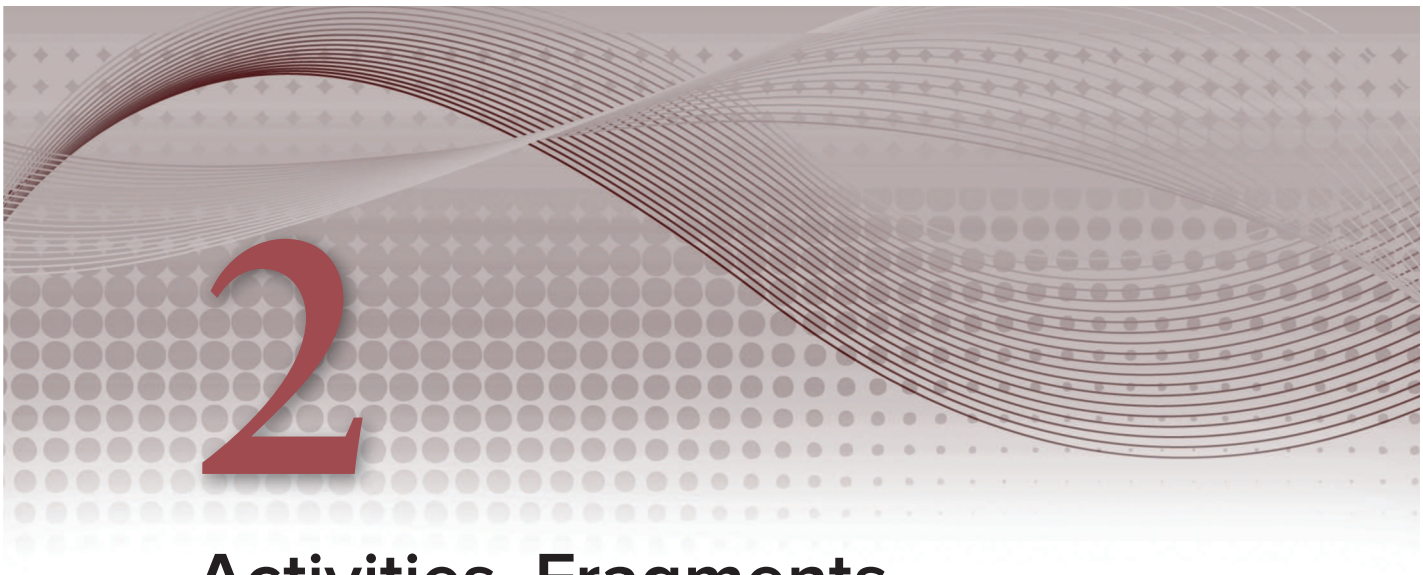

2

Activities, Fragments, and Intents

WHAT YOU WILL LEARN IN THIS CHAPTER

- The life cycles of an activity
- Using fragments to customize your UI
- Applying styles and themes to activities
- How to display activities as dialog windows
- Understanding the concept of intents
- Using the Intent object to link activities
- How intent filters help you selectively connect to other activities
- Displaying alerts to the user using notifications

In Chapter 1, you learned that an activity is a window that contains the user interface of your application. An application can have zero or more activities. Typically, applications have one or more activities; and the main purpose of an activity is to interact with the user. From the moment an activity appears on the screen to the moment it is hidden, it goes through a number of stages, known as an activity's *life cycle*. Understanding the life cycle of an activity is vital to ensuring that your application works correctly. In addition to activities, Android 4.0 also supports a feature that was introduced in Android 3.0 (for tablets): fragments. Think of fragments as "miniature" activities that can be grouped to form an activity. In this chapter, you will learn about how activities and fragments work together.

Apart from activities, another unique concept in Android is that of an *intent*. An intent is basically the "glue" that enables different activities from different applications to work together seamlessly, ensuring that tasks can be performed as though they all belong to one single application. Later in this chapter, you will learn more about this very important concept and how you can use it to call built-in applications such as the Browser, Phone, Maps, and more.

UNDERSTANDING ACTIVITIES

This chapter begins by looking at how to create an activity. To create an activity, you create a Java class that extends the `Activity` base class:

```
package net.learn2develop.Activity101;

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

public class Activity101Activity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

Your activity class loads its UI component using the XML file defined in your `res/layout` folder. In this example, you would load the UI from the `main.xml` file:

```
        setContentView(R.layout.main);
```

Every activity you have in your application must be declared in your `AndroidManifest.xml` file, like this:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.learn2develop.Activity101"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="14" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:label="@string/app_name"
            android:name=".Activity101Activity" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

The `Activity` base class defines a series of events that govern the life cycle of an activity. The `Activity` class defines the following events:

- `onCreate()` — Called when the activity is first created
- `onStart()` — Called when the activity becomes visible to the user
- `onResume()` — Called when the activity starts interacting with the user

- `onPause()` — Called when the current activity is being paused and the previous activity is being resumed
- `onStop()` — Called when the activity is no longer visible to the user
- `onDestroy()` — Called before the activity is destroyed by the system (either manually or by the system to conserve memory)
- `onRestart()` — Called when the activity has been stopped and is restarting again

By default, the activity created for you contains the `onCreate()` event. Within this event handler is the code that helps to display the UI elements of your screen.

Figure 2-1 shows the life cycle of an activity and the various stages it goes through — from when the activity is started until it ends.

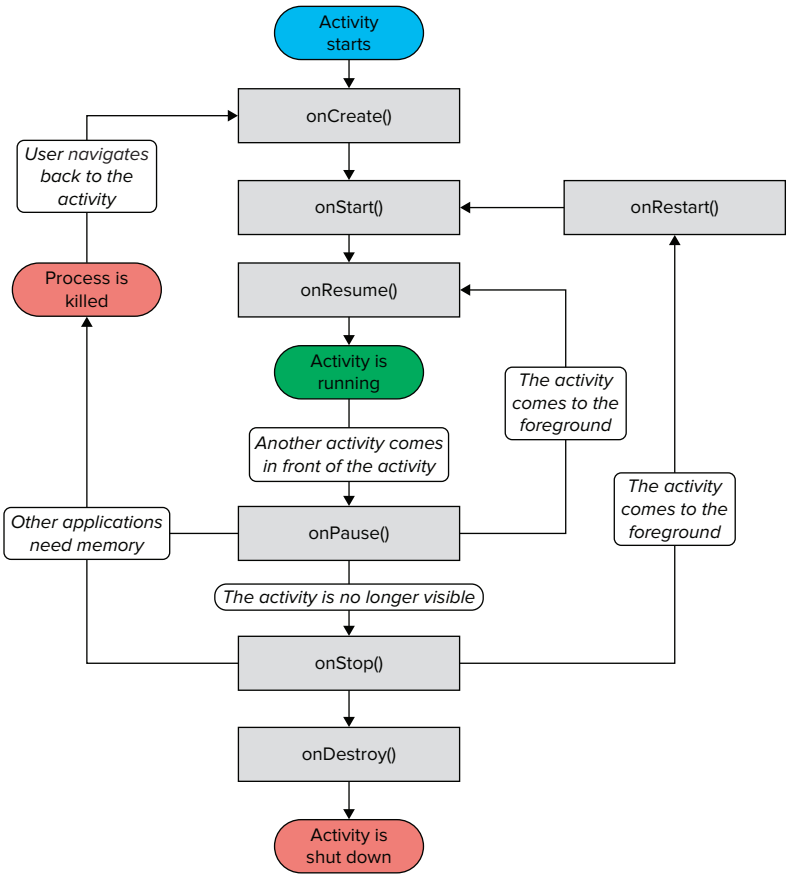

FIGURE 2-1

The best way to understand the various stages of an activity is to create a new project, implement the various events, and then subject the activity to various user interactions.

TRY IT OUT Understanding the Life Cycle of an Activity

codefile Activity101.zip available for download at Wrox.com

1. Using Eclipse, create a new Android project and name it **Activity101**.
2. In the `Activity101Activity.java` file, add the following statements in bold:

```
package net.learn2develop.Activity101;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;

public class Activity101Activity extends Activity {
    String tag = "Lifecycle";

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Log.d(tag, "In the onCreate() event");
    }

    public void onStart()
    {
        super.onStart();
        Log.d(tag, "In the onStart() event");
    }

    public void onRestart()
    {
        super.onRestart();
        Log.d(tag, "In the onRestart() event");
    }

    public void onResume()
    {
        super.onResume();
        Log.d(tag, "In the onResume() event");
    }

    public void onPause()
    {
        super.onPause();
        Log.d(tag, "In the onPause() event");
    }

    public void onStop()
```

```
    {
        super.onStop();
        Log.d(tag, "In the onStop() event");
    }

    public void onDestroy()
    {
        super.onDestroy();
        Log.d(tag, "In the onDestroy() event");
    }
}
```

3. Press F11 to debug the application on the Android emulator.

4. When the activity is first loaded, you should see something very similar to the following in the LogCat window (click the Debug perspective; see also Figure 2-2):

```
11-16 06:25:59.396: D/Lifecycle(559): In the onCreate() event
11-16 06:25:59.396: D/Lifecycle(559): In the onStart() event
11-16 06:25:59.396: D/Lifecycle(559): In the onResume() event
```

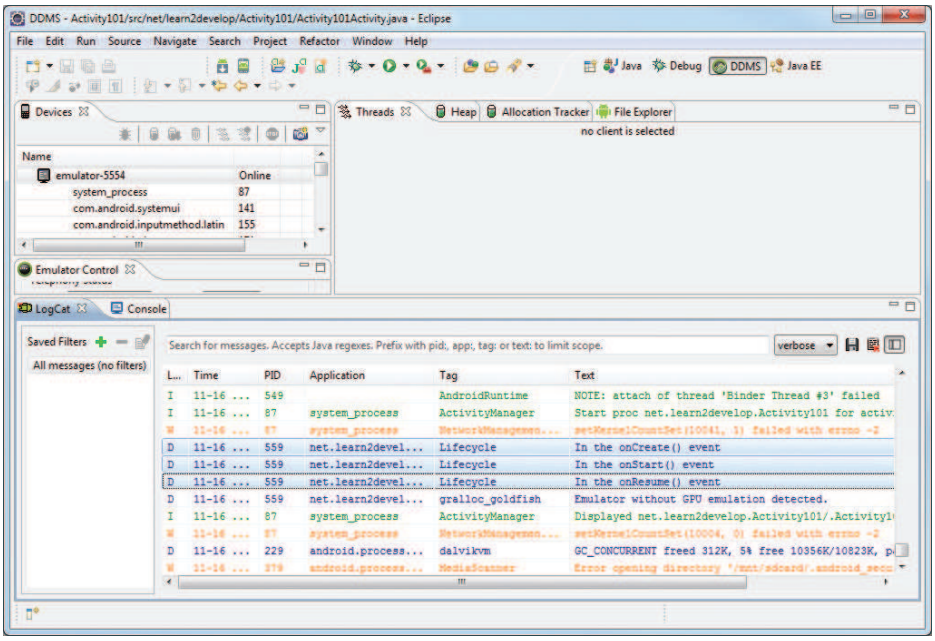

FIGURE 2-2

5. If you click the Back button on the Android emulator, the following is printed:

```
11-16 06:29:26.665: D/Lifecycle(559): In the onPause() event
11-16 06:29:28.465: D/Lifecycle(559): In the onStop() event
11-16 06:29:28.465: D/Lifecycle(559): In the onDestroy() event
```

6. Click the Home button and hold it there. Click the `Activities` icon and observe the following:

```
11-16 06:31:08.905: D/Lifecycle(559): In the onCreate() event
11-16 06:31:08.905: D/Lifecycle(559): In the onStart() event
11-16 06:31:08.925: D/Lifecycle(559): In the onResume() event
```

7. Click the Phone button on the Android emulator so that the activity is pushed to the background. Observe the output in the LogCat window:

```
11-16 06:32:00.585: D/Lifecycle(559): In the onPause() event
11-16 06:32:05.015: D/Lifecycle(559): In the onStop() event
```

8. Notice that the `onDestroy()` event is not called, indicating that the activity is still in memory. Exit the phone dialer by clicking the Back button. The activity is now visible again. Observe the output in the LogCat window:

```
11-16 06:32:50.515: D/Lifecycle(559): In the onRestart() event
11-16 06:32:50.515: D/Lifecycle(559): In the onStart() event
11-16 06:32:50.515: D/Lifecycle(559): In the onResume() event
```

The `onRestart()` event is now fired, followed by the `onStart()` and `onResume()` methods.

How It Works

As you can see from this simple example, an activity is destroyed when you click the Back button. This is crucial to know, as whatever state the activity is currently in will be lost; hence, you need to write additional code in your activity to preserve its state when it is destroyed (Chapter 3 shows you how). At this point, note that the `onPause()` method is called in both scenarios — when an activity is sent to the background, as well as when it is killed when the user presses the Back button.

When an activity is started, the `onStart()` and `onResume()`methods are always called, regardless of whether the activity is restored from the background or newly created. When an activity is created for the first time, the `onCreate()` method is called.

From the preceding example, you can derive the following guidelines:

- Use the `onCreate()` method to create and instantiate the objects that you will be using in your application.
- Use the `onResume()` method to start any services or code that needs to run while your activity is in the foreground.
- Use the `onPause()` method to stop any services or code that does not need to run when your activity is not in the foreground.
- Use the `onDestroy()` method to free up resources before your activity is destroyed.

NOTE *Even if an application has only one activity and the activity is killed, the application will still be running in memory.*

Applying Styles and Themes to an Activity

By default, an activity occupies the entire screen. However, you can apply a dialog theme to an activity so that it is displayed as a floating dialog. For example, you might want to customize your activity to display as a pop-up, warning users about some actions that they are going to perform. In this case, displaying the activity as a dialog is a good way to get their attention.

To apply a dialog theme to an activity, simply modify the `<Activity>` element in the `AndroidManifest.xml` file by adding the `android:theme` attribute:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.learn2develop.Activity101"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="14" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@android:style/Theme.Dialog">
        <activity
            android:label="@string/app_name"
            android:name=".Activity101Activity" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

This will make the activity appear as a dialog, as shown in Figure 2-3.

FIGURE 2-3

Hiding the Activity Title

You can also hide the title of an activity if desired (such as when you just want to display a status update to the user). To do so, use the `requestWindowFeature()` method and pass it the `Window.FEATURE_NO_TITLE` constant, like this:

```
import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
import android.view.Window;

public class Activity101Activity extends Activity {
```

```
        String tag = "Lifecycle";

        /** Called when the activity is first created. */
        @Override
        public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            //---hides the title bar---
            requestWindowFeature(Window.FEATURE_NO_TITLE);

            setContentView(R.layout.main);
            Log.d(tag, "In the onCreate() event");
        }
    }
```

This will hide the title bar, as shown in Figure 2-4.

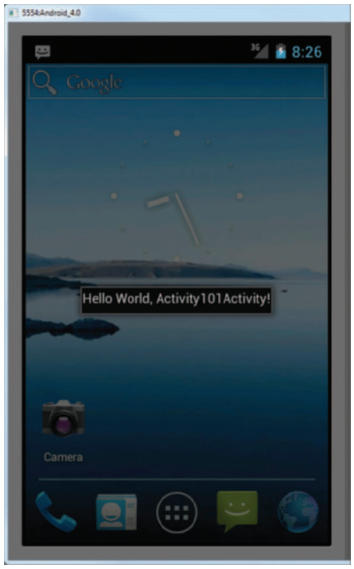

FIGURE 2-4

Displaying a Dialog Window

There are times when you need to display a dialog window to get a confirmation from the user. In this case, you can override the `onCreateDialog()` protected method defined in the `Activity` base class to display a dialog window. The following Try It Out shows you how.

TRY IT OUT Displaying a Dialog Window Using an Activity

codefile Dialog.zip available for download at Wrox.com

1. Using Eclipse, create a new Android project and name it **Dialog**.
2. Add the following statements in bold to the `main.xml` file:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

<Button
    android:id="@+id/btn_dialog"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Click to display a dialog"
    android:onClick="onClick" />

</LinearLayout>
```

3. Add the following statements in bold to the `DialogActivity.java` file:

```
package net.learn2develop.Dialog;

import android.app.Activity;
```

```
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class DialogActivity extends Activity {
    CharSequence[] items = { "Google", "Apple", "Microsoft" };
    boolean[] itemsChecked = new boolean [items.length];

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void onClick(View v) {
        showDialog(0);
    }

    @Override
    protected Dialog onCreateDialog(int id) {
        switch (id) {
        case 0:
            return new AlertDialog.Builder(this)
            .setIcon(R.drawable.ic_launcher)
            .setTitle("This is a dialog with some simple text...")
            .setPositiveButton("OK",
                new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int whichButton)
                    {
                        Toast.makeText(getBaseContext(),
                                "OK clicked!", Toast.LENGTH_SHORT).show();
                    }
                }
            )
            .setNegativeButton("Cancel",
                new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int whichButton)
                    {
                        Toast.makeText(getBaseContext(),
                                "Cancel clicked!", Toast.LENGTH_SHORT).show();
                    }
                }
            )
            .setMultiChoiceItems(items, itemsChecked,
                new DialogInterface.OnMultiChoiceClickListener() {
                    public void onClick(DialogInterface dialog,
                    int which, boolean isChecked) {
                        Toast.makeText(getBaseContext(),
                            items[which] + (isChecked ? " checked!":" unchecked!"),
                            Toast.LENGTH_SHORT).show();
```

```
                    }
                }
            ).create();


        }
        return null;
    }

}
```

4. Press F11 to debug the application on the Android emulator. Click the button to display the dialog (see Figure 2-5). Checking the various checkboxes will cause the `Toast` class to display the text of the item checked/unchecked. To dismiss the dialog, click the OK or Cancel button.

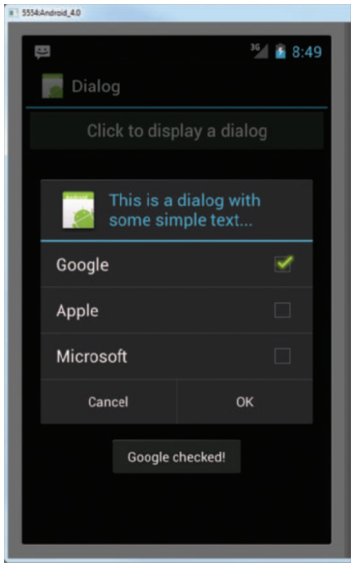

FIGURE 2-5

How It Works

To display a dialog, you first implement the `onCreateDialog()` method in the `Activity` class:

```
@Override
protected Dialog onCreateDialog(int id) {
    //...
}
```

This method is called when you call the `showDialog()` method:

```
public void onClick(View v) {
    showDialog(0);
}
```

The `onCreateDialog()` method is a callback for creating dialogs that are managed by the activity. When you call the `showDialog()` method, this callback will be invoked. The `showDialog()` method accepts an integer argument identifying a particular dialog to display. In this case, we used a `switch` statement to identify the different types of dialogs to create, although the current example creates only one type of dialog. Subsequent Try It Out exercises will extend this example to create different types of dialogs.

To create a dialog, you use the `AlertDialog` class's `Builder` constructor. You set the various properties, such as icon, title, and buttons, as well as checkboxes:

```
@Override
protected Dialog onCreateDialog(int id) {
    switch (id) {
    case 0:
        return new AlertDialog.Builder(this)
        .setIcon(R.drawable.ic_launcher)
        .setTitle("This is a dialog with some simple text...")
        .setPositiveButton("OK",
            new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int whichButton)
```

```
                {
                    Toast.makeText(getBaseContext(),
                            "OK clicked!", Toast.LENGTH_SHORT).show();
                }
            }
        )
        .setNegativeButton("Cancel",
            new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int whichButton)
                {
                    Toast.makeText(getBaseContext(),
                            "Cancel clicked!", Toast.LENGTH_SHORT).show();
                }
            }
        )
        .setMultiChoiceItems(items, itemsChecked,
            new DialogInterface.OnMultiChoiceClickListener() {
                public void onClick(DialogInterface dialog,
                int which, boolean isChecked) {
                    Toast.makeText(getBaseContext(),
                        items[which] + (isChecked ? " checked!":" unchecked!"),
                        Toast.LENGTH_SHORT).show();
                }
            }
        ).create();


    }
    return null;
}
```

The preceding code sets two buttons, OK and Cancel, using the `setPositiveButton()` and `setNegativeButton()` methods, respectively. You also set a list of checkboxes for users to choose via the `setMultiChoiceItems()` method. For the `setMultiChoiceItems()` method, you passed in two arrays: one for the list of items to display and another to contain the value of each item, to indicate if they are checked. When each item is checked, you use the `Toast` class to display a message indicating the item that was checked.

