CHAPTER 3

GUI Design for Android Apps, Part 3: Designing Complex Applications

In the previous chapter, you learned about Android interface design by creating a simple application called `GuiExam`. The chapter also covered the state transition of activities, the `Context` class, and an introduction to intents and the relationship between applications and activities. You learned how to use a layout as an interface, and how button, event, and inner event listeners work. In this chapter, you learn how to create an application with multiple activities; examples introduce the explicit and implicit trigger mechanisms of activities. You see an example of an application with parameters triggered by an activity in a different application, which will help you understand the exchange mechanism for the activity's parameters.

Applications with Multiple Activities

The application in the previous example has only one activity: the main activity, which is displayed when the application starts. This chapter demonstrates an application with multiple activities, using the activity-intent mechanism, and shows the changes needed in the `AndroidManifest.xml` file.

As previously described, an activity is triggered by an intent. There are two kinds of intent-resolution methods: *explicit match* (also known as *direct intent*) and *implicit match* (also known as *indirect intent*). A triggering activity can also have parameters and return values. Additionally, Android comes with a number of built-in activities, and therefore a triggered activity can come from Android itself, or it can be customized. Based on these situations, this chapter uses four examples to illustrate different activities. For the explicit match, you see an application with or without parameters and return values. For the implicit match, you see an application that uses activities that come from the Android system or are user defined.

Triggering an Explicit Match of Activities with No Parameters

Using explicit match without parameters is the simplest trigger mechanism of the activity intent. This section first uses an example to introduce this mechanism and later covers more complex mechanisms.

The code framework of the activity-intent triggering mechanism for explicit matching includes two parts: the activities of the callee (being triggered) and those of the caller (trigger). The trigger is not limited to activities; it can also be a service, such as a broadcast intent receiver. But because you have only seen the use of activities so far, the triggers for all the examples in this section are activities.

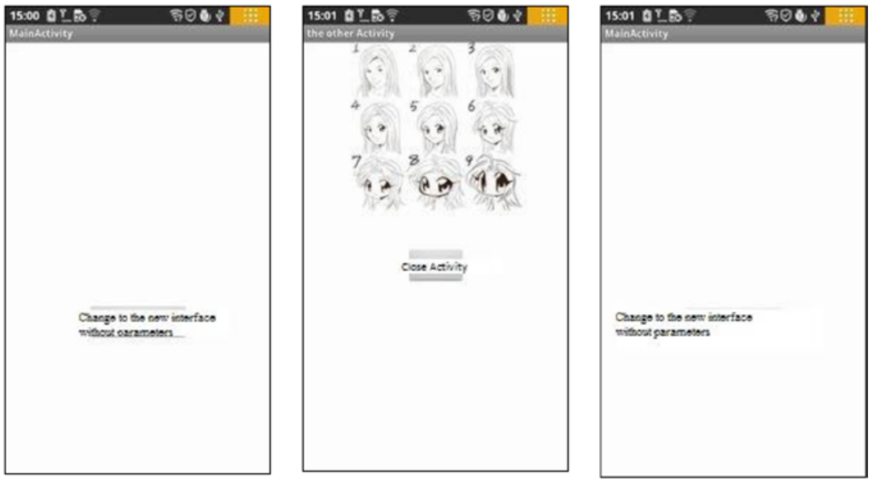
1. The source code framework for the activity of the callee does the following:
    a. Defines a class that inherits from the activity.
    b. If there are parameters that need to be passed, then the source code framework of the activity calls the `Activity.getIntent()` function in the `onCreate` function to obtain the `Intent` object that triggers this activity, and then gets the parameters being passed through functions like `Intent.getData ()`, `Intent.getXXXExtra ()`, `Intent.getExtras ()`, and so on.
    c. Writes code for the normal activity patterns.
    d. If the trigger returns values, does the following before exiting the activity:
        i. Defines an `Intent` object
        ii. Sets data values for the intent with functions like `Intent.putExtras()`
        iii. Sets the return code of the activity by calling the `Activity.setResult()` function
    e. Adds the code for the activity of the callee in the `AndroidManifest.xml` file.
2. The code framework for the activity of the callee does the following:
    a. Defines the `Intent` object, and specifies the trigger's context and the `class` attribute of the triggered activity.
    b. If parameters need to be passed to the activity, sets the parameters for the `Intent` object by calling functions of the intent like `setData()`, `putExtras()`, and so on.

c. Calls `Activity.startActivity(Intent intent)` function to trigger an activity without parameters, or call `Activity.startActivityForResult(Intent intent, int requestCode)` to trigger an activity with parameters.

d. If the activity needs to be triggered by the return value, then the code framework rewrites the `onActivityResult()` function of the `Activity` class, which takes different actions depending on the request code (`requestCode`), result code (`resultCode`), and intentions (`Intent`) values.

In step 2a, the class attribute of the triggered activity is used, which involves a Java mechanism called *reflection*. This mechanism can create and return an object of the class according to the class name. The object of the triggered activity is not constructed before the triggering; therefore triggering the activity also means creating an object of that class so that subsequent operations can continue. That is, triggering the activity includes the operation of the newly created class objects.

The following two examples illustrate the code framework in detail. This section describes the first one. In this example, the triggered activity belongs to the same application as the activity of the trigger, and the triggered activity does not require any parameters and does not return any values. The new activity is triggered via a button, and its activity interface is similar to the interface of the example in the section “Exit Activities and Application.” in Chapter 2, Figure 2-16. The entire application interface is shown in Figure 3-1.


15:00
MainActivity
Change to the new interface without parameters
15:01
the other Activity
1
2
3
4
5
6
7
8
9
Close Activity
15:01
MainActivity
Change to the new interface without parameters

(a) Interface when the app starts

(b) Interface when Change To The New Interface Without Parameters is clicked

(c) Interface when Close Activity is clicked

Figure 3-1. *The application interface with multiple activities in the same application without parameters*

After the application starts, the application's main activity is displayed, as shown in Figure 3-1(a). When the Change To The New Interface Without Parameters button is clicked, the app displays the new activity, as shown in Figure 3-1(b). Clicking the Close Activity button causes the interface to return to the application's main activity, as shown in Figure 3-1(c).

Create this example by modifying and rewriting the example in the `GuiExam` section in Chapter 2, as follows:

1. Generate the corresponding layout file for the triggered activity:

   a. Right-click the shortcut menu in the `res\layout` subdirectory of the application, and select New ➤ Other Items. A New dialog box pops up. Select the `\XML\XML File` subdirectory, and click Next to continue. In the New XML File dialog box, enter the file name (in this case `noparam_otheract.xml`), and click Finish. The entire process is shown in Figure 3-2.

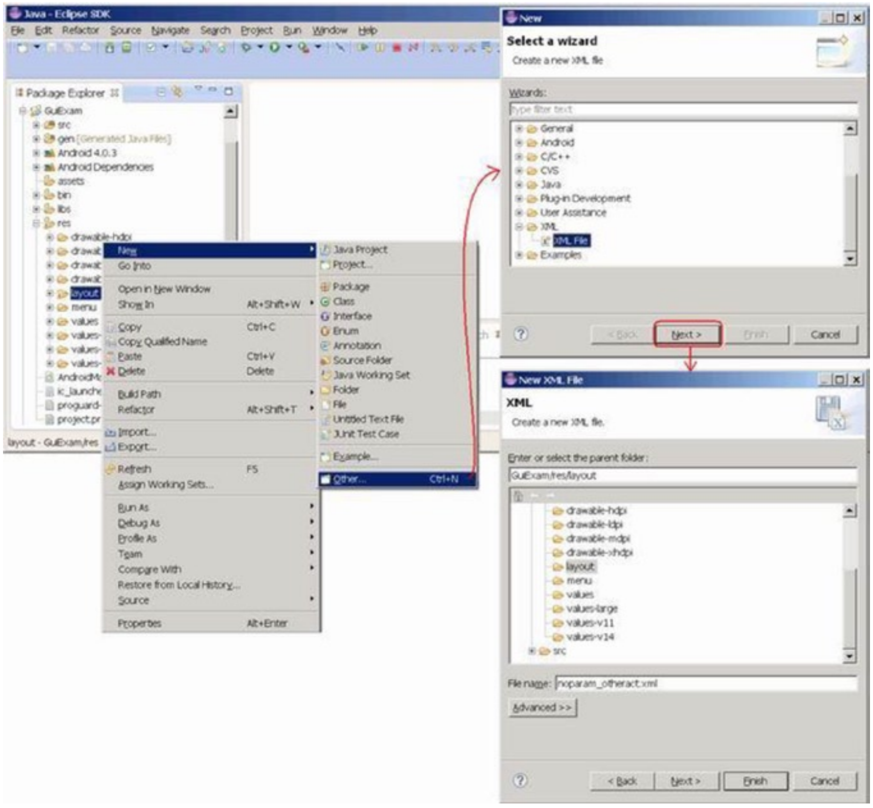
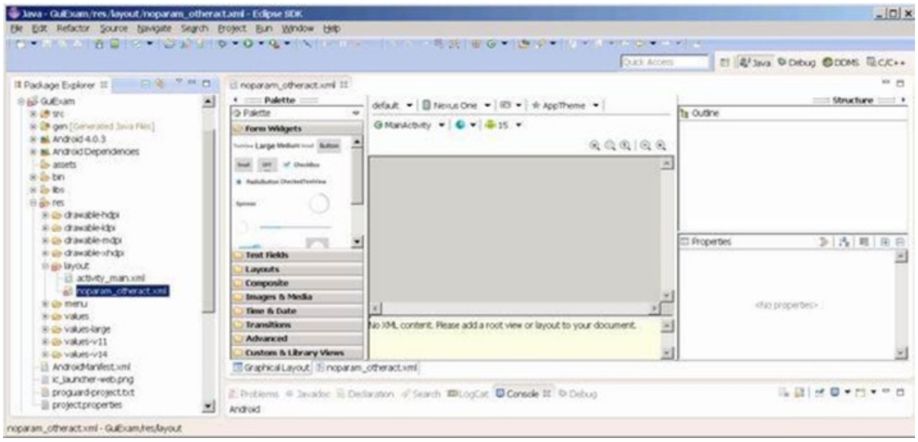


Figure 3-2. *The layout file for the triggered activity*

■ **Note** The file name is the name of the layout file. You must use only lowercase letters for compilation to be successful; otherwise you will get the error "Invalid file name: must contain only a-z0-9_."

