# 55. An Introduction to Android 7 Multi-Window Support

Android 7 introduced a new feature in the form of multi-window support. Unlike previous versions of Android, multi-window support in Android 7 allows more than one activity to be displayed on the device screen at one time. In this chapter, an overview of Android multi-window modes will be provided from both user and app developer perspectives.

Once the basics of multi-window support have been covered, the next chapter will work through a tutorial outlining the practical steps involved in working with multi-window mode when developing Android apps.

## 55.1 Split-Screen, Freeform and Picture-in-Picture Modes

Multi-window support in Android provides three different forms of window support. Split-screen mode, available on most phone and tablet devices, provides a split screen environment where two activities appear either side by side or one above the other. A moveable divider is provided which, when dragged by the user, adjusts the percentage of the screen assigned to each of the adjacent activities:

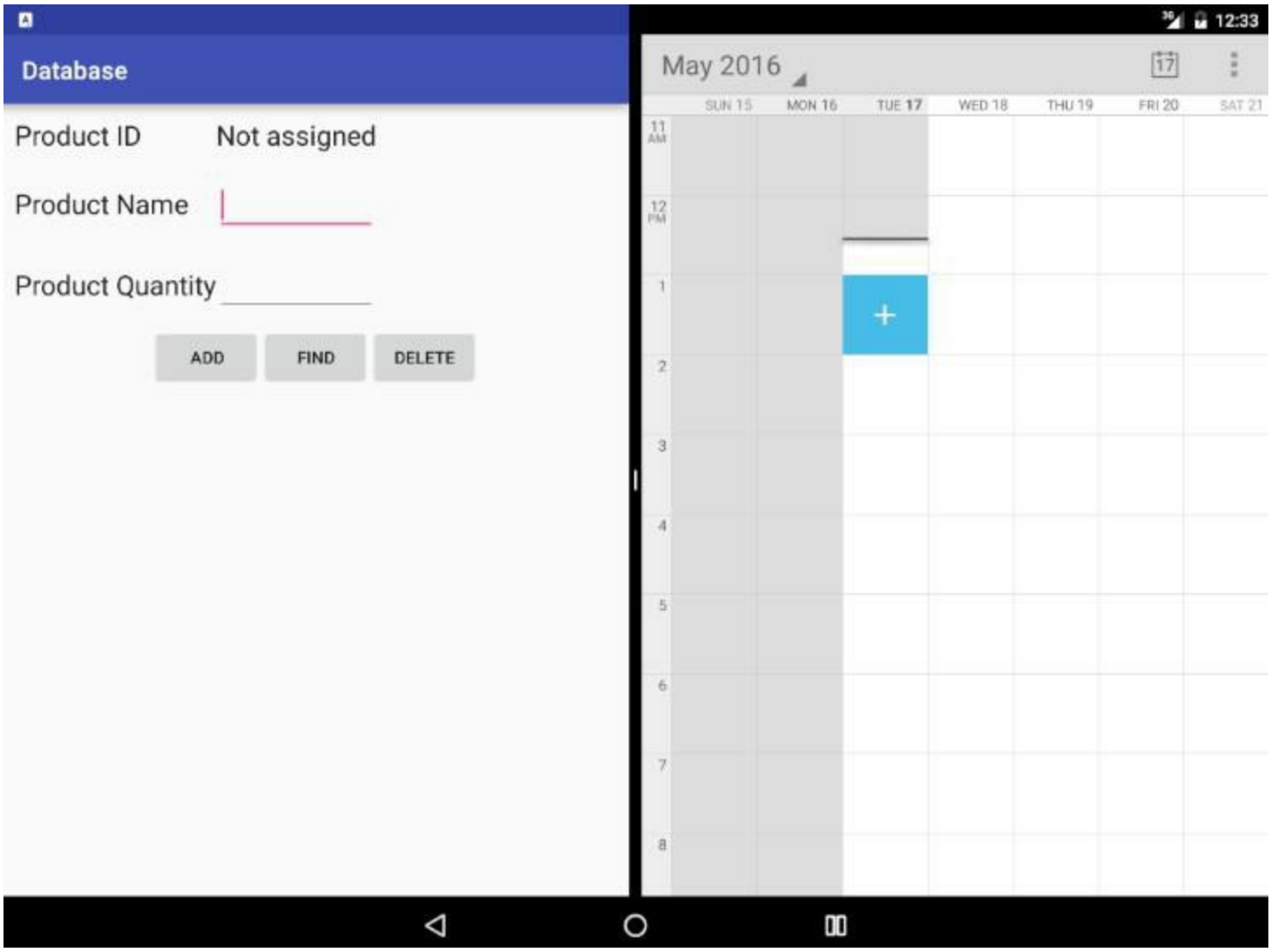


Figure 55-1

Freeform mode provides a windowing environment on devices with larger screens and is currently

enabled at the discretion of the device manufacturer. Freeform differs from split-screen mode in that it allows each activity to appear in a separate, resizable window and is not limited to two activities being displayed concurrently. Figure 55-2, for example, shows a device in freeform mode with the Calculator and Contacts apps displayed in separate windows:

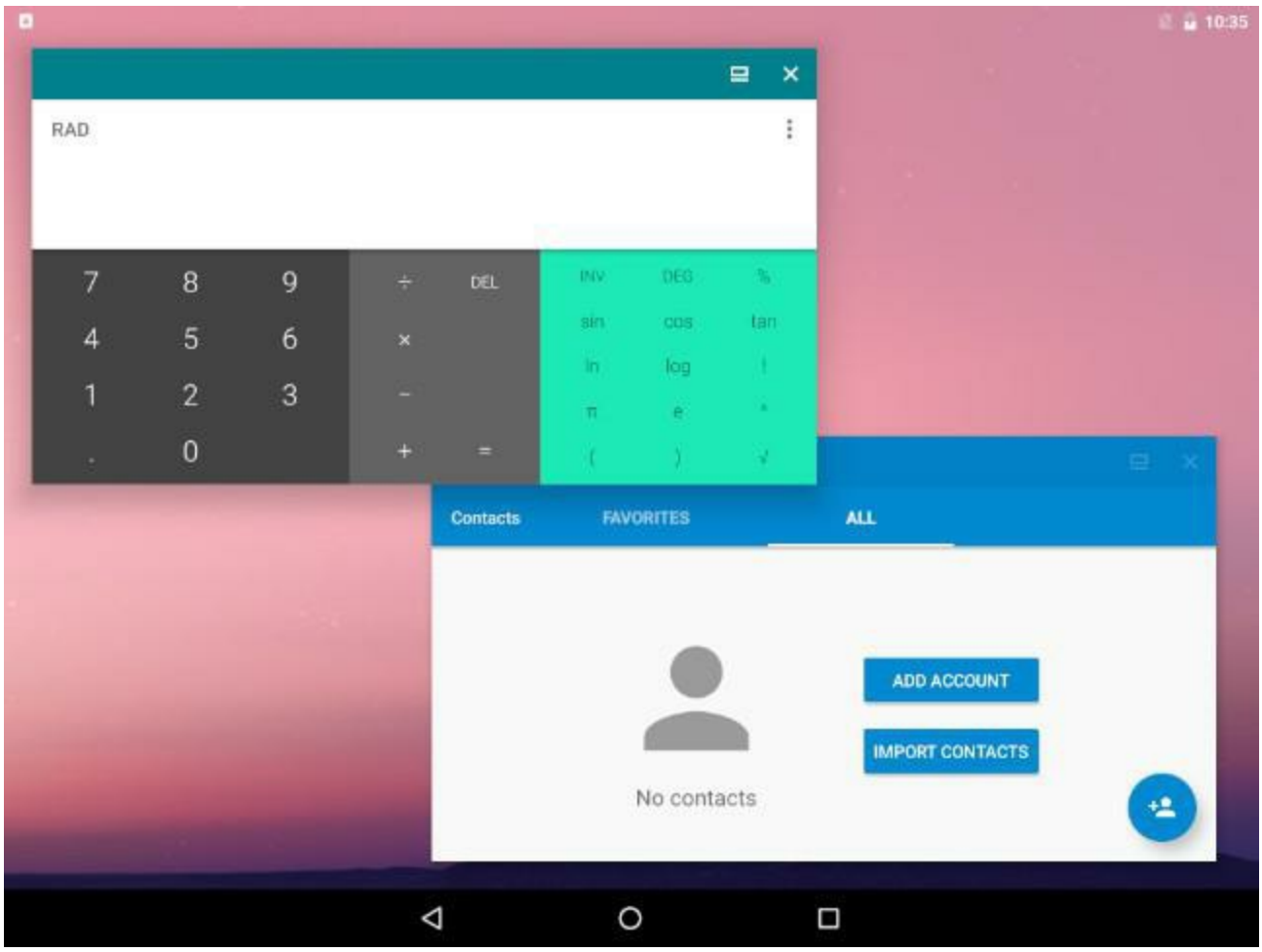


Figure 55-2

Picture-in-picture support, as the name suggests, allows video playback to continue in a smaller window while the user performs other tasks. At present this feature is only available on Android TV and, as such, is outside the scope of this book.

