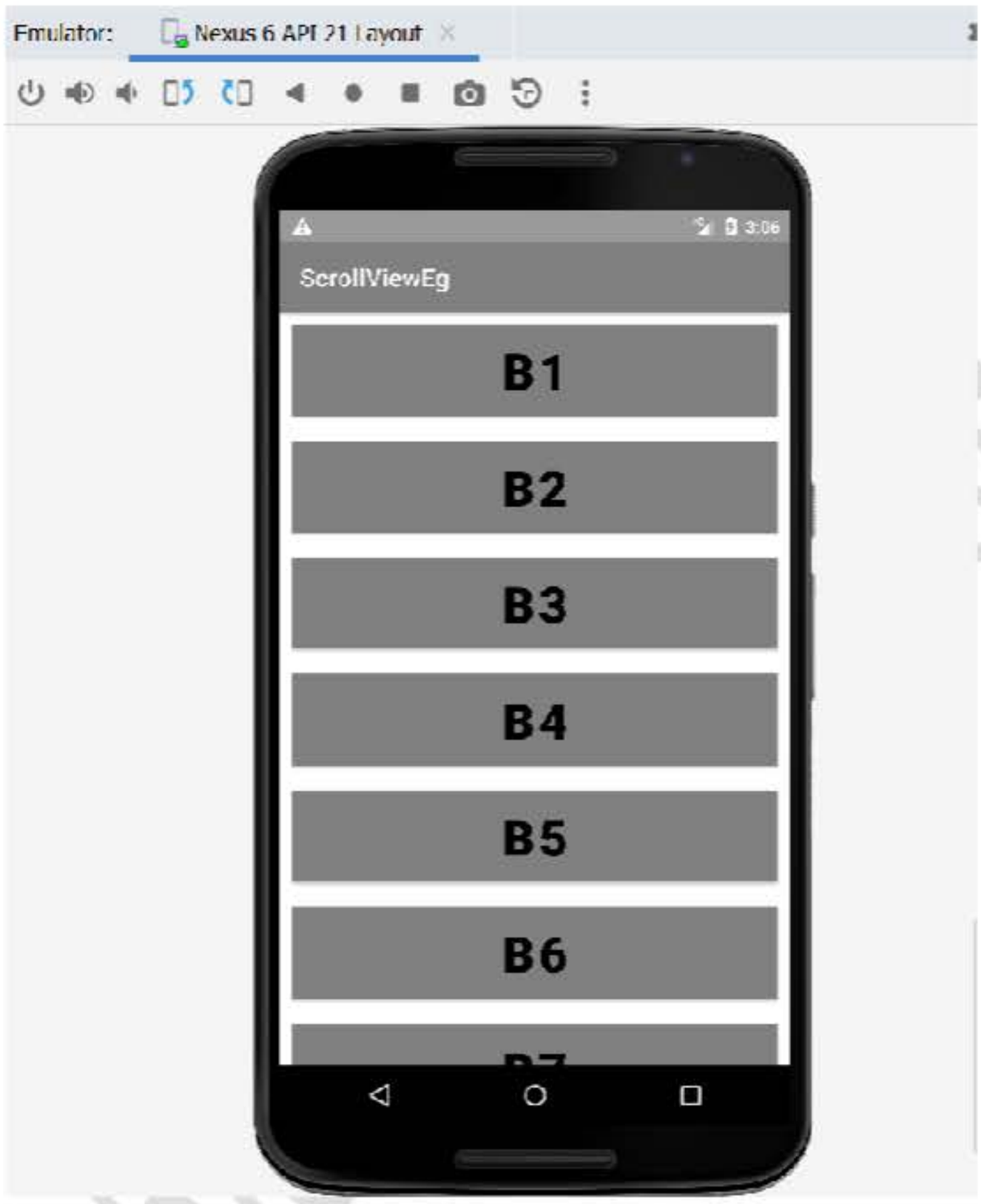


Figure 5.17: Display of ScrollView with Button controls scrollable vertically

5.4 Views

Views are used for creating the user interface in an android application. Views can be created either programmatically using java code or by declaring it in XML layout files.

In Android, there are 3 types of Views

- **Basic** Views
- **Picker** Views
- **List** Views

5.4.1 Basic Views

Basic Views are the general views/widgets/controls commonly used in any application.

For example- **TextView**, **EditView, Button, RadioButton**, etc.

The commonly used attributes with basic views are

Table 5.7: Basic View commonly used attributes

Sr. No.	Attribute	Description
1	**android:id**	**ID** is used to uniquely identify the view The value can be an integer or string. Internally during compilation, the ID is referenced as an **integer.**
2	**layout_width**	It is used to specify the width of the view. The value can be given in terms of **units(px, dp, sp etc), "wrap_content ", "match_parent ", "fill_parent ".**
3	**layout_height**	It is used to specify the height of the view. The value can be given in terms of **units(px, dp, sp etc), "wrap_content", "match_parent", "fill_parent ".**
4	**android:text**	It is used to specify the **text** to be displayed in the view.
5	**android:textSize**	It is used to specify the **size** of the text. The value can be given in terms of units(px, dp, sp, in, etc).
6	**android:textColor**	It is used to specify the **color** of the text. The value can be either a hexadecimal value "#FFFFFF" or the name of the string defined in the colors.xml file.

Graphic Era University

7	**android:textStyle**	It is used to specify the **style** of the text. The value can be either "bold", "italic" or "normal".
8	**android:background**	It is used to specify the **background** of the view. The value can be either a color value or the name of the drawable(image) file placed in the drawable folder.
9	**android:hint**	When a view is empty it is used to display the **text as a hint**.
10	**android:gravity**	It is used to specify the **direction** of the **contents** in the view.
11	**android:padding**	It specifies the **space inside** the view i.e. **around the contents** of the view. The padding at the top, bottom, left and right can be set by using the appropriate property

5.4.1.1 Text View

It is used to display pre-defined text on the screen. It acts like a label and it is used alongside other views for e.g. while creating forms. By default this view is un-editable but it can be configured to be editable.

Example of TextView

```
<TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="Graphic Era Deemed University Dehradun"
        android:textSize="30sp"
        android:textStyle="bold"
        android:textColor="@color/black"
        android:gravity="center"/>
```

Figure 5.18: Snippet for TextView with essential attributes

5.4.1.2 Edit View

It is an editable view/control. It is used to provide input from the keyboard.

Example of EditText

```
<EditText
      android:id="@+id/ETcustname"
      android:layout_width="wrap_content"
      android:layout_height="40dp"
      android:text="  Enter Customer Name  "
      android:textSize="20sp"
      android:textStyle="bold"
      android:layout_margin="10dp"
      android:background="@drawable/border_text"  />
```

Figure 5.19: Snippet for EditText with essential attributes

5.4.1.3 Button and ImageButton

A `Button` view can be defined either with text, an icon, or both.

An XML layout file can be defined in 3 ways

- A button with text
- A button with both text and an icon
- A button with an image/icon

- A button with text using `<Button>` view

```
<Button
      android:id="@+id/Btn_AddCart"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:background="#6F7171"
      android:text="ADD TO CART"
      android:textColor="@color/white"
      android:textSize="20sp"
      android:textStyle="bold" />
```

Figure 5.20: Button view with text and essential attributes

Graphic Era University

- A button with text and icon using `<Button>` view and `android:drawableLeft` attribute

```
<Button
    android:id="@+id/Btn_AddCart"
    android:layout_width="150dp"
    android:layout_height="wrap_content"
    android:background="#6F7171"
    android:text="Add To Cart"
    android:textColor="@color/white"
    android:textSize="20sp"
    android:textStyle="bold"
    android:drawableLeft="@drawable/ic_baseline_shopping_cart_24"
/>
```

Figure 5.21: Button view with text, icon, and essential attributes

- A button with only an image/icon

  A `Button` with an only icon is defined using `ImageButton` class

```
<ImageButton
    android:layout_width="150dp"
    android:layout_height="100dp"
    android:src="@drawable/ic_baseline_add_shopping_cart_34" />
```

Figure 5.22: ImageButton view with essential attributes

A Button is a view/widget which is used to perform an action when it is clicked/pressed. It belongs to the push-button category.

GraphicEra University

An event handler i.e. the method to be called on click event is specified either in the XML layout file by setting the **android:onClick** attribute or by setting on click event listener in the activity class.

android:onClick attribute of the **<Button>** element is set in an XML layout file to perform an action. In other words, the name of the method to be called on Click event must be set as the value of the **android:onClick** attribute.

For example **android:onClick** = "AddtoCart".

Here "AddtoCart" is the method that is called on click of the button, so it is called an event handler.

The button element in the XML layout file is defined as follows

```
<Button
    android:id="@+id/Btn_AddCart"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="#6F7171"
    android:text="ADD TO CART"
    android:textColor="@color/white"
    android:textSize="20sp"
    android:textStyle="bold"
    android:onClick="AddtoCart" />
```

Figure 5.23: Button view with text, onClick, and essential attributes

The method AddtoCart() to be called is defined in the Activity class referencing the layout file. The method should have public access and should not return a value. The return type of the method is void. The object of view (the view that was clicked) is the only parameter passed to the method.

```
public void AddtoCart(View view) {
    // java Code here to perform an action on the click of a button
   }
```

Figure 5.24: Declaration of AddtoCart method in Activity class

Setting on click event listener in activity class

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

  // referencing activity_main.xml file
     setContentView(R.layout.activity_main);

  // creating an instance of a button by referencing its ID Btn_AddtoCart
     Button bobj = (Button) findViewById(R.id.Btn_AddtoCart);

  // setting on click event listener
      bobj.setOnClickListener(new View.OnClickListener() {
         public void onClick(View v) {
           // Code here to perform an action on click to add data to the cart
            Toast.makeText(getApplicationContext()," Data Added to Cart",
                        Toast.LENGTH_LONG).show();
         }
      });
    }
}
```

Figure 5.25: Snippet of setting click event listener for Button in Activity class

5.4.1.4 Radio Group and Radio Button

RadioGroup is used to group more than one RadioButton view/widget. A radio button is used when the user wants to select only one single radio button from a group of radio buttons. It has two states – checked and unchecked.

When the user checks one radio button then all other radio buttons in a group become unchecked. A radio button can be either in a checked or unchecked state.

An unchecked radio button can be checked by the user by just clicking on it, but when the same radio button is to be unchecked the user has to click on another radio button.

android:checked attribute is used to set its value to either "true" or "false ".

By default, it is "false ".

To set an icon for the radio button depending on the position of the display an appropriate attribute like android:drawableBottom, android:drawableTop, android:drawableRight, android:drawableLeft can be used.

Example of Radiobuttons and RadioGroup

An activity_login layout showing declaration of Radiobuttons in RadioGroup along with TextView and Button view

```
<RelativeLayout
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:layout_marginTop="10dp">
      <TextView
          android:id="@+id/TVappname1"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:layout_marginStart="16dp"
          android:text = "@string/app_name"
          android:textSize="20sp"
          android:textColor="@color/white"
          android:textStyle="bold" />
      <TextView
          android:id="@+id/TVappmsg"
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
          android:text="Welcome to GEU Online Shopping"
          android:textSize="20sp"
          android:textStyle="bold"
          android:layout_below="@+id/TVappname1"
          android:layout_marginLeft="20dp"
          android:layout_centerHorizontal="true"  />
      <RadioGroup
          android:id="@+id/RBGusr1"
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
          android:layout_below="@+id/TVappmsg"
          android:layout_marginTop="20dp" >
          <RadioButton
              android:id="@+id/RBnewusr"
              android:layout_width="match_parent"
```

```
            android:layout_height="wrap_content"
            android:text="Create a new account"
            android:textSize="20sp"
            android:textStyle="bold"
            android:layout_marginLeft="20dp"/>
        <RadioButton
            android:id="@+id/RBregdusr"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Sign In"
            android:textSize="20sp"
            android:textStyle="bold"
            android:layout_marginLeft="20dp"
            android:checked="true" />
    </RadioGroup>
    <Button
        android:id="@+id/BTcontinue"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/RBGusr1"
        android:text= "Continue"
        android:textSize="20sp"
        android:textStyle="bold"
        android:padding="20dp"    />
</RelativeLayout>
```

Figure 5.26: RadioGroup and RadioButton with text and essential attributes

Add on click listeners i.e. setOnClickListener() for a radio button in the corresponding Activity class. A user can choose either of the radio buttons

- Create a New account
- Sign In

Note: Activities LoginNewActivity and LoginRegdActivity have to be created so that they can be started by selecting the radio buttons

```
public class LoginActivity extends AppCompatActivity {
    RadioButton RBnewlogin;
    RadioButton RBregdusr;
    Button BTcontinue;
```

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

    RBnewlogin = (RadioButton) findViewById(R.id.RBnewusr);
    RBregdusr = (RadioButton) findViewById(R.id.RBregdusr);
    BTcontinue = (Button) findViewById(R.id.BTcontinue);

    RBnewlogin.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
              // Code for moving to new user registration
          Intent intent=new Intent(LoginActivity.this,
                                   LoginNewActivity.class);
          startActivity(intent);
         }
     });

    RBregdusr.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
             // Code for moving to registered user
        Intent intent=new Intent(LoginActivity.this,
                                 LoginRegdActivity.class);
        startActivity(intent);
         }
     });

    BTcontinue.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
        // Code for moving to another activity
        }
    });
}
```

Figure 5.27: Declaration of onclick event listeners for Radio buttons in Activity class

On Running the Application the following User Interface gets displayed. New users have to register themselves and registered users can sign in.

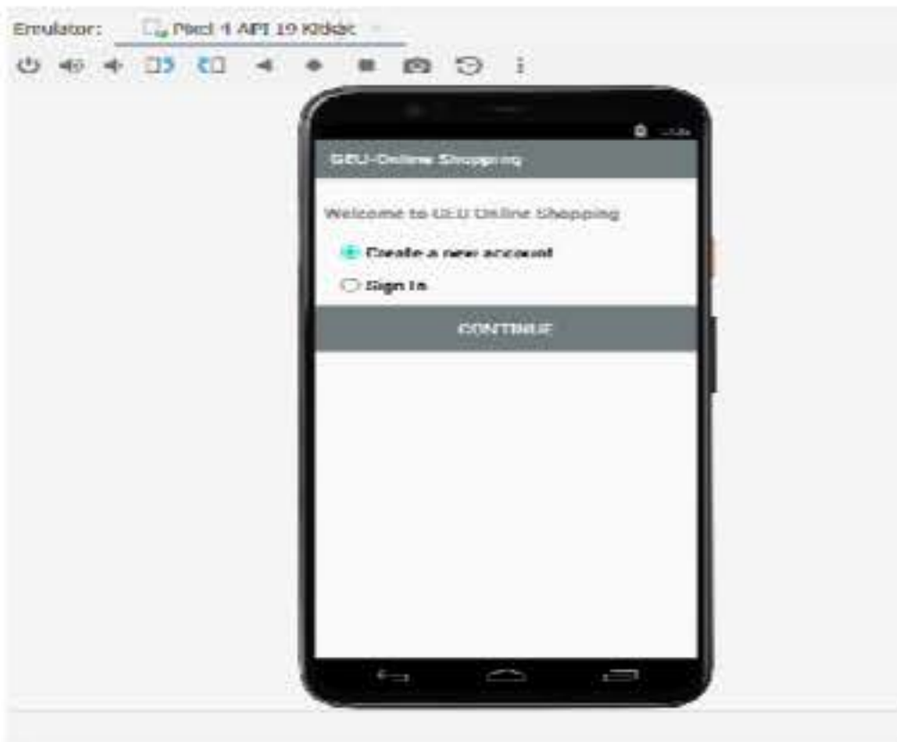


Figure 5.28: User Interface for login Activity

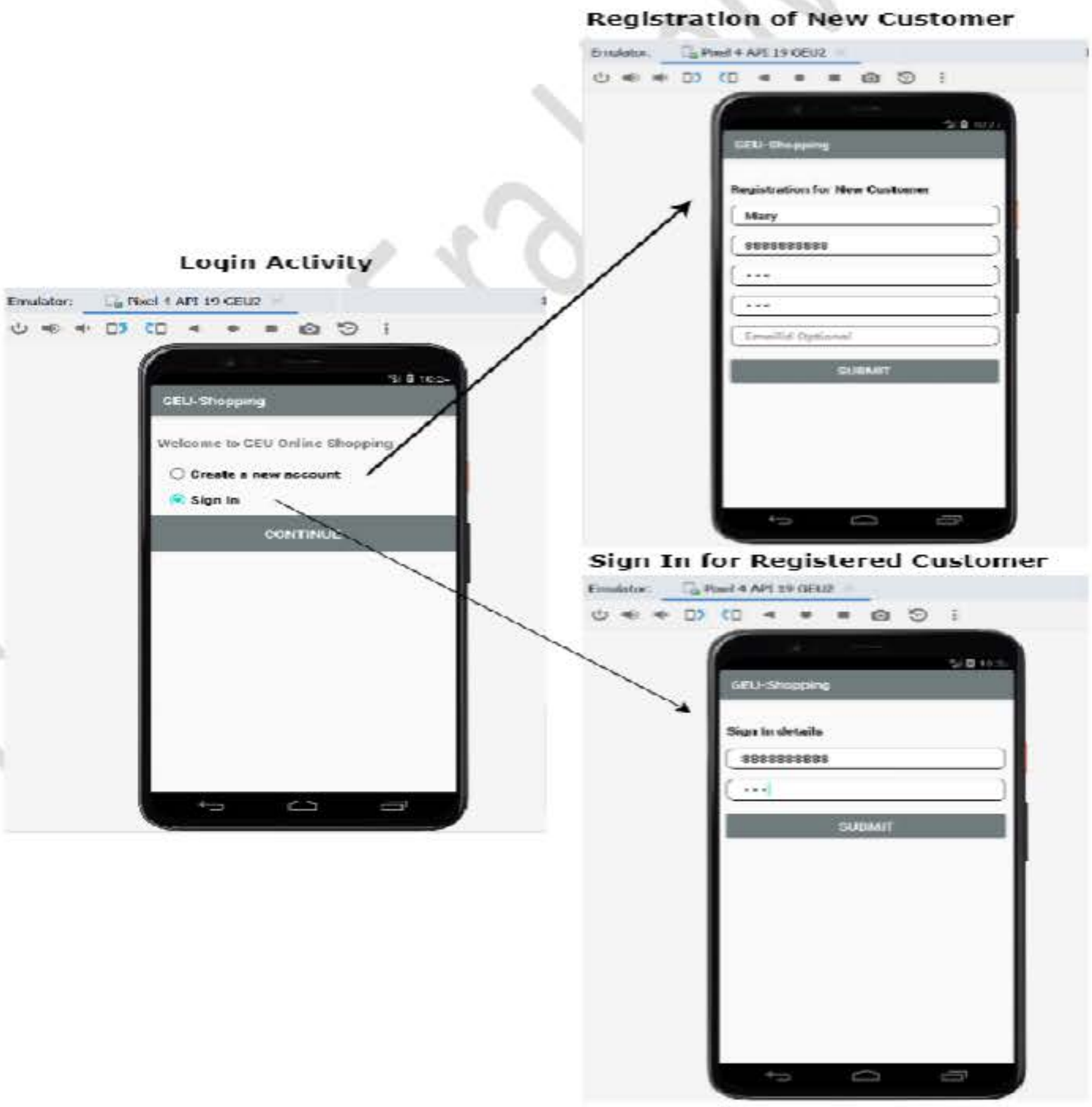


Figure 5.29: Display of LoginActivity, Registration of new Customer Activity, and Registered customer Activity

5.4.1.5 Check Box

A checkbox button is usually used when the user wants to select more than one checkbox in a group. A checkbox, like a radio button, has two states - checked and unchecked. By default all checkboxes are in an unchecked state i.e. they have a value "false". When the user clicks the checkbox, it gets selected and re-clicking on the same unchecks/deselects it. The methods "isChecked()" is used in the code to verify whether a checkbox is selected by the user or not. This method returns a boolean value either "true " or "false ". The value of the checkbox can be retrieved by using the getText() and toString() methods.