You can see the newly added `xxx.xml` file (in this case, `noparam_otheract.xml`) in the project's Package Explorer, as shown in Figure 3-3.



No XML content. Please add a root view or layout to your document.

Figure 3-3. Initial interface of the application's newly added layout file

■ **Note** The layout editor window on the right is still empty, and there is no visible interface so far.

b. Select the `Layouts` subdirectory in the left palette, and drag the layout control (in this case, `RelativeLayout`) onto the window in the right pane. You immediately see a visible (phone-screen shaped) interface, as shown in Figure 3-4.

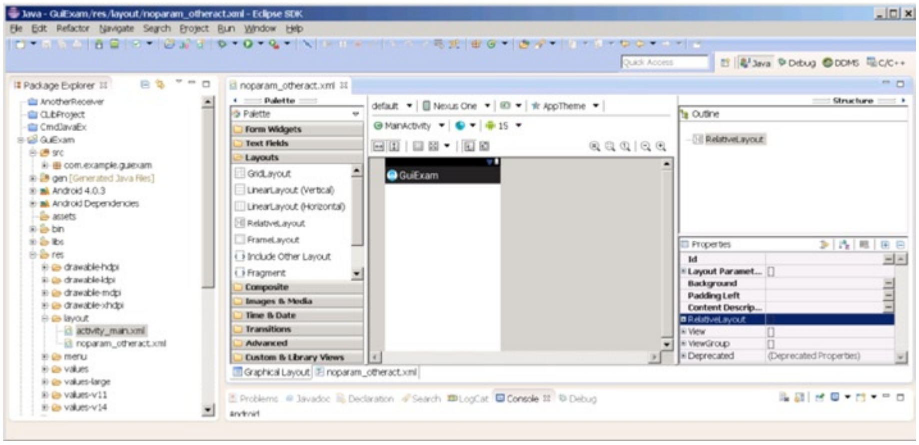


Figure 3-4. *Drag-and-drop layout for the newly added layout file*

c. Based on the same methodology described in the section "Using ImageView" in Chapter 2, place an `ImageView` and a button in the new layout file. Set the `ImageView` widget's `ID` attribute to `@+id/picture` and the `Button` widget's `ID` attribute to `@+id/closeActivity`. The `Text` property is "Close Activity," as shown in Figure 3-5. Finally, save the layout file.

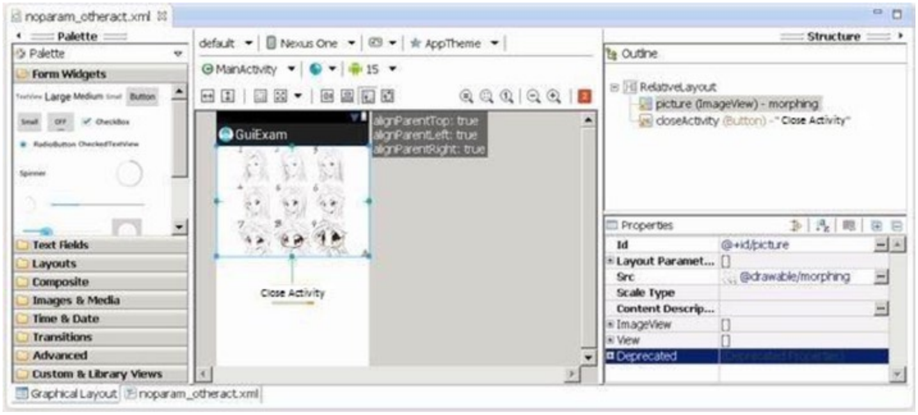
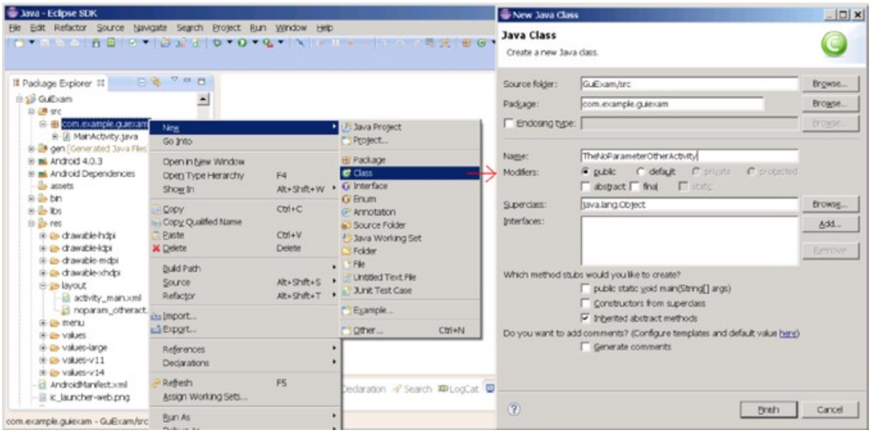


Figure 3-5. *Final configuration of the newly added layout file*

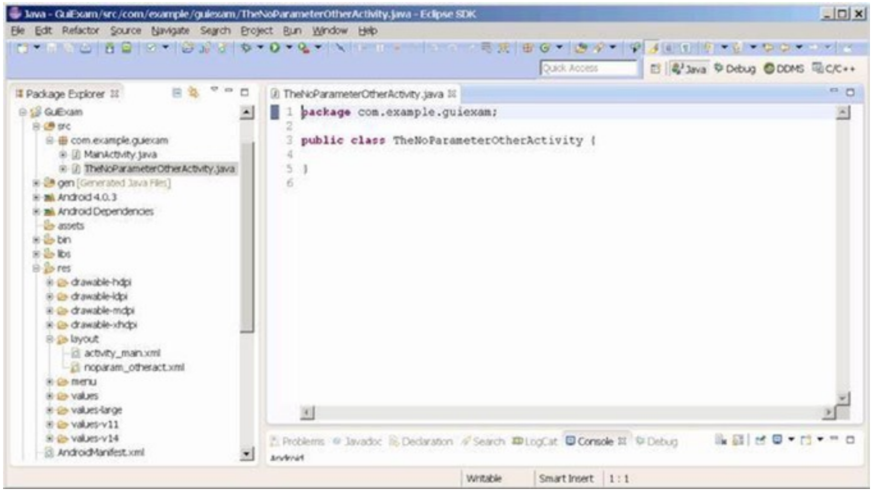
2. Add the corresponding `Activity` class for the layout file (Java source files). To do so, right-click `\src\com.example.XXX` under the project directory, and select New ➤ Class on the shortcut menu. In the New Java Class dialog box, for Name, enter the `Activity` class name corresponding to the new layout file (in this case, `TheNoParameterOtherActivity`). Click Finish to close the dialog box. The whole process is shown in Figure 3-6.



Source folder: GuiExam/src
Package: com.example.guiexam
Name: TheNoParameterOtherActivity
Modifiers: [✓ public] default private protected
Superclass: java.lang.Object
Which method stubs would you like to create?
public static void main(String[] args)
Constructors from superclass
[✓ Inherited abstract methods]
Do you want to add comments? (Configure templates and default value here)
Generate comments

Figure 3-6. Corresponding class for the newly added layout file

You can see the newly added Java files (in this case, `TheNoParameterOtherActivity.java`) and the initial code, as shown in Figure 3-7.


Java - GuiExam/src/com/example/guiexam/TheNoParameterOtherActivity.java - Eclipse SDK
Package Explorer
TheNoParameterOtherActivity.java

```
package com.example.guiexam;

public class TheNoParameterOtherActivity {

}
```

Figure 3-7. Corresponding class and initial source code of the newly added layout

3. Edit the newly added `.java` file (`TheNoParameterOtherActivity.java`). This class executes the activity of the triggered activity (callee). Its source code is as follows (bold text is added or modified):

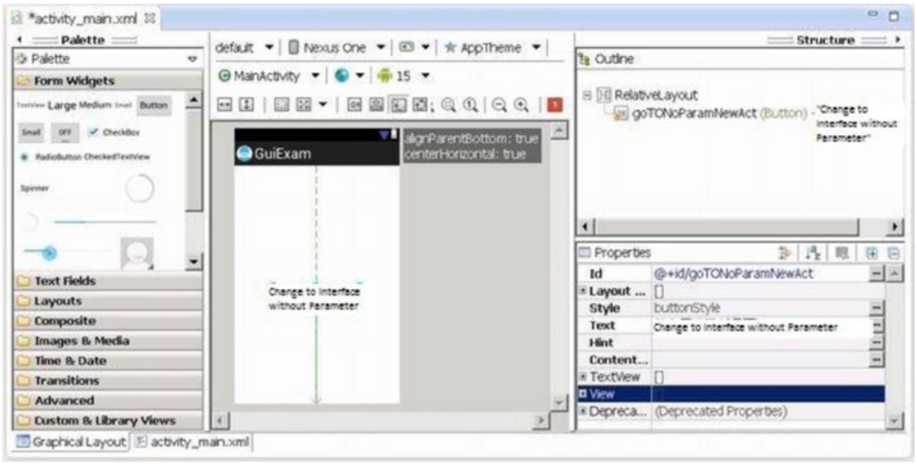
```
Line #        Source Code
1  package com.example.guiexam;
2  import android.os.Bundle;                     // Use Bundle class
3  import android.app.Activity;                  // Use Activity Class
4  import android.widget.Button;                 // Use Button class
5  import android.view.View;                     // Use View class
6  import android.view.View.OnClickListener; // Use OnClickListener Class

7  public class TheNoParameterOtherActivity extends Activity {
8  // Define Activity subclass
9     @Override
10 protected void onCreate(Bundle savedInstanceState) {
11 // Define onCreate method
12        super.onCreate(savedInstanceState);
13 // onCreate method of calling parent class
14        setContentView(R.layout.noparam_otheract);
15 // Set layout file
16        Button btn = (Button) findViewById(R.id.closeActivity);
17 // Set responding code for <Close Activity> Button
18        btn.setOnClickListener(new /*View.*/OnClickListener(){
19          public void onClick(View v) {
              finish();
   // Close this activity
          }
          });
   }
   }
```

In line 7, you add the superclass `Activity` for the newly created class. The code in lines 8 through 18 is similar to the application's main activity. Note that in line 14, the code calls the `setContentView()` function to set the layout for `Activity`, where the parameter is the prefix name of the new layout XML file created in the first step.