## 55.2 Entering Multi-Window Mode

Split-screen mode can be entered by pressing and holding the square Overview button until the display switches mode. Once in split-screen mode, the Overview button will change to display two rectangles as shown in Figure 55-3 and the current activity will fill one half of the screen. The Overview screen will appear in the adjacent half of the screen allowing the second activity to be selected for display:

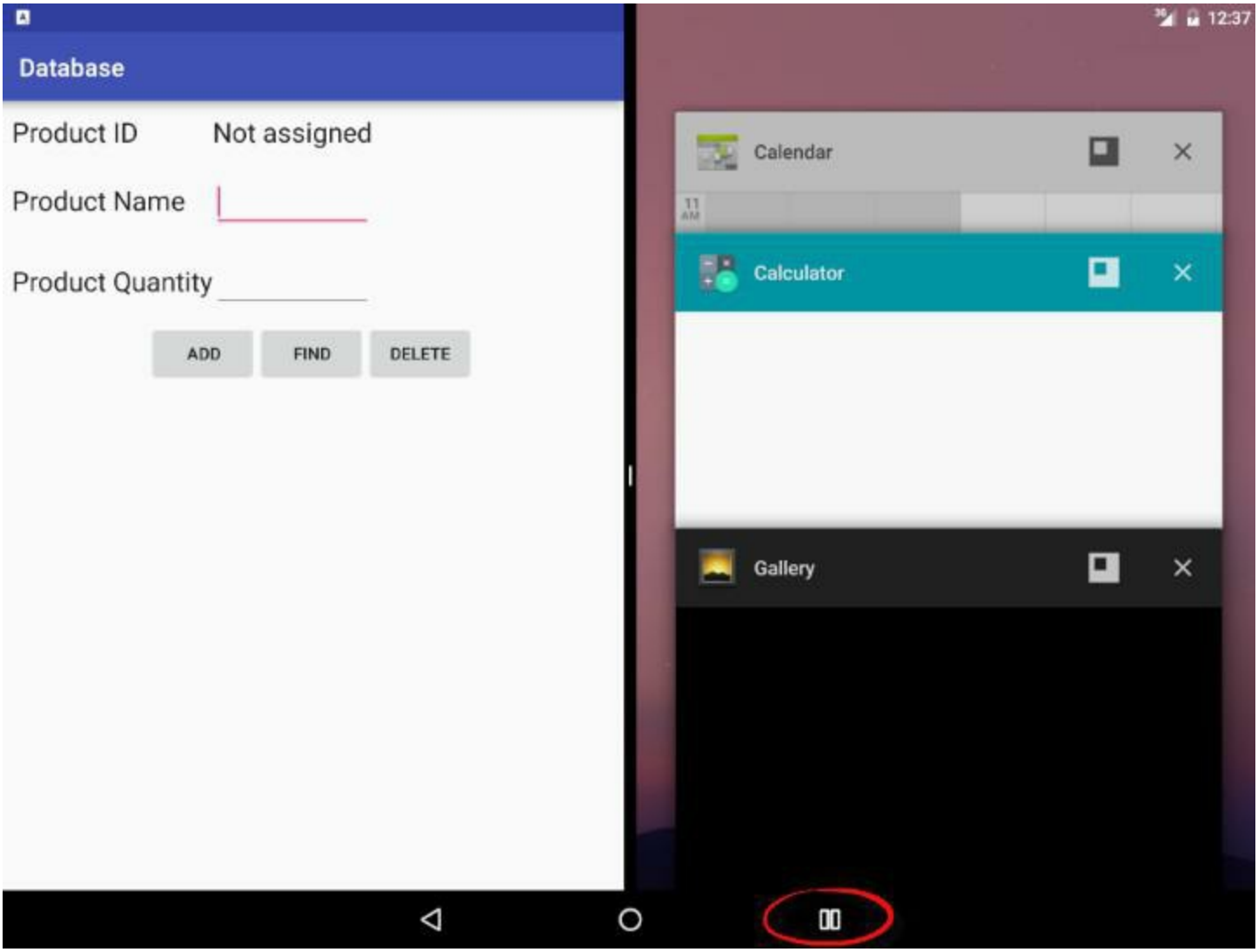


Figure 55-3

Alternatively, an app may be placed in split-screen mode by displaying the Overview screen, pressing and holding the title bar of a listed app and then dragging and dropping the app onto the highlighted section of the screen.

To exit split-screen mode, simply drag the divider separating the two activities to a far edge so that only one activity fills the screen, or press and hold the Overview button until it reverts to a single square.

