Q18. Create an Android application that displays the ... Modify the code of the

Ans. Follow the steps given in this chapter to create the required Android application.

Java file as follows:

```
package com.kogent.hello;

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

public class HelloWorld extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)     {
        super.onCreate(savedInstanceState);
        TextView message = new TextView(this);
        message.setText("Welcome to the world of Android");
        setContentView(message);
    }
}
```

**Listing 3.1:** Showing the Code of the `MainActivity.java` File in the `StartActivityDemo` Application

```
package com.kogent.android;

import android.net.Uri;
import android.os.Bundle;
import android.provider.ContactsContract;
import android.app.Activity;
import android.content.Intent;
import android.view.MotionEvent;

public class MainActivity extends Activity {
  /** Called when the activity is first created. */
   static final int PICK_CONTACT_REQUEST = 0;
  @Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_main);
   }
public boolean onTouchEvent(MotionEvent motionEvent)
  {
   if (motionEvent.getAction() ==MotionEvent.ACTION_DOWN)
     {
                        // This is invoked when  user touches the screen
        startActivityForResult(new Intent(Intent.ACTION_PICK,
                        ContactsContract.Contacts.CONTENT_URI),
                        PICK_CONTACT_REQUEST);
        return true;
     }
  return false;

  }
 protected void onActivityResult(int requestCode, int resultCode,Intent data)
 {
  if (requestCode == PICK_CONTACT_REQUEST)
  {
        if (resultCode == RESULT_OK)
        {
              // The picked contact would be displayed to the user.
              Uri uri = data.getData();
           Intent intent = new Intent(Intent.ACTION_VIEW, uri);
              startActivity(intent);
              }
        }
  }
}
```

In Listing 3.1, the `onTouchEvent()` event would be raised when a user touch down anywhere at the screen. This event invokes the `startActivityForResult()` method, which accepts three arguments, namely, `Intent` object, and URI of the `Contacts` activity, and the `int` type value. The `startActivityForResult()` method returns true if the appropriate action has been performed by the user; otherwise, it returns false. Further, the `onActivityResult()` method is invoked, which displays the contact picked up by the user.

Let's now run the application.

## Running the StartActivityDemo Application

Once you have created the `StartActivityDemo` application, you can execute the application to view its output. *Right-click* the `StartActivityDemo` application in the **Package Explorer** and *select* the **Run As→Android Application** option. The output of the `StartActivityDemo` application is shown in Figure 3.5:

1. *Create* the `ManageActivityCycle` application in the Eclipse IDE under the `com.kogent.android` package.
2. *Replace* the code of the `MainActivity.java` file, located in the **/src/com/kogent/android** folder, with the code shown in Listing 3.2:

**Listing 3.2:** Displaying the Code of the `MainActivity.java` File in the `ManageActivityCycle` Application

```
package com.kogent.android;
import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
public class MainActivity extends Activity
{
    String message="Event being invoked";
  /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Log.d(message, "onCreate() Method");
    }
    public void onStart()
    {
         super.onStart();
         Log.d(message, "onStart() Method");
    }
    public void onRestart()
    {
         super.onRestart();
         Log.d(message, "onRestart() Method");
    }
    public void onResume()
    {
         super.onResume();
         Log.d(message, "onResume() Method");
    }
    public void onPause()
    {
         super.onPause();
         Log.d(message, "onPause() Method");
    }
    public void onStop()
    {
         super.onStop();
         Log.d(message, "onStop() Method");
    }
    public void onDestroy()
    {
         super.onDestroy();
         Log.d(message, "onDestroy() Method");
    }
}
```

Listing 3.2 provides the implementation of all the lifecycle callback methods of an activity, such as `onStart()`, `onResume()`, `onPause()`, and `onStop()`.

3. *Save* the changes in the `ManageActivityCycle` application. *Select* the application folder in the **Package Explorer** pane and *click* the **Run** button from the toolbar of Eclipse IDE. The application is executed as shown in Figure 3.9:

2. *Replace* the code of the `MainActivity.java` file, located in the **/src/com/kogent/android** folder, with the code shown in Listing 3.3:

**Listing 3.3:** Showing the Code of `MainActivity.java` File in the `Themes` Application

```
package com.kogent.android;

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

public class MainActivity extends Activity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
  }
  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
  }
}
```

3. *Replace* the code of the `strings.xml` file, which is located in the **/res/values** folder, with the code shown in Listing 3.4:

**Listing 3.4:** Showing the Code of the `strings.xml` File in the `Themes` Application

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Themes</string>
    <string name="action_settings">Settings</string>
     <string name="hello_world">We use themes to override the default look and feel of an
Android application. A theme is a set of styles. Android theme was inspired by CSS styles.
We normally separate the styles to CSS file in our web
applications, in the same way we are separating Android styles to themes.
</string>
</resources>
```

4. *Add* a new XML file `Blackbook_style.xml` file, which is located in the **/res/values** folder, as shown in Listing 3.5:

**Listing 3.5:** Showing the Code of the `Blackbook_style.xml` File in the `Themes` Application

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="new_style" parent="@android:style/TextAppearance.Small">
        <item name="android:textColor">#008000 </item>
        <item name="android:typeface">sans</item>
        <item name="android:textSize">20sp</item>
    </style>
</resources>
```

5. *Replace* the code of the `activity_main.xml` file, which is located in the **/res/layout** folder, with the code shown in Listing 3.6:

**Listing 3.6:** Showing the Code of the `activity_main.xml` File in the `Themes` Application

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >
    <TextView
        android:id="@+id/textView2"
```

abstract description of the task that needs to be performed. Moreover, an Intent object also holds the description of what has happened and what needs to be announced.

**Q11. Explain fragments in Android programming.**

Ans. In these days, you see a variety of Android devices available in the market, which have different resolution and screen density. It is a challenge for Android application developers to develop applications that support all types of devices. In such a case, fragments come into picture. Fragments allow you to arrange and combine together to provide different layouts.

`Fragment` is a class or component that is used by an activity. It encapsulates functionality, which can be reused within activities and layouts. The concept of fragment is introduced after Android 3.0. A fragment runs in the context of an activity, but has its own lifecycle and user interface. It is also possible to define fragments without any user interface, which are also called headless fragments. Fragments can be added statically or dynamically to an activity. In fact, a single activity can contain multiple fragments. To add, remove, or replace fragments dynamically at runtime from an activity, you need to use the methods of the `FragmentManager` class.

**Q12. Create an Android application, which pops up an Alert Dialog with three buttons.**

Ans. Create application with name `AlertDialog`. Replace the code of the `MainActivity.java` file with the code shown below:

```
package com.kogent.android.alertdialog;

import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.widget.TextView;

public class MainActivity extends Activity
{
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        final TextView message = new TextView(this);
        message.setText("");
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setMessage(R.string.AlertBoxMessage);
        builder.setPositiveButton(R.string.Yes, new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int id) {
                message.setText("Your Answer is Yes" );

            }
        });
         builder.setNegativeButton(R.string.No, new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int id) {
                message.setText("Your Answer is No");

            }

        });
         builder.setNeutralButton(R.string.CanNotSay, new
         DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int id) {
                message.setText("Your Answer is Can Not Say");

            }

        });


        builder.show();
```

```
        setContentView(message);

    }
}
```

Replace the code of the strings.xml file with the code shown below:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="action_settings">Settings</string>
    <string name="app_name">HelloWorld</string>
    <color name="BackgroundColor">#337700</color>
    <string name="AlertBoxMessage">Do You Like Icecream</string>
    <string name="Yes">Yes</string>
    <string name="No">No</string>
    <string name="CanNotSay">Cant Say</string>
</resources>
```

**Q13. Create an Android application using intents and two activities. On the first activity, user can add name and then on pressing the OK button, the second activity should be started. The second activity must greet the user by using the text that entered on Activity one (e.g., Hi User).**

Ans. Create an application named, `HiUser`. The `MainActivity.java` file shown in the following code:

```
package com.kogent.android.hiuser;
import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;

public class MainActivity extends Activity  {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        final EditText  Name  = (EditText) findViewById(R.id.TextBox);
        Button btnValidate = (Button)findViewById(R.id.Submit);
        btnValidate.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                        Intent intent = new Intent(MainActivity.this,Activity2.class);
                intent.putExtra("Name",Name.getText().toString());
                startActivity(intent);

                }
        });

  }

}
```

Add the `Activity2.java` file shown in the following code:

```
package com.kogent.android.hiuser;
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class Activity2 extends Activity {

  protected void onCreate(Bundle savedInstanceState)
    {
        // TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        final TextView message = new TextView(this);
```

```
            </intent-filter>
        </activity>
        </application>

</manifest>
```

**Q14. Make an Android Application using the intent object to invoke the built-in browser and open google.com in it.**

Ans. Create application with name, `BuiltInBrowser`. Replace the code of the `MainActivity.java` file with the code shown below:

```
package com.kogent.android.builtinbrowser;
import android.net.Uri;
import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.view.View;
import android.widget.Button;

public class MainActivity extends Activity  {

   @Override
   protected void onCreate(Bundle savedInstanceState) {
         super.onCreate(savedInstanceState);
         setContentView(R.layout.activity_main);
         Button btnValidate = (Button)findViewById(R.id.Submit);
         btnValidate.setOnClickListener(new View.OnClickListener() {

               @Override
               public void onClick(View v) {
                     Intent viewIntent = new Intent("android.intent.action.VIEW",
                                 Uri.parse("http://www.google.com"));
                     startActivity(viewIntent);

               }
         });
   }
}
```

Replace the code of the `activity_main.xml` file with the code shown below:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity" >
    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginTop="20dp"
        android:text="Press Submit to open Google.com" />
    <Button
        android:id="@+id/Submit"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/textView1"
        android:layout_below="@+id/textView1"
        android:layout_marginTop="21dp"
        android:text="Submit" />
</RelativeLayout>
```

Developers of Android applications design the User Interface (UI) of the applications by using the Views and ViewGroups objects. There are various types of views and view groups, which are based on the View class. This class serves as the basis for designing interactive UI components, such as buttons, text fields, and check boxes. A View is a rectangular area on a user's screen on which the components are drawn as well as the events associated with these components are handled.

You can add views programmatically by providing the appropriate code in the Activity class. You can also provide the tree of views in the Extensible Markup Language (XML) layout files. Apart from defining the tree of views, you can also set the properties, listeners, focus, and visibility of the views. A specific view can be identified from the tree of views with the help of the ids that are defined in the XML layout files.

In this chapter, you learn about various types of ViewGroups, such as LinearLayout and ScrollView. Next, you learn how to work with Views and bind data using the AdapterView view. Then the chapter discusses about the AutoCompleteTextView view. It explains how to manage screen orientation and design UI programmatically. Further, the chapter explains how to handle events in Android and special fragments use. Finally, you learn to create various menus, such as context menu and options menu.

Let us learn how to create different layouts.

## Working with View Groups

Generally, an activity contains views and view groups in different styles and arrangements. A view is a widget that is displayed on the screen of the users, such as buttons, text boxes, check boxes, and radio buttons. The base class of a view is `android.view.View`. The grouping of one or more views is known as `ViewGroup`. A `ViewGroup` helps you in specifying the layout according to which the views can be placed and arranged. Some of the layouts that can be defined for the views are `LinearLayout`, `RelativeLayout`, and `TableLayout`. A `ViewGroup` is derived from the `android.view.ViewGroup` base class.

Let us create an Android application, named Layouts, in which each layout is added by defining the Android XML file. For example, the linear_layout.xml file would be created for the `LinearLayout` layout and tree of views defined in this file would be set in an Activity class by invoking the `setContentView()` method. You need to create the Layouts application in the Eclipse IDE within the com.kogent.android package. While creating the Layouts application, by default, an activity class, the activity_main.xml file, and the Androidmanifest.xml file are created. You need to define the buttons for each layout so that when the user clicks a button, the appropriate layout is displayed. In our case, we have defined five buttons for each layout, i.e., `LinearLayout`, `RelativeLayout`, `TableLayout`, `ScrollingView`, and `TableLayout`. Listing 4.1 shows the code of the `activity_main.xml` file located in the /res/layout folder:

**Listing 4.1:** Showing the Code of the `activity_main.xml` File in the Layouts Application

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" />
    <Button
        android:id="@+id/linearlayout"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="left"
        android:text="Linear Layout" />
    <Button
        android:id="@+id/relativelayout"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="left"
        android:text="Relative Layout" />
    <Button
```

```
        android:id="@+id/scrollingview"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="left"
        android:text="Scroll View" />
    <Button
        android:id="@+id/tablelayout"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="left"
        android:text="Table Layout" />
    <Button
        android:id="@+id/framelayout"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="left"
        android:text="Frame Layout" />
</LinearLayout>
```