4. Edit the code for the trigger (caller) activity. The trigger activity is the main activity of the application. The source code is `MainActivity.java`, and the layout file is `activity_main.xml`. The steps for editing are as follows:

   a. Edit the layout file, delete the original `TextView` widgets, and add a button. Set its ID property to `@+id/goTONoParamNewAct` and its `Text` property to "Change to interface without Parameter," as shown in Figure 3-8.



Properties

Id	@+id/goTONoParamNewAct
Layout ...	
Style	buttonStyle
Text	Change to interface without Parameter
Hint	
Content...	
TextView	
View	
Depreca...	(Deprecated Properties)

Figure 3-8. Layout configuration for the trigger activity

b. Edit the source code file of the trigger activity (in this case, `MainActivity.java`) as follows (bold text is either added or modified):

```
Line #        Source Code
1  package com.example.guiexam;
2  import android.os.Bundle;
3  import android.app.Activity;
4  import android.view.Menu;
5  import android.content.Intent;           // Use Intent class
6  import android.widget.Button;            // Use Button class
7  import android.view.View.OnClickListener;
8  import android.view.View;

9  public class MainActivity extends Activity {
10     @Override
11     public void onCreate(Bundle savedInstanceState) {
12         super.onCreate(savedInstanceState);
13         setContentView(R.layout.activity_main);
14     Button btn = (Button) findViewById(R.id.goTONoParamNewAct);
15     btn.setOnClickListener(new /*View.*/OnClickListener(){
16         public void onClick(View v) {
17             Intent intent = new Intent(MainActivity.this,
                                    TheNoParameterOtherActivity.class);
18             startActivity(intent);
19         }
20     });
21     }
22     @Override
23     public boolean onCreateOptionsMenu(Menu menu) {
24         getMenuInflater().inflate(R.menu.activity_main, menu);
25         return true;
26     }
27 }
```

The code in line 17 defines an intent. The constructor function prototype in this case is

```
Intent(Context packageContext, Class<?> cls)
```

The first parameter is the trigger activity, in this case the main activity; `this`, because it is used inside the inner classes, is preceded by class-name modifiers. The second parameter is the class of the callee (being triggered) activity. It uses the `.class` attribute to construct its object (all Java classes have the `.class` attribute).

Line 18 calls `startActivity`, which runs the intent. This function does not pass any parameters to the triggered activity. The function prototype is

```
void Activity.startActivity(Intent intent)
```

5. Edit the `AndroidManifest.xml` file. Add descriptive information for the callee activity (bold text is added) to register the new `Activity` class:

```
Line #        Source Code
1  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
2      package="com.example.guiexam"
3      android:versionCode="1"
4      android:versionName="1.0" >
...      ... ...
10     <application
11         android:icon="@drawable/ic_launcher"
12         android:label="@string/app_name"
13         android:theme="@style/AppTheme" >
14         <activity
15             android:name=".MainActivity"
16             android:label="@string/title_activity_main" >
17             <intent-filter>
18                 <action android:name="android.intent.action.MAIN" />
19
20                 <category android:name="android.intent.category.LAUNCHER" />
21             </intent-filter>
22         </activity>
23         <activity android:name=".TheNoParameterOtherActivity"
            android:label="the other Activity"/>
24     </application>
25
26 </manifest>
```

You can also replace this XML code with the following methods:

- Method 1:

  ```
  <activity android:name="TheNoParameterOtherActivity"
  android:label=" the other Activity"> </activity>
  ```

- Method 2:

  ```
  <activity android:name=".TheNoParameterOtherActivity " />
  ```

- Method 3:

  ```
  <activity android:name=".TheNoParameterOtherActivity">
  </activity>
  ```

The content of the `android: name` text field is the class name of the callee's activity: `TheNoParameterOtherActivity`.

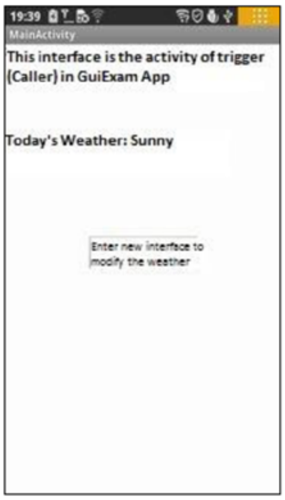
Note that if a period (`.`) is added before the name of the `Activity` class `android: name`, the compiler will give you the following warning at this line in the XML file (only a warning, not a compile error):

```
Exported activity does not require permission
```

Triggering Explicit Matching of an Activity with Parameters of Different Applications

The previous sections introduced triggering another activity without parameters in the same application. The activity of the trigger is that the callee allows the exchange of parameters: the trigger can specify certain parameters to the callee, and the callee can return those parameter values to the trigger on exit. Additionally, the callee and the trigger can be in completely different applications. This section shows an example of an application with parameters triggered by an activity in a different application. This example will help you understand the exchange mechanism for the activity's parameters.

Use the same `GuiExam` application from Chapter 2. The interface is shown in Figure 3-9.


19:39

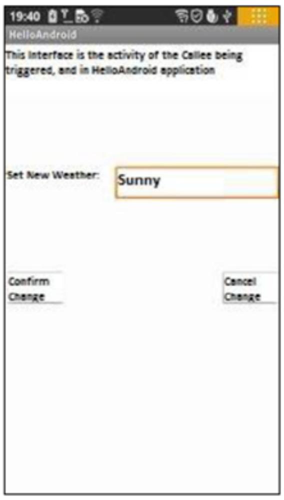
MainActivity

This interface is the activity of trigger (Caller) in GuiExam App

Today's Weather: Sunny

Enter new interface to modify the weather

(a) Interface when the GuiExam application starts


19:40

HelloAndroid

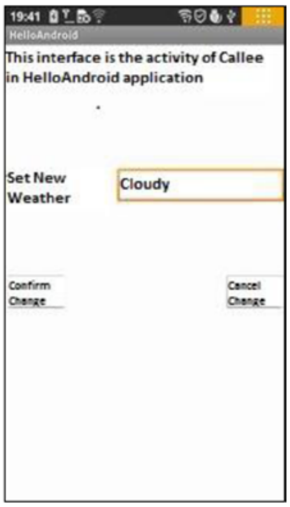
This interface is the activity of the Callee being triggered, and in HelloAndroid application

Set New Weather: Sunny

Confirm Change

Cancel Change

(b) Interface after clicking Enter New Interface To Modify the Weather


19:41

HelloAndroid

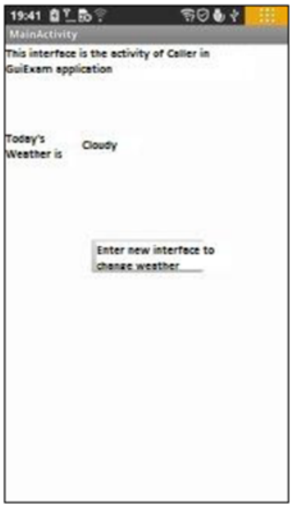
This interface is the activity of Callee in HelloAndroid application

Set New Weather Cloudy

Confirm Change

Cancel Change

(c) Entering a new value in the Set New Weather text box


19:41

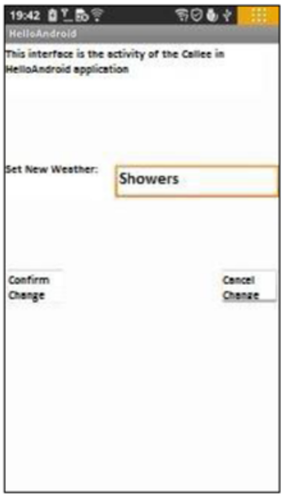
MainActivity

This interface is the activity of Caller in GuiExam application

Today's Weather is Cloudy

Enter new interface to change weather

(d) Interface after clicking Confirm Change


19:42

HelloAndroid

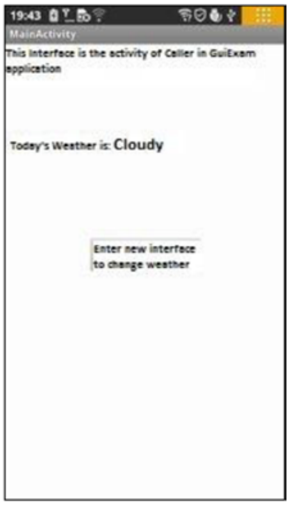
This interface is the activity of the Callee in HelloAndroid application