CheckBox Example :

Here a list of item categories is displayed. The user can select more than one item and the selected items are displayed on the screen using Toast. A layout showing declaration Checkboxes along with a Button view.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout 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"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/TVitmcatg"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Select Item Category"
        android:gravity="center"
        android:textColor="@color/black"
        android:textSize="20sp"
        android:textStyle="bold"
        android:layout_margin="10dp"
        />
    <CheckBox
        android:id="@+id/chkitem1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:text="Laptop"
        android:textColor="@color/black"
        android:textSize="20sp"
        android:textStyle="bold"
        android:layout_margin="10dp"
        />
    <CheckBox
        android:id="@+id/chkitem2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Watches"
        android:textColor="@color/black"
        android:textSize="20sp"
        android:textStyle="bold"
        android:layout_margin="10dp"
        />
    <CheckBox
        android:id="@+id/chkitem3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Mobile"
        android:textColor="@color/black"
        android:textSize="20sp"
        android:textStyle="bold"
        android:layout_margin="10dp"
        />
    <Button
        android:id="@+id/btn1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text = "Show selected items"
        android:textColor="@color/white"
        android:textSize="20sp"
        android:textStyle="bold"
        android:layout_margin="10dp"/>
</LinearLayout>
```

Figure 5.30: Checkboxes with textView and Button with essential attributes

A `setOnClickListener()` is added to the button. The items which are checked are appended to the string. With a click of a button, the items selected by the user are displayed using Toast.

```
public class MainActivity extends AppCompatActivity {
    Button btn1;
    CheckBox chklaptop, chkmobile, chkwatch;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        chklaptop=(CheckBox)findViewById(R.id.chkitem1);
        chkwatch=(CheckBox)findViewById(R.id.chkitem2);
        chkmobile=(CheckBox)findViewById(R.id.chkitem3);
        btn1=(Button)findViewById(R.id.btn1);
        btn1.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                StringBuilder itemsselect=new StringBuilder();
                itemsselect.append("Selected Items are:");
                if(chklaptop.isChecked())
                {
                    itemsselect.append("\n Laptop");
                }
                if(chkwatch.isChecked())
                {
                    itemsselect.append("\n Watches");
                }
                if(chkmobile.isChecked())
                {
                    itemsselect.append("\n Mobile");
                }
                //Displaying selected items using toast
                Toast.makeText(getApplicationContext(), itemsselect.toString(),
                                        Toast.LENGTH_LONG).show();
            }
        });
    }
}
```

Figure 5.31: Declaration of onclick event listeners for button in Activity class

On Running the Application the following User Interface gets displayed.

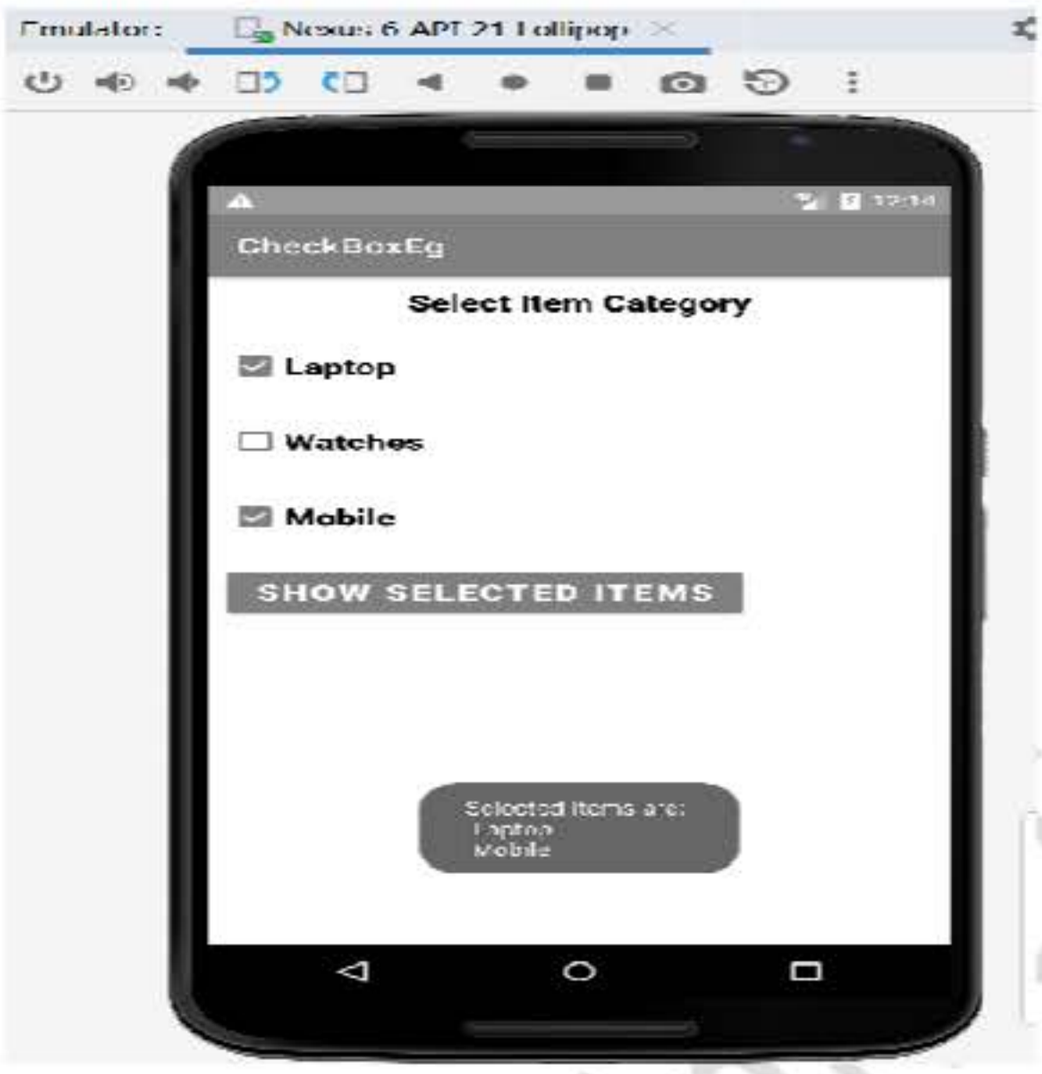


Figure 5.32: User Interface showing Checkboxes

5.4.1.6 Image View

It is used to display images on the user interface.

Commonly used attributes of ImageView

Table 5.8: Image View commonly used attributes

Sr.No.	Attribute	Description
1	android:id	ID is used to uniquely identify ImageView. The value can be an integer or string. Internally during compilation, the ID is referenced as an integer.
2	android:maxWidth	It is used to specify the maximum width for an ImageView
3	android:maxHeight	It is used to specify the maximum height for an ImageView
4	android:src	It is used to specify the image file name which is displayed as the content of

		ImageView. The image file should be present in the res/drawable folder.
5	**android:scaletype**	It is used to control resizing of the image. The value can be set to "center", "fitCenter", "fitStart" etc
6	**android:background**	It is used to set the background color to an image which is displayed when the entire area of ImageView is not used by the image.

Example of ImageView defined within Linear Layout in the XML layout file

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout 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"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:text="Displaying GRElogo using ImageView"
        android:textSize="20sp"
        android:textStyle="bold"
        android:textColor="@color/black"/>

    <ImageView
        android:id="@+id/GRElogo"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:scaleType="fitCenter"
        android:src="@drawable/grelogo"
        android:background="#D9E4EC"/>
</LinearLayout>
```

Figure 5.33: ImageView with src, background, and essential attributes

Setting the Image Resource for an ImageView Object in the Activity class

```
public class MainActivity extends AppCompatActivity {
    ImageView imageViewobj;

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

  // Create an ImageView Object by referencing the ID of ImageView
  // in layout

  ImageView imageViewobj =(ImageView)
                        findViewById(R.id.GRElogo);

  //set the image source file name present in the drawable folder
    imageViewobj.setImageResource(R.drawable.grelogo);
    }
}
```

Figure 5.34: Setting Image Resource for ImageView object in Activity class

On running the application the image is displayed on the screen as follows

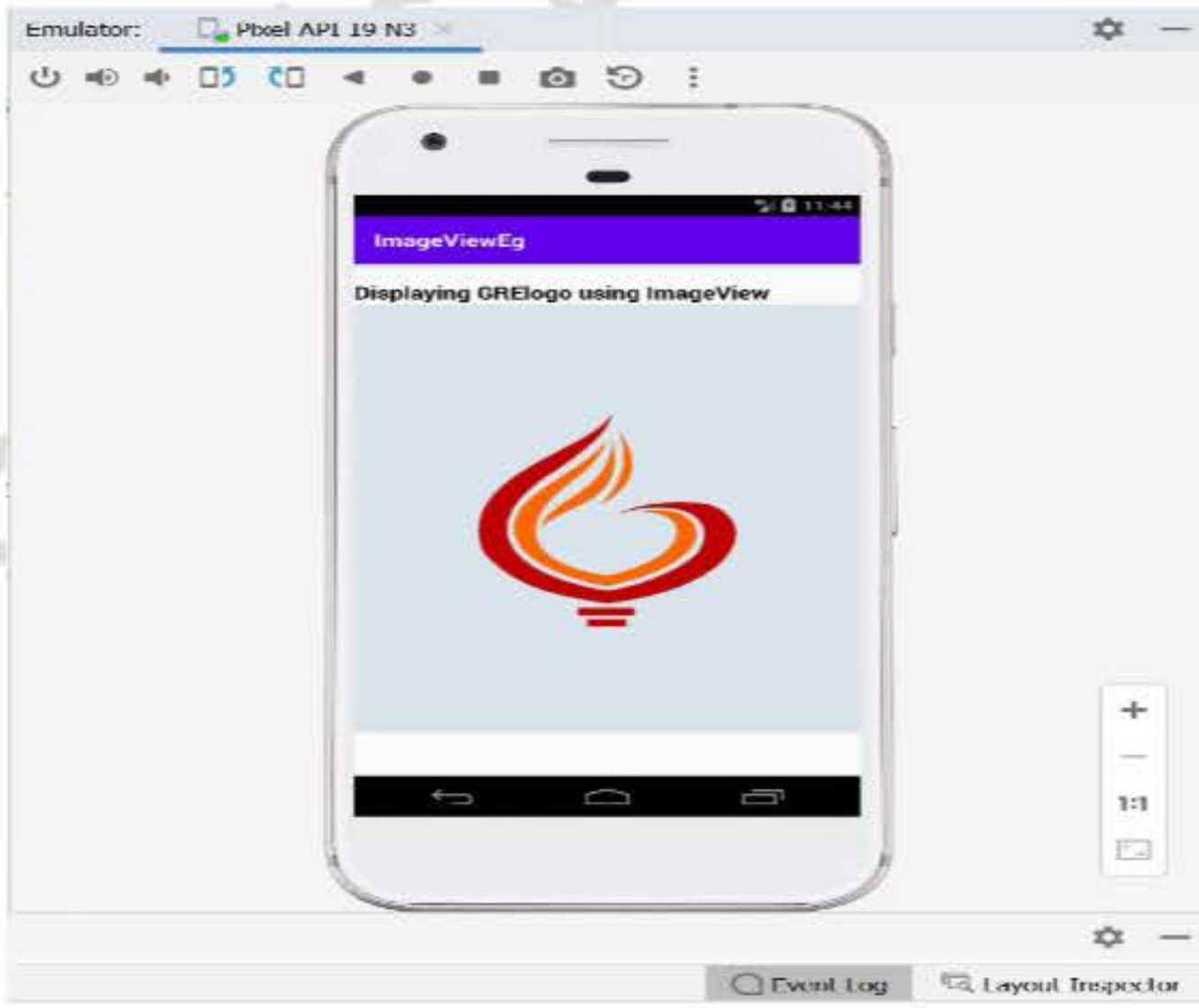


Figure 5.35: Displaying an image i.e. GRE logo using ImageView

5.4.2 Picker Views

Pickers are used in android for selecting date and time from the DatePicker and TimePicker controls respectively.

5.4.2.1 TimePicker View

A TimePicker control is used in an android application so that the user selects a valid time from the display. The user can select the required time of a day either in AM/PM mode or 24 hours clock. TimePicker requires API level 21 and above to be used in an android application.

TimePicker control has 2 modes of displayed - Clock and Spinner
The mode can be set by using the attribute "timePickerMode".

- android:timePickerMode = "clock"
- android:timePickerMode = "spinner"

The following is the XML layout showing the TimePicker to be displayed in "Clock" mode.

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

<LinearLayout 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"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Displaying Time Picker in Clock Mode"
        android:layout_marginTop="20dp"
        android:gravity="center_horizontal"
        android:textSize="25sp"
        android:textColor="@color/black"
        android:textStyle="bold"/>
```

```
<TimePicker
    android:id="@+id/Tmpicker2"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="40dp"
    android:timePickerMode="clock"
    android:layout_centerHorizontal="true"
    />

</LinearLayout>
```

Figure 5.36: TimePicker with TimePickerMode set to “Clock” and other essential attributes

On running the application the clock is displayed on the screen in clock mode as follows

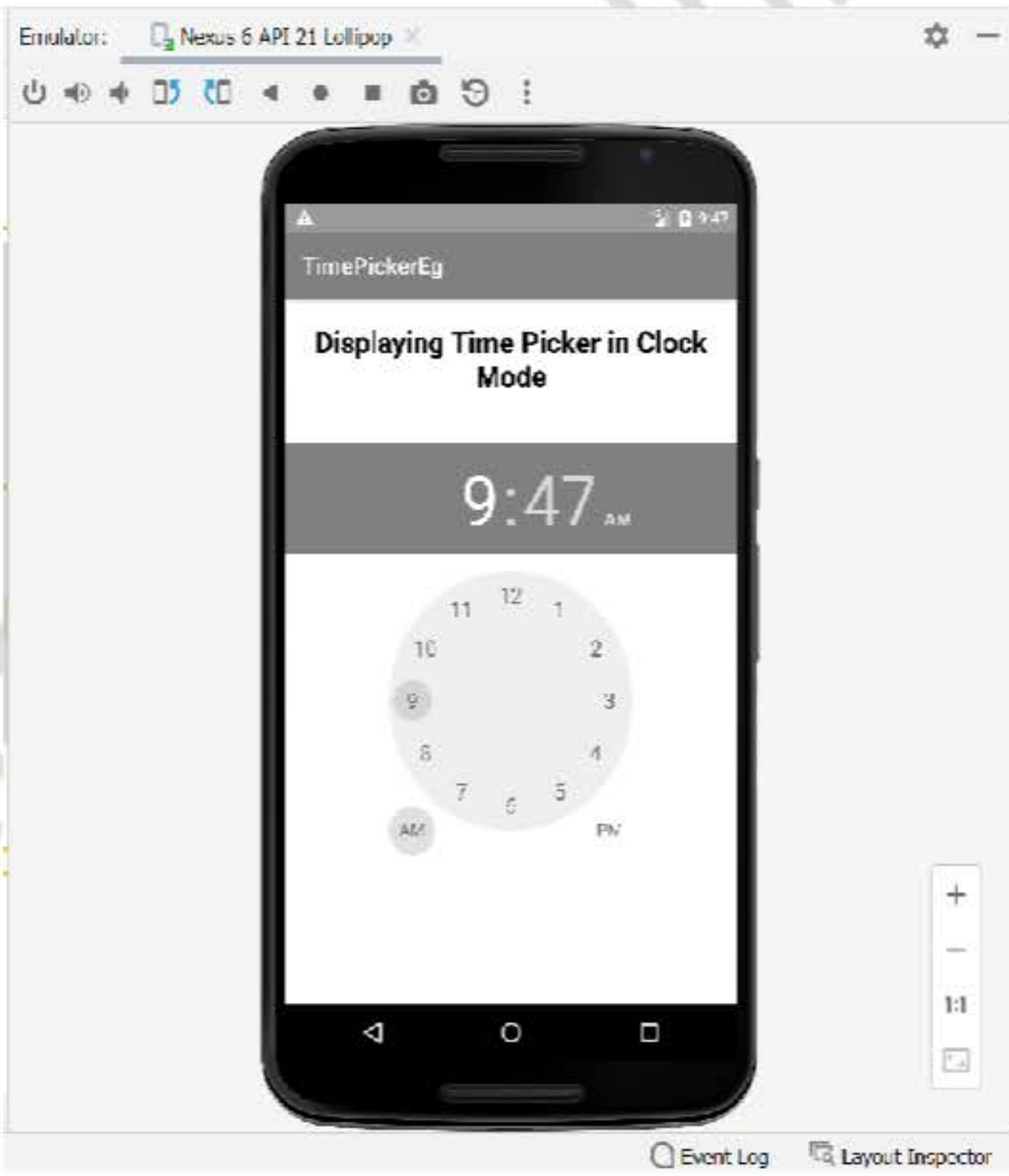


Figure 5.37: Displaying the Time selected from TimePicker in Clock Mode

The following is the XML layout showing the TimePicker to be displayed in "Spinner" mode.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout 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"
    android:orientation="vertical"
    tools:context=".MainActivity">
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Displaying Time Picker in Spinner Mode"
        android:layout_marginTop="20dp"
        android:gravity="center_horizontal"
        android:textSize="25sp"
        android:textColor="@color/black"
        android:textStyle="bold"/>
    <TimePicker
        android:id="@+id/Tmpicker3"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="40dp"
        android:timePickerMode="spinner"
        android:layout_centerHorizontal="true"
        />
    <TextView
       android:id="@+id/TV_time"
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:layout_marginTop="100dp"
       android:gravity="center_horizontal"
       android:textSize="25sp"
       android:textColor="@color/black"
       android:textStyle="bold"/>
</LinearLayout>
```

Figure 5.38: TimePicker with TimePickerMode set to “spinner” and essential attributes

The time selected by the user during runtime is retrieved by setting a setOnTimeChangedListener() event listener in the corresponding activity class.