After you have defined all the required views in the `activity_main.xml` file, as shown in Listing 4.1, you can use these views in the activity class of the `Layouts` application. In our case, we have accessed these views (defined in Listing 4.1) in the `Layouts` activity class through their ids. Listing 4.2 shows the code of the `Layouts` activity class:

**Listing 4.2:** Showing the Code of the `Layouts` Activity Class in the Layouts Application

```
package com.kogent.android;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
public class Layouts extends Activity implements OnClickListener {
  Button button_linearlayout, button_relativelayout, button_scrollview,
                button_tablelayout, button_framelayout;
  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // Initializing the buttons declared as view components
        button_linearlayout = (Button) findViewById(R.id.linearlayout);
        button_relativelayout = (Button) findViewById(R.id.relativelayout);
        button_scrollview = (Button) findViewById(R.id.scrollingview);
        button_tablelayout = (Button) findViewById(R.id.tablelayout);
        button_framelayout = (Button) findViewById(R.id.framelayout);

        // Setting the click event for layout button
        button_linearlayout.setOnClickListener(this);
        button_relativelayout.setOnClickListener(this);
        button_scrollview.setOnClickListener(this);
        button_tablelayout.setOnClickListener(this);
        button_framelayout.setOnClickListener(this);
  }

  // Verifying which button has been clicked and accordingly the Activity is
  // being started
  public void onClick(View view) {
        if (view == findViewById(R.id.linearlayout)) {
```

```
            public void onClick(View v) {
        // Provide the code to implement the click operation
                    Toast.makeText(ButtonView.this, "ButtonView is clicked!",
                                Toast.LENGTH_SHORT).show();
                }
        });

    }
}
```

In Listing 4.23, the ButtonView class extends the Activity class and displays a button, "Click Me". When a user clicks this button, the "ButtonView is clicked!" message is displayed, as shown in Figure 4.12:

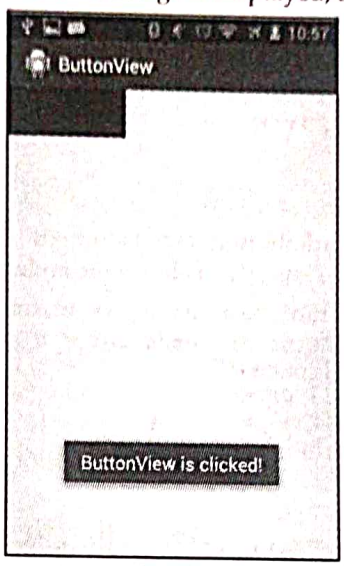


**Figure 4.12: Displaying the Output of the ButtonView Activity**

Let us now create the RadioButton view.

## Using the RadioButton View

The RadioButton view has two states, i.e., either checked or unchecked. A user can click the unchecked radio button to change its state. However, you should note that the RadioButton view cannot be unchecked by re-clicking it; you need to click the other RadioButton view. Generally, the radio buttons are grouped together in a RadioGroup. The following code snippet allows you to create the RadioButton view:

```
<RadioGroup
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:orientation="vertical">
<RadioButton android:id="@+id/radio_male"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Male"
android:textSize="30dp"/>
<RadioButton android:id="@+id/radio_female"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Female"
android:textSize="30dp"/>
</RadioGroup>
```

In the preceding code snippet, a RadioGroup view has been defined with two RadioButton views, namely, radio_male and radio_female. To provide some action that would be performed on clicking of either of the radio buttons, you can provide the following code snippet:

```
    protected void onCreate(Bundle savedInstanceState) {
          super.onCreate(savedInstanceState);
          setContentView(R.layout.activity_main);
          RadioButton radio_male = (RadioButton) findViewById(R.id.radio_male);
          RadioButton radio_female = (RadioButton) findViewById(R.id.radio_female);
          radio_male.setOnClickListener(radiolistener);
          radio_female.setOnClickListener(radiolistener);

    }
     private OnClickListener radiolistener = new OnClickListener()
     {
     public void onClick(View view)
     {
          RadioButton radiobutton = (RadioButton) view;
          Toast.makeText(MainActivity.this, radiobutton.getText(),
   Toast.LENGTH_LONG).show();
          }
     };
```

In the InputControl application, the radiobuttonview_layout.xml file is created in the /res/layout location, which displays the use of radio buttons. Listing 4.24 shows the code of the radiobuttonview_layout.xml file:

**Listing 4.24:** Displaying the Code of the radiobuttonview_layout.xml File in the InputControl Application

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
 //Defining the RadioGroup to hold RadioButtons

    <RadioGroup
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical" >
 //Defining the First RadioButton

        <RadioButton
            android:id="@+id/radio_male"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Male"
            android:textSize="30dp" />
 //Defining the Second RadioButton

        <RadioButton
            android:id="@+id/radio_female"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Female"
            android:textSize="30dp" />
    </RadioGroup>

</LinearLayout>
```

In Listing 4.24, a radio group is created with two radio buttons displaying, Male and Female, texts. Corresponding to the radiobuttonview_layout.xml file, the RadioButtonView Java class is created in the InputControl application to display the use of radio buttons. Listing 4.25 shows the code of the RadioButtonView.java file:

**Listing 4.25:** Displaying the Code of the RadioButtonView.java File in the InputControl Application