Set New Weather: Showers

Confirm Change

Cancel Change

(e) Interface after clicking Enter New Interface To Modify The Weather and entering a new value in the Set New Weather text box


19:43

MainActivity

This interface is the activity of Caller in GuiExam application

Today's Weather is: Cloudy

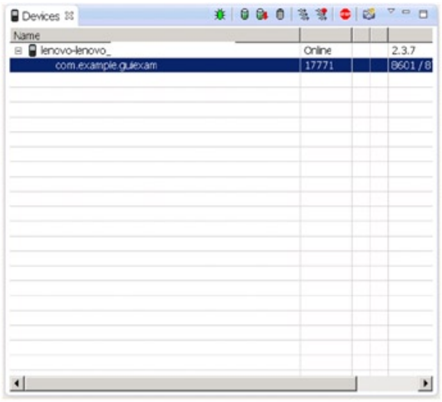
Enter new interface to change weather

(f) Interface after clicking Cancel Change

Figure 3-9. *The interface of multiple activities in different applications*

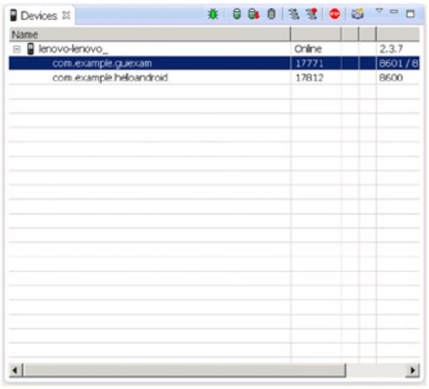
As shown in Figure 3-9, the trigger activity is in the `GuiExam` application, where there is a variable to accept the weather condition entry. The interface in Figure 3-9(a) displays when the `GuiExam` application is opened. Click the Enter New Interface To Modify The Weather box to trigger the activity in `HelloAndroid`. When this activity starts, it displays the new weather condition passed in the Set New Weather text box, as shown in Figure 3-9(b). Now enter a new weather condition value in the Set New Weather, and click OK Change to close the trigger's activity. The new value returned from Set New Weather refreshes the `Weather` variable in the trigger's activity, as shown in Figure 3-9(d). If you click Cancel Change, it does the same thing and closes the activity, but the value `Weather` does not change, as shown in Figure 3-9(f).

The process list for the executing application is shown in Figure 3-10 (displayed in the DDMS window of the host machine in Eclipse).



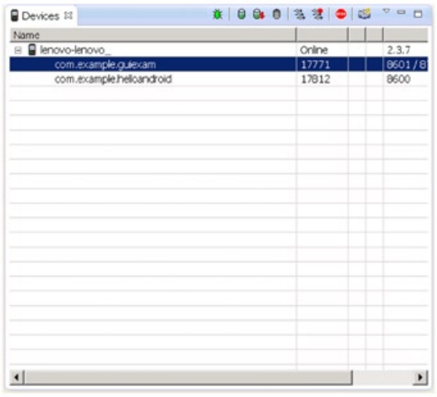
Name			
lenovo-lenovo_	Online		2.3.7
com.example.guiexam	17771		8601 / 8

(a) When the **GuiExam** application starts



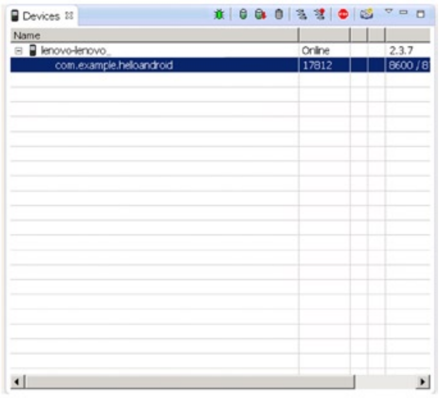
Name			
lenovo-lenovo_	Online		2.3.7
com.example.guiexam	17771		8601 / 8
com.example.helloandroid	17812		8600

(b) After clicking Enter New Interface To Modify The Weather



Name			
lenovo-lenovo_	Online		2.3.7
com.example.guiexam	17771		8601 / 8
com.example.helloandroid	17812		8600

(c) After clicking Confirm Change or Cancel Change



Name			
lenovo-lenovo_	Online		2.3.7
com.example.helloandroid	17812		8600 / 8

(d) After the **GuiExam** application exits

Figure 3-10. *Process list in DDMS for the multiple-activity application*

Figure 3-10 shows that when the application starts, only the process for the trigger, `GuiExam`, is running. But when you click Enter New Interface To Modify The Weather, the new activity is triggered and the process for the new activity `HelloAndroid` runs, as shown in Figure 3-10(b). When you click Confirm Change or Cancel Change, the triggered activity turns off, but the `HelloAndroid` process does not quit, as shown in Figure 3-10(c). Interestingly, even though the `GuiExam` trigger process exits, the `HelloAndroid` process to which the triggered activity belongs is still in the running state.

The build steps are as follows:

1. Modify the `GuiExam` code of the trigger application:

   a. Edit the main layout file (`activity_main.xml` in this case) by deleting the original `TextView` widgets; then add three new `TextView` widgets and a button. Set their properties as follows: set the `Text` property for two `TextViews` to "This interface is the activity of the Caller in GuiExam application" and "Today's Weather:". Set the third `TextView`'s `ID` property to `@+id/weatherInfo`. The `Text` property of the button is "Enter New Interface to Change Weather", and its `ID` attribute is `@+id/modifyWeather`. Adjust the size and position of each widget as shown in Figure 3-11.

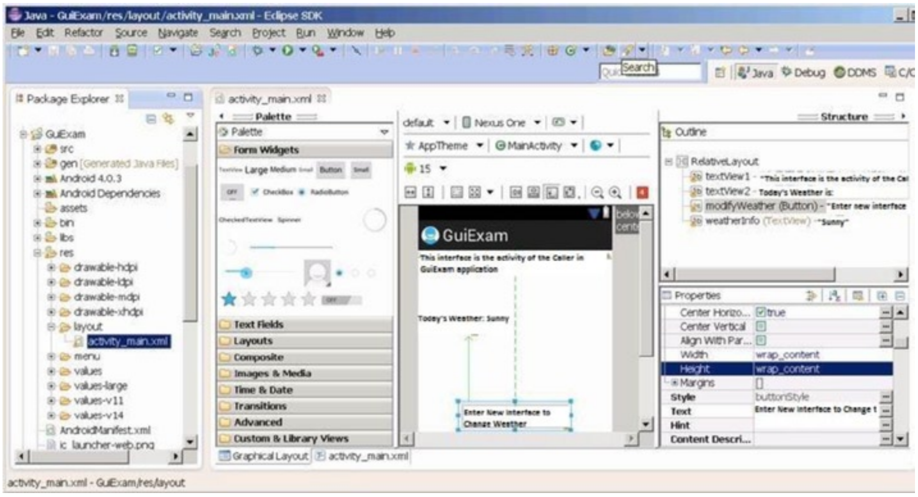


Figure 3-11. The main layout design for the `GuiExam` trigger application

b. Modify the content of `MainActivity.java` as shown here:

```
Line#        Source Code
package com.example.guiexam;
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.widget.Button;            // Use Button class
import android.view.View;                // Use View class
import android.view.View.OnClickListener; // Use View.OnClickListener class
import android.widget.TextView;          // Use TextView class
import android.content.Intent;           // Use Intentclass

public class MainActivity extends Activity {
    public static final String INITWEATHER = "Sunny; // /Initial Weather
    public static final int MYREQUESTCODE =100;
//Request Code of triggered Activity
    private TextView tv_weather;
// The TextView Widget that displays Weather info
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    tv_weather = (TextView)findViewById(R.id.weatherInfo);
    tv_weather.setText(INITWEATHER);
    Button btn = (Button) findViewById(R.id.modifyWeather);
//Get Button object according to resource ID #
    btn.setOnClickListener(new /*View.*/OnClickListener(){
//Set responding code click event
       public void onClick(View v) {
           Intent intent = new Intent();
           intent.setClassName("com.example.helloandroid",
// the package ( application) that the triggered Activity is located
          "com.example.helloandroid.TheWithParameterOtherActivity");
//triggered class ( full name)
             String wthr = tv_weather.getText().toString();
// Acquire the value of weather TextView
             intent.putExtra("weather",wthr); // Set parameter being
                                                passed to Activity
                startActivityForResult(intent, MYREQUESTCODE);
//Trigger Activity
            }
        });
    }
39
40      @Override
41      protected void onActivityResult(int requestCode, int resultCode,
                                        Intent data) {
42 //Triggered Activity finish return
43          super.onActivityResult(requestCode, resultCode, data);
44          if (requestCode == MYREQUESTCODE) {
45 // Determine whether the specified Activity end of the run
              if (resultCode == RESULT_CANCELED)
46                  {       }
47 // Select "Cancel" to exit the code, this case is empty
48              else if (resultCode == RESULT_OK) {
49 // Select <OK> to exit code
50                  String wthr = null;
51                  wthr = data.getStringExtra("weather");
   // Get return value
                    if (wthr != null)
                        tv_weather.setText(wthr);
   // Update TextView display of weather content
                }
            }
        }

        @Override
        public boolean onCreateOptionsMenu(Menu menu) {
            getMenuInflater().inflate(R.menu.activity_main, menu);
           return true;
        }
   }
```

The code in lines 23–28 triggers the activity with parameters in other applications. Lines 23–25 establish the trigger intent, which uses the `Intent.setClassName()` function. The prototype is

```
Intent Intent.setClassName(String packageName, String className);
```

The first parameter is the name of the package where the triggered activity is located, and the second parameter is the class name (required to use the full name) of the triggered activity. By using the `startActivity ...` function to trigger the activity, the system can accurately locate the application and activity classes.