The following code in the activity class is used for retrieving the selected time by the user and setting the text of TextView control which is used for displaying the selected time.

```
public class MainActivity extends AppCompatActivity {
    TimePicker timepicker;
    TextView textview;

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

        // textview used for displaying selected time
        textview = (TextView) findViewById(R.id.TV_time);
        timepicker = (TimePicker) findViewById(R.id.Tmpicker3);
        timepicker.setIs24HourView(true);

        timepicker.setOnTimeChangedListener(new
                    TimePicker.OnTimeChangedListener(){

        @Override
         public void onTimeChanged(TimePicker view, int hourOfDay,
                                                    int minute)
         {
                int hr = hourOfDay;
                int min = minute;
                // setting time to be displayed
          textview.setText("Selected time is " + hr + ":" + min);
         }

        });
    }
}
```

Figure 5.39: Setting selected time in TextView for display in Activity class

On running the application the clock is displayed on the screen in spinner mode as follows

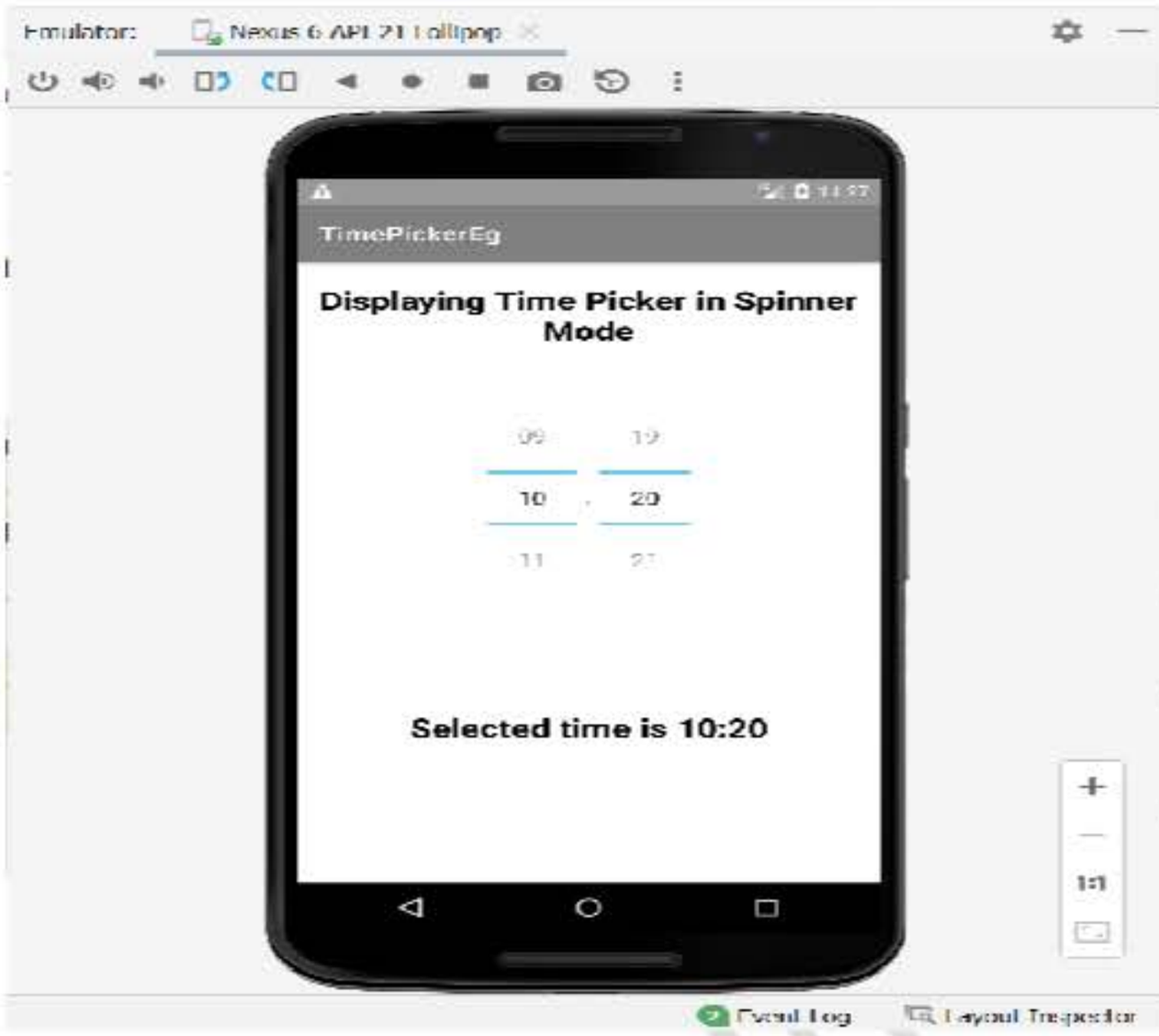


Figure 5.40: Displaying the selected time in TextView from TimePicker in Spinner Mode

5.4.2.2 DatePicker View

A DatePicker control is used in an android application so that the user selects valid data from the display. The user can select the required date, month, and year from the DatePicker control.

DatePicker requires API level 21 and above to be used in an android application.

DatePicker control has 2 modes of displaying - Calendar and Spinner.

The mode can be set by using the attribute "datePickerMode".

- android:datePickerMode = "calendar"
- android:datePickerMode = "spinner"

"calendar" is the default mode of DatePicker.

1. The following is the XML layout showing the DatePicker to be displayed in the "calendar" mode.

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

```
    android:orientation="vertical"
    tools:context=".MainActivity">
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Displaying Date Picker in Calendar Mode"
        android:layout_marginTop="20dp"
        android:gravity="center_horizontal"
        android:textSize="25sp"
        android:textColor="@color/black"
        android:textStyle="bold"/>
    <DatePicker
        android:id="@+id/DatePicker1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="25dp"
        android:datePickerMode="calendar"
        android:padding="20dp"
        android:background="@color/white" />
    <Button
        android:id="@+id/btn_date"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="GetDate"
        android:textStyle="bold"
        android:gravity="center_horizontal"
        android:textSize="20sp"
        android:textColor="@color/white" />
    <TextView
        android:id="@+id/TV_date"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="5dp"
        android:text="Selected date is "
        android:gravity="center_horizontal"
        android:textSize="25sp"
        android:textColor="@color/black"
        android:textStyle="bold"/>
</LinearLayout>
```

Figure 5.41: DatePicker with datePickerMode set to “calendar” and other essential attributes

In the activity class which launches the DatePicker control, a setOnClickListener() is set for the button so that when the button is clicked the selected date by the user is displayed in the textView.

```
public class MainActivity extends AppCompatActivity {

    DatePicker dtpicker;
    TextView textView;
    Button btndate;

    @RequiresApi(api = Build.VERSION_CODES.O)
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        dtpicker = (DatePicker) findViewById(R.id.DatePicker1);
        textView = (TextView) findViewById(R.id.TV_date);
        btndate = (Button) findViewById(R.id.btn_date);

        btndate.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

                int day = dtpicker.getDayOfMonth();
                int month = dtpicker.getMonth() + 1;
                int year = dtpicker.getYear();

textView.setText("Selected Date: " + day + "-" + month + "-" +
                                                        year );


            }
        });
    }
}
```

Figure 5.42: Setting selected date in TextView for display in Activity class

On running the application the selected date is displayed on the screen in calendar mode as follows

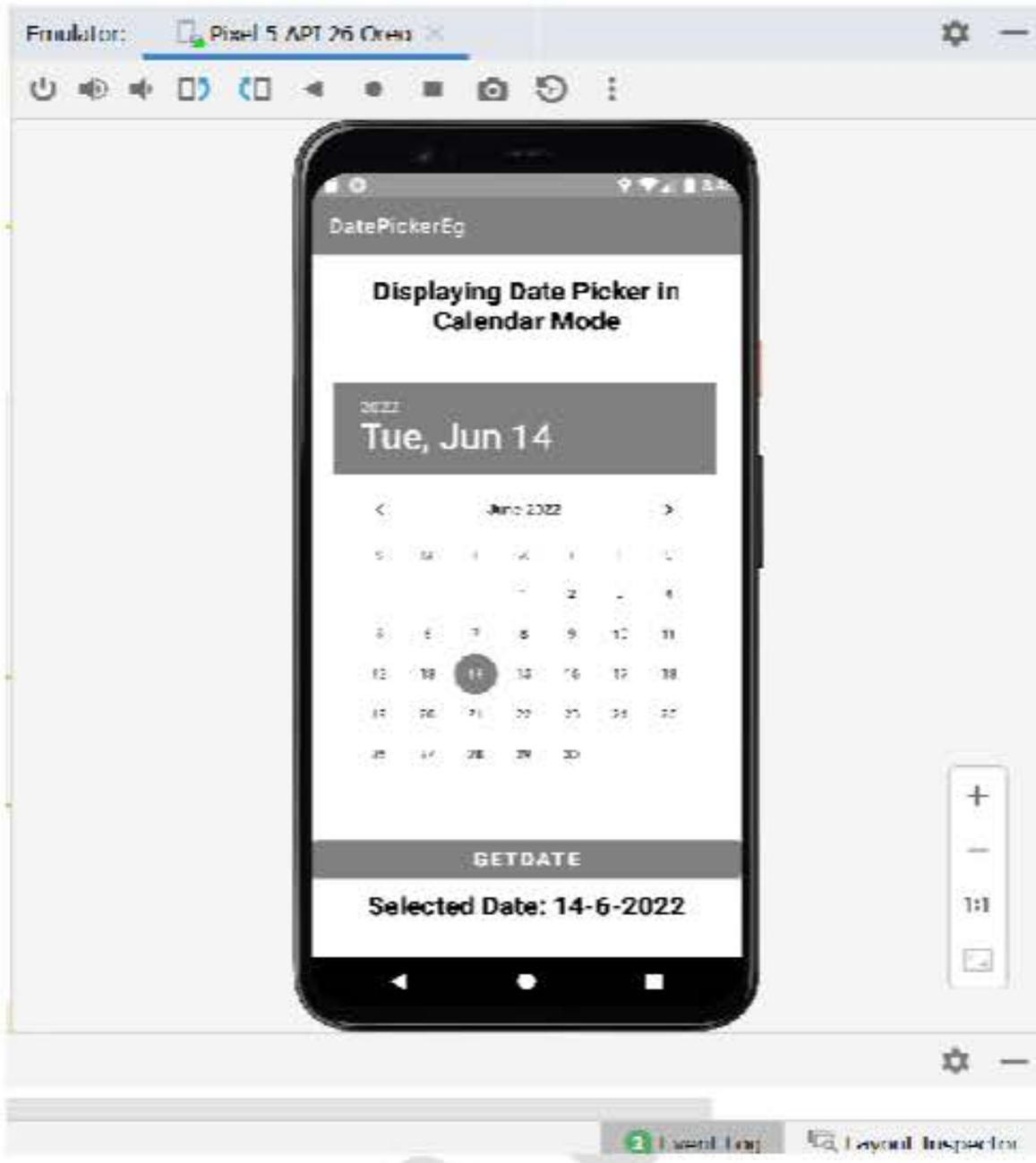


Figure 5.43: Displaying the selected date in TextView from DatePicker in calendar Mode

2. The following is the XML layout showing the DatePicker to be displayed in the "spinner" mode.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
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"
    android:orientation="vertical"
    tools:context=".MainActivity">
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Displaying Date Picker in Spinner Mode"
        android:layout_marginTop="20dp"
```

```
        android:gravity="center_horizontal"
        android:textSize="25sp"
        android:textColor="@color/black"
        android:textStyle="bold"/>

    <DatePicker
        android:id="@+id/DatePicker1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="25dp"
        android:datePickerMode="spinner"
        android:padding="20dp"
        android:background="@color/white" />

    <Button
        android:id="@+id/btn_date"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="GetDate"
        android:textStyle="bold"
        android:gravity="center_horizontal"
        android:textSize="20sp"
        android:textColor="@color/white" />