The preceding code for creating the dialog looks complicated, but it could easily be rewritten as follows:

```
package net.learn2develop.Dialog;

import android.app.Activity;
import android.app.AlertDialog;
import android.app.AlertDialog.Builder;
import android.app.Dialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class DialogActivity extends Activity {
```

```
    CharSequence[] items = { "Google", "Apple", "Microsoft" };
    boolean[] itemsChecked = new boolean [items.length];

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void onClick(View v) {
        showDialog(0);
    }

    @Override
    protected Dialog onCreateDialog(int id) {
        switch (id) {
        case 0:
            Builder builder = new AlertDialog.Builder(this);
            builder.setIcon(R.drawable.ic_launcher);
            builder.setTitle("This is a dialog with some simple text...");
            builder.setPositiveButton("OK",
                new DialogInterface.OnClickListener() {
                   public void onClick(DialogInterface dialog,    int whichButton) {
                        Toast.makeText(getBaseContext(),
                                "OK clicked!", Toast.LENGTH_SHORT).show();
                    }
                }
            );

            builder.setNegativeButton("Cancel",
                new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int whichButton) {
                        Toast.makeText(getBaseContext(),
                            "Cancel clicked!", Toast.LENGTH_SHORT).show();
                    }
                }
            );

            builder.setMultiChoiceItems(items, itemsChecked,
                new DialogInterface.OnMultiChoiceClickListener() {
                    public void onClick(DialogInterface dialog,
                    int which, boolean isChecked) {
                        Toast.makeText(getBaseContext(),
                            items[which] + (isChecked ? " checked!":" unchecked!"),
                            Toast.LENGTH_SHORT).show();
                    }
                }
            );
            return builder.create();
        }
        return null;
    }
}
```

THE CONTEXT OBJECT

In Android, you often encounter the `Context` class and its instances. Instances of the `Context` class are often used to provide references to your application. For example, in the following code snippet, the first parameter of the `Toast` class takes in a `Context` object:

```
.setPositiveButton("OK",
    new DialogInterface.OnClickListener() {
       public void onClick(DialogInterface dialog, int whichButton)
        {
            Toast.makeText(getBaseContext(),
                    "OK clicked!", Toast.LENGTH_SHORT).show();
        }
    }
```

However, because the `Toast()` class is not used directly in the activity (it is used within the `AlertDialog` class), you need to return an instance of the `Context` class by using the `getBaseContext()` method.

You also encounter the `Context` class when creating a view dynamically in an activity. For example, you may want to dynamically create a `TextView` from code. To do so, you instantiate the `TextView` class, like this:

```
TextView tv = new TextView(this);
```

The constructor for the `TextView` class takes a `Context` object; and because the `Activity` class is a subclass of `Context`, you can use the `this` keyword to represent the `Context` object.

Displaying a Progress Dialog

One common UI feature in an Android device is the "Please wait" dialog that you typically see when an application is performing a long-running task. For example, the application may be logging in to a server before the user is allowed to use it, or it may be doing a calculation before displaying the result to the user. In such cases, it is helpful to display a dialog, known as a *progress dialog*, so that the user is kept in the loop.

The following Try It Out demonstrates how to display such a dialog.

TRY IT OUT Displaying a Progress (Please Wait) Dialog

1. Using the same project created in the previous section, add the following statements in bold to the `main.xml` file:

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

```
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

<Button
    android:id="@+id/btn_dialog"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Click to display a dialog"
    android:onClick="onClick" />

<Button
    android:id="@+id/btn_dialog2"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Click to display a progress dialog"
    android:onClick="onClick2" />

</LinearLayout>
```

2. Add the following statements in bold to the `DialogActivity.java` file:

```
package net.learn2develop.Dialog;

import android.app.Activity;
import android.app.AlertDialog;
import android.app.AlertDialog.Builder;
import android.app.Dialog;
import android.app.ProgressDialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class DialogActivity extends Activity {
    CharSequence[] items = { "Google", "Apple", "Microsoft" };
    boolean[] itemsChecked = new boolean [items.length];

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void onClick(View v) {
        showDialog(0);
    }

    public void onClick2(View v) {
        //---show the dialog---
        final ProgressDialog dialog = ProgressDialog.show(
            this, "Doing something", "Please wait...", true);
        new Thread(new Runnable(){
```

```
            public void run(){
                try {
                    //---simulate doing something lengthy---
                    Thread.sleep(5000);
                    //---dismiss the dialog---
                    dialog.dismiss();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }).start();
    }

    @Override
    protected Dialog onCreateDialog(int id) { ... }

}
```

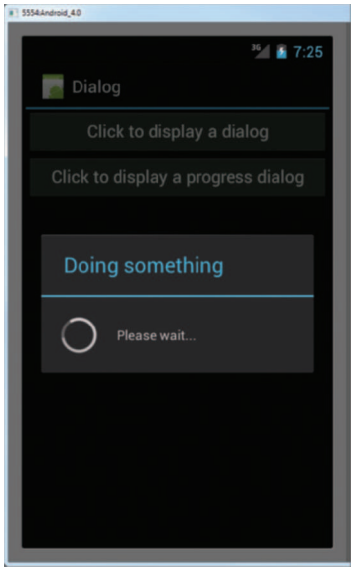

FIGURE 2-6

3. Press F11 to debug the application on the Android emulator. Clicking the second button will display the progress dialog, as shown in Figure 2-6. It will go away after five seconds.

How It Works

Basically, to create a progress dialog, you created an instance of the `ProgressDialog` class and called its `show()` method:

```
//---show the dialog---
final ProgressDialog dialog = ProgressDialog.show(
    this, "Doing something", "Please wait...", true);
```

This displays the progress dialog that you have just seen. Because this is a modal dialog, it will block the UI until it is dismissed. To perform a long-running task in the background, you created a `Thread` using a `Runnable` block (you will learn more about threading in Chapter 11). The code that you placed inside the `run()` method will be executed in a separate thread, and in this case you simulated it performing something for five seconds by inserting a delay using the `sleep()` method:

```
new Thread(new Runnable(){
    public void run(){
        try {
            //---simulate doing something lengthy---
            Thread.sleep(5000);
            //---dismiss the dialog---
            dialog.dismiss();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}).start();
```

After the five seconds elapse, you dismiss the dialog by calling the `dismss()` method.

Displaying a More Sophisticated Progress Dialog

Besides the generic "please wait" dialog that you created in the previous section, you can also create a dialog that displays the progress of an operation, such as the status of a download.

The following Try It Out shows you how to display a specialized progress dialog.