```
package com.kogent.android;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
```

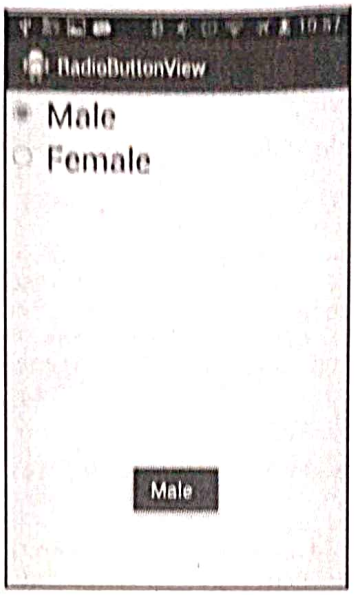


**Figure 4.14: Selecting a Radio Button**

Let us now understand how to create the CheckBox view.

## Using the CheckBox View

Similar to the RadioButton view, the CheckBox view is also two-state button that can be selected or unselected by the user. However, unlike RadioButton view, you can clear the selected CheckBox view by re-clicking the view. You need to define the CheckBox view in the layout XML file first and then implement View.onClickListener interface for the CheckBox view that would be checked by the user. The following code snippet shows how to create the CheckBox view in the layout:

```
<CheckBox

   android:id="@+id/checkbox1"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Qualification"
   android:textSize="30dp" />
```

To perform some action by check boxes, you have to use the OnClickListener interface for the CheckBox view. In the following code snippet, the text would be displayed, depending upon whether or not the CheckBox view is selected:

```
CheckBox checkbox1 = (CheckBox) findViewById(R.id.checkbox1);
checkbox1.setOnClickListener(new OnClickListener() {
public void onClick(View view) {

if (((CheckBox) view).isChecked()) {
Toast.makeText(FormDisplayActivity.this,   "This   CheckBox   has   been   Selected",
  Toast.LENGTH_LONG).show();
}
else {
Toast.makeText(FormDisplayActivity.this,   "This   CheckBox   has   not   been   selected",
  Toast.LENGTH_LONG).show();
}
}});
```

In the InputControl application, the checkboxview_layout.xml file is created in the /res/layout location, which displays the use of the <CheckBox> element. Listing 4.26 shows the code of the checkboxview_layout.xml file:

**Listing 4.26:** Displaying the Code of the checkboxview_layout.xml File in the InputControl Application

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <CheckBox
        android:id="@+id/checkbox1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Qualification"
        android:textSize="30dp" />

</LinearLayout>
```

In Listing 4.26, a CheckBox control is added, which displays the text, Qualification. Corresponding to the checkboxview_layout.xml file, the CheckBoxView Java class is created in the InputControl application to display the use of a check box view. Listing 4.27 shows the code of the CheckBoxView.java file:

**Listing 4.27:** Displaying the Code of the CheckBoxView.java File in the InputControl Application

```
package com.kogent.android;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.CheckBox;
import android.widget.Toast;

public class CheckBoxView extends Activity {

  @Override
  public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.checkboxview_layout);
        CheckBox checkbox1 = (CheckBox) findViewById(R.id.checkbox1);
        checkbox1.setOnClickListener(new OnClickListener() {
                public void onClick(View view) {
                        if (((CheckBox) view).isChecked()) {
                                Toast.makeText(CheckBoxView.this,
                                                "This CheckBox has been Selected",
                                                Toast.LENGTH_LONG).show();
                        } else {
                                Toast.makeText(CheckBoxView.this,
                                                "This CheckBox has not been selected",
                                                Toast.LENGTH_LONG).show();
                        }
                }
        });

  }
}
```

In Listing 4.27, an object of CheckBox class is created. This object will display a message with the help of the OnClickListener interface, whenever a user checks or unchecks the check box.

When you execute the InputControl application and click the CheckBox View button, you see a check box view, as shown in Figure 4.15:

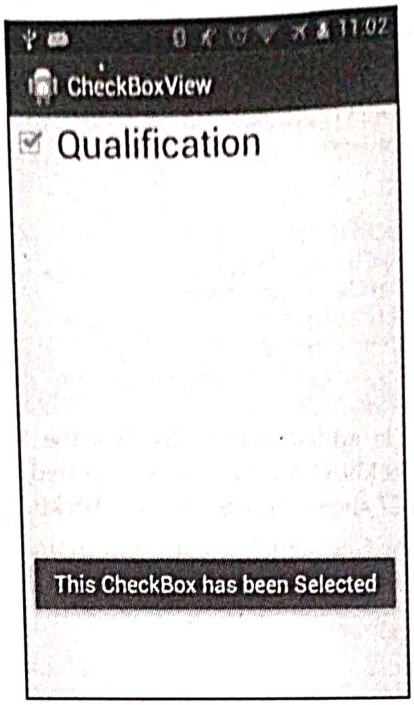


**Figure 4.15: Displaying the Output of the CheckBoxView Activity**

If you check the Qualification CheckBox view, the message is displayed that this checkbox has been selected.

## Using the ImageButton View

The ImageButton view is similar to the Button view, but it displays an image instead of the text. Generally, the background of the ImageButton view changes during the various states of a button, such as clicked or focused. The following code snippet shows how to create an image button:

```
<ImageButton
        android:id="@+id/imageButton1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/android_icon"
       android:contentDescription="@string/image_description"/>
```

In the preceding code snippet, android_icon is the name of the image file placed in the drawable folder. The contentDescription attribute provides the description about the image. This description can be set by defining a string text in the string.xml file. The src attribute provides the location of the image.

If you want that the image button do some action when you click it, use the OnClickListener interface, as shown in the following code snippet:

```
super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        final ImageButton       imageButton= (ImageButton)
findViewById(R.id.imageButton1);
                                        imageButton.setOnClickListener(new
OnClickListener() {
                                public void onClick(View v) {
                                Toast.makeText(FormDisplayActivity.this,"ImageButton is
clicked!", Toast.LENGTH_SHORT).show();
                                }
        });
```

In the preceding code snippet, an object of the ImageButton class is created. Then, the OnClickListener interface provides the onClick() method, which displays a message when a user click the image button.