    <TextView
        android:id="@+id/TV_date"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="5dp"
        android:text="Selected date is "
        android:gravity="center_horizontal"
        android:textSize="25sp"
        android:textColor="@color/black"
        android:textStyle="bold"/>
</LinearLayout>
```

Figure 5.44: DatePicker with datePickerMode set to “spinner” and other essential attributes

On running the application the selected date is displayed on the screen in spinner mode as follows

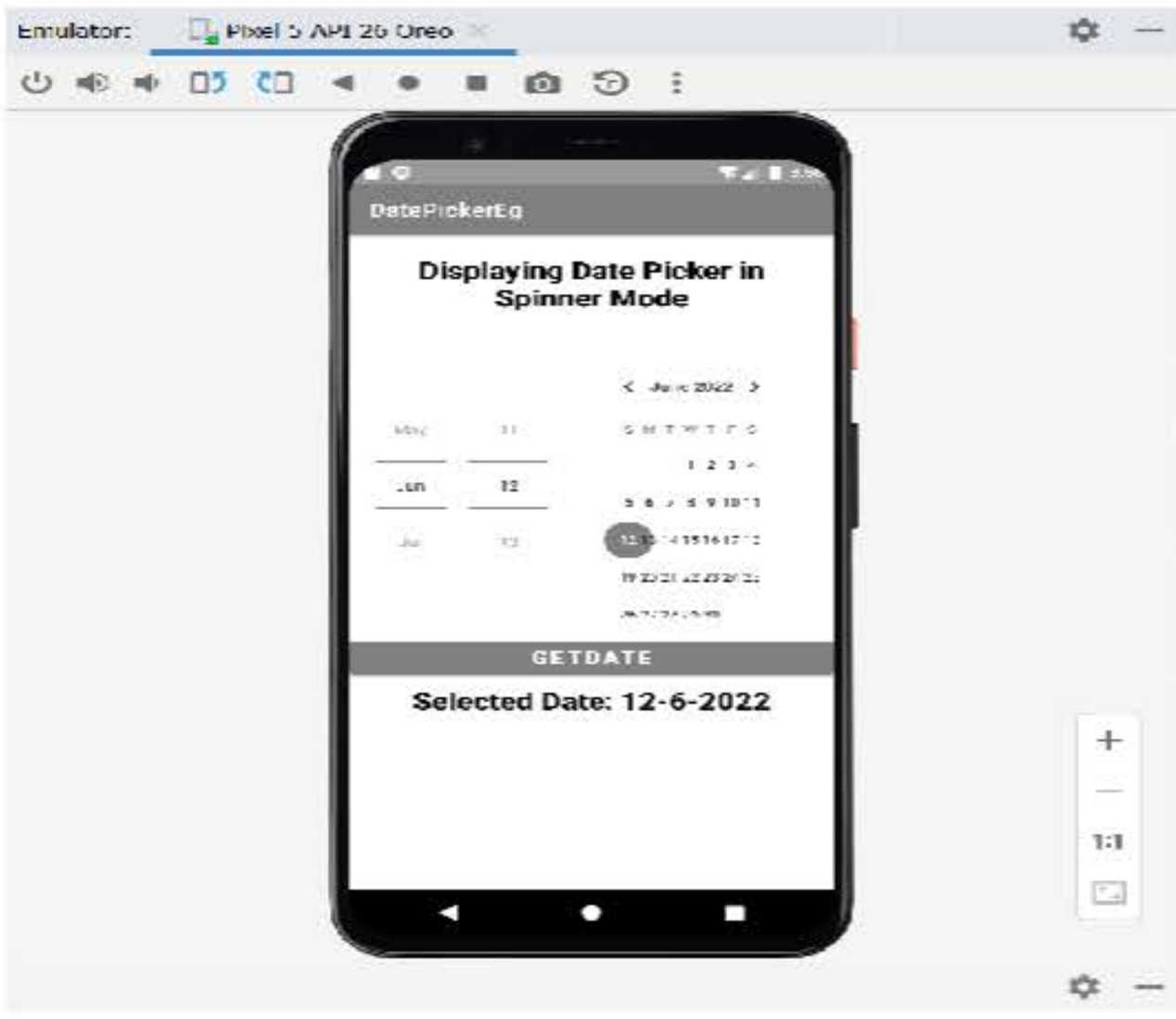


Figure 5.45: Displaying the selected date in TextView from DatePicker in Spinner Mode

5.5 Adapter and Adapter Views

An Adapter connects the data source and the adapter views acting as a bridge between them. Data can be from any source but should be in a structured form. For e.g. Array objects, List Objects, etc. It is the responsibility of an adapter, to fetch the data from the data source convert it into a view, and fill it into the adapter view to display it.

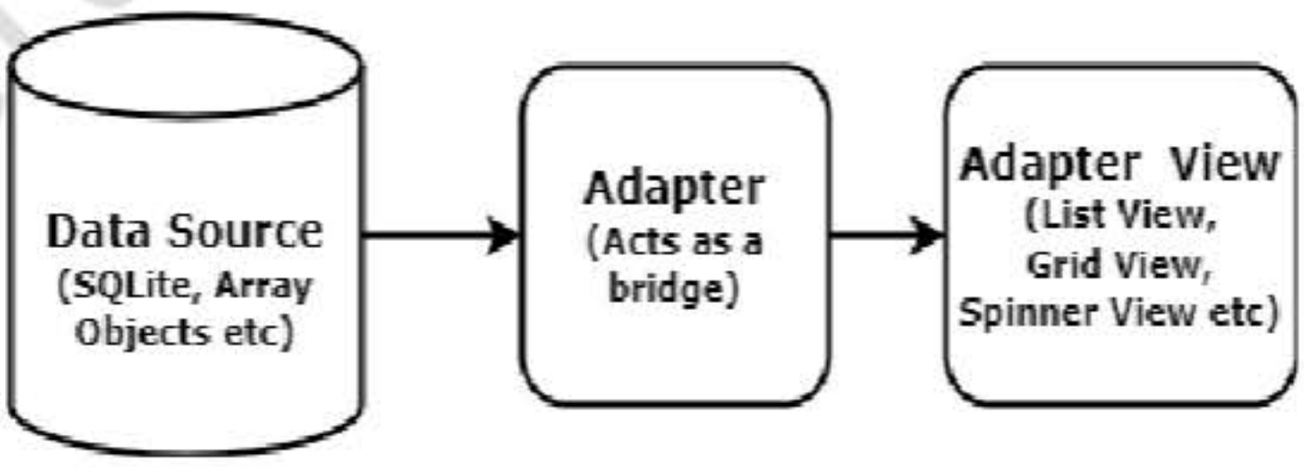


Figure 5.46: The Adapter acts as a bridge between Data-Source and Adapter View

An adapter view can efficiently display a large set of data in the form of ListView, SpinnerView, and GridView.

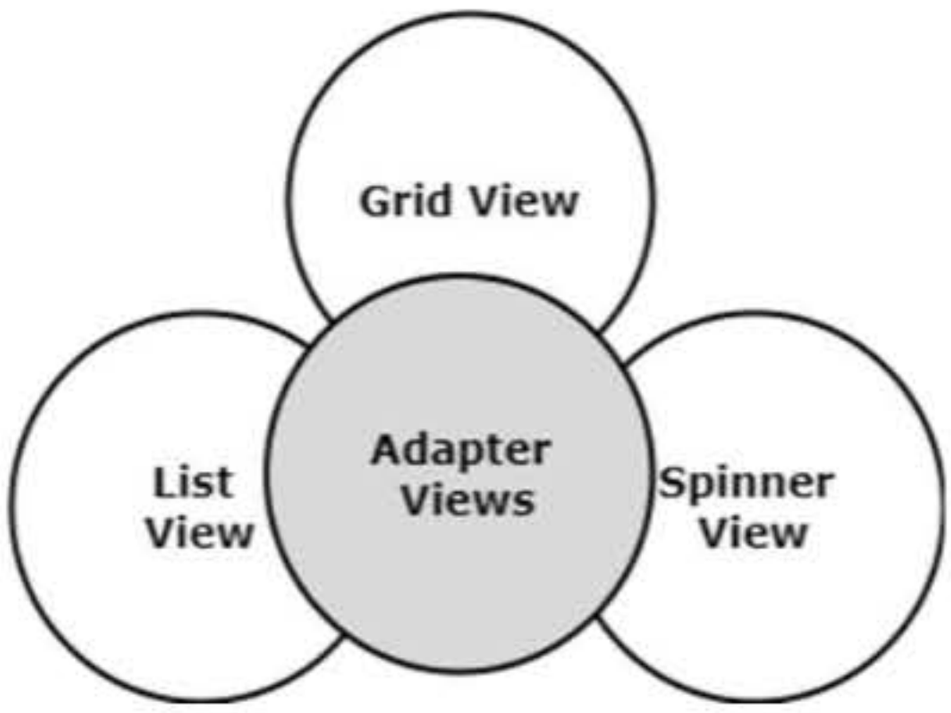


Figure 5.47: Adapter Views – List View, Grid View, and Spinner View

5.5.1 List view

It is a view that allows the creation of a list of items that are scrollable. In fact, the list view contains a large set of data items of which only a few are visible that fit on the screen, other items are visible on scrolling vertically up and down.

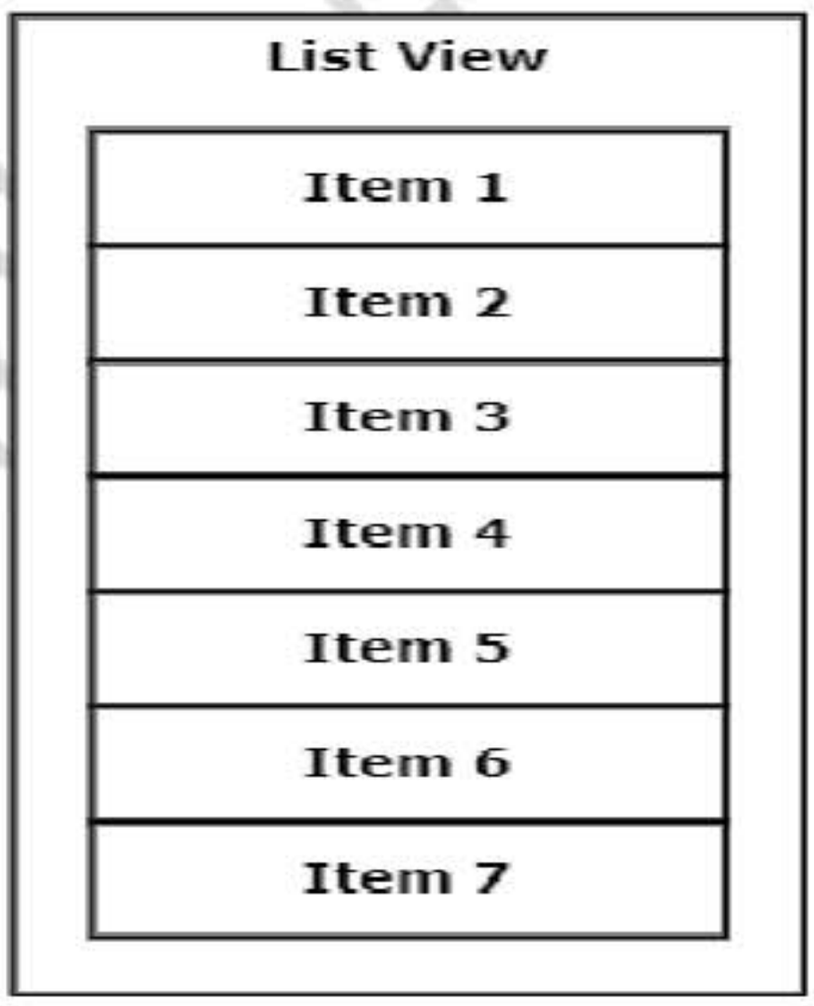


Figure 5.48: List View displaying items 1 to 7

Graphic Era University

5.5.2 Spinner View

It is similar to a dropdown list, from where we can select items.

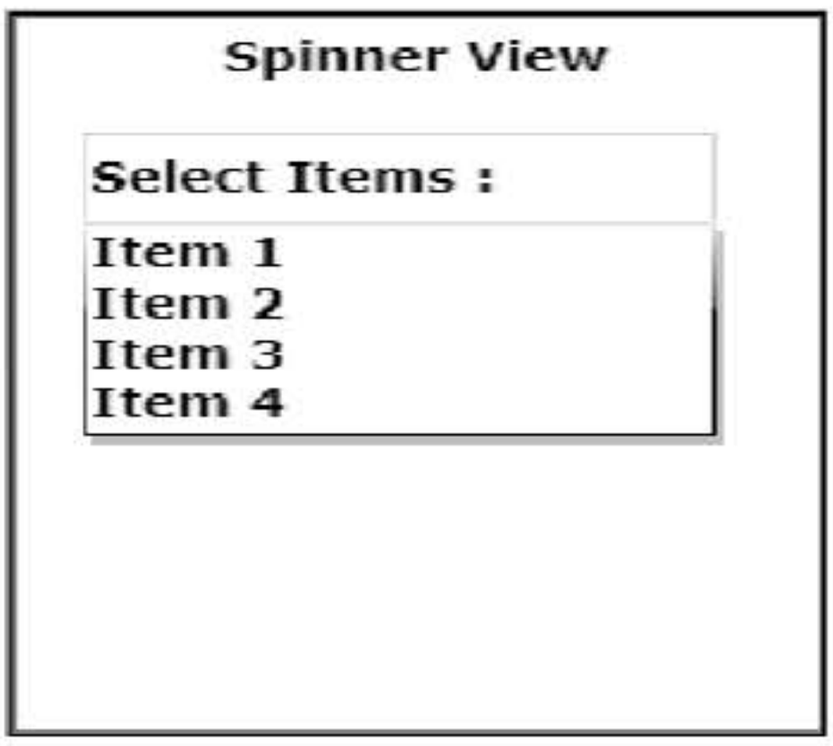


Figure 5.49: Spinner View displaying items 1 to 4 in a drop-down

5.5.3 Grid View

Items are displayed in a two-dimensional grid that is scrollable (in form of rows and columns). The number of columns can be two or more.

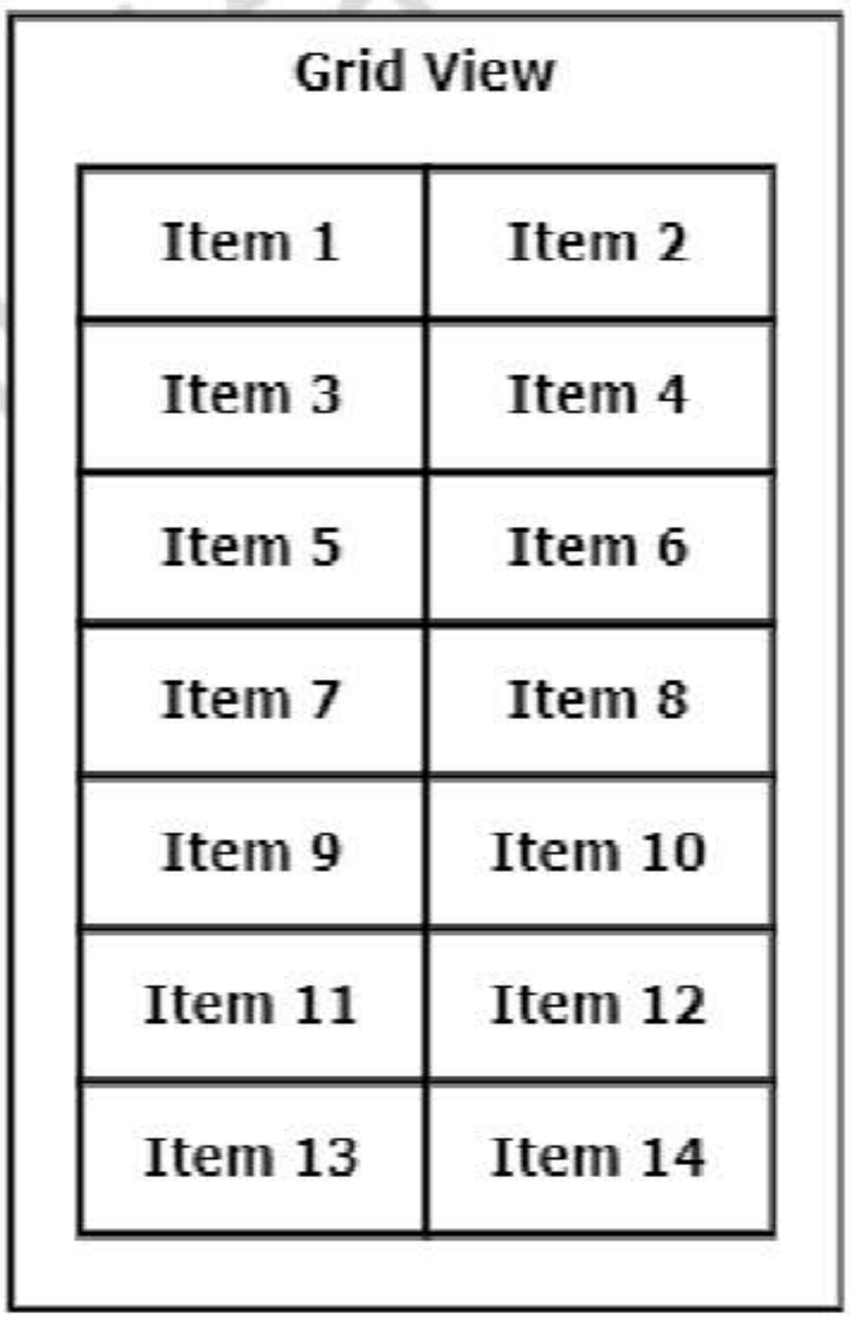


Figure 5.50: Grid View displaying items 1 to 14 in rows and columns

Graphic Era University

5.6 Self-Assessment Questions

Q1. Define View. Explain different types of views. [8 marks, L2]

Q2. What are views and viewgroups? [3 marks, L2]

Q3. List and discuss basic views in Android with suitable code segments. [8 marks, L2]

Q4. Explain the following views in brief [6 marks each, L2]

- Button and ImageButton
- RadioGroup and Radio
- CheckBox
- ImageView

Q5. What are the different types of Layouts in Android? Explain in brief. [8 marks, L2]

Q6. Describe the following layouts w.r.t Android with an example. [6 marks each, L2]

- Linear Layout
- Relative Layout
- Table Layout
- Frame Layout
- ScrollView

Q7. List the common attributes of Layouts/ViewGroups.[8 marks, L2]

Q8. Differentiate between RelativeLayout and FrameLayout. [4 marks, L4]

Q9. What are the benefits of using table layouts? [4 marks, L2]

Q10. Write notes on [6 marks each, L2]

- DatePicker View
- TimePicker View

Q11. Explain adapter and adapter views. [8 marks, L2]

Q12. Explain the different types of adapter views in brief. [6 marks, L2]

5.7 Self-Assessment Activities

Note: While developing an android application use an appropriate emulator for execution.

A1. Create an Android application and use a linear layout, add appropriate controls so that on the execution of the application, the output should be as shown in Figure 5.5.

A2. Create an Android application, and add appropriate controls so that on the execution of the application the output should be as shown in Figure 5.13.

A3. Create an Android application, and add appropriate controls so that on the execution of the application the output should be as shown in Figure 5.29.

A4. Create an Android application, and add appropriate controls so that on the execution of the application the output should be as shown in Figure 5.48.

Graphic Era University

5.8 Multiple-Choice Questions

1. ___layout lets child views specify their position relative to the parent view or each other.
   a) Linear Layout [1 mark, L1]
   b) Relative layout
   c) Table Layout
   d) None

2. __ layout is used to align all subviews within it either vertically or horizontally.[1 mark, L1]
   a) Linear Layout
   b) Relative layout
   c) Table Layout
   d) None

3. ____ attribute is used to set the text of the TextView. [1 mark, L1]
   a) android:lines
   b) android:text
   c) android:textAppearance
   d) android:textSize

4. ____ layout is used to align child views in rows and columns.[1 mark, L1]
   a) Linear Layout
   b) Relative layout
   c) Table Layout
   d) None

5. Frame Layout is designed to display _____ item(s) at a time. [1 mark, L1]
   a) Single
   b) Two
   c) Multiple
   d) None of the above

6. The value that the attribute datePickerMode can take is_____.[1 mark, L1]
   a) Quarterly
   b) monthly
   c) yearly

Graphic Era University

d) Spinner

7. _____ acts as a bridge between the data source and adapter view.[1 mark, L1]
   a) Layout
   b) Grid
   c) Adapter
   d) Array

8. The attribute that is used to specify the space around the view is_____. [1 mark, L1]
   a) layout_padding
   b) layout_width
   c) layout_height
   d) layout_margin

9. The attribute that is used to specify the space inside the view i.e. the space around the contents within the view is _____. [1 mark, L1]
   a) layout_ width
   b) layout_padding
   c) layout_height
   d) layout_margin

10. The view that groups several items and displays them in a vertical scrollable manner is _____. [1 mark, L1]
   a) Spinner View
   b) Grid View
   c) List View
   d) None of the above

5.9 Key Answers to multiple-choice questions

1. Relative Layout lets child views specify their position relative to the parent view or each other.[b]
2. Linear layout is used to align all subviews within it either vertically or horizontally. [a]
3. **`android:text`** attribute is used to set the text of the TextView.[b]
4. Table layout is used to align child views in rows and columns.[c]
5. Frame Layout is designed to display a single item(s) at a time.[a]

Graphic Era University

6. The value that the attribute datePickerMode can take is a spinner.[d]
7. Adapter acts as a bridge between the data source and the adapter view.[c]
8. The attribute that is used to specify the space around the view is layout_margin.[d]
9. The attribute that is used to specify the space inside the view i.e. the space around the contents within the view is layout_padding.[b]
10. The view that groups several items and displays them in a vertical scrollable manner is List View. [c]

5.10 Summary

Activity in an android application contains the user interface with which the users interact. In other words, an activity's user interface is a collection of **views** and **viewgroups.**