TRY IT OUT Displaying the Progress of an Operation

1. Using the same project created in the previous section, add the following lines in bold to the `main.xml` file:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

<Button
    android:id="@+id/btn_dialog"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Click to display a dialog"
    android:onClick="onClick" />

<Button
    android:id="@+id/btn_dialog2"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Click to display a progress dialog"
    android:onClick="onClick2" />

<Button
    android:id="@+id/btn_dialog3"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Click to display a detailed progress dialog"
    android:onClick="onClick3" />

</LinearLayout>
```

2. Add the following statements in bold to the `DialogActivity.java` file:

```
package net.learn2develop.Dialog;

import android.app.Activity;
import android.app.AlertDialog;
import android.app.AlertDialog.Builder;
import android.app.Dialog;
import android.app.ProgressDialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.View;
```

```
import android.widget.Toast;

public class DialogActivity extends Activity {
    CharSequence[] items = { "Google", "Apple", "Microsoft" };
    boolean[] itemsChecked = new boolean [items.length];

    ProgressDialog progressDialog;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) { ... }

    public void onClick(View v) { ... }

    public void onClick2(View v) { ... }

    public void onClick3(View v) {
        showDialog(1);
        progressDialog.setProgress(0);

        new Thread(new Runnable(){
            public void run(){
                for (int i=1; i<=15; i++) {
                    try {
                        //---simulate doing something lengthy---
                        Thread.sleep(1000);
                        //---update the dialog---
                        progressDialog.incrementProgressBy((int)(100/15));
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
                progressDialog.dismiss();
            }
        }).start();
    }

    @Override
    protected Dialog onCreateDialog(int id) {
        switch (id) {
        case 0:
            return new AlertDialog.Builder(this)
                //...
            ).create();

        case 1:
            progressDialog = new ProgressDialog(this);
            progressDialog.setIcon(R.drawable.ic_launcher);
            progressDialog.setTitle("Downloading files...");
            progressDialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
            progressDialog.setButton(DialogInterface.BUTTON_POSITIVE, "OK",
                new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog,
                    int whichButton)
```

```
                {
                    Toast.makeText(getBaseContext(),
                            "OK clicked!", Toast.LENGTH_SHORT).show();
                }
            });
            progressDialog.setButton(DialogInterface.BUTTON_NEGATIVE, "Cancel",
                new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog,
                        int whichButton)
                    {
                        Toast.makeText(getBaseContext(),
                                "Cancel clicked!", Toast.LENGTH_SHORT).show();
                    }
            });
            return progressDialog;
        }

        return null;
    }
}
```

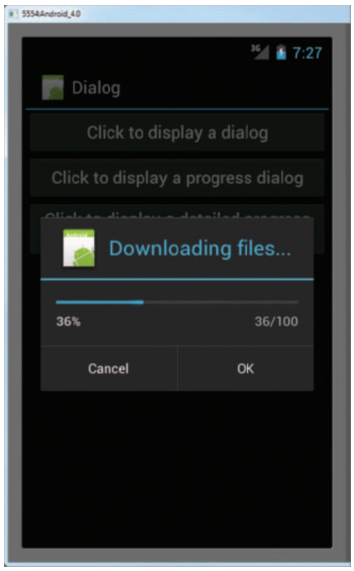

FIGURE 2-7

3. Press F11 to debug the application on the Android emulator. Click the third button to display the progress dialog (see Figure 2-7). Note that the progress bar will count up to 100%.

How It Works

To create a dialog that shows the progress of an operation, you first create an instance of the `ProgressDialog` class and set its various properties, such as icon, title, and style:

```
progressDialog = new ProgressDialog(this);
progressDialog.setIcon(R.drawable.ic_launcher);
progressDialog.setTitle("Downloading files...");
progressDialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
```

You then set the two buttons that you want to display inside the progress dialog:

```
progressDialog.setButton(DialogInterface.BUTTON_POSITIVE, "OK",
    new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog,
        int whichButton)
        {
            Toast.makeText(getBaseContext(),
                    "OK clicked!", Toast.LENGTH_SHORT).show();
        }
});
progressDialog.setButton(DialogInterface.BUTTON_NEGATIVE, "Cancel",
    new DialogInterface.OnClickListener() {
```

```
                public void onClick(DialogInterface dialog,
                    int whichButton)
                {
                    Toast.makeText(getBaseContext(),
                            "Cancel clicked!", Toast.LENGTH_SHORT).show();
                }
            });
            return progressDialog;
```

The preceding code causes a progress dialog to appear (see Figure 2-8).

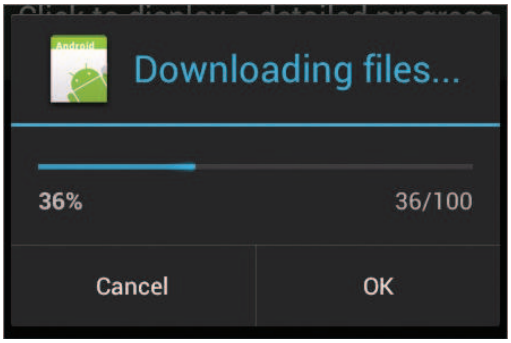

FIGURE 2-8

To display the progress status in the progress dialog, you can use a `Thread` object to run a `Runnable` block of code:

```
progressDialog.setProgress(0);

new Thread(new Runnable(){
    public void run(){
        for (int i=1; i<=15; i++) {
            try {
                //---simulate doing something lengthy---
                Thread.sleep(1000);
                //---update the dialog---
                progressDialog.incrementProgressBy((int)(100/15));
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        progressDialog.dismiss();
    }
}).start();
```

In this case, you want to count from 1 to 15, with a delay of one second between each number. The `incrementProgressBy()` method increments the counter in the progress dialog. When the progress dialog reaches 100%, it is dismissed.

LINKING ACTIVITIES USING INTENTS

An Android application can contain zero or more activities. When your application has more than one activity, you often need to navigate from one to another. In Android, you navigate between activities through what is known as an *intent*.

The best way to understand this very important but somewhat abstract concept in Android is to experience it firsthand and see what it helps you to achieve. The following Try It Out shows how to add another activity to an existing project and then navigate between the two activities.

TRY IT OUT Linking Activities with Intents

codefile UsingIntent.zip available for download at Wrox.com

1. Using Eclipse, create a new Android project and name it **UsingIntent**.
2. Right-click on the package name under the `src` folder and select New ➪ Class (see Figure 2-9).

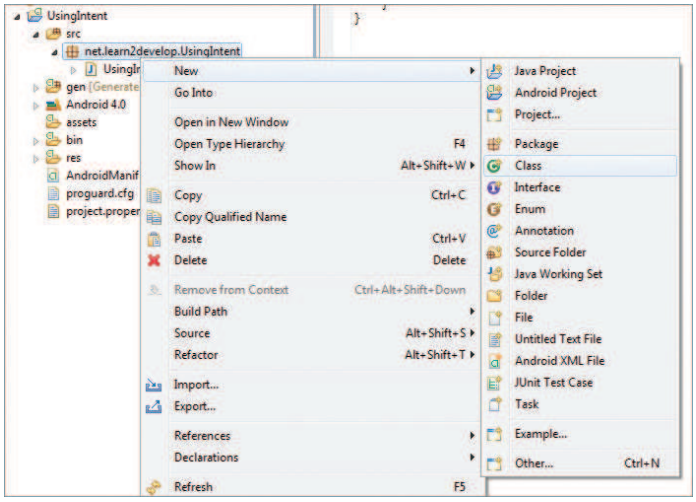

FIGURE 2-9

3. Name the new class **SecondActivity** and click Finish.
4. Add the following statements in bold to the `AndroidManifest.xml` file:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.learn2develop.UsingIntent"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="14" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:label="@string/app_name"
            android:name=".UsingIntentActivity" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:label="Second Activity"
            android:name=".SecondActivity" >
```

```
            <intent-filter >
                <action android:name="net.learn2develop.SecondActivity" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

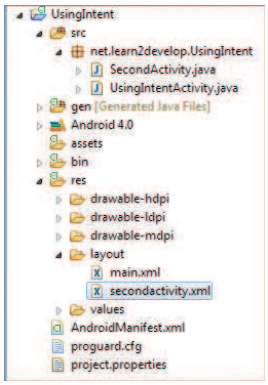

FIGURE 2-10

5. Make a copy of the `main.xml` file (in the `res/layout` folder) by right-clicking on it and selecting `Copy`. Then, right-click on the `res/layout` folder and select `Paste`. Name the file `secondactivity.xml`. The `res/layout` folder will now contain the `secondactivity.xml` file (see Figure 2-10).

6. Modify the `secondactivity.xml` file as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<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"
        android:text="This is the Second Activity!" />

</LinearLayout>
```