Line 28 attaches the parameter as additional data to the intent. `Intent` has a series of `putExtra` functions to attach additional data and another series of `getXXXExtra` functions to extract data from the intent. Additional data can also be assembled by the `Bundle` class. `Intent` provides a `putExtras` function to add data and a `getExtras` function to get the data. `putExtra` uses a *property-value* data pairing or *variable name-value* data pairing to add and retrieve data. In this example, `Intent.putExtra("weather", "XXX")` saves the data pair consisting of the name of the `weather` variable and the value “XXX” as additional data for the intent.

The code line with `Intent.getStringExtra("weather")` gets the value of the `weather` variable from the attached intent data and returns the string type.

More details about these functions and the `Bundle` class can be found in the documentation on the Android web site. They are not discussed any further here.

In lines 33–46, you rewrite the `onActivityResult` function of the `Activity` class. This function is called when the triggered activity is closed. In line 36, you first determine which activity is closed and returned according to the request code. Then you judge whether it is returned by an OK or a Cancel click, based on the result code and the request code. Lines 40–50 get the negotiated variable values from the returned intent. Line 42 updates the interface based on the return value of the variable. In this function, if the user clicks Cancel to return, you do nothing.

2. Modify the code of the callee application `HelloAndroid` as shown in Figure 3-12:

    a. Using the method described in the section "Triggering Explicit Matching of an Activity with Parameters of Different Applications earlier in this chapter, add a layout file (in this case named `param_otheract.xml`), and drag and drop a `RelativeLayout` layout into the file.

    b. Edit this layout file by adding two `TextView` widgets, an `EditText`, and two `Button` widgets. Set their properties as follows:

        - `Text` property for the two `TextView` widgets: "This interface is the activity of the caller in HelloAndroid application" and "Set new weather as:"
        - `ID` property for the `EditText`: `@+id/editText_NewWeather`
        - `Text` property for the two `Buttons`: "Confirm Changes" and "Cancel Changes"
        - `ID` attribute for the two `Buttons`: `@+id/button_Modify` and `@+id/button_Cancel`

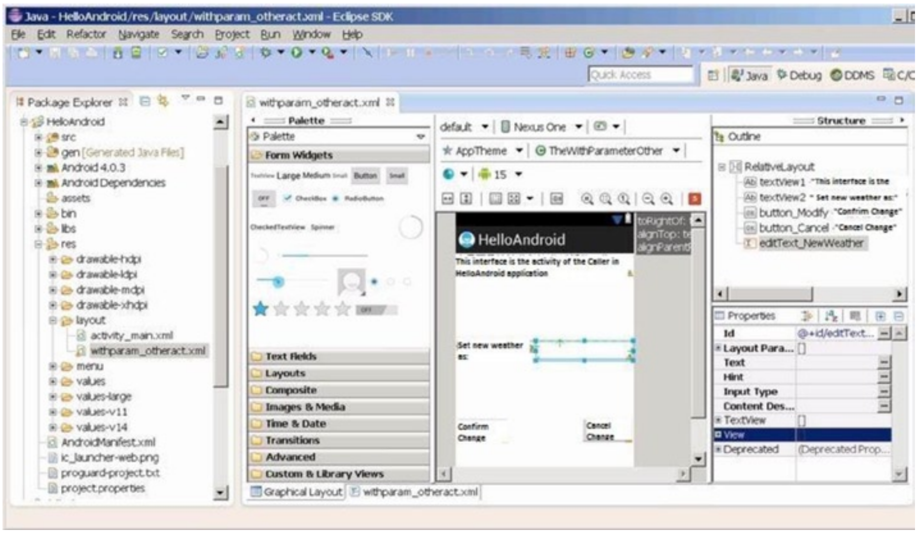
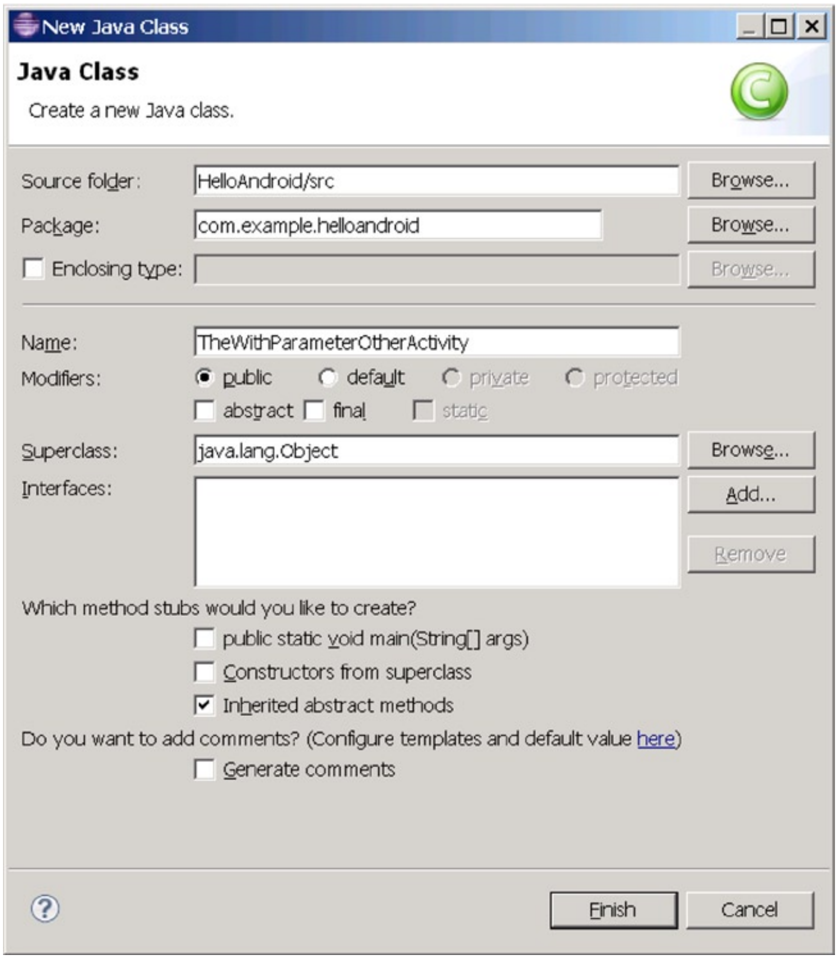


Figure 3-12. *New layout design of the triggered (callee) application* `HelloAndroid`

Then adjust their size and position.

c. As described in the section “Triggering Explicit Matching of an Activity with Parameters of Different Applications,” add the corresponding class (in this case, `TheWithParameterOtherActivity`) for the new layout file, as shown in Figure 3-13.



New Java Class

Java Class

Create a new Java class.

Source folder: HelloAndroid/src

Package: com.example.helloandroid

[ ] Enclosing type:

Name: TheWithParameterOtherActivity

Modifiers: [✓ public] default private protected

[ ] abstract [ ] final [ ] static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

[ ] public static void main(String[] args)

[ ] Constructors from superclass

[✓] Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

[ ] Generate comments

***Figure 3-13.** Add the corresponding class for the newly added layout file in the `HelloAndroid` project*

d. Edit the class file for the newly added layout file (in this example, `TheWithParameterOtherActivity.java`). The content is as follows:

```
Line#        Source Code
1  package com.example.helloandroid;
2  import android.os.Bundle;                // Use Bundle Class
3  import android.app.Activity;             // Use Activity Class
4  import android.content.Intent;           // Use Intent Class
5  import android.widget.Button;            // Use Button Class
6  import android.view.View;                // Use View Class
```

```
import android.view.View.OnClickListener;    // Use OnClickListener Class
import android.widget.EditText;              // Use  EditText Class

public class TheWithParameterOtherActivity extends Activity {
    private String m_weather;
// Save new weather variable
    @Override
    protected void onCreate(Bundle savedInstanceState) {
// Define onCreate method
        super.onCreate(savedInstanceState);
// method of call onCreate Super Class
        setContentView(R.layout.withparam_otheract); // Set layout file
        Intent intent = getIntent();
// Get Intent of triggering this Activity
        m_weather = intent.getStringExtra("weather");
// Get extra data from Intent
        final EditText et_weather = (EditText)
                       findViewById(R.id.editText_NewWeather);
        et_weather.setText(m_weather,null);
// Set initial value of "New Weather" EditText according to extra data of
   the Intent
        Button btn_modify = (Button) findViewById(R.id.button_Modify);
        btn_modify.setOnClickListener(new /*View.*/OnClickListener(){
// Set corresponding code of <Confirm Change>
            public void onClick(View v) {
                Intent intent = new Intent();
// Create and return the Intent of Data storage
                String wthr = et_weather.getText().toString();
// Get new weather value from EditText
                intent.putExtra("weather",wthr);
// Put new weather value to return Intent
                setResult(RESULT_OK, intent);
// Set <Confirm> and return data
                finish(); // Close Activity
            }
        });
        Button btn_cancel = (Button) findViewById(R.id.button_Cancel);
        btn_cancel.setOnClickListener(new /*View.*/OnClickListener(){
// Set corresponding code for <Cancel Change>
            public void onClick(View v) {
                setResult(RESULT_CANCELED, null);
// Set return value for <Cancel>
                finish(); // Close this Activity
           }
        });
    }
}
```

This code follows the framework of an activity. It sets the activity layout in line 11 such that the layout name is the same as the layout file name created in step 1 (no extension). In lines 19–22, it first constructs an intent for the return and then adds extra data to the `Intent` object as the return data. In line 21, it sets the return value of the activity and the intent as a return data carrier. The prototype of the `setResult` function is

```
final void Activity.setResult(int resultCode, Intent data);
```

If `resultCode` is `RESULT_OK`, the user has clicked OK to return; and if it is `RESULT_CANCELLED`, the user has clicked Cancel to return. In this condition, the return data carrier intent can be null, which is set in line 27.