A **view** is a control/widget/element that is displayed on the screen. Buttons, textboxes, labels, etc. are examples of views. **android.view.View** is the base class for all views.

Viewgroup is a collection of one or more views. ViewGroup helps in specifying different types of layouts according to the arrangements and positions of the views/controls on the screen. Linear layout, relative layout, absolute layout, etc. are different types of layouts used by viewgroups. A ViewGroup is a subclass of View. It acts as an invisible container for other views and viewgroups. **android.view.ViewGroup** is the base class for all ViewGroups.

Layouts are used as containers to define the user interface i.e. the arrangement of elements/controls on the screen in an android application. There are different types of layouts. The most commonly used layouts are - Linear Layout, Relative Layout, Absolute Layout, Table Layout, Frame Layout, and Scroll view. Layouts are declared in 2 ways – Declaring views at design time and creating elements/views at runtime.

Basic Views are the general views/widgets/controls commonly used in any application.

For example- **TextView**, **EditView, Button, RadioButton**, etc.

Pickers Views are used in android for selecting date and time from the DatePicker and TimePicker controls respectively.

An Adapter connects the data source and the adapter views acting as a bridge between them. Data can be from any source but should be in a structured form. For e.g. Array objects, List Objects, etc. It is the responsibility of an adapter, to fetch the data from the data source convert it into a view and fill it into the adapter view to display it.

An adapter view can efficiently display a large set of data in the form of ListView, SpinnerView, and GridView.

ListView is a view that allows the creation of a list of items that are scrollable.
SpinnerView is similar to a dropdown list, from where we can select items.
GridView - Items are displayed in a two-dimensional grid that is scrollable.

5.11 Keywords

- View and Viewgroup
- Linear Layout
- Relative Layout
- Frame Layout
- Table Layout
- Absolute Layout
- Basic Views
- Picker Views
- Adapter
- Adapter View
- DatePicker
- TimePicker

5.12 Recommended Resources for Further Reading

Text Book(s)

1. Pradeep Kothari, "Android Application Development" Black-Book, Dreamtech Press,2015
2. Wei-Meng Lee, "Beginning Android Application Development", Wiley 2011.
3. Dawn Griffiths and David Griffiths, "Head First Android Development", O'Reilly, 2017.

References

- https://www.javatpoint.com/android-tutorial
- https://www.tutorialspoint.com/android/index.htm
- https://developer.android.com/guide/components/fundamentals

--*--

Graphic Era University

Contents

Topics	Page No.
Unit-2: Working with Android User Interface and handling UI Events	1
Chapter-6: Menus, Action Bar, and Event Handling	
6.0 Structure of Menus, Action Bar, and Event Handling.....	1
6.1 Learning Outcomes	1
6.2 Menus	1
6.2.1 Creating Menus	2
6.2.1.1 Creating Options Menu	2
6.2.1.2 Creating Popup Menu.....	8
6.2.1.3 Creating Context Menu.....	11
6.3 Action Bar	13
6.4 Event Handling: Events, Event Handlers, and Event Listener Registration.....	17
6.5 Alert Dialog	24
6.6 Self-Assessment Questions	28
6.7 Self-Assessment Activities	28
6.8 Multiple-Choice Questions	29
6.9 Key Answers to Multiple-Choice Questions	30
6.10 Summary	31
6.11 Keywords	32
6.12 Recommended Resources for Further Reading	32

Graphic Era University

UNIT 2 - Working with Android User Interface and handling UI Events

CHAPTER 6 – Menus, Action Bar, and Event Handling

Structure of Menus, Action Bar, and Event Handling

6.1 Learning Outcomes
6.2 Menus
6.3 Action Bars
6.4 Event Handling: Events, Event Handlers, and Event Listener Registration
6.5 Alert Dialog
6.6 Self-Assessment Questions
6.7 Self-Assessment Activities
6.8 Multiple-Choice Questions
6.9 Key answers to multiple-choice questions
6.10 Summary
6.11 Keywords
6.12 Recommended resources for further reading

6.1 Learning Outcomes

After the successful completion of this chapter, the student will be able to:

- Describe the types of menus in Android
- Demonstrate the creation of different types of menus in Android with suitable examples
- Explain events, event handlers, and event listeners in Android
- Demonstrate event handling mechanism in Android with suitable examples

---

6.2 Menus

Menus are used in the android application to provide a user-friendly interface. Users can select the required option from the multiple options provided in the menu and perform the appropriate tasks.

There are 3 types of Menus

- Option Menu
- Popup Menu
- Context Menu

Graphic Era University

Option Menu

An Options Menu is usually used to provide additional information about an application. For e.g. Help, About Us, Search, Settings, History, Bookmarks, Recent tabs, downloads, etc.

Popup Menu

A Popup Menu usually displays a vertical list with items in it. By default, it appears below the element and in case there is no space available then it appears above. Popups appear on the occurrence of an event like a button click, hovering over an element, on item touch, etc.

Context Menu

A Context Menu appears when the user right-clicks on the element or presses the element for a long duration usually 500 milliseconds. It is a floating menu.

For e.g. After right-clicking on a button that is displayed on a screen, a context menu gets displayed. The user can choose from a set of options displayed in the menu like sending SMS, making a call, sending email, etc.

6.2.1 Creating Menus

6.2.1.1 Creating Option menu in an android application

1. Create a new project, open android

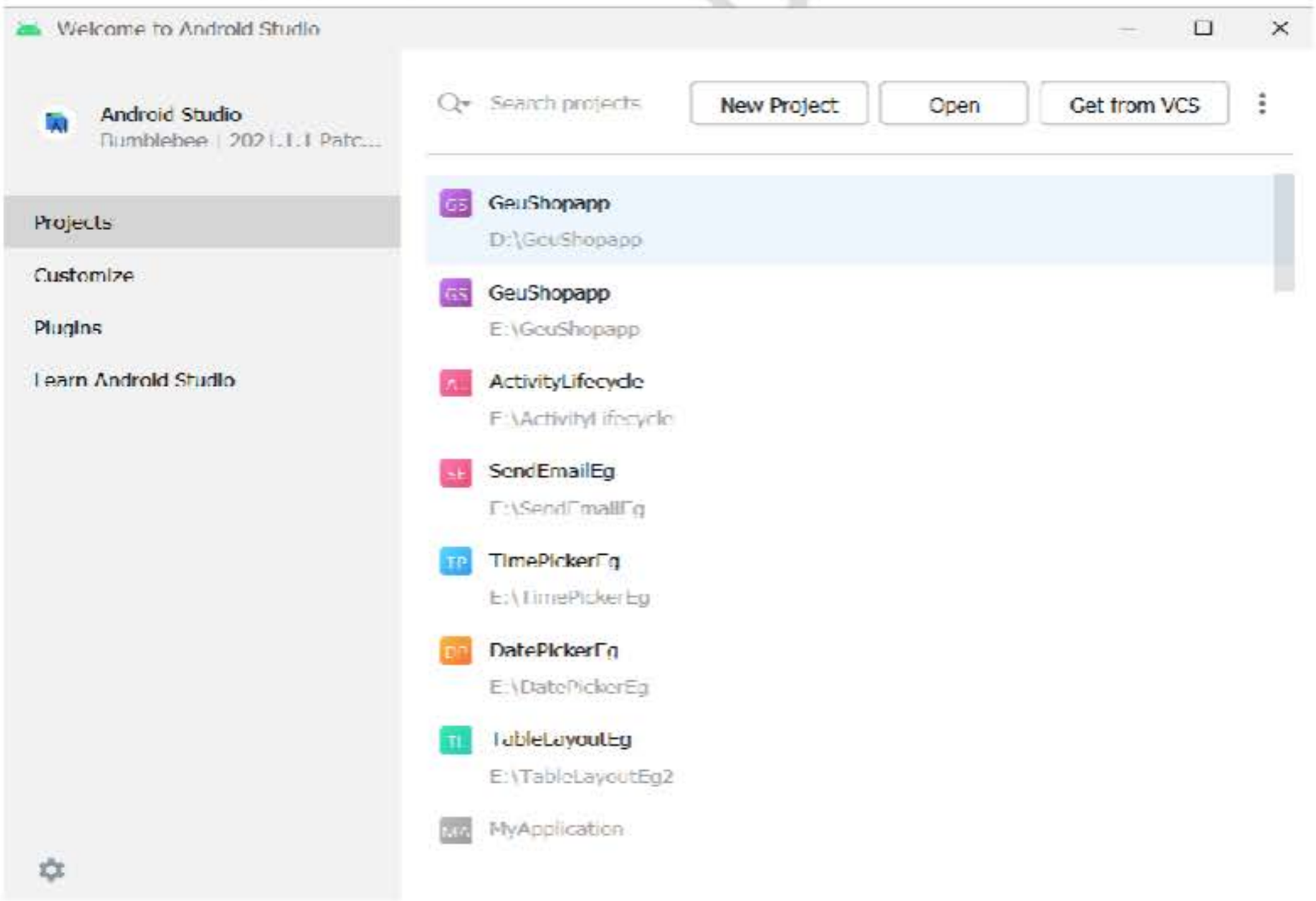


Figure 6.1: Creating a New Project in Android

2. Select an Empty Activity and click on Next

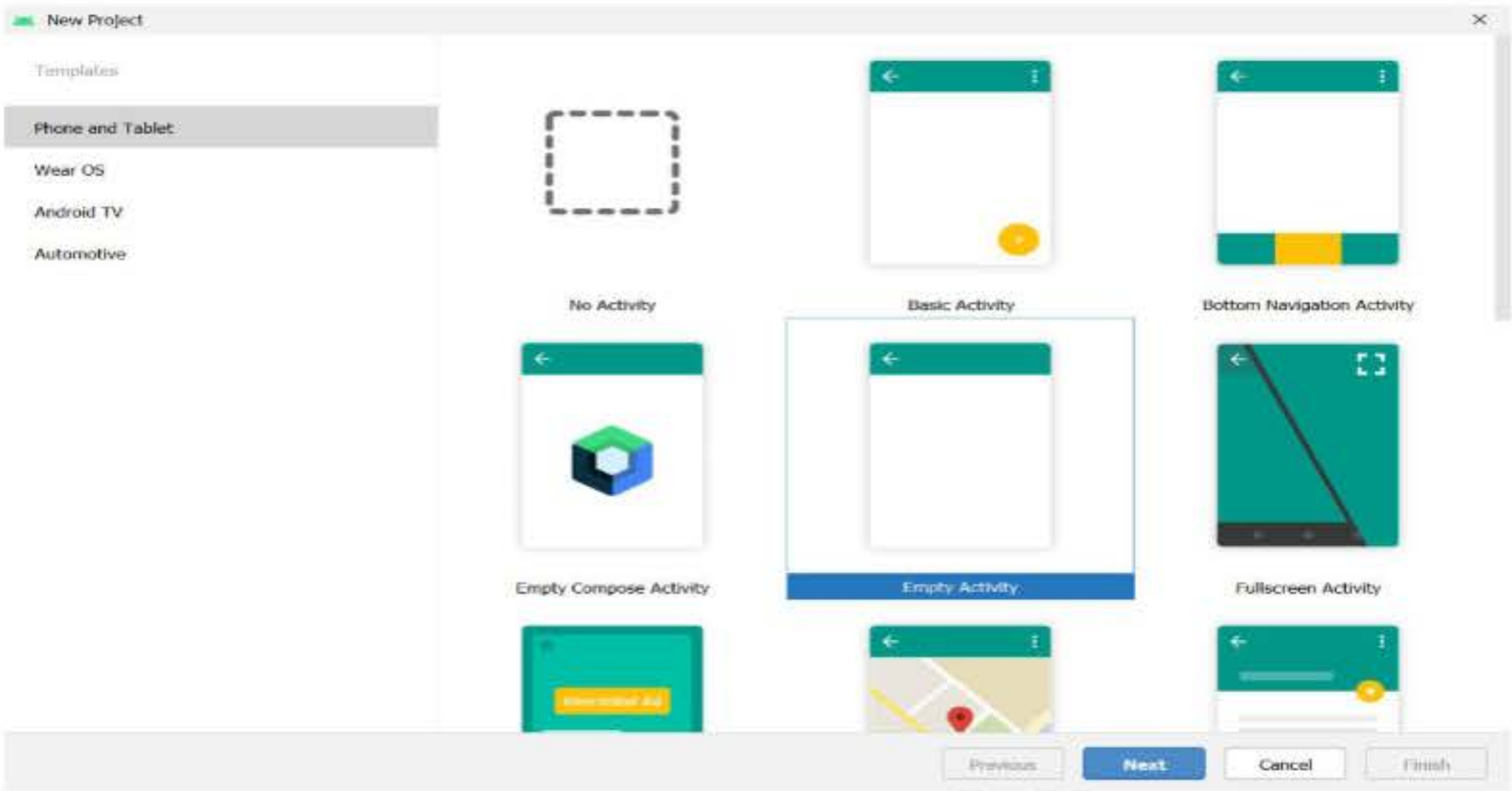


Figure 6.2: Select an Empty Activity in Android

3. Give the project name as OptionMenuEg and select Minimum SDK as API 19:Android 4.4 (KitKat) and click on Finish

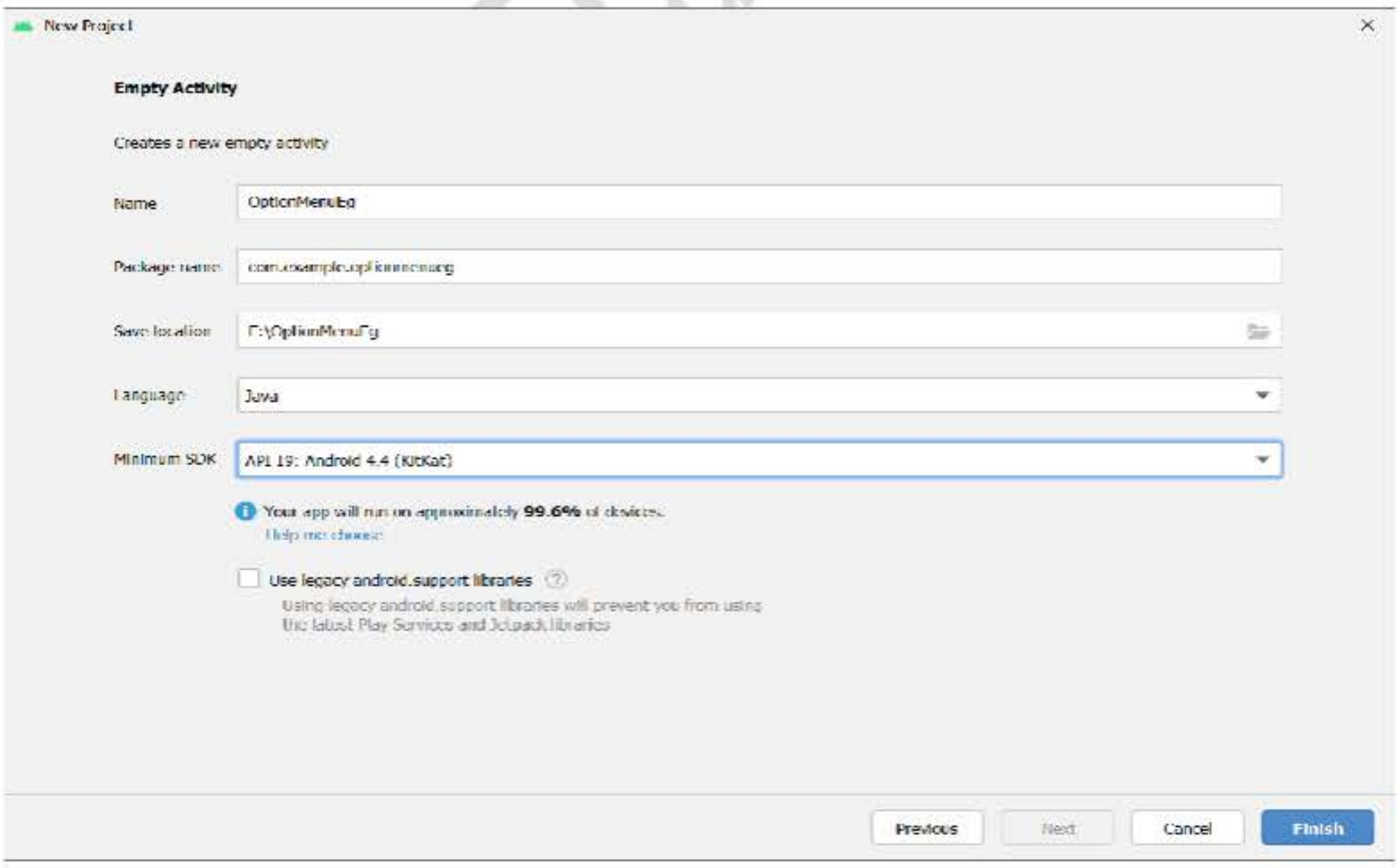


Figure 6.3: Empty Activity Window in Android

An Android application gets created with the MainActivity.java and activity_main.xml layout file with the default codes in it.

Change the text attribute of TextView to be displayed on the screen and the layout to LinearLayout with orientation set to "vertical" as given in figure 6.4 showing the code of the activity_main.xml file

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

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Option Menu Example"
        android:layout_marginTop="200dp"
        android:gravity="center"
        android:textSize="25sp"
        android:textColor="@color/black"
        android:textStyle="bold"/>
</LinearLayout>
```