7. In the `SecondActivity.java` file, add the following statements in bold:

```
package net.learn2develop.UsingIntent;

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

public class SecondActivity extends Activity{
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.secondactivity);
    }
}
```

8. Add the following lines in bold to the `main.xml` file:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

<Button
```

```
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Display second activity"
    android:onClick="onClick"/>

</LinearLayout>
```

9. Modify the `UsingIntentActivity.java` file as shown in bold:

```
package net.learn2develop.UsingIntent;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class UsingIntentActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void onClick(View view) {
        startActivity(new Intent("net.learn2develop.SecondActivity"));
    }
}
```

10. Press F11 to debug the application on the Android emulator. When the first activity is loaded, click the button and the second activity will now be loaded (see Figure 2-11).

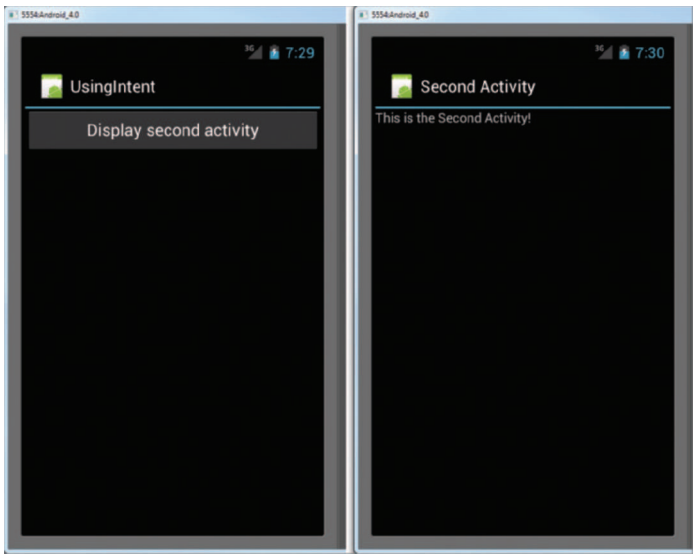

FIGURE 2-11

How It Works

As you have learned, an activity is made up of a UI component (for example, `main.xml`) and a class component (for example, `UsingIntentActivity.java`). Hence, if you want to add another activity to a project, you need to create these two components.

In the `AndroidManifest.xml` file, specifically you have added the following:

```
<activity
    android:label=" Second Activity"
    android:name=".SecondActivity" >
    <intent-filter >
        <action android:name="net.learn2develop.SecondActivity" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

Here, you have added a new activity to the application. Note the following:

- The name (class) of the new activity added is `SecondActivity`.
- The label for the new activity is named `Second Activity`.
- The intent filter name for the new activity is `net.learn2develop.SecondActivity`. Other activities that wish to call this activity will invoke it via this name. Ideally, you should use the reverse domain name of your company as the intent filter name in order to reduce the chances of another application having the same intent filter name. (The next section discusses what happens when two or more activities have the same intent filter.)
- The category for the intent filter is `android.intent.category.DEFAULT`. You need to add this to the intent filter so that this activity can be started by another activity using the `startActivity()` method (more on this shortly).

When the button is clicked, you use the `startActivity()` method to display `SecondActivity` by creating an instance of the `Intent` class and passing it the intent filter name of `SecondActivity` (which is `net.learn2develop.SecondActivity`):

```
public void onClick(View view) {
    startActivity(new Intent("net.learn2develop.SecondActivity"));
}
```

Activities in Android can be invoked by any application running on the device. For example, you can create a new Android project and then display `SecondActivity` by using its `net.learn2develop.SecondActivity` intent filter. This is one of the fundamental concepts in Android that enables an application to invoke another easily.

If the activity that you want to invoke is defined within the same project, you can rewrite the preceding statement like this:

```
startActivity(new Intent(this, SecondActivity.class));
```

However, this approach is applicable only when the activity you want to display is within the same project as the current activity.

Resolving Intent Filter Collision

In the previous section, you learned that the `<intent-filter>` element defines how your activity can be invoked by another activity. What happens if another activity (in either the same or a separate application) has the same filter name? For example, suppose your application has another activity named `Activity3`, with the following entry in the `AndroidManifest.xml` file:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.learn2develop.UsingIntent"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="14" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:label="@string/app_name"
            android:name=".UsingIntentActivity" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:label="Second Activity"
            android:name=".SecondActivity" >
            <intent-filter >
                <action android:name="net.learn2develop.SecondActivity" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>

        <activity
            android:label="Third Activity"
            android:name=".ThirdActivity" >
            <intent-filter >
                <action android:name="net.learn2develop.SecondActivity" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>

    </application>

</manifest>
```

If you call the `startActivity()` method with the following intent, then the Android OS will display a selection of activities, as shown in Figure 2-12:

```
startActivity(new Intent("net.learn2develop.SecondActivity"));
```

If you check the "Use by default for this action" item and then select an activity, then the next time the intent "`net.learn2develop.SecondActivity`" is called again, it will launch the previous activity that you have selected.

To clear this default, go to the Settings application in Android and select Apps ➪ Manage applications, and then select the application name (see Figure 2-13). When the details of the application are shown, scroll down to the bottom and click the Clear defaults button.

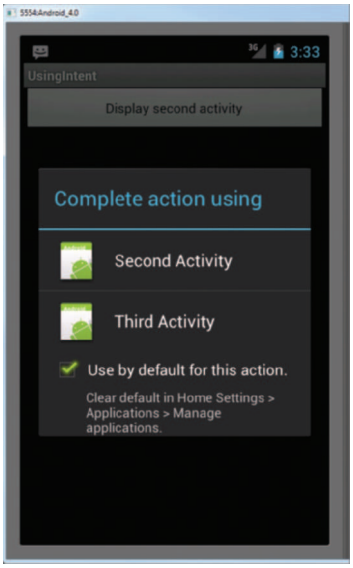

FIGURE 2-12

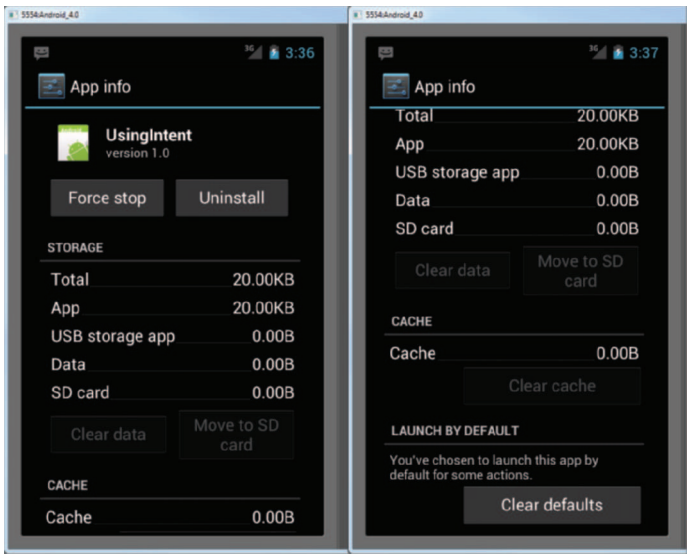

FIGURE 2-13

Returning Results from an Intent

The `startActivity()` method invokes another activity but does not return a result to the current activity. For example, you may have an activity that prompts the user for user name and password. The information entered by the user in that activity needs to be passed back to the calling activity for further processing. If you need to pass data back from an activity, you should instead use the `startActivityForResult()` method. The following Try It Out demonstrates this.

TRY IT OUT Obtaining a Result from an Activity

1. Using the same project from the previous section, add the following statements in bold to the `secondactivity.xml` file:

```
<?xml version="1.0" encoding="utf-8"?>
<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"
    android:text="This is the Second Activity!" />

<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Please enter your name" />

<EditText
    android:id="@+id/txt_username"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />

<Button
    android:id="@+id/btn_OK"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="OK"
    android:onClick="onClick"/>

</LinearLayout>
```

2. Add the following statements in bold to `SecondActivity.java`:

```
package net.learn2develop.UsingIntent;

import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;

public class SecondActivity extends Activity{
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.secondactivity);
    }

    public void onClick(View view) {
        Intent data = new Intent();

        //---get the EditText view---
        EditText txt_username =
            (EditText) findViewById(R.id.txt_username);

        //---set the data to pass back---
        data.setData(Uri.parse(
```

```
            txt_username.getText().toString()));
        setResult(RESULT_OK, data);

        //---closes the activity---
        finish();
    }
}
```

3. Add the following statements in bold to the `UsingIntentActivity.java` file:

```
package net.learn2develop.UsingIntent;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class UsingIntentActivity extends Activity {
    int request_Code = 1;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void onClick(View view) {
        //startActivity(new Intent("net.learn2develop.SecondActivity"));
        //or
        //startActivity(new Intent(this, SecondActivity.class));
        startActivityForResult(new Intent(
            "net.learn2develop.SecondActivity"),
            request_Code);
    }

    public void onActivityResult(int requestCode, int resultCode, Intent data)
    {
        if (requestCode == request_Code) {
            if (resultCode == RESULT_OK) {
                Toast.makeText(this,data.getData().toString(),
                    Toast.LENGTH_SHORT).show();
            }
        }
    }
}
```

4. Press F11 to debug the application on the Android emulator. When the first activity is loaded, click the button. `SecondActivity` will now be loaded. Enter your name (see Figure 2-14) and click the OK button. The first activity will display the name you have entered using the `Toast` class.

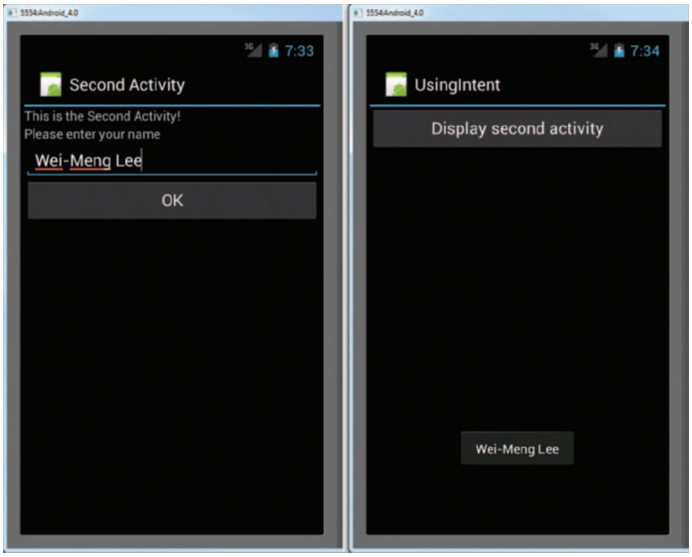

FIGURE 2-14

How It Works

To call an activity and wait for a result to be returned from it, you need to use the `startActivityForResult()` method, like this:

```
startActivityForResult(new Intent(
    "net.learn2develop.SecondActivity"),
    request_Code);
```

In addition to passing in an `Intent` object, you need to pass in a request code as well. The request code is simply an integer value that identifies an activity you are calling. This is needed because when an activity returns a value, you must have a way to identify it. For example, you may be calling multiple activities at the same time, and some activities may not return immediately (for example, waiting for a reply from a server). When an activity returns, you need this request code to determine which activity is actually returned.

NOTE *If the request code is set to -1, then calling it using the* `startActivityForResult()` *method is equivalent to calling it using the* `startActivity()` *method. That is, no result will be returned.*

In order for an activity to return a value to the calling activity, you use an `Intent` object to send data back via the `setData()` method:

```
Intent data = new Intent();

//---get the EditText view---
```

```
EditText txt_username =
    (EditText) findViewById(R.id.txt_username);

//---set the data to pass back---
data.setData(Uri.parse(
    txt_username.getText().toString()));
setResult(RESULT_OK, data);

//---closes the activity---
finish();
```

The `setResult()` method sets a result code (either `RESULT_OK` or `RESULT_CANCELLED`) and the data (an `Intent` object) to be returned back to the calling activity. The `finish()` method closes the activity and returns control back to the calling activity.

In the calling activity, you need to implement the `onActivityResult()` method, which is called whenever an activity returns:

```
public void onActivityResult(int requestCode, int resultCode,
Intent data)
{
    if (requestCode == request_Code) {
        if (resultCode == RESULT_OK) {
            Toast.makeText(this,data.getData().toString(),
                Toast.LENGTH_SHORT).show();
        }
    }
}
```

Here, you check for the appropriate request and result codes and display the result that is returned. The returned result is passed in via the `data` argument; and you obtain its details through the `getData()` method.

Passing Data Using an Intent Object

Besides returning data from an activity, it is also common to pass data to an activity. For example, in the previous example you may want to set some default text in the `EditText` view before the activity is displayed. In this case, you can use the `Intent` object to pass the data to the target activity.

The following Try It Out shows you the various ways in which you can pass data between activities.

TRY IT OUT Passing Data to the Target Activity

1. Using Eclipse, create a new Android project and name it **PassingData**.
2. Add the following statements in bold to the `main.xml` file:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
```

```
    android:orientation="vertical" >

<Button
    android:id="@+id/btn_SecondActivity"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Click to go to Second Activity"
    android:onClick="onClick"/>

</LinearLayout>
```

3. Add a new XML file to the `res/layout` folder and name it `secondactivity.xml`. Populate it as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<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"
    android:text="Welcome to Second Activity" />

<Button
    android:id="@+id/btn_MainActivity"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Click to return to main activity"
    android:onClick="onClick"/>

</LinearLayout>
```

4. Add a new Class file to the package and name it `SecondActivity`. Populate the `SecondActivity.java` file as follows:

```
package net.learn2develop.PassingData;

import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class SecondActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.secondactivity);

        //---get the data passed in using getStringExtra()---
        Toast.makeText(this,getIntent().getStringExtra("str1"),
                Toast.LENGTH_SHORT).show();

        //---get the data passed in using getIntExtra()---
```

```
        Toast.makeText(this,Integer.toString(
                getIntent().getIntExtra("age1", 0)),
                Toast.LENGTH_SHORT).show();

        //---get the Bundle object passed in---
        Bundle bundle = getIntent().getExtras();

        //---get the data using the getString()---
        Toast.makeText(this, bundle.getString("str2"),
                Toast.LENGTH_SHORT).show();

        //---get the data using the getInt() method---
        Toast.makeText(this,Integer.toString(bundle.getInt("age2")),
                Toast.LENGTH_SHORT).show();
    }

    public void onClick(View view) {
        //---use an Intent object to return data---
        Intent i = new Intent();

        //---use the putExtra() method to return some
        // value---
        i.putExtra("age3", 45);

        //---use the setData() method to return some value---
        i.setData(Uri.parse(
                "Something passed back to main activity"));

        //---set the result with OK and the Intent object---
        setResult(RESULT_OK, i);

        //---destroy the current activity---
        finish();
    }
}
```

5. Add the following statements in bold to the `AndroidManifest.xml` file:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.learn2develop.PassingData"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="14" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:label="@string/app_name"
            android:name=".PassingDataActivity" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
```

```
        <activity
            android:label="Second Activity"
            android:name=".SecondActivity" >
            <intent-filter >
                <action android:name="net.learn2develop.PassingDataSecondActivity" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

6. Add the following statements in bold to the `PassingDataActivity.java` file:

```
package net.learn2develop.PassingData;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class PassingDataActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void onClick(View view) {
        Intent i = new
                Intent("net.learn2develop.PassingDataSecondActivity");
        //---use putExtra() to add new name/value pairs---
        i.putExtra("str1", "This is a string");
        i.putExtra("age1", 25);

        //---use a Bundle object to add new name/values
        // pairs---
        Bundle extras = new Bundle();
        extras.putString("str2", "This is another string");
        extras.putInt("age2", 35);

        //---attach the Bundle object to the Intent object---
        i.putExtras(extras);

        //---start the activity to get a result back---
        startActivityForResult(i, 1);
    }

    public void onActivityResult(int requestCode,
    int resultCode, Intent data)
    {
        //---check if the request code is 1---
        if (requestCode == 1) {

            //---if the result is OK---
```

```
            if (resultCode == RESULT_OK) {

                //---get the result using getIntExtra()---
                Toast.makeText(this, Integer.toString(
                    data.getIntExtra("age3", 0)),
                    Toast.LENGTH_SHORT).show();

                //---get the result using getData()---
                Toast.makeText(this, data.getData().toString(),
                    Toast.LENGTH_SHORT).show();
            }
        }
    }

}
```

7. Press F11 to debug the application on the Android emulator. Click the button on each activity and observe the values displayed.

How It Works

While this application is not visually exciting, it does illustrate some important ways to pass data between activities.

First, you can use the `putExtra()` method of an `Intent` object to add a name/value pair:

```
//---use putExtra() to add new name/value pairs---
i.putExtra("str1", "This is a string");
i.putExtra("age1", 25);
```

The preceding statements add two name/value pairs to the `Intent` object: one of type `string` and one of type `integer`.

Besides using the `putExtra()` method, you can also create a `Bundle` object and then attach it using the `putExtras()` method. Think of a `Bundle` object as a dictionary object — it contains a set of name/value pairs. The following statements create a `Bundle` object and then add two name/value pairs to it. It is then attached to the `Intent` object:

```
//---use a Bundle object to add new name/values pairs---
Bundle extras = new Bundle();
extras.putString("str2", "This is another string");
extras.putInt("age2", 35);

//---attach the Bundle object to the Intent object---
i.putExtras(extras);
```

On the second activity, to obtain the data sent using the `Intent` object, you first obtain the `Intent` object using the `getIntent()` method. Then, call its `getStringExtra()` method to get the string value set using the `putExtra()` method:

```
//---get the data passed in using getStringExtra()---
Toast.makeText(this,getIntent().getStringExtra("str1"),
        Toast.LENGTH_SHORT).show();
```

In this case, you have to call the appropriate method to extract the name/value pair based on the type of data set. For the integer value, use the `getIntExtra()` method (the second argument is the default value in case no value is stored in the specified name):

```
//---get the data passed in using getIntExtra()---
Toast.makeText(this,Integer.toString(
        getIntent().getIntExtra("age1", 0)),
        Toast.LENGTH_SHORT).show();
```

To retrieve the `Bundle` object, use the `getExtras()` method:

```
//---get the Bundle object passed in---
Bundle bundle = getIntent().getExtras();
```

To get the individual name/value pairs, use the appropriate method. For the string value, use the `getString()` method:

```
//---get the data using the getString()---
Toast.makeText(this, bundle.getString("str2"),
        Toast.LENGTH_SHORT).show();
```

Likewise, use the `getInt()` method to retrieve an integer value:

```
//---get the data using the getInt() method---
Toast.makeText(this,Integer.toString(bundle.getInt("age2")),
        Toast.LENGTH_SHORT).show();
```

Another way to pass data to an activity is to use the `setData()` method (as used in the previous section), like this:

```
//---use the setData() method to return some value---
i.setData(Uri.parse(
        "Something passed back to main activity"));
```

Usually, you use the `setData()` method to set the data on which an `Intent` object is going to operate (such as passing a URL to an `Intent` object so that it can invoke a web browser to view a web page; see the section "Calling Built-In Applications Using Intents" later in this chapter for more examples).

To retrieve the data set using the `setData()` method, use the `getData()` method (in this example `data` is an `Intent` object):

```
//---get the result using getData()---
Toast.makeText(this, data.getData().toString(),
    Toast.LENGTH_SHORT).show();
```