In the InputControl application, create a folder, named drawable, in the /res location and add an image file in it. In our case the file is android_icon.jpeg. Then, add the code in the imagebuttonview_layout.xml file, located in

the /res/layout folder, which displays the use of the <ImageButton> element. Listing 4.28 shows the code of the imagebuttonview_layout.xml file:

**Listing 4.28:** Displaying the Code of the imagebuttonview_layout.xml File in the InputControl Application

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <ImageButton
        android:id="@+id/imageButton1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:contentDescription="image_description"
        android:src="@drawable/android_icon" />
</LinearLayout>
```

In Listing 4.28, an ImageButton control is added, which displays an image. Corresponding to the imagebuttonview_layout.xml file, the ImageButtonView Java class is created in the InputControl application to display the use of an image button view. Listing 4.29 shows the code of the ImageButtonView.java file:

**Listing 4.29:** Displaying the Code of the ImageButtonView.java File in the InputControl Application

```
package com.kogent.android;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.ImageButton;
import android.widget.Toast;
public class ImageButtonView extends Activity {
  @Override
  public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.imagebuttonview_layout);
        final ImageButton imageButton = (ImageButton) findViewById(R.id.imageButton1);
        imageButton.setOnClickListener(new OnClickListener() {
                @Override
                public void onClick(View v) {
                        Toast.makeText(ImageButtonView.this, "ImageButton is clicked!",
                                        Toast.LENGTH_SHORT).show();
                }

        });
  }
}
```

In Listing 4.29, an instance of the ImageButton is created, which displays the "ImageButton is clicked!" message when a user clicks it.

When you execute the InputControl application and click the ImageButton View button, you see an image button, as shown in Figure 4.16:

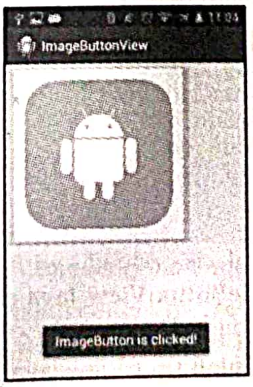


**Figure 4.16: Showing the Image Button**

When the user clicks the given icon, the floating view -"ImageButton is clicked!" appears. Next, let us create the ToggleButton view.

## Using the ToggleButton View

The ToggleButton view shows whether or not the state of the button has been changed (checked/unchecked) with the help of the light indicator. Also, you can set the ON and OFF text to be displayed when the state of the ToggleButton view changes. You can use the following code snippet to define the ToggleButton view in the XML file:

```
<ToggleButton
  android:id="@+id/toggle_button"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:textOn="Silent Mode on"
  android:textOff="Silent Mode off"
  android:textSize="30dp"/>
```

Then, you need to implement the OnClickListener interface in an activity class, as shown in the following code snippet:

```
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        final ToggleButton      toggle_button = (ToggleButton)
findViewById(R.id.toggle_button);
        toggle_button.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
        if (toggle_button.isChecked()) {
                Toast.makeText(FormDisplayActivity.this,"ToggleButton is On",
Toast.LENGTH_LONG).show();
                                }
        else {
                Toast.makeText(FormDisplayActivity.this,"ToggleButton is Off",
Toast.LENGTH_LONG).show();
                                }
                        }
                });
        }
```

In the InputControl application, the togglebuttonview_layout xml file is created in the /res/layout location, which displays the use of the <ToggleButton> element. Listing 4.30 shows the code of the togglebuttonview_layout.xml file:

**Listing 4.30:** Displaying the Code of the togglebuttonview_layout.xml File in the InputControl Application

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <ToggleButton
        android:id="@+id/toggle_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textOff="Silent Mode off"
        android:textOn="Silent Mode on"
        android:textSize="30dp" />

</LinearLayout>
```

In Listing 4.30, a ToggleButton control is added with different attributes. Corresponding to the togglebuttonview_layout.xml file, the ToggleButtonView Java class is created in the InputControl application to display the use of a toggle button view. Listing 4.31 shows the code of the ToggleButtonView.java file:

**Listing 4.31:** Displaying the Code of the ToggleButtonView.java File in the InputControl Application

```
package com.kogent.android;
```

```
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Toast;
import android.widget.ToggleButton;

public class ToggleButtonView extends Activity {
  @Override
  public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.togglebuttonview_layout);
        final ToggleButton toggle_button = (ToggleButton)
findViewById(R.id.toggle_button);
        toggle_button.setOnClickListener(new OnClickListener() {
                @Override
                public void onClick(View v) {
                        if (toggle_button.isChecked()) {
                                Toast.makeText(ToggleButtonView.this, "ToggleButton is
On",
                                                Toast.LENGTH_LONG).show();
                        } else {
                                Toast.makeText(ToggleButtonView.this,
                                                "ToggleButton is Off",
Toast.LENGTH_LONG).show();
                        }
                }

        });
  }

}
```

In Listing 4.31, a ToggleButton is created, which displays the text on the basis of whether it is checked or not.

When you execute the InputControl application and click the ToggleButton View button, you see a toggle button, displaying the "Silent Mode off" text, as shown in Figure 4.17:

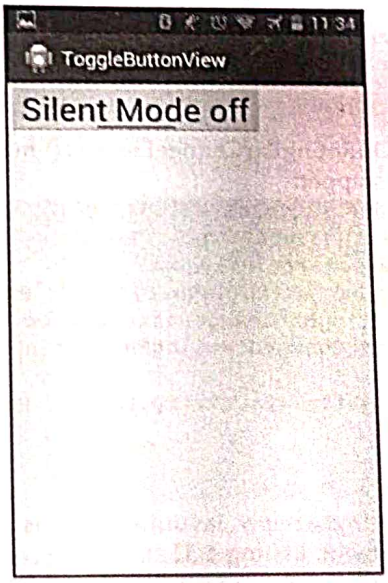


**Figure 4.17: Showing the Toggle Button**

Now, when you click the toggle button, the text changes as the state of the view has been changed to checked. You can see the light indicator in Figure 4.18, depicting that the ToggleButton view is checked:

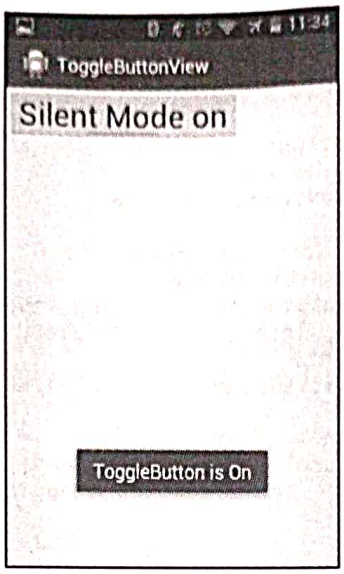


**Figure 4.18: Switching On the Toggle Button**

Let us move ahead to understand how to create the rating bar.