Figure 6.4: Snippet of activity_main.xml file

Next, a separate xml file for the menu is to be created

4. For Creating an XML Menu file:
   - Create a menu directory as a subdirectory of the res folder.
   - Right-click on the res directory and navigate to New ->Android resource directory
   - Give the Directory menu as a menu and select resource type as a menu from the drop-down and click on OK

   Under the res folder, a sub-directory menu gets created

5. Next Create an XML resource file by right-clicking on the menu folder and navigating to New->menu resource file. Give the file name as menu_optioneg. A file menu_optioneg.xml gets created under the menu folder.

6. Adding Items to the Menu

   By default, items are not present in the menu resource file. Items are to be added to the menu resource file. Each item has a unique ID, title, and icon attribute.

   Here 3 items for the options menu with title attributes values as – "Help","About Us" and "Settings" are added as shown in figure 6.5.

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

<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/menu_help"
        android:title="Help">
    </item>
       <item
        android:id="@+id/menu_abtus"
        android:title="About Us" >
    </item>
         <item
        android:id="@+id/menu_set"
        android:title="Settings" >
    </item>
</menu>
```

Figure 6.5: Menu resource file with items added

7. Right-click on the MainActivity.java file inside the class after the onCreate() method and select Generate
8. Select Override Methods from the popup window
9. Next Select the methods to override
   - `onCreateOptionsMenu(Menu menu)` and
   - `onOptionsItemSelected(@NonNull MenuItem item)`

10. The overridden methods get generated in the MainActivty.xml file

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    return super.onCreateOptionsMenu(menu);
}

@Override
public boolean onOptionsItemSelected(@NonNull MenuItem item) {
    return super.onOptionsItemSelected(item);
}
```

Figure 6.6: Methods generated in MainActivity.java file

11. In the `onCreateOptionsMenu(Menu menu)` method of the MainActivity.xml file, inflate/fill the menu by making a call to `inflate()` method of MenuInflater class and override the `onCreateOptionsMenu(Menu menu)` to perform event handling on the items of the menu as shown in figure 6.7

```
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)
    {
      /*  Inflating - filling menu by calling inflate() method of
          MenuInflater class */

        MenuInflater inflaterobj = getMenuInflater();
        inflaterobj.inflate(R.menu.menu_optioneg, menu);
       // menu_optioneg.xml file is referenced while inflating
        return true;
    }
    @Override
    public boolean onOptionsItemSelected(@NonNull MenuItem item)
```

```
    {
        Toast.makeText(this, "You have selected : " + item.getTitle(),
                                Toast.LENGTH_LONG).show();
            switch (item.getItemId())
        {
            case R.id.menu_help:
                    return true;
            case R.id.menu_abtus:
                    return true;
            case R.id.menu_set:
                    return true;
            default:
                return super.onOptionsItemSelected(item);
        }
    }
}
```

Figure 6.7: Methods are overridden in MainActivity.java file

12. Execute the application to display the output

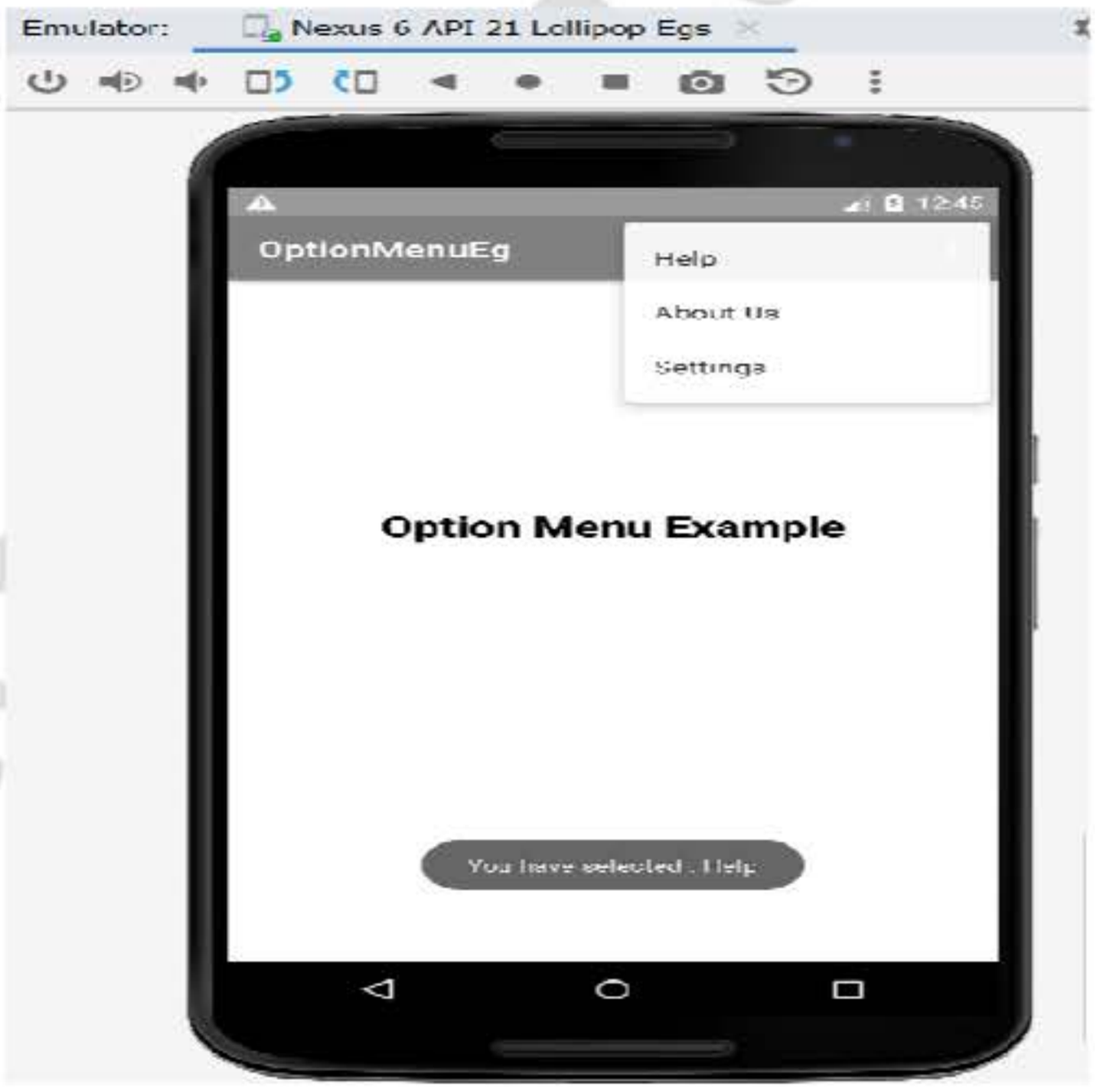


Figure 6.8: Displaying Option Menu and the selected item from the menu

6.2.1.2 Creating Popup menu in an android application

1. Create a new project named PopupMenuEg as given in the example of the option menu
2. Make the changes to the default activity_main.xml file as shown in figure 6.9
   Change the layout to LinearLayout and add a button to the user interface.
   Here on Click of a button, a popup gets displays

```
<Button
  android:id="@+id/btn_popup"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_margin="50dp"
  android:text="Click for Send Message"
  android:textSize="20sp"
  android:textColor="@color/white"
  android:textStyle="bold"/>
```

Figure 6.9: Snippet of activity_main.xml file user interface for Popup menu

3. Create a menu folder and menu resource file as given in the example of creating an options menu. Add items in the menu resource file as shown in figure 6.10
   The menu resource file name is given as: menu_popupeg.xml

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

<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/menu_sms"
        android:title="SMS">
    </item>
    <item
        android:id="@+id/menu_email"
        android:title="Email" >
    </item>
    <item
        android:id="@+id/menu_help"
        android:title="Help" >
    </item>
</menu>
```

Figure 6.10: New Resource file with items added

4. Make Changes to the MainActivity.java file as shown in figure 6.11

   Here the methods are not overridden as in the case of the Options Menu but event listeners are added

   - Add a listener to the button to be clicked to show the popup menu
   - Create an object of PopupMenu and add a menu to it
   - Add menu item click listener to the popup menu object and display the item selected
   - Show the popup menu

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ...
        ...
        ...
    btnpopup.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

    // create a PopupMenu object and add a menu to it
PopupMenu menuobj = new PopupMenu(MainActivity.this,v);
menuobj.getMenuInflater().inflate(R.menu.menu_popupeg,
                                                menuobj.getMenu());

   // set Menu item click listener to selected item
menuobj.setOnMenuItemClickListener(new
                            PopupMenu.OnMenuItemClickListener()
{
    @Override
    public boolean onMenuItemClick(MenuItem item) {

    // display the selected item from the popup menu
    Toast.makeText(MainActivity.this,item.getTitle(),
                                        Toast.LENGTH_LONG).show();
    return false;
```

```
}
});

  // show the popup menu
  menuobj.show();
    }
  });
 }
}
```

Figure 6.11: Snippet of MainActivity.java file after adding listeners

5. Execute the application to display the output

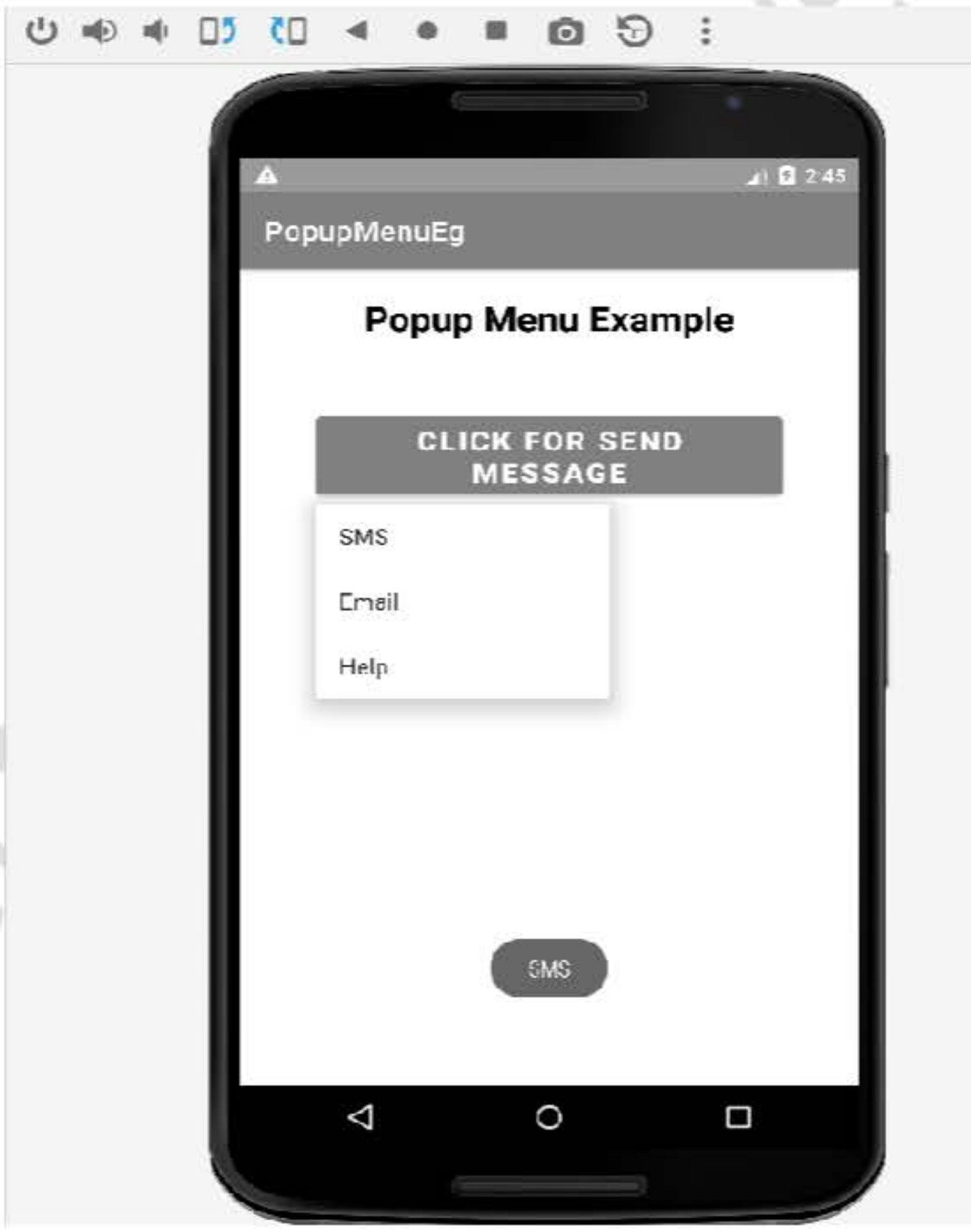


Figure 6.12: Displaying Popup Menu on click of the button

6.2.1.3 Creating Context menu in an android application

1. Create a new project named ContextMenuEg
2. Make the changes to the default activity_main.xml file as shown in figure 6.13
   Change the layout to LinearLayout and add a button to the user interface.
   Here on a Long press of the button, a Context Menu gets displays

```
<Button
    android:id="@+id/btn_context"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_margin="50dp"
    android:text="Long Press me"
    android:textSize="20sp"
    android:textColor="@color/white"
    android:textStyle="bold"/>
```

Figure 6.13: Snippet of activity_main.xml file user interface for Context menu

3. Create a menu folder and menu resource file as given in the example of creating an options menu. Add items in the menu resource file as shown in figure 6.14
   The menu resource file name is given as: menu_contexteg.xml

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

<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/menu_sms"
        android:title="SMS">
    </item>
    <item
        android:id="@+id/menu_email"
        android:title="Email" >
    </item>
    <item
        android:id="@+id/menu_help"
        android:title="Help" >
    </item>
</menu>
```

Figure 6.14: Menu Resource file with items added for Context Menu

4. Make Changes to the MainActivity.java file as shown in figure 6.15
   - Register the button to be "pressed long" to show the Context menu
   - Generate the methods to be overridden.
   - In the `onCreateContextMenu()` method of the MainActivity.xml file, inflate/fill the menu by making a call to inflate() method of MenuInflater class and override the `onContextItemSelected()` to display the selected item by using Toast as shown in figure 6.15.

```
public class MainActivity extends AppCompatActivity {
    ..
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        btncontext = (Button) findViewById(R.id.btn_context);
        // register the button for the context menu
        registerForContextMenu(btncontext);
    }
    @Override
    public void onCreateContextMenu(ContextMenu menu, View v,
                        ContextMenu.ContextMenuInfo menuInfo)
   {
        super.onCreateContextMenu(menu, v, menuInfo);

        // set the header for the Context Menu
        menu.setHeaderTitle("Select the Action");
        MenuInflater menuInflaterobj = getMenuInflater();
        menuInflaterobj.inflate(R.menu.menu_contexteg, menu);
    }
    @Override
    public boolean onContextItemSelected(@NonNull MenuItem item) {