In the case of freeform mode, an additional button appears within the title bar of the apps when listed in the Overview screen. When selected, this button (highlighted in Figure 55-4) causes the activity to appear in a freeform window:


Figure 55-4

The additional button located in the title bar of a freeform activity (shown in Figure 55-5) may be pressed to return the activity to full screen mode:

Figure 55-5

Alternatively, freeform activities may be switched to full screen mode from within the Overview screen by tapping the full screen button located in the title bar:


Figure 55-6

## 55.3 Checking for Freeform Support

As outlined earlier in the chapter, Google is leaving the choice of whether to enable freeform multi-window mode to the individual Android device manufacturers. Since it only makes sense to use freeform on larger devices, there is no guarantee that freeform will be available on every device on which an app is likely to run. Fortunately all of the freeform specific methods and attributes are ignored by the system if freeform mode is not available on a device, so using these will not cause the app to crash on a non-freeform device. Situations might arise, however, where it may be useful to be able to detect if a device supports freeform multi-window mode. Fortunately, this can be achieved by checking for the freeform window management feature in the package manager. The following code example checks for freeform multi-window support and returns a Boolean value based on the result of the test:

```
public Boolean checkFreeform() {
    return getPackageManager().hasSystemFeature(
            PackageManager.FEATURE_FREEFORM_WINDOW_MANAGEMENT);
}
```

## 55.4 Enabling Multi-Window Support in an App

The *android:resizableActivity* manifest file setting controls whether multi-window behavior is supported by an app. This setting can be made at either the application or individual activity levels. The following fragment, for example, configures the activity named MainActivity to support both split-screen and freeform multi-window modes:

```
<activity
    android:name=".MainActivity"
    android:resizeableActivity="true"
    android:label="@string/app_name"
    android:theme="@style/AppTheme.NoActionBar">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

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

Setting the property to false will prevent the activity from appearing in split-screen or freeform mode.

Launching an activity for which multi-window support is disabled will result in a message appearing indicating that the app does not support multi-window mode and the activity filling the entire screen. When a device is in multi-window mode, the title bar of such activities will also display a message within the Overview screen indicating that multi-window mode is not supported by the activity (Figure 55-7):


Figure 55-7

## 55.5 Specifying Multi-Window Attributes

A number of attributes are available as part of the <layout> element for specifying the size and placement of an activity when it is launched into a multi-window mode. The initial height, width and position of an activity when launched in freeform mode may be specified using the following attributes:

- **android:defaultWidth** – Specifies the default width of the activity.
- **android:defaultHeight** – Specifies the default height of the activity.
- **android:gravity** – Specifies the initial position of the activity (start, end, left, right, top etc.).

Note that the above attributes apply to the activity only when it is displayed in freeform mode. The following example configures an activity to appear with a specific height and width at the top of the starting edge of the screen:

```
<activity android:name=".MainActivity ">
    <layout android:defaultHeight="350dp"
          android:defaultWidth="450dp"
          android:gravity="start|end" />
</activity>
```

The following <layout> attributes may be used to specify the minimum width and height to which an activity may be reduced in either split-view or freeform modes:

- **android:minimalHeight** – Specifies the minimum height to which the activity may be reduced while in split-screen or freeform mode.
- **android:minimalWidth** - Specifies the minimum width to which the activity may be reduced while in split-screen or freeform mode.

When the user slides the split-screen divider beyond the minimal height or width boundaries, the system will stop resizing the layout of the shrinking activity and simply clip the user interface to make room for the adjacent activity.

The following manifest file fragment implements the minimal width and height attributes for an activity:

```
<activity android:name=".MainActivity ">
    <layout android:minimalHeight="400dp"
          android:minimalWidth="290dp" />
</activity>
```

## 55.6 Detecting Multi-Window Mode in an Activity

Situations may arise where an activity needs to detect whether it is currently being displayed to the user in multi-window mode. The current status can be obtained via a call to the *isInMultiWindowMode()* method of the Activity class. When called, this method returns a true or false value depending on whether or not the activity is currently full screen:

```
if (this.isInMultiWindowMode()) {
    // Activity is running in Multi-Window mode
} else {
    // Activity is not in Multi-Window mode
}
```

## 55.7 Receiving Multi-Window Notifications

An activity will receive notification that it is entering or exiting multi-window mode if it overrides the *onMultiWindowModeChanged()* callback method. The argument passed to this method is true on entering multi-window mode, and false when the activity exits the mode:

```
@Override
public void onMultiWindowModeChanged(boolean isInMultiWindowMode) {
    super.onMultiWindowModeChanged(isInMultiWindowMode);

    if (isInMultiWindowMode) {
        // Activity has entered multi-window mode
    } else {
        // Activity has exited multi-window mode
    }
}
```

## 55.8 Launching an Activity in Multi-Window Mode

In the *Android Explicit Intents – A Worked Example* chapter of this book, an example app was created in which an activity uses an intent to launch a second activity. By default, activities launched via an intent are considered to reside in the same *task stack* as the originating activity. An activity can, however, be launched into a new task stack by passing through the appropriate flags with the intent.

When an activity in multi-window mode launches another activity within the same task stack, the new activity replaces the originating activity within the split-screen or freeform window (the user returns to the original activity via the back button).

When launched into a new task stack in split-screen mode, however, the second activity will appear in the window adjacent to the original activity, allowing both activities to be viewed simultaneously. In the case of freeform mode, the launched activity will appear in a separate window from the original activity.

In order to launch an activity into a new task stack, the following flags must be set on the intent before it is started:

- Intent.FLAG_ACTIVITY_LAUNCH_ADJACENT
- Intent.FLAG_ACTIVITY_MULTIPLE_TASK
- Intent.FLAG_ACTIVITY_NEW_TASK

The following code, for example, configures and launches a second activity designed to appear in a separate window:

```
Intent i = new Intent(this, SecondActivity.class);
```

```
i.addFlags(Intent.FLAG_ACTIVITY_LAUNCH_ADJACENT|
           Intent.FLAG_ACTIVITY_MULTIPLE_TASK|
           Intent.FLAG_ACTIVITY_NEW_TASK);

startActivity(i);
```

## 55.9 Configuring Freeform Activity Size and Position

By default, an activity launched into a different task stack while in freeform mode will be positioned in the center of the screen at a size dictated by the system. The location and dimensions of this window can be controlled by passing *launch bounds* settings to the intent via the *ActivityOptions* class. The first step in this process is to create a *Rect* object configured with the left (X), top (Y), right (X) and bottom (Y) coordinates of the rectangle representing the activity window. The following code, for example, creates a Rect object in which the top-left corner is positioned at coordinate (100, 800) and the bottom-right at (900, 700):

```
Rect rect = new Rect(100, 800, 900, 700);
```

The next step is to create a basic instance of the ActivityOptions class and initialize it with the Rect settings via the *setLaunchBounds()* method:

```
ActivityOptions options = ActivityOptions.makeBasic();
ActivityOptions bounds = options.setLaunchBounds(rect);
```

Finally, the ActivityOptions instance is converted to a Bundle object and passed to the *startActivity()* method along with the Intent object:

```
startActivity(i, bounds.toBundle());
```

Combining these steps results in a code sequence that reads as follows:

```
Intent i = new Intent(this, SecondActivity.class);
i.addFlags(Intent.FLAG_ACTIVITY_LAUNCH_ADJACENT|
           Intent.FLAG_ACTIVITY_MULTIPLE_TASK|
           Intent.FLAG_ACTIVITY_NEW_TASK);

Rect rect = new Rect(100, 800, 900, 700);

ActivityOptions options = ActivityOptions.makeBasic();
ActivityOptions bounds = options.setLaunchBounds(rect);