## Using the RatingBar View

The RatingBar view is used to display rating in the form of stars, which can be set by a user by touching, dragging, or using arrow keys. You can set the number of stars to be displayed by using the setNumStars(int) method in the Activity class or by using the android:numStars XML attribute in the XML layout. You must ensure that the layout_width of the RatingBar view should be set to wrap_content to display the number of stars being set. However, if another layout_width is being set, then the output may be unpredictable.

You can define the RatingBar view in the XML layout file by using the following code snippet:

```
<RatingBar
  android:id="@+id/ratingbar"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:numStars="7"
  android:stepSize="1.0"/>
```

Then, you need to implement the setOnRatingBarChangeListener() method for the RatingBar view in an Activity class, as shown in the following code snippet:

```
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
          RatingBar rating_bar = (RatingBar) findViewById(R.id.ratingbar);
         rating_bar.setOnRatingBarChangeListener(new OnRatingBarChangeListener() {
         public void onRatingChanged(RatingBar ratingBar, float rating, boolean
fromUser) {
         Toast.makeText(FormDisplayActivity.this, "User Rating is:" + rating,
Toast.LENGTH_LONG).show();
         }});
    }
```

In the InputControl application, the ratingbarview_layout.xml file is created in the /res/layout location, which displays the use of the <RatingBar> element. Listing 4.32 shows the code of the ratingbarview_layout.xml file:

**Listing 4.32:** Displaying the Code of the ratingbarview_layout.xml File in the InputControl Application

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
```

```
    <RatingBar
        android:id="@+id/ratingbar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:numStars="7"
        android:stepSize="1.0" />

</LinearLayout>
```

In Listing 4.32, a RatingBar control is added with various attributes. Corresponding to the ratingbarview_layout.xml file, the RatingBarView Java class is created in the InputControl application to display the use of a rating bar view. Listing 4.33 shows the code of the RatingBarView.java file:

**Listing 4.33:** Displaying the Code of the RatingBarView.java File in the InputControl Application

```
package com.kogent.android;

import android.app.Activity;
import android.os.Bundle;
import android.widget.RatingBar;
import android.widget.RatingBar.OnRatingBarChangeListener;
import android.widget.Toast;

public class RatingBarView extends Activity {
  @Override
  public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.ratingbarview_layout);
        RatingBar rating_bar = (RatingBar) findViewById(R.id.ratingbar);
        rating_bar.setOnRatingBarChangeListener(new OnRatingBarChangeListener() {
  public void onRatingChanged(RatingBar ratingBar,
                                        float rating, boolean fromUser) {
                                Toast.makeText(RatingBarView.this,
                                        "User Rating is:" + rating,
Toast.LENGTH_LONG)

                                        .show();
                        }
                });

  }
}
```

In Listing 4.33, an object of the RatingBar class is created. Then, the OnRatingBarChangeListener interface is used to display a message when a user gives rates. When you execute the InputControl application and click the Ratingbar View button, you see a rating bar with seven stars, as shown in Figure 4.19:

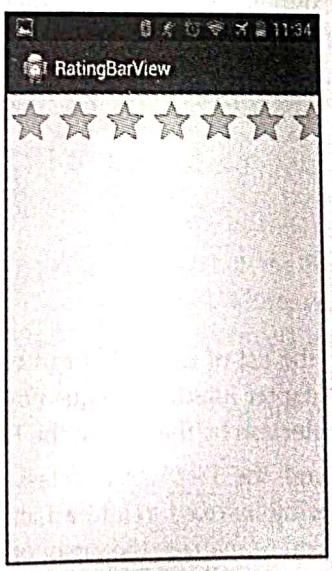


**Figure 4.19: Showing the Rating Bar in the FormDisplay Application**

```
            PrefFragment prefFragment = new PrefFragment();
            FragmentManager fragmentManager = getFragmentManager();
            FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
            fragmentTransaction.replace(android.R.id.content, prefFragment);
            fragmentTransaction.addToBackStack(null);
            fragmentTransaction.commit();

        }
    }
```

In Listing 4.51, the FragmentManager and the FragmentTransaction classes are used to display the preference fragment in an activity. The addToBackStack() method is used to add the preference fragment to the back stack so that the user can dismiss the fragment by clicking the back button.

6. *Execute* the PreferenceFragments application. Figure 4.32 shows the output of the PreferenceFragments application:

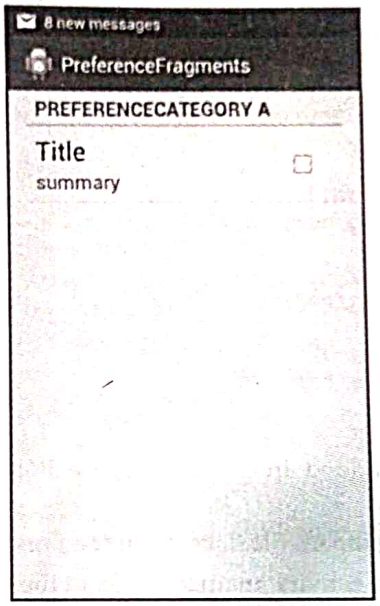


**Figure 4.32: Displaying the Output of the PreferenceFragments Application**

Let us now learn how to create various types of menus in Android.

# Creating Menus

Note that in the devices based on Android 4 platform, there are different types of menus, such as options menu or context menu. The Android SDK provides the simple framework to add different types of menus to your applications. In this section, you would learn how to create the following types of menus:

- The Options menu
- The Context menu
- The Submenus

## *The Options Menu*

Generally, the options menu is created for the applications, such as email, in which a user should have various options to be performed. For example, in the email application, a user can compose an email, reply to the email, or forward the e-mail. Let us create the OptionsMenu application in which the MainActivity class would be displaying the options menu by creating the instance of the MenuInflater class.

Perform the following steps to create and execute the OptionsMenu application:

1. *Create* the OptionsMenu application in Eclipse IDE within the com.kogent.android package.
2. *Modify* the code of the strings.xml file, located in the /res/values folder, as shown in Listing 4.52:

**Listing 4.52:** Displaying the Code of the strings.xml File in the OptionsMenu Application

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">OptionsMenu</string>
    <string name="hello_world">Hello world!</string>
    <string name="action_settings">Settings</string>
    <string name="new_mail">Compose</string>
    <string name="reply">Reply</string>
    <string name="forward">Forward</string>
</resources>
```

In Listing 4.52, some new strings are defined, which will be required in the application. These are new_mail, reply, and forward.