       // display the selected Context Menu item using Toast
        Toast.makeText(this, "Selected Item is : " + item.getTitle(),
                                    Toast.LENGTH_LONG).show();
        return super.onContextItemSelected(item);
    }
}
```

Figure 6.15: Code in MainActivity.java file showing registration of button for Context Menu and listeners on item selected

5. Execute the application to display the output

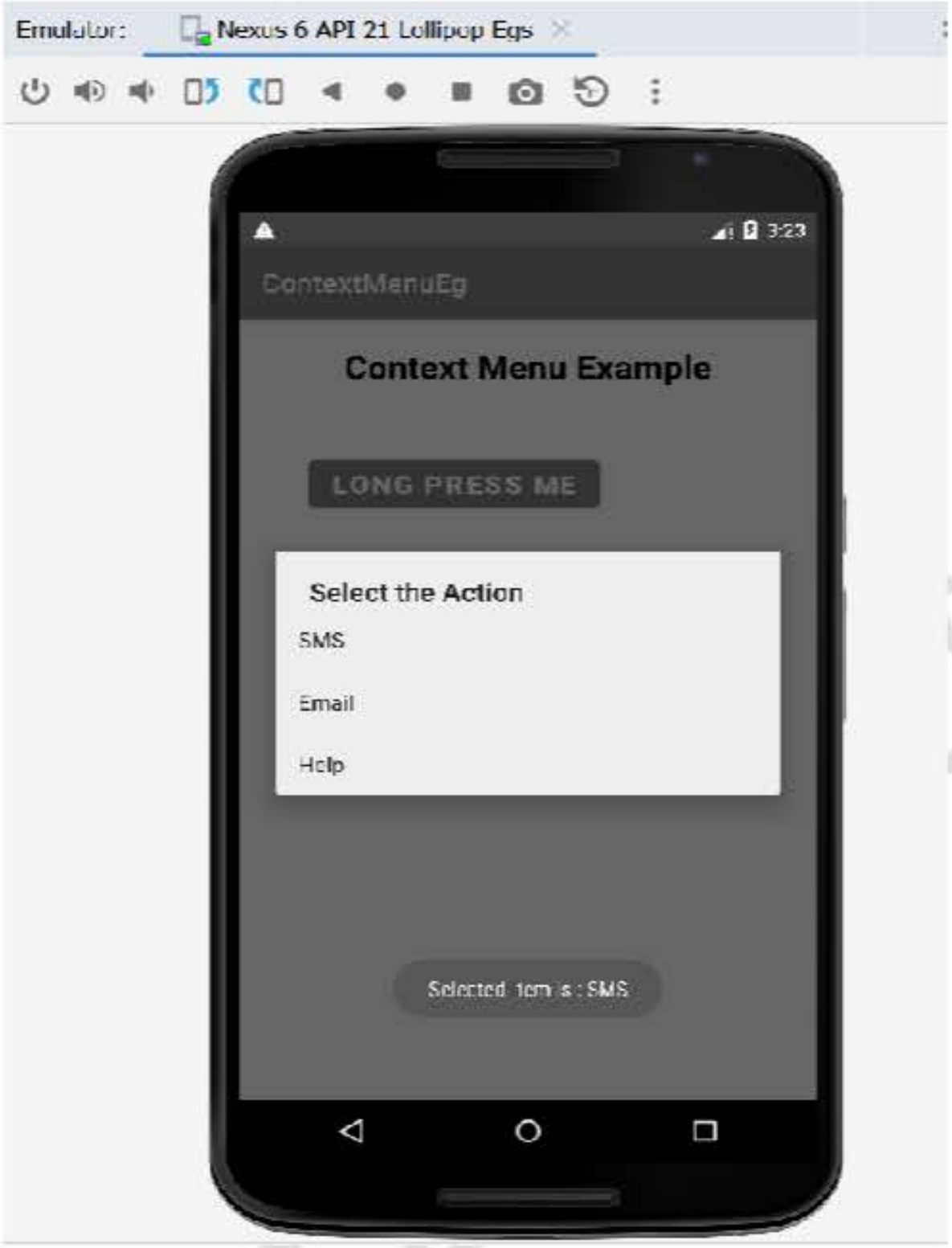


Figure 6.16: Displaying Context Menu on "Long Press" of the button and the item selected from the Context Menu

6.3 Action Bar

An Action Bar element in an android application is usually present/displayed at the top of the screen/activity. The action bar is also known as App Bar.

An Action Bar consists of the following components

- Application logo – icon
- Application or Activity title
- Application or Activity Sub-title
- Primary Action Buttons/Icons for an Activity
- Action Overflow – where less used actions are placed

Example: Creating an Action Bar in Android Application

1. Create a new project name ActionBarEg
2. Make the changes to the default activity_main.xml file as shown in figure 6.17 Change the layout to LinearLayout.

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

<LinearLayout 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"
    android:orientation="vertical"
    tools:context=".MainActivity">
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Action Bar Example"
        android:layout_marginTop="200dp"
        android:gravity="center"
        android:textSize="25sp"
        android:textColor="@color/black"
        android:textStyle="bold"/>

</LinearLayout>
```

Figure 6.17: Code of activity_main.xml file user interface for Action Bar

3. Create a menu directory under the res directory as explained earlier in Creating Option Menu
4. For adding icons to the action bar create vector images.
5. Next Create an XML resource file by right-clicking on the menu folder and navigating to New->menu resource file. Give the file name as- menu_actionbareg. A file menu_actionbareg.xml gets created under the menu folder.

Add items as follows as shown in figure 6.18

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

```
    <item
        android:id="@+id/menu_signin"
        android:icon="@drawable/ic_baseline_person_24"
        android:title="SignIn"
        app:showAsAction="ifRoom"
        android:titleCondensed="Sign-In">
    </item>

    <item
        android:id="@+id/menu_cartitem"
        android:icon="@drawable/ic_baseline_add_shopping_cart_24"
        android:title="Get Cart Info"
        app:showAsAction="ifRoom"
        android:titleCondensed="Cart-Info">
    </item>
</menu>
```

Figure 6.18: Menu Resource file with items (icons) added for Action Bar

6. Change the default app name string value in the string.xml file as shown in figure 6.19

```
<resources>
    <string name="app_name">GEU-Shopping</string>
</resources>
```

Figure 6.19: Strings.xml file

7. Make changes in the MainActivity.java file and add the code as shown in figure 6.20

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

       ActionBar actionBarobj = getSupportActionBar();

       // subtitle for the ActionBar
       actionBarobj.setSubtitle(" Designing a custom Action Bar");
```

```
    // adding an icon and displaying it in the ActionBar
     actionBarobj.setIcon(R.drawable.geulogo);
     actionBarobj.setDisplayUseLogoEnabled(true);
     actionBarobj.setDisplayShowHomeEnabled(true);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu)
    {

 //Inflating - filling menu by overriding inflate() method of MenuInflater
// class

        MenuInflater inflaterobj = getMenuInflater();
        inflaterobj.inflate(R.menu.menu_actionbareg, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(@NonNull MenuItem item){

      Toast.makeText(this, "You have selected : " + item.getTitle(),
                                        Toast.LENGTH_LONG).show();

       switch (item.getItemId()) {
            case R.id.menu_signin:
                return true;
            case R.id.menu_cartitem:
                return true;
            default:
                return super.onOptionsItemSelected(item);
        }
    }
}
```

Figure 6.20: Snippet of MainActivity.java file for Action Bar

8. Execute the application, action bar with the icons gets displayed

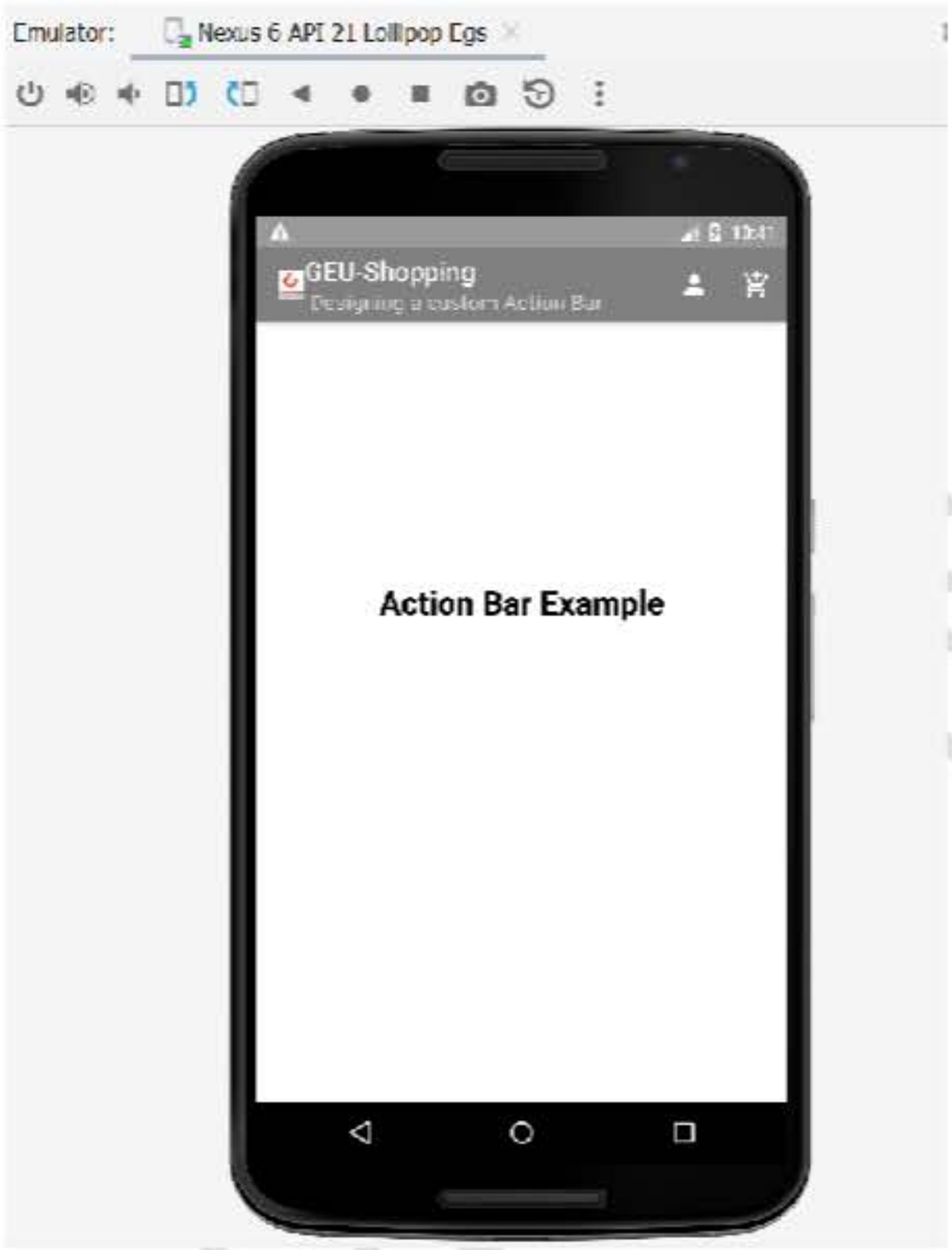


Figure 6.21: Displaying Action Bar with logo, title, subtitle and icons

6.4 Android Event Handling – Events, Event Handlers, and Event Listener Registration

Users interact with the android application user interface elements like buttons, radio buttons, checkboxes, text boxes, images, etc. by either using a mouse, keyboard, or touch screen. The action that takes place due to the user's interaction is called an event.
For e.g. click, long press, touch, keypress, etc are all events.

Managing these events is called Event Handling. When an event occurs a specific task is performed in response to an event, this is called event handling. All possible events that can occur have to be handled by an application. It should not happen that a user clicks a button and there is no response.

For handling events, the android framework maintains an event queue where the events are stored as they occur and are handled on a First In – First Out basis.

For managing event handling in android, we need to understand the following 3 concepts

- Event Handlers
- Event Listeners
- Event Listener Registration

Event Handlers

The method which is invoked/called to perform a specific task on the occurrence of an event is called an event handler. These event handlers have to be registered with the event listeners, only then they can be invoked/called.

For e.g. onClick(), onLongClick(), onKey() are event handlers

Event Listeners

An Event Listener is an interface within the View Class. It contains a call-back method.

For e.g. OnClickListener(), OnLongClickListener(), OnFocusChangeListener() are event listeners

Event Listener Registration

The process of registering an event handler with an event listener is called as event listener registration. The event handler is invoked when the event listener fires an event.

Table 6.1: Commonly used event handlers and event listeners in Android

Event Handler	Event Listeners	Description
onClick()	**OnClickListener()**	This is called when the user **clicks** on a **Button, ImageView,** etc
onLongClick()	**OnLongClickListener()**	This is called when the user **clicks** on a Button for **more than a second**
onKey()	**OnKeyListener()**	This is called when the user has **focused** on an **item** and **presses** a key.
onTouch()	**OnTouchListener()**	This is called when the user **touches** the **element** on the screen
onCreateContextMenu()	**onCreateContextMenuListener()**	This is called when the **context menu** is being built

Event Listener Registration in Android is done in 3 ways

- By specifying the event handler directly in the layout file i.e. activity_main.xml file
- By using an Activity class that implements the Listener interface
- By using an Anonymous Class

Example: An Android application with 3 buttons to demonstrate the 3 different ways of event registration with event handlers.

- The First button is registered with the event handler directly in xml file using the attribute
- The Second button is registered using an inner class that implements OnClickListener
- Third Button is registered using an Anonymous Class

Creating an android application to demonstrate 3 ways of event listener registration

1. Create an android application
2. Make changes in activity_main.xml file as shown in figure 6.22

   XML Activity file showing the code of User interface with 3 Buttons

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout 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"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Event Listener Registration Example"
        android:layout_marginTop="30dp"
        android:gravity="center"
        android:textSize="25sp"
        android:textColor="@color/black"
        android:textStyle="bold"/>
```

```
<Button
    android:id="@+id/btn_1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="20dp"
    android:layout_marginStart="20dp"
    android:text="Using xml"
    android:textSize="20sp"
    android:textColor="@color/white"
    android:textStyle="bold"
    android:onClick="displaymsg"/>

<Button
    android:id="@+id/btn_2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="20dp"
    android:layout_marginStart="20dp"
    android:text="Using Inner Class"
    android:textSize="20sp"
    android:textColor="@color/white"
    android:textStyle="bold"/>

<Button
    android:id="@+id/btn_3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="20dp"
    android:layout_marginStart="20dp"
    android:text="Using Anonymous Class"
    android:textSize="20sp"
    android:textColor="@color/white"
    android:textStyle="bold"/>

</LinearLayout>
```

Figure 6.22: Code of activity_main.xml file user interface with 3 Buttons to show different ways of event listener registration

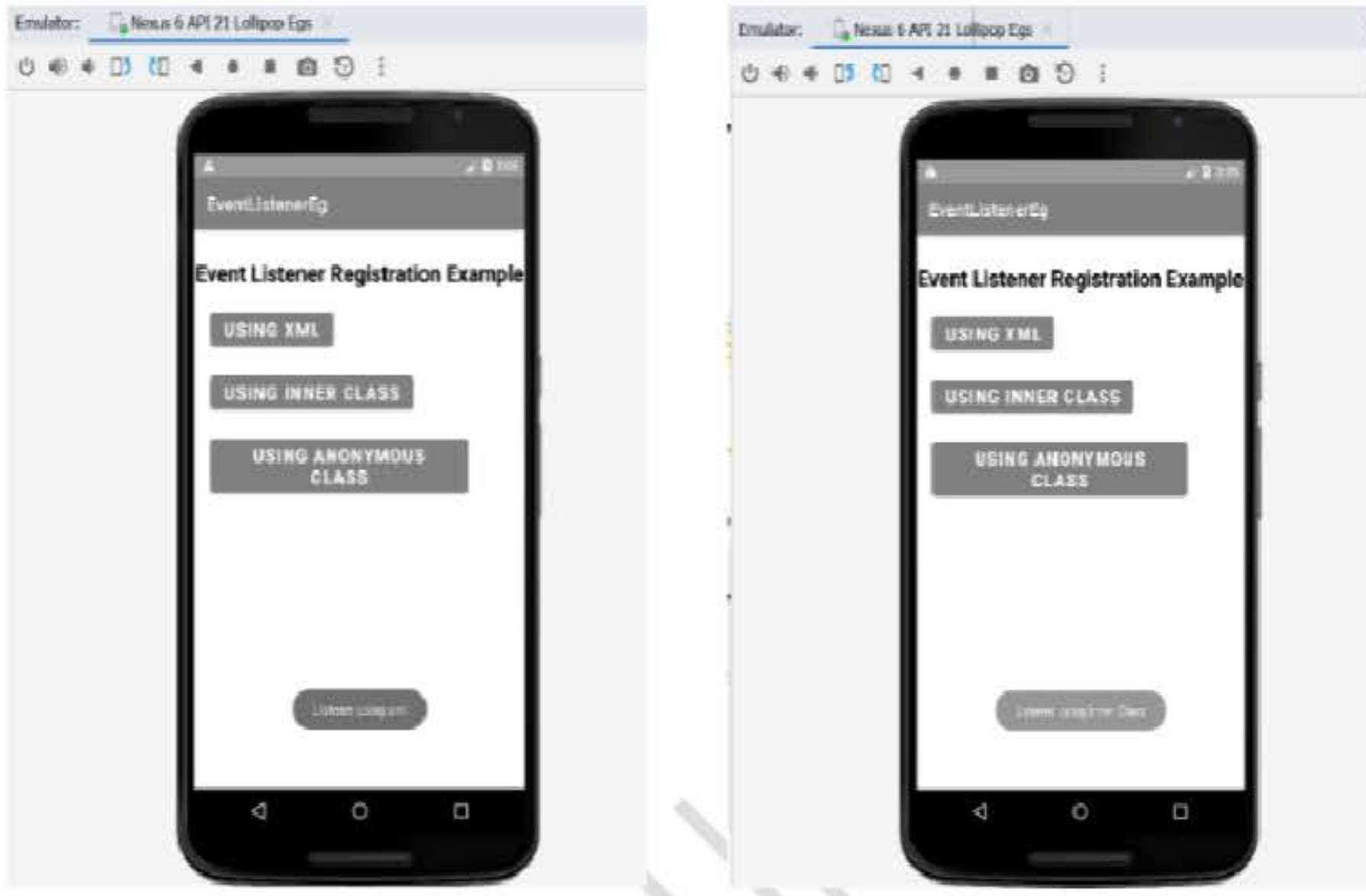
3. Make Changes in the MainActivity.java file as shown in figure 6.23
   MainActivity.java file showing 3 different ways of event listener registration

```
package com.example.eventlistenereg;

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

public class MainActivity extends AppCompatActivity {

    Button btn1;
    Button btn2;
    Button btn3;

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

    // reference the buttons in activity_main.xml file

    btn1 = (Button) findViewById(R.id.btn_1);  // used for xml

    btn2 = (Button) findViewById(R.id.btn_2);  // used for inner class

    btn3 = (Button) findViewById(R.id.btn_3);  // used for anonymous class

    /*  2nd Button event listener registration
        Event handler Registration using inner class */

        InnerClass inobj = new InnerClass();
        btn2.setOnClickListener(inobj);
```

```
/* 3rd Button event listener registration
   Event handler Registration using Anonymous class */
   btn3.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View v) {
    Toast.makeText(MainActivity.this,
     "Listener using Anonymous Class",Toast.LENGTH_LONG).show();
    }
  });
}

// Inner class definition used for 2nd Button
class InnerClass implements View.OnClickListener
{
    @Override
    public void onClick(View v) {

     Toast.makeText(MainActivity.this,
     "Listener using Inner Class",Toast.LENGTH_LONG).show();

    }
}

/* Event handler Registration done in xml activity_main.xml file using
  attribute onClick  android:onClick="displaymsg".
  1st Button calls this method on Click  */

 public void displaymsg(View view)
 {
    {
      Toast.makeText(this,"Listener using xml",Toast.LENGTH_LONG).show();
    }
}
```