startActivity(i, bounds.toBundle());
```

When the second activity is launched by the intent while the originating activity is in freeform mode, the new activity window will appear with the location and dimensions defined in the Rect object.

## 55.10 Summary

Android 7 introduced multi-window support, a system whereby more than one activity is displayed on the screen at any one time. The three modes provided by multi-window support are split-screen, freeform and picture-in-picture. In split-screen mode, the screen is split either horizontally or vertically into two panes with an activity displayed in each pane. Freeform mode, which is only supported on certain Android devices, allows each activity to appear in a separate, movable and resizable window. Picture-in-picture mode is only available on Android TV and allows video

playback to continue in a small window while the user is performing other tasks.

As outlined in this chapter, a number of methods and property settings are available within the Android SDK to detect, respond to and control multi-window behavior within an app.

# 56. An Android Studio Multi-Window Split-Screen and Freeform Tutorial

With the basics of Android multi-window support covered in the previous chapter, this chapter will work through the steps involved in implementing multi-window support within an Android app. This project will be used to demonstrate the steps involved in configuring and managing both split-screen and freeform behavior within a multi-activity app.

## 56.1 Creating the Multi-Window Project

Start Android Studio and create a new project, entering *MultiWindow* into the Application name field and *ebookfrenzy.com* as the Company Domain setting before clicking on the *Next* button.

On the form factors screen, enable the *Phone and Tablet* option and set the minimum SDK setting to API 24: Android 7.0 (Nougat). Continue through the remaining setup screens, requesting the creation of an Empty Activity named *FirstActivity* with a corresponding layout file named *activity_first*.

## 56.2 Designing the FirstActivity User Interface

The user interface will need to be comprised of a single Button and a TextView. Within the Project tool window, navigate to the *activity_first.xml* layout file located in *app -> res -> layout* and double-click on it to load it into the Layout Editor tool. With the tool in Design mode, select and delete the *Hello World!* TextView object.