3. Modify `AndroidManifest.xml`, which is triggered by the application, with the following code:

```
Line #        Source Code
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.helloandroid"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="15" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/title_activity_main" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name="TheWithParameterOtherActivity">
            <intent-filter>
                <action android:name="android.intent.action.DEFAULT" />
               </intent-filter>
        </activity>
    </application>

</manifest>
```

4. Lines 24–29 are new. As in previous sections, you add an additional activity description and specify its class name, which is the class name of the triggered activity generated in the second step. See the section "Triggering an Explicit Match of Activities with No Parameters" for information about modifying the `AndroidManifest.xml` file. Unlike in that section, you add not only an activity and the documentation of its `name` attribute, but also the intent-filter instructions and state to accept the default actions described in the `Action` element and trigger this `Activity` class. The activity cannot be activated in the absence of the intent-filter description of the activity.

5. Run the callee application to register components of the activity. The modifications to `AndroidManifest.xml` file are not registered to the Android system until the callee application, `HelloAndroid`, is executed once. Thus this is an essential step to complete the registration of its included activity.

Implicit Matching of Built-In Activities

In the examples in the previous two sections, before you trigger the activity of the same application or different applications through the `Activity.startActivity()` function or the `Activity.startActivityForResult()` function, the constructor of the `Intent` objects explicitly specifies the class, either through the `.class` attribute or through the class name in a string. This way, the system can find the class to be triggered. This approach is called *explicit intent matching*. The next example shows how to trigger a class that is not specified. Instead, the system figures it out using an approach called *implicit intent matching*.

In addition, Android has a number of activities that have already been implemented, such as dialing, sending text messages, and so on. Examples in this section explain how you use can implicit matching to trigger these built-in activities. The application interface is shown in Figure 3-14.

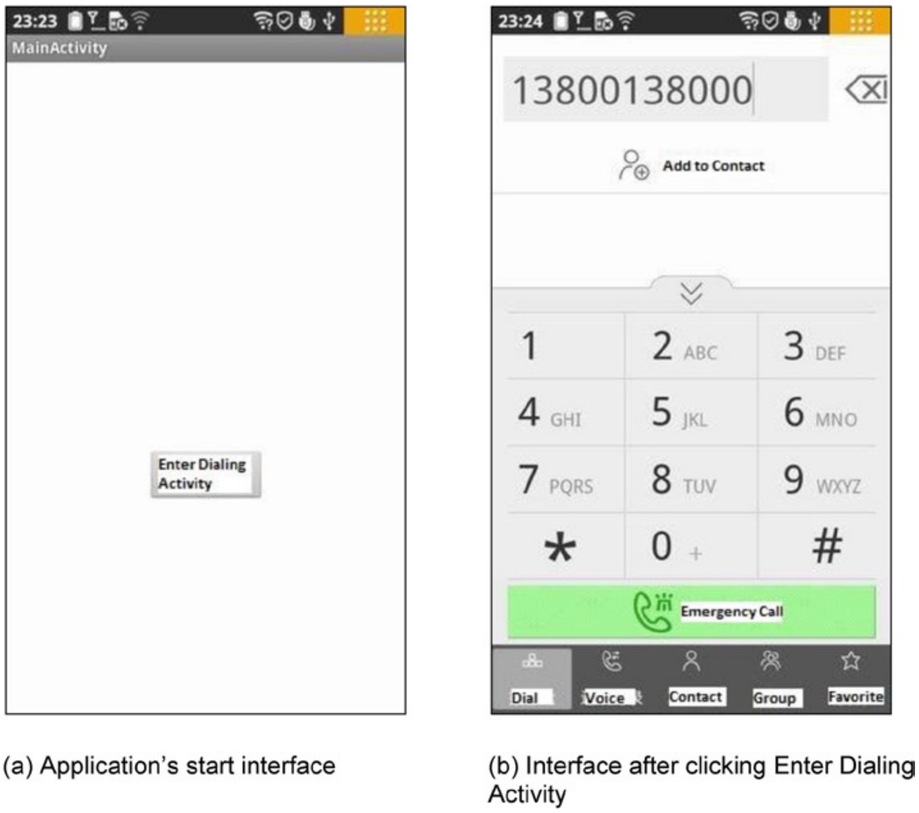
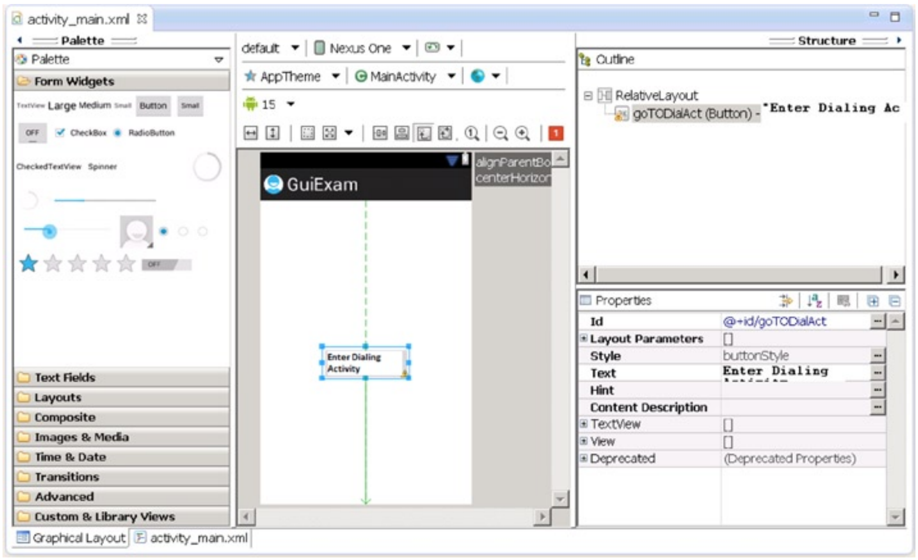


Figure 3-14. *The application interface when using implicit intent to trigger a built-in activity*

The user start the `GuiExam` application and clicks the Enter Dialing Activity button on the screen. It triggers dial-up activities that come with the system.

In this case, you modify the `GuiExam` project and use this application as a trigger. The implicit match triggered activity is the dial-up activity. The steps to build this example are as follows.

1. In the layout file (`activity_main.xml`) of the `GuiExam` application, delete the original `TextView` widgets, add a button, and set its `ID` attribute to `@+id/goTODialAct` and its `Text` property to "Enter Dialing Activity". Adjust its size and position as shown in Figure 3-15.



Properties

Id	@+id/goTODialAct
Layout Parameters	
Style	buttonStyle
Text	Enter Dialing
Hint	
Content Description	
TextView	
View	
Deprecated	(Deprecated Properties)

Figure 3-15. *Layout file of the application for the implicit match built-in activity*

2. Modify the source code file (MainActivity.java) as follows:

```
Line#        Source Code
1  package com.example.guiexam;
2  import android.os.Bundle;
3  import android.app.Activity;
4  import android.view.Menu;
5  import android.widget.Button;            // Use Button Class
6  import android.view.View;                // Use View Class
7  import android.view.View.OnClickListener; // Use View.OnClickListener Class
8  import android.content.Intent;           // Use Intent Class
9  import android.net.Uri;                  // Use URI Class

10 public class MainActivity extends Activity {
11     @Override
12     public void onCreate(Bundle savedInstanceState) {
13         super.onCreate(savedInstanceState);
14         setContentView(R.layout.activity_main);
15         Button btn = (Button) findViewById(R.id.goTODialAct);
16         btn.setOnClickListener(new /*View.*/OnClickListener(){
17 // Set corresponding Code for Click Activity
18             public void onClick(View v) {
19                 Intent intent = new Intent(Intent.ACTION_DIAL,
                   Uri.parse("tel:13800138000"));
20                  startActivity(intent); // Trigger corresponding Activity
21              }
22          });
        }
23
24      @Override
25      public boolean onCreateOptionsMenu(Menu menu) {
26          getMenuInflater().inflate(R.menu.activity_main, menu);
27          return true;
28      }
    }
```

The code in line 16 defines an *indirect intent* (that is, intent of implicit match. It is called an indirect intent because the class that needs to be triggered is not specified in the constructor of the object; the constructor only describes the function of the class that needs to be triggered to complete dialing. The constructor functions for the indirect intent are as follows:

```
Intent(String action)
Intent(String action, Uri uri)
```

These functions require the classes (activities) that can complete the specified action when they are called. The only difference between the two is that the second function also comes with data.

This example uses the second constructor, which requires the activity that can complete the dialing and extra data as a string of phone numbers. Because the application does not specify the trigger type, Android finds the class to handle this action (for example, `Activity`) from the registered class list and triggers the start of the event.

If multiple classes can handle the specified action, Android pops up a selection menu, and users can select which one to run.

The parameter `action` can use the system-predefined string. In the previous example, `Intent.ACTION_DIAL` is the string constant of `ACTION_DIAL`, which is defined by the `Intent` class. Some system-predefined `ACTION` examples are shown in Table 3-1.