3. *Add* the code of the options_menu.xml file, located in the /res/menu folder, as shown in Listing 4.53:

**Listing 4.53**: Displaying the Code of the options_menu.xml File in the OptionsMenu Application

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

    <item
        android:id="@+id/new_mail"
        android:title="@string/new_mail"/>
    <item
        android:id="@+id/reply"
        android:title="@string/reply"/>
    <item
        android:id="@+id/forward"
        android:title="@string/forward"/>

</menu>
```

In Listing 4.53, three items are added in a menu. These items display text, which are defined in the strings.xml file.

4. Modify the code of the activity_main.xml file, located in the /res/layout folder, as shown in Listing 4.54:

**Listing 4.54:** Displaying the Code of the activity_main.xml File in the OptionsMenu Application

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.kogent.android.MainActivity"
    tools:ignore="MergeRootFrame" />
```

In Listing 4.54, a frame layout is set to display an options menu.

5. Modify the code of the MainActivity class, located in the /src/com/kogent/android folder, as shown in Listing 4.55:

**Listing 4.55:** Displaying the Code of the MainActivity.java File in the OptionsMenu Application

```
package com.kogent.android;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.widget.Toast;


public class MainActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_main);
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.options_menu, menu);
        return true;
    }
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.new_mail:     Toast.makeText(this, "A new mail is composed!",
                Toast.LENGTH_LONG).show();
                                    break;
            case R.id.reply:      Toast.makeText(this, "Reply to the mail sender!",
                Toast.LENGTH_LONG).show();
                                    break;
            case R.id.forward: Toast.makeText(this, "Forward your mail!",
                Toast.LENGTH_LONG).show();
                                    break;
        }
        return true;
    }
}
```

In Listing 4.55, the instance of the MenuInflater class has been created and the options_main.xml file of the res/menu folder is set as the menu to be inflated by invoking the inflate() method on the instance. Note that we have overridden the onCreateOptionsMenu() method to create the options menu. Further, if you want to perform any action on the click of the option displayed in the options menu, then you need to override the onOptionsItemSelected() method.

6. Execute the OptionsMenu application. The options menu is displayed, when you click the Menu button on the mobile device, as shown in Figure 4.33:

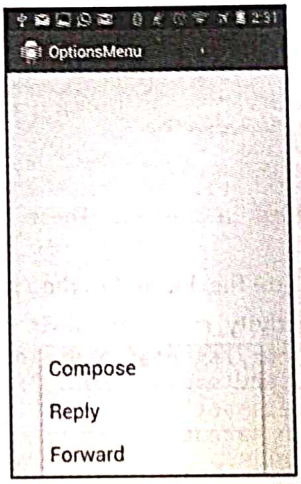


**Figure 4.33: Displaying the OptionsMenu Application**

Whenever you click an option from the options menu, shown in Figure 4.33, the appropriate message is displayed.

Let us now learn how to create the context menu.

## The Context Menu

In the Android devices, a context menu is similar to the menu displayed when a user performs a right-click on the computer. Generally, the context menu is used in devices to display various options related to an item in UI. On Android devices, a context menu is displayed when a user presses and holds an item for a long time.

Let us create a new application ContextMenu with activity class, named ContextMenuActivity. The ContextMenuActivity class would display the context menu options when the button is clicked for a long time.

Perform the following steps to create and execute the ContextMenu application:

1. *Create* the ContextMenu application in Eclipse IDE within the com.kogent.android package.
2. *Modify* the code of the strings.xml file, located in the /res/values folder, as shown in Listing 4.56:

**Listing 4.56:** Displaying the Code of the strings.xml File in the ContextMenu Application

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

    <string name="app_name">ContextMenu</string>
    <string name="hello_world">Hello world!</string>
    <string name="action_settings">Settings</string>
    <string name="new_mail">Compose</string>
    <string name="reply">Reply</string>
    <string name="forward">Forward</string>

</resources>
```

In Listing 4.56, some new strings are defined, which will be required in the application. These are new_mail, reply, and forward.

3. Modify the code of the context_menu.xml file, located in the /res/menu folder, as shown in Listing 4.57:

**Listing 4.57:** Displaying the Code of the context_menu.xml File in the ContextMenu Application

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context="com.kogent.android.ContextMenuActivity" >

    <item
        android:id="@+id/new_mail"
        android:title="@string/new_mail"/>
    <item
        android:id="@+id/reply"
        android:title="@string/reply"/>
    <item
        android:id="@+id/forward"
        android:title="@string/forward"/>

</menu>
```

In Listing 4.57, three items are added in a menu. These items display text, which are defined in the strings.xml file.

4. *Modify* the code of the activity_main.xml file, located in the /res/layout folder, as shown in Listing 4.58:

**Listing 4.58:** Displaying the Code of the activity_main.xml File in the ContextMenu Application

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
  <Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Click Me" />
 </LinearLayout>
```

In Listing 4.58, a linear layout is set with a Button control, which will display the Click Me text.

5. Modify the code of the ContextMenuActivity class, located in the /src/com/kogent/android folder, as shown in Listing 4.59:

**Listing 4.59:** Displaying the Code of the ContextMenuActivity.java File in the ContextMenu Application

```
package com.kogent.android;
import android.app.Activity;
```

```
import android.os.Bundle;
import android.view.ContextMenu;
import android.view.ContextMenu.ContextMenuInfo;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.widget.AdapterView.AdapterContextMenuInfo;
import android.widget.Button;
import android.widget.Toast;

public class ContextMenuActivity extends Activity
{
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
  {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button button1 = (Button) findViewById(R.id.button1);
        button1.setOnCreateContextMenuListener(this);
       }