Figure 6.23: Code in MainActivity.java file showing event listener registration of button using 3 different ways

4. Execute the application and display the output



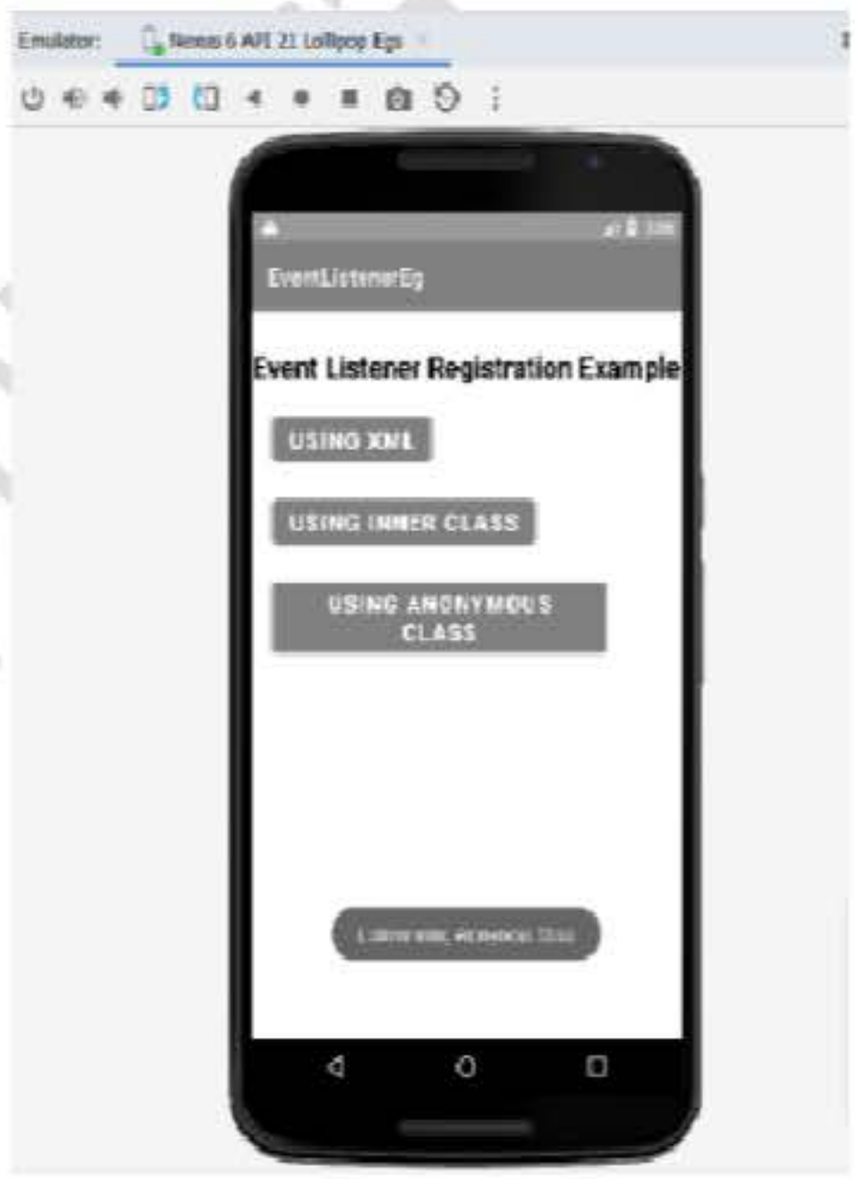


Figure 6.24: Displaying Outputs on Click of 3 buttons

6.5 Alert Dialog

An Alert Dialog is a small floating window that is displayed with an alert message without moving to the next activity and allows the user to take decision and interact. The user may have to give an answer either by selecting the "OK/Cancel" , "Yes/No" buttons and depending on the response the application may continue or discontinue.

Alert Dialog in android consists of 3 regions/areas –

- Title
- Message [content]
- Action buttons

There are some methods which are commonly used with alert dialog to set the required values.

- setTitle() - this method is used to set the title for the alert dialog box
- setMessage() - this method is used to set the message to be displayed in the alert dialog box
- setIcon() - this method is used to set icon before the title to be displayed on alert dialog box
- setCancelable() - this method is used to set a boolean value true/false. If a value false is set then the alert dialog box is cancellable
- setPositiveButton() - this method adds a positive button in the alert dialog box which is used by the user, when the user wants something to happen
- setNegativeButton() - this method adds a negative button in the alert dialog box which is used by the user, when the user does not want something to happen

To create an alert dialog first we need to create an object of AlertDialog.Builder Class.

```
AlertDialog.Builder adbuilderobj = new AlertDialog.Builder(this);
```

Then use the methods with the object as follows

```
abuilderobj.settitle()
abuilderobj.setMessage()
abuilderobj.setPositiveButton()
abuilderobj.setNegativeButton()
```

Example: To create an alert dialog in an android application

1. Create an android project, name it as AlertdialogEg
2. Make changes in the activity_main.xml file as shown in figure 6.25

   XML Activity file showing the code of the User interface with a button

```
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Alert Dialog Example"
    android:layout_marginTop="20dp"
    android:gravity="center"
    android:textSize="25sp"
    android:textColor="@color/black"
    android:textStyle="bold"/>


<Button
    android:id="@+id/btn_context"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_margin="50dp"
    android:text="Click for Alert Dialog Box"
    android:textSize="20sp"
    android:textColor="@color/white"
    android:textStyle="bold"
    android:onClick="displayalert"/>
```

Figure 6.25: Snippet of activity_main.xml file user interface with a Button

3. Make Changes in the MainActivity.java file as shown in figure 6.26

   MainActivity.java file showing the code to create an alert dialog box and display

```
public class MainActivity extends AppCompatActivity {


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

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

    public void displayalert(View view) {

        /* Create the object of AlertDialog Builder class */
        AlertDialog.Builder abuilderobj = new
                            AlertDialog.Builder(MainActivity.this);


        /* Set Alert dialog box Title */
        abuilderobj.setTitle("Alert Dialog Box Example!");


        /* Set the message for Alert Dialog */
        abuilderobj.setMessage("Do you want to exit ?");


         /* Set Cancelable false */
         abuilderobj.setCancelable(false);

        /* Set the positive button with a yes value */
         abuilderobj.setPositiveButton("Yes", new
            DialogInterface.OnClickListener() {

          @Override
          public void onClick(DialogInterface dialog, int which)
            {
                finish(); // The app will close When the user clicks yes button
            }
          });

        /* Set the Negative button with No value */
         abuilderobj.setNegativeButton("No", new
            DialogInterface.OnClickListener() {

          @Override
          public void onClick(DialogInterface dialog, int which){
           dialog.cancel();
```

```
    // The alert dialog box is canceled when the user clicks No
    }
});

        /* Create the Alert dialog */
        AlertDialog alertDialogobj = abuilderobj.create();

        /* Show the Alert Dialog box */
        alertDialogobj.show();
    }
}
```

Figure 6.26: Code in MainActivity.java file showing creating and displaying Alert Dialog Box

4. Execute the application and display the output

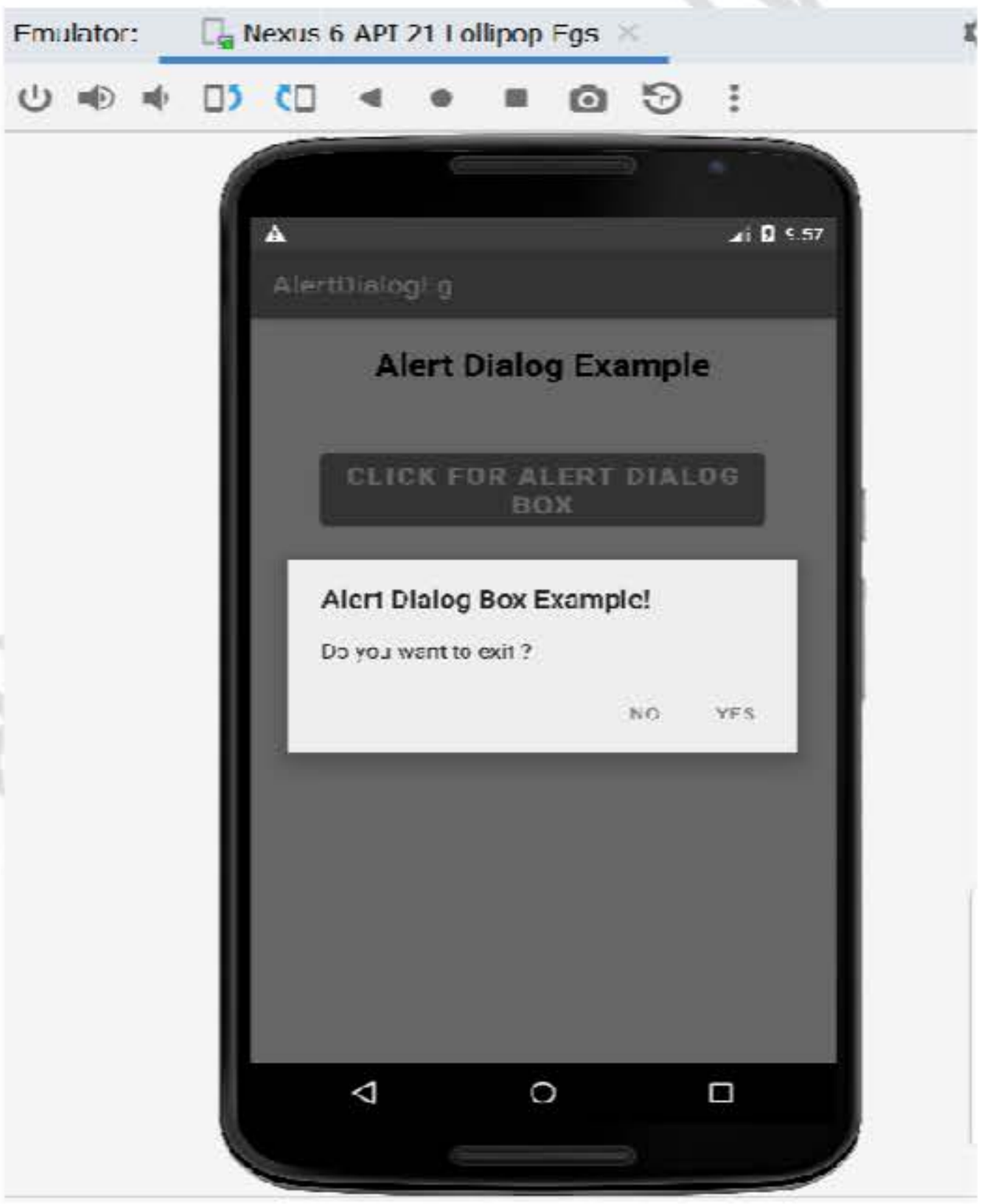


Figure 6.27: Displaying Alert Dialog on click of a button

6.6 Self-Assessment Questions

Q1. Explain different types of menus in Android.[8 marks, L2]

Q2. Differentiate between Option Menu and Context Menu in Android.[4 marks, L4]

Q3. Explain the following menus in Android with an example. [5 marks each, L2]

- Option Menu
- Context Menu
- Popup Menu

Q4. Explain the following w.r.t android in brief. [5 marks each, L2]

- Event handlers
- Event Listeners
- Event Listener Registration

Q5. Explain the 3 different ways of event registration with event handlers in Android with suitable code. [8 marks, L2]

Q6. What is an Action Bar? Explain in brief the methods used with Action Bar with suitable code and examples.[8 marks, L2]

Q7. Illustrate the use Action Bar in Android with a suitable example. [8 marks, L2]

Q8. What is an Alert dialog? Explain the methods used with Alert Dialog with suitable code and an example [8 marks, L2]

Q9. Illustrate the use of Alert dialog in Android with a suitable example.[8 marks, L2]

Q10. Explain the various Event Listeners that can be used to handle events with suitable examples. [6 marks, L2]

6.7 Self-Assessment Activities

Note: Use an appropriate emulator for the execution of an android application.

A1. Create an Android application, and add appropriate controls so that on the execution of the application, the output of the menus should be as shown in figure 6.8.

A2. Create an Android application, and add appropriate controls so that on the execution of the application, the output of the menus should be as shown in figure 6.12.

A3. Create an Android application, and add appropriate controls so that on the execution of the application, the output of the menus should be as shown in figure 6.16.

A4. Create an Android application, and add appropriate controls so that on the execution of the application, the output with an action bar should be as shown in figure 6.21

A5. Create an Android application, and add appropriate controls so that on the execution of the application, the output with an alert dialog should be as shown in figure 6.27

Graphic Era University

6.8 Multiple-Choice Questions

1. _____ of the following is not a menu in Android. [1 mark, L1]
   a) Option Menu
   b) Popup Menu
   c) Context Menu
   d) Design Menu

2. The menu which gets displayed when an element is long-pressed is ____. [1 mark, L1]
   a) Option Menu
   b) Popup Menu
   c) Context Menu
   d) None

3. The **"menu"** directory is created as a subdirectory of the _____ folder.[1 mark, L1]
   a) res
   b) drawable
   c) layout
   d) java

4. The menu resource file is created under the ____ folder. [1 mark, L1]
   a) res
   b) drawable
   c) layout
   d) menu

5. While creating a menu, menu items are added to _____. [1 mark, L1]
   a) activity_main.xml file
   b) MainActivity.java file
   c) Menu resource file
   d) None of the above

6. A small floating window that is displayed with an alert message and waits for user confirmation in android is _____. [1 mark, L1]
   a) Toast
   b) Alert dialog
   c) Message Box

Graphic Era University

d) None

7. For handling events, the android framework maintains an event queue where the events are stored as they occur and are handled on _____. [1 mark, L1]
   a) First In – First Out basis
   b) Last In – First Out basis
   c) None

8. The element which is usually placed at the top of the activity screen is ____. [1 mark, L1]
   a) Alert dialog
   b) Action Bar
   c) Message Box
   d) Toast

9. The element which is used to display information and is not interactive is___. [1 mark, L1]
   a) Alert dialog
   b) Action Bar
   c) Message Box
   d) Toast

10. setMessage() method is used to set the message to be displayed in____. [1 mark, L1]
   a) Toast
   b) Alert dialog
   c) Action Bar
   d) None

6.9 Key Answers to multiple-choice questions

1. Design Menu of the following is not a menu in Android.[d]
2. The menu which gets displayed when an element is long-pressed is Context Menu.[c]
3. The “menu” directory is created as a subdirectory of the res folder.[a]
4. The menu resource file is created under the menu folder.[d]
5. While creating a menu, menu items are added to the menu resource file.[c]
6. A small floating window that is displayed with an alert message and waits for user confirmation in android is Alert Dialog.[b]
7. For handling events, the android framework maintains an event queue where the events are stored as they occur and are handled on a First In – First Out basis.[a]

Graphic Era University

8. The element which is usually placed at the top of the activity screen is Action Bar.[b]
9. The element which is used to display information and is not interactive is Toast. [d]
10. setMessage() method is used to set the message to be displayed in Alert Dialog.[b]

6.10 Summary

Menus are used in the android application to provide a user-friendly interface. There are 3 types of Menus - Options Menu, Popup Menu, and Context Menu

Options Menu - An Options Menu is usually used to provide additional information about an application.

Popup Menu - A Popup Menu usually displays a vertical list with items in it, on the occurrence of an event like a button click, hovering over an element, on item touch, etc.

Context Menu - A Context Menu appears when the user right-clicks on the element or presses the element for a long duration usually 500 milliseconds. It is a floating menu.

An Action Bar element in an android application is usually present/displayed at the top of the screen/activity. The action bar is also known as App Bar.

An Action Bar consists of the following components - Application logo – icon, Application or Activity title, Application or Activity Sub-title, Primary Action Buttons/Icons for an Activity, and Action Overflow – where less used actions are placed.

The action that takes place due to the **user's interaction is called an event**.

Managing these events is called Event Handling. When an event occurs a specific task is performed in response to an event, this is called event handling. For handling events, the android framework maintains an event queue where the events are stored as they occur and are handled on a First In – First Out basis.

For managing event handling in android Event Handlers, Event Listeners, and Event Listener Registration is used.

Event Handlers - The method which is invoked/called to perform a specific task on the occurrence of an event is called an event handler.

Event Listeners - An Event Listener is an interface within the View Class. It contains a call-back method.

Event Listener Registration - The process of registering an event handler with an event listener is called event listener registration.

An Alert Dialog is a small floating window that is displayed with an alert message without moving to the next activity and allows the user to take decision and interact. The user may have to give an answer either by selecting the **"OK/Cancel" , "Yes/No"** buttons and depending on the response the application may continue or discontinue.

6.11 Keywords

- Menu
- Option Menu
- Popup Menu
- Context Menu
- Action Bar
- Event
- Event Handler
- Event Handler Registration
- Alert Dialog

6.12 Recommended Resources for Further Reading

Text Book(s)

1. Pradeep Kothari, "Android Application Development" Black-Book, Dreamtech Press,2015
2. Wei-Meng Lee, "Beginning Android Application Development", Wiley 2011.
3. Dawn Griffiths and David Griffiths, "Head First Android Development", O'Reilly, 2017.

References

- https://www.javatpoint.com/android-tutorial
- https://www.tutorialspoint.com/android/index.htm
- https://developer.android.com/guide/components/fundamentals

--*--

Graphic Era University