Table 3-1. *Some System-Predefined ACTION Constants*

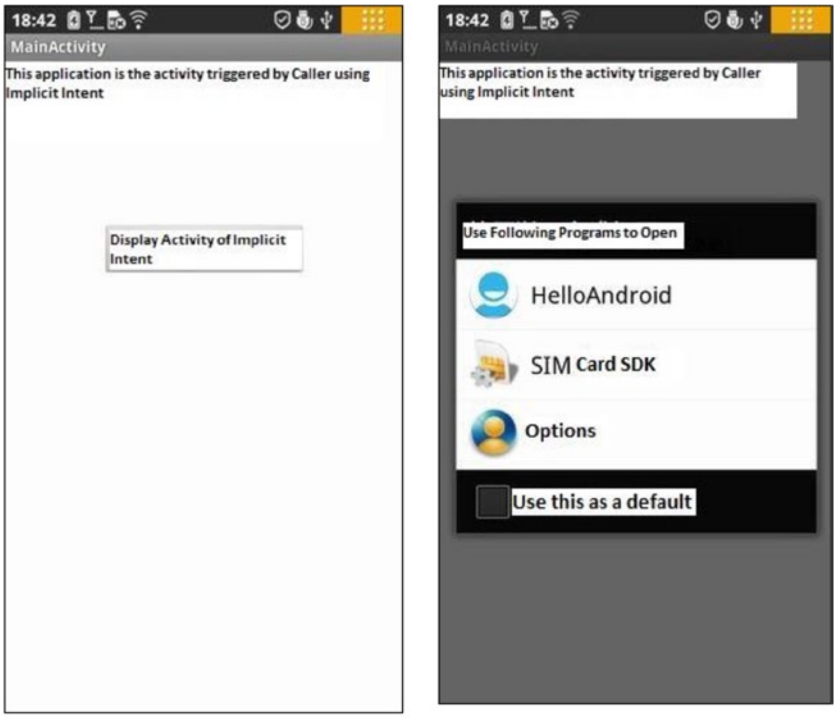
ACTION Constant Name	Value	Description
`ACTION_MAIN`	`android.intent.action.MAIN`	Start up as the initial activity of a task with no data input and no returned output.
`ACTION_VIEW`	`android.intent.action.VIEW`	Display the data in the intent URI.
`ACTION_EDIT`	`android.intent.action.EDIT`	Request an activity to edit data.
`ACTION_DIAL`	`android.intent.action.DIAL`	Start a phone dialer, and use preset numbers in the data to dial.
`ACTION_CALL`	`android.intent.action.CALL`	Initiate a phone call, and immediately use the number in the data URI to initiate a call.
`ACTION_SEND`	`android.intent.action.SEND`	Start an activity to send specific data (the recipient is selected by activity resolution).
`ACTION_SENDTO`	`android.intent.action.SENDTO`	Generally, start an activity to send a message to a contact designated in the URI.
`ACTION_ANSWER`	`android.intent.action.ANSWER`	Open an activity to process an incoming call. Currently it is handled by a local phone-dialing tool.
`ACTION_INSERT`	`android.intent.action.INSERT`	Open an activity that can insert a new project at the addition cursor in a specific data field. When it is called as the child activity, it must return the URI of the newly inserted project.
`ACTION_DELETE`	`android.intent.action.DELETE`	Start an activity to delete a data port at the URI position.
`ACTION_WEB_SEARCH`	`android.intent.action.WEB_SEARCH`	Open an activity, and run a web page search based on the text in the URI data.

The `ACTION` constant name is the first parameter used in the constructor of the implicit-match intent. The value of the `ACTION` constant, used in the `AndroidManifest.xml` statement of the activity that receives this action, is not used in this section, but is used in the next section. You can find more information about predefined `ACTION` values in the `android.content.Intent` help documentation.

Implicit Match that Uses a Custom Activity

The previous example used implicit matching to trigger activities that come with the Android system. In this section, you see an example of how to use an implicit match to trigger a custom activity.

The configuration of this example application is similar to the one in the section "Triggering Explicit Matching of an Activity with Parameters of Different Applications." The triggering application is hosted in the `GuiExam` project, and the custom activity triggered by implicit match is in the `HelloAndroid` application. The interface is shown in Figure 3-16.



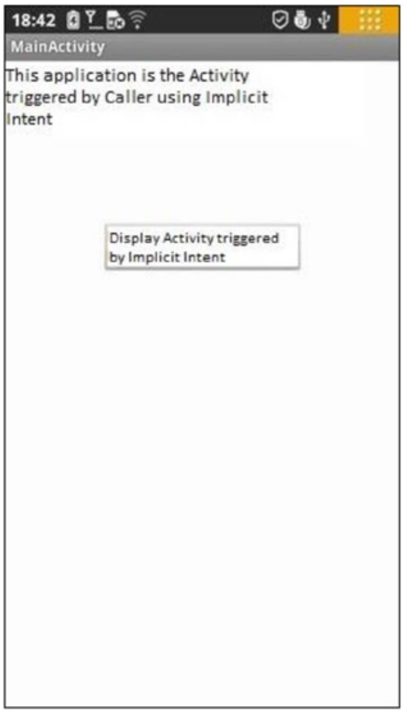
This application is the activity triggered by Caller using Implicit Intent

This application is the activity triggered by Caller using Implicit Intent

(a) Application's start interface

(b) Interface after clicking Display Activity Of Implicit Intent

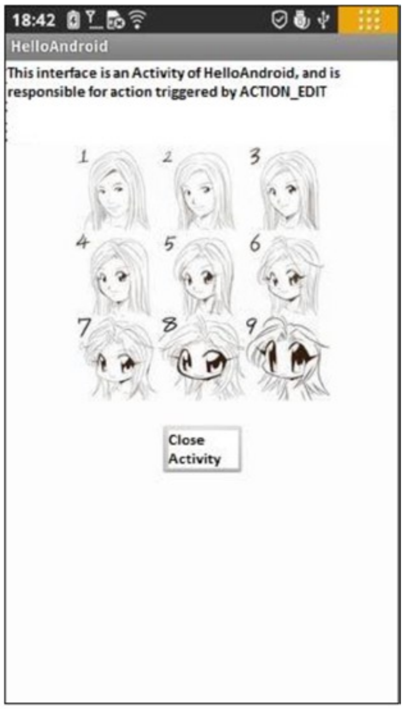
Figure 3-16. *The interface of implicit match that uses a custom activity*


18:42

MainActivity

This application is the Activity triggered by Caller using Implicit Intent

c) Interface after selecting HelloAndroid


18:42

HelloAndroid

This interface is an Activity of HelloAndroid, and is responsible for action triggered by ACTION_EDIT

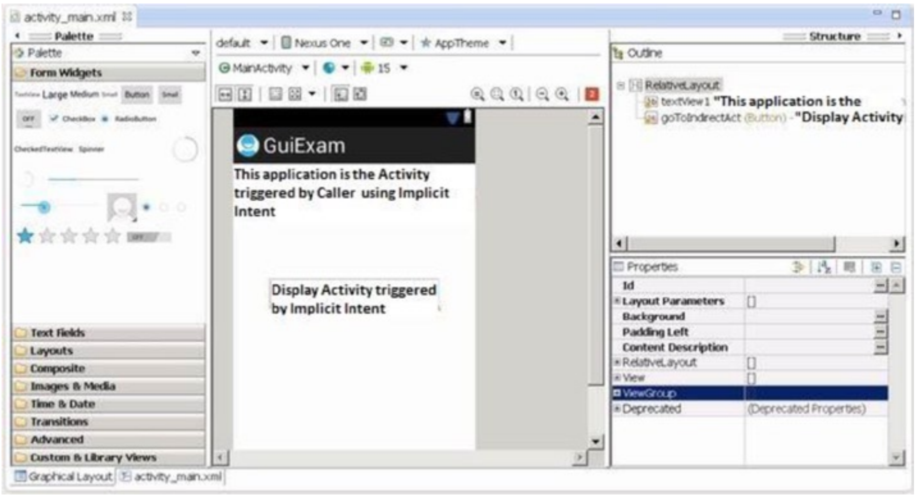
(d) Interface after clicking Close Activity

Figure 3-16. (*continued*)

Figure 3-16(a) shows the interface when the GuiExam trigger application starts. When you click the Display Activity Of Implicit Intent button, the system finds qualified candidates for activities according to the requirements of the ACTION_EDIT action and displays a list of events of these candidates (b). When the user-defined HelloAndroid application is selected, the activity that can receive the ACTION_EDIT action as claimed in the intent-filter in HelloAndroid application is displayed (c). When you click the Close Activity button, the application returns to the original GuiExam activity interface (d).

Like the previous ones, this example is based on modifying the GuiExam project. The steps are as follows:

1. Edit the main layout file (activity_main.xml). Delete the original TextView widgets, and then add a TextView and a button. Set the TextView's Text property to "This application is the Activity triggered by Caller using Implicit Intent". Set the button's Text property to "Display Activity triggered by Implicit Intent" and its ID attribute to @+id/goToIndirectAct, as shown in Figure 3-17.



RelativeLayout
textView1 "This application is the
goToIndirectAct (Button) - "Display Activity

GuiExam

This application is the Activity triggered by Caller using Implicit Intent

Display Activity triggered by Implicit Intent

Figure 3-17. The main layout design for the `GuiExam` *trigger application*

2. Edit `MainActivity.java` as follows:

```
Line#         Source Code
1  package com.example.guiexam;
2  import android.os.Bundle;
3  import android.app.Activity;
4  import android.view.Menu;
5  import android.widget.Button;            // Use Button Class
6  import android.view.View;                // Use View class
7  import android.view.View.OnClickListener; // Use View.OnClickListener class
8  import android.content.Intent;           // Use Intent Class

9  public class MainActivity extends Activity {
10     @Override
11     public void onCreate(Bundle savedInstanceState) {
12         super.onCreate(savedInstanceState);
13         setContentView(R.layout.activity_main);
14         Button btn = (Button) findViewById(R.id.goToIndirectAct);
15         btn.setOnClickListener(new /*View.*/OnClickListener(){
16 // Set respond Code for Button Click event
17             public void onClick(View v) {
18                 Intent intent = new Intent(Intent.ACTION_EDIT);
19 //Construct implicit Inent
20                 startActivity(intent); // Trigger Activity
21             }
         });
22     }
23
24      @Override
25      public boolean onCreateOptionsMenu(Menu menu) {
26          getMenuInflater().inflate(R.menu.activity_main, menu);
27          return true;
        }
    }
```

The code in lines 16 and 17 defines the implicit intent and triggers the corresponding activity, which is basically the same as the earlier code that triggers implicit activity, but here it uses the constructor function of the intent that has no data.