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

    @Override
    public boolean onContextItemSelected(MenuItem item)
    {
        AdapterContextMenuInfo info = (AdapterContextMenuInfo) item.getMenuInfo();
        switch (item.getItemId()) {
           case R.id.new_mail:
                  Toast.makeText(this, "A new mail is composed!",
               Toast.LENGTH_LONG).show();
          break;

           case R.id.reply:
                     Toast.makeText(this, "Reply to the mail sender!" ,
               Toast.LENGTH_LONG).show();
          break;

           case R.id.forward:
                     Toast.makeText(this, "Forward your mail!" ,
               Toast.LENGTH_LONG).show();
          break;

        }
        return true;
        }
}
```

In Listing 4.59, we have created the context menu by overriding the onCreateContextMenu() method in which the instance of the MenuInflater class has been created. Moreover, to implement the clicking of an option from the context menu, we have overridden the onContextItemSelected() method.

6. Execute the ContextMenu application. In the output, you see a button with the text, "Click Me". When you press the button for a long time, the context menu appears, as shown in Figure 4.34:

**Q14. Describe the role of the AdapterView class.**

Ans. In Android programming, you can display the stored data using the AdapterView subclass of the ViewGroup class. With the help of the Adapter interface, the AdapterView class can bind the data to a particular view. The data to be displayed using the AdapterView class can be stored in an external resource, such as strings or drawables. Some examples of the AdapterView subclasses are ListView, Spinner, and Gallery, that allow you to bind the stored data and display the data in a specific manner.

**Q15. Explain about preference fragment.**

Ans. Preference fragment allows you to set preferences in your Android applications. Preferences means some settings related to a user's choice concerning the using of applications. For example, some users like and allow saving the login credentials that they need to access their Web resources. In Android, the PreferenceActivity base class allows you to display an activity for editing preferences. However, from Android 3.0 version onwards, you can use the PreferenceFragment class to do the same thing.

To create a list of preferences in your Android application, you first need to create an XML file and populate it with the various XML elements. This XML file defines the various items that you want to set in your application. Then, you create the fragment class by extending the PreferenceFragment base class. Use the addPreferencesFromResource() method to load the XML file of preferences in the preference fragment.

**Q16. Create an Android application with three toggle buttons named toggle 1, toggle2, and toggle 3. Use an extra button called states, which on tapping shows the current states (i.e., ON or OFF) of three toggle buttons through a toast notification.**

Ans. Create an application, FormDisplay. Modify the code of the strings.xml file in the \res\values folder as follows:

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

    <string name="app_name">FormDisplay</string>
    <string name="hello_world">Hello world!</string>
    <string name="menu_settings">Settings</string>
    <string name="editText">This is simple edit text view</string>
    <string name="clickHere">Click Here</string>
    <string name="maleDisplay">This might be displayed for male</string>
    <string name="Android_Image">This is an image button</string>
    <drawable name="button_pressed">#FF3300</drawable>
    <drawable name="button_focused">#FF0000</drawable>

</resources>
```

Modify the code of the activity_main.xml file in the \res\layout folder as follows:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

<ToggleButton
  android:id="@+id/toggle_button1"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:textOn="Silent Mode on"
  android:textOff="Silent Mode off"
  android:textSize="30dp"/>

<ToggleButton
  android:id="@+id/toggle_button2"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
```

```
    android:textOn="Silent Mode on"
    android:textOff="Silent Mode off"
    android:textSize="30dp"/>

<ToggleButton
  android:id="@+id/toggle_button3"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:textOn="Silent Mode on"
  android:textOff="Silent Mode off"
  android:textSize="30dp"/>

<Button
        android:id="@+id/btnDisplay"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/clickHere" />

</LinearLayout>
```

Create the FormDisplayActivity.java file as follows:

```
package com.kogent.android;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.ImageButton;
import android.widget.RadioButton;
import android.widget.RatingBar;
import android.widget.RatingBar.OnRatingBarChangeListener;
import android.widget.Toast;
import android.widget.ToggleButton;

public class FormDisplayActivity extends Activity {
private ToggleButton toggle_button1, toggle_button2, toggle_button3;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        onClickButton();

  }

  public void onClickButton() {


        toggle_button1 = (ToggleButton) findViewById(R.id.toggle_button1);
        toggle_button2 = (ToggleButton) findViewById(R.id.toggle_button2);
        toggle_button3 = (ToggleButton) findViewById(R.id.toggle_button3);
       Button btnDisplay = (Button) findViewById(R.id.btnDisplay);

       //final ToggleButton toggle_button = (ToggleButton)
 findViewById(R.id.toggle_button);
       btnDisplay.setOnClickListener(new OnClickListener() {

                public void onClick(View v) {
```

```
                        StringBuffer result = new StringBuffer();
                                result.append("ToggleButton1 :
").append(toggle_button1.getText());
                                result.append("\nToggleButton2 :
").append(toggle_button2.getText());
                                result.append("\nToggleButton3 :
").append(toggle_button3.getText());

                                Toast.makeText(FormDisplayActivity.this, result.toString(),
                                                Toast.LENGTH_SHORT).show();
                        }
                });
        }


}
```

**Q17. Write an Android application using the multiple radio buttons in Alert Dialog.**

Ans. Create an application, alertdialogbox. Modify the code of the activity_main.xml file in the \res\layout folder as follows:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="@string/hello_world" />

</RelativeLayout>
```

Create the MainActivity.java file as follows:

```
package com.Kogent.android.alertdialogbox;

import android.os.Bundle;
import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.view.Menu;
import android.widget.Toast;


public class MainActivity extends Activity {
    private  String gen;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

         final CharSequence[] gender = {"Male","Female"};
            AlertDialog.Builder alert = new AlertDialog.Builder(this);
            alert.setTitle("Select Gender");
            alert.setSingleChoiceItems(gender,-1, new DialogInterface.OnClickListener()
            {
                @Override
                public void onClick(DialogInterface dialog, int which)
                {
                                Toast.makeText(
```

```
                                                MainActivity.this,
                        gender[which]+"Select ",
                        Toast.LENGTH_SHORT
                        )
                        .show();
                }
            });
        //    AlertDialog alert = builder.create();
            alert.show();

    }}
```