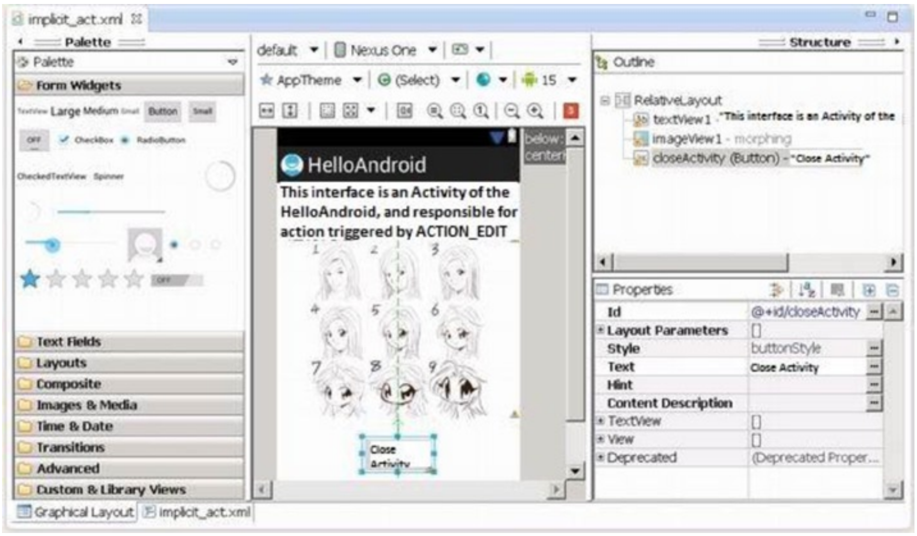
3. Modify the HelloAndroid application that includes a custom activity with the corresponding implicit intent:

    a. Based on the method described in the section "Triggering an Explicit Match of Activities with No Parameters," earlier in this chapter, add a layout file (implicit_act.xml) to the project and drag and drop a RelativeLayout layout into the file.

    b. Edit the layout file, and add TextView, ImageView, and Button widgets. Set the attributes as follows:

        - Text property of the TextView: "This interface is an Activity of the HelloAndroid, which is responsible for action triggered by the ACTION_EDIT"
        - ImageView: Set up exactly as in the section "Using ImageView" in Chapter 2.
        - Text property of the Button: "Close Activity"
        - ID property of the Button: @+id/closeActivity.

Then adjust their respective size and position, as shown in Figure 3-18.



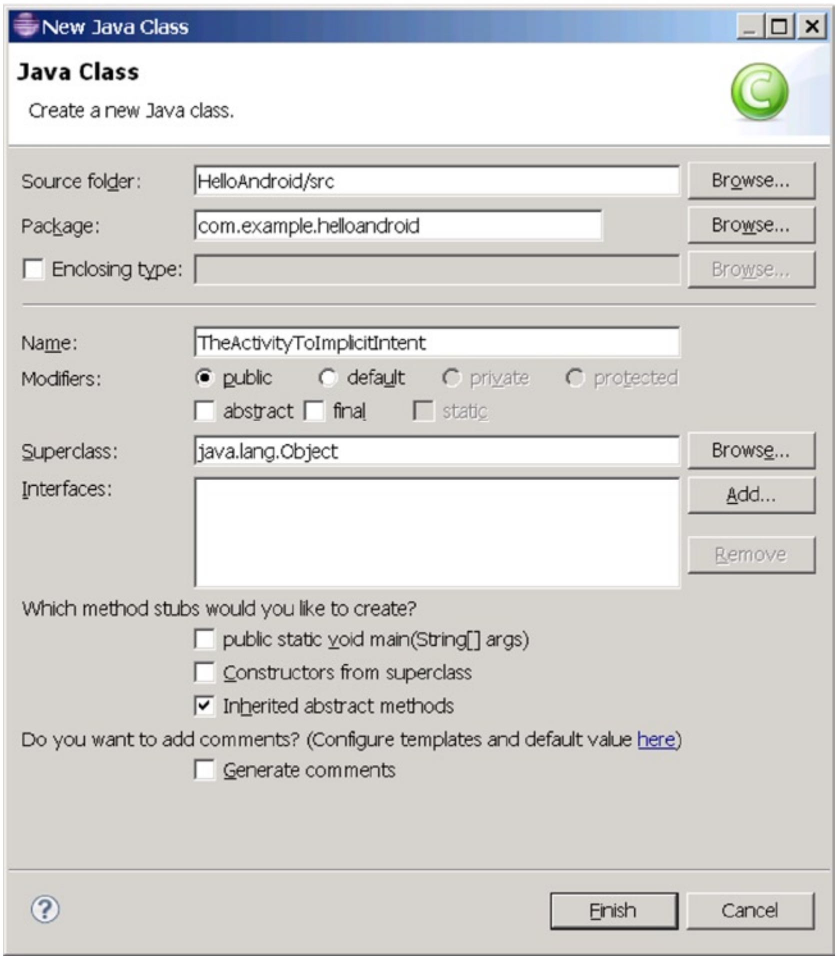
This interface is an Activity of the HelloAndroid, and responsible for action triggered by ACTION_EDIT

```
RelativeLayout
    textView1 - "This interface is an Activity of the
    imageView1 - morphing
    closeActivity (Button) - "Close Activity"
```

Id	@+id/closeActivity
Layout Parameters	[]
Style	buttonStyle
Text	Close Activity
Hint	
Content Description	
TextView	[]
View	[]
Deprecated	(Deprecated Proper...

Figure 3-18. *Layout file for the custom activity of the corresponding implicit intent*

4. Similar to the process described in the section of this chapter "Triggering an Explicit Match of Activities with No Parameters," add the corresponding class to the project for the new layout file (`TheActivityToImplicitIntent`), as shown in Figure 3-19.



New Java Class

Java Class

Create a new Java class.

Source folder: HelloAndroid/src [Browse...]

Package: com.example.helloandroid [Browse...]

☐ Enclosing type: [Browse...]

Name: TheActivityToImplicitIntent

Modifiers: [● public] ○ default ○ private ○ protected

☐ abstract ☐ final ☐ static

Superclass: java.lang.Object [Browse...]

Interfaces: [Add...] [Remove]

Which method stubs would you like to create?

☐ public static void main(String[] args)

☐ Constructors from superclass

☑ Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

☐ Generate comments

Figure 3-19. *New class for the custom activity of the corresponding implicit intent*

5. Edit the class file for the newly added layout file (`TheActivityToImplicitIntent.java`), which reads as follows:

```
Line#         Source Code
1  package com.example.helloandroid;
2  import android.os.Bundle;
3  import android.app.Activity;
4  import android.widget.Button;           // Use Button Class
5  import android.view.View;               // Use View class
6  import android.view.View.OnClickListener; // Use View.OnClickListener class
```

```
public class TheActivityToImplicitIntent extends Activity {
@Override
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.implicit_act);
        Button btn = (Button) findViewById(R.id.closeActivity);
        btn.setOnClickListener(new /*View.*/OnClickListener(){
// Set response code for <Close Activity> Click
            public void onClick(View v) {
                finish();
            }
        });
    }
}
```

6. Modify the AndroidManifest.xml file of the HelloAndroid custom application containing the corresponding implicit intent, as follows:

```
Line#          Source Code
<manifest xmlns:android="http://schemas.android.com/apk/res/android"

    package="com.example.helloandroid"

    android:versionCode="1"

    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="15" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/title_activity_main" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
```

```
26          <activity
27              android:name="TheActivityToImplicitIntent">
28              <intent-filter>
29                  <action android:name="android.intent.action.DEFAULT" />
30                  <action android:name="android.intent.action.EDIT" />
31                  <category android:name="android.intent.category.DEFAULT" />
32              </intent-filter>
33          </activity>
        </application>

    </manifest>
```

The code in lines 24–32 (in bold) gives the activity information for receiving the implicit intent. Line 26 specifies that you can receive an `android.intent.action.EDIT` action. This value corresponds to the constant value of the `ACTION` parameter `Intent.ACTION_EDIT` of the trigger's intent constructor function (the `MainActivity` class of `GuiExam`). This is a predetermined contact signal between the trigger and the callee. Line 27 further specifies that the default data type can also be received.

7. Run the application `HelloAndroid`, which now contains a custom activity for the corresponding implicit intent and registers its `AndroidManifest.xml` file in the system.

So far, three chapters have covered Android interface design. The simple `GuiExam` application has demonstrated the state transition of an activity, the `Context` class, intents, and the relationship between applications and activities. You also learned how to use a layout as an interface and how the button, event, and inner event listener work. Examples with multiple activities introduced the explicit and implicit trigger mechanisms for activities. You saw an example of an application with parameters triggered by an activity in a different application, and you now understand the exchange mechanism for the activity's parameters.

The application interface discussed so far is basically similar to a dialog interface. The drawback of this mode is that it is difficult to obtain accurate touchscreen input, making it difficult to display accurate images based on the input interface. The next chapter, which covers the last part of Android interface design, introduces the view-based interaction style interface. In this interface, you can enter information with accurate touchscreen input and display detailed images, as required by many game applications.