With Autoconnect mode enabled in the Layout Editor toolbar, drag a TextView widget from the palette and position it in the center of the layout. Next, drag a Button object and position it beneath the TextView. Edit the text on the Button so that it reads “Launch”. If any constraints are missing from the layout, simply click on the Infer Constraints button in the Layout Editor toolbar to add them. On completion of these steps, the layout should resemble that shown in Figure 56-1:

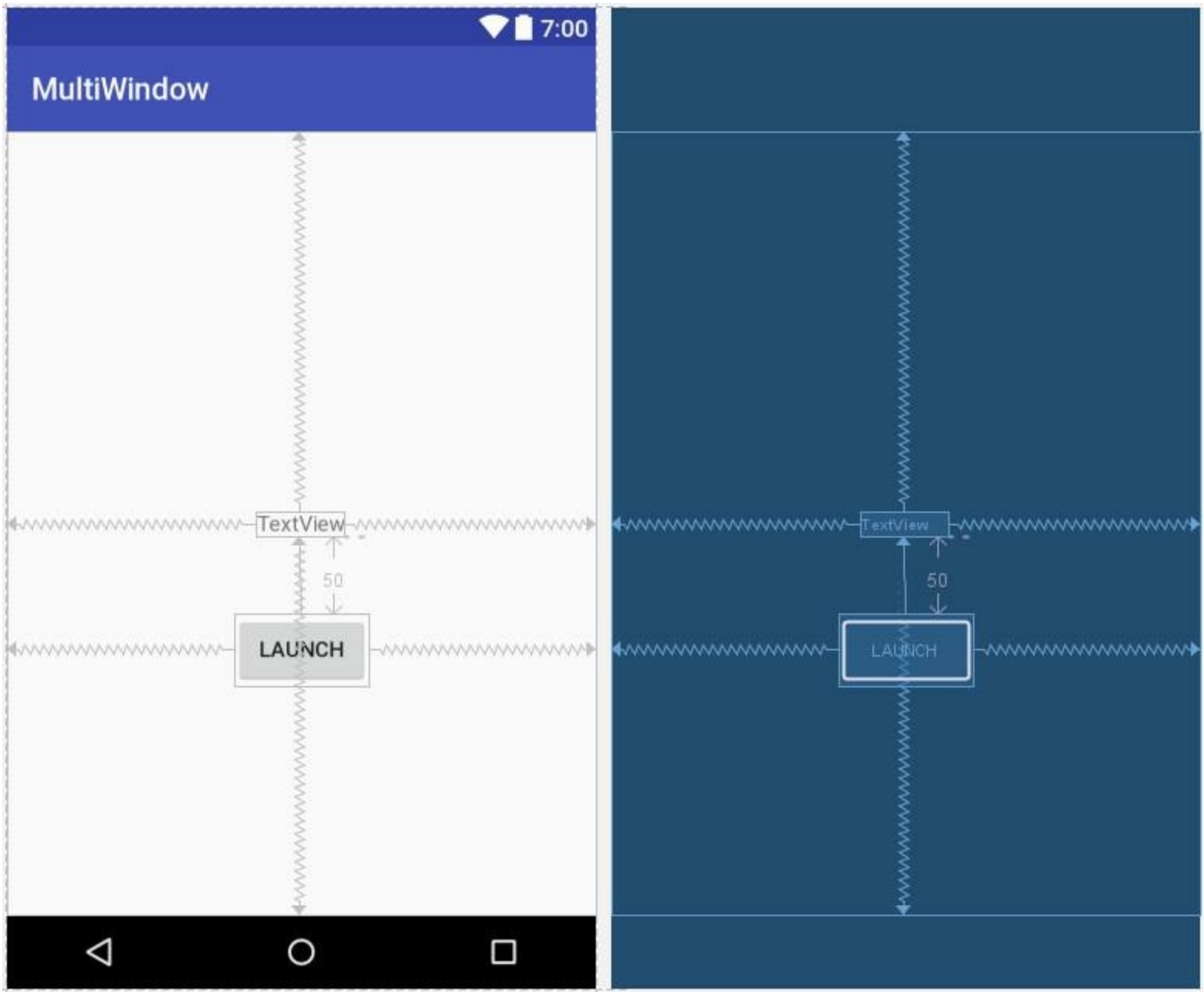


Figure 56-1

In the properties panel, change the widget ID for the TextView to *myTextView* and assign an *onClick* property to the button so that it calls a method named *launchIntent* when selected by the user.

## 56.3 Adding the Second Activity

The second activity will be launched when the user clicks on the button in the first activity. Add this new activity by right-clicking on the *com.ebookfrenzy.multiwindow* package name located in *app -> java* and select the *New -> Activity -> Empty Activity* menu option to display the *New Android Activity* dialog.

Enter *SecondActivity* into the Activity Name and Title fields and name the layout file *activity_second*. Since this activity will not be started when the application is launched (it will instead be launched via an intent by FirstActivity when the button is pressed), it is important to make sure that the *Launcher Activity* option is disabled before clicking on the Finish button.

Open the layout for the second activity (*app -> res -> layout -> activity_second.xml*) and convert the layout to a ConstraintLayout if necessary. Drag and drop a TextView widget so that it is positioned in the center of the layout. Edit the text of the TextView so that it reads “Second Activity”

and set the layout_width property to *wrap_content*:

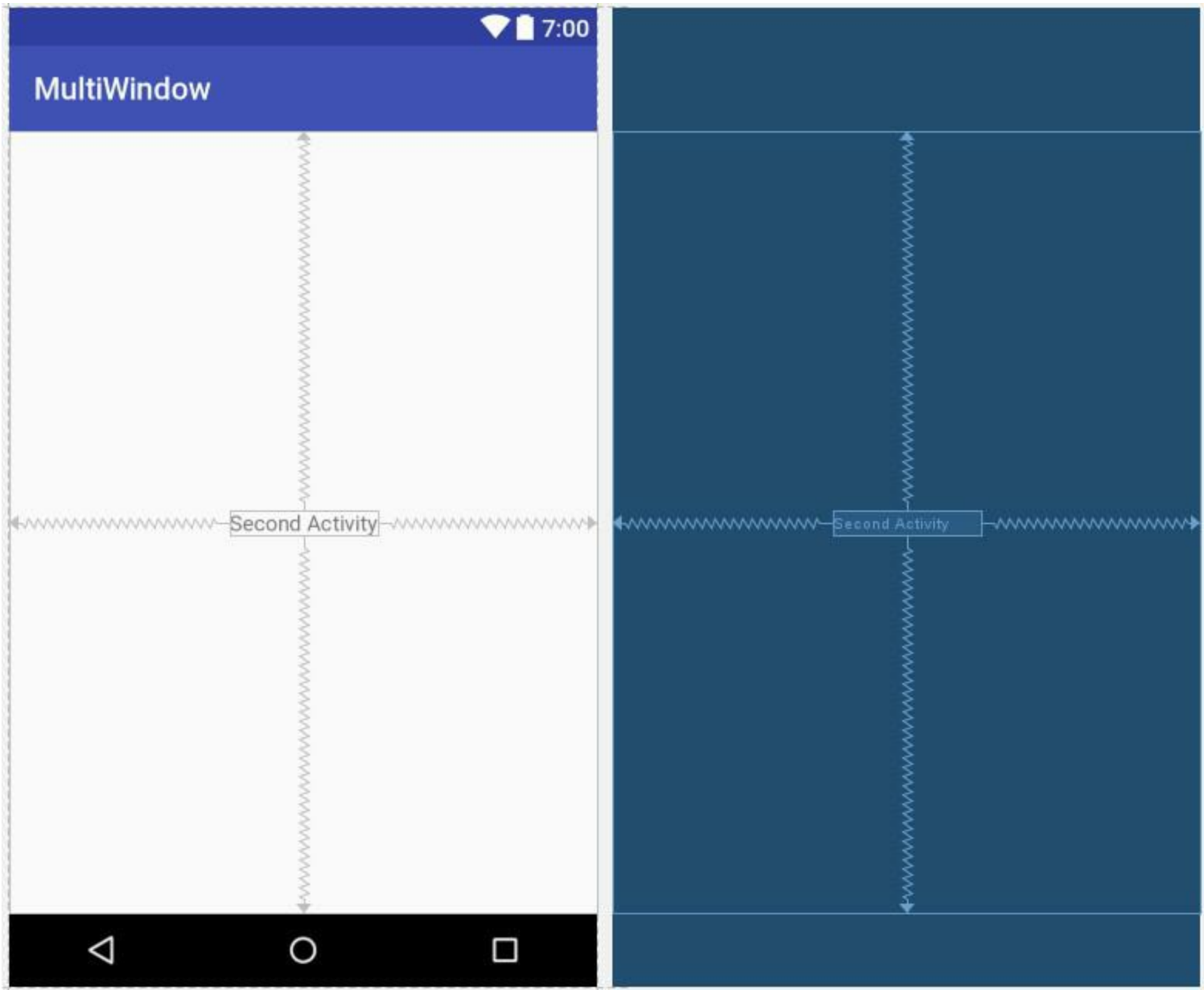


Figure 56-2

## 56.4 Launching the Second Activity

The next step is to add some code to the *FirstActivity.java* class file to implement the *launchIntent()* method. Edit the *FirstActivity.java* file and implement this method as follows:

```
package com.ebookfrenzy.multiwindow;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;

public class FirstActivity extends AppCompatActivity {

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

```
    }

    public void launchIntent(View view) {
        Intent i = new Intent(this, SecondActivity.class);
        startActivity(i);
    }
}
```

Compile and run the app and verify that the second activity is launched when the Launch button is clicked.

## 56.5 Enabling Multi-Window Mode

Edit the *AndroidManifest.xml* file and add the directive to enable multi-window support for the app as follows:

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity
            android:name=".FirstActivity"
            android:resizeableActivity="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

Note that, at the time of writing, multi-window support is enabled by default. The above step, however, is recommended for the purposes of completeness and to defend against the possibility that this default behavior may change in the future.

## 56.6 Testing Multi-Window Support

Build and run the app once again and, once running, press and hold the Overview button as outlined in the chapter entitled *An Introduction to Android 7 Multi-Window Support* to switch to split-screen mode. From the Overview screen in the second half of the screen, choose an app to appear in the adjacent panel:

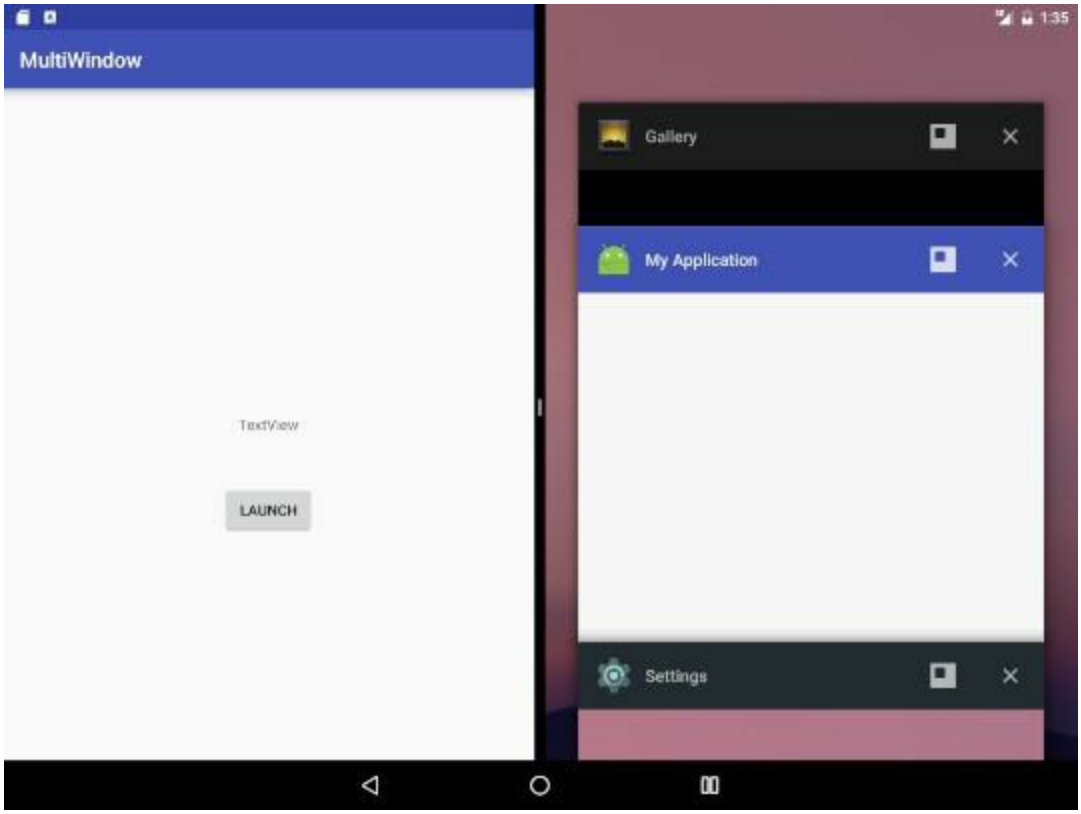


Figure 56-3

Click on the Launch button and note that the second activity appears in the same panel as the first activity.

If the app is running on a device or emulator session that supports freeform mode, press and hold the Overview button a second time until multi-window mode exits. Click in the Overview button once again and, in the resulting Overview screen, select the freeform button located in the title bar of the MultiWindow app as outlined in Figure 56-4:

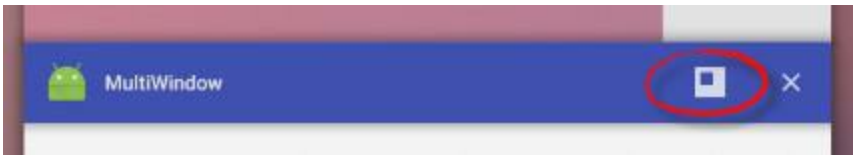


Figure 56-4

Once selected, the activity should appear in freeform mode as illustrated in Figure 56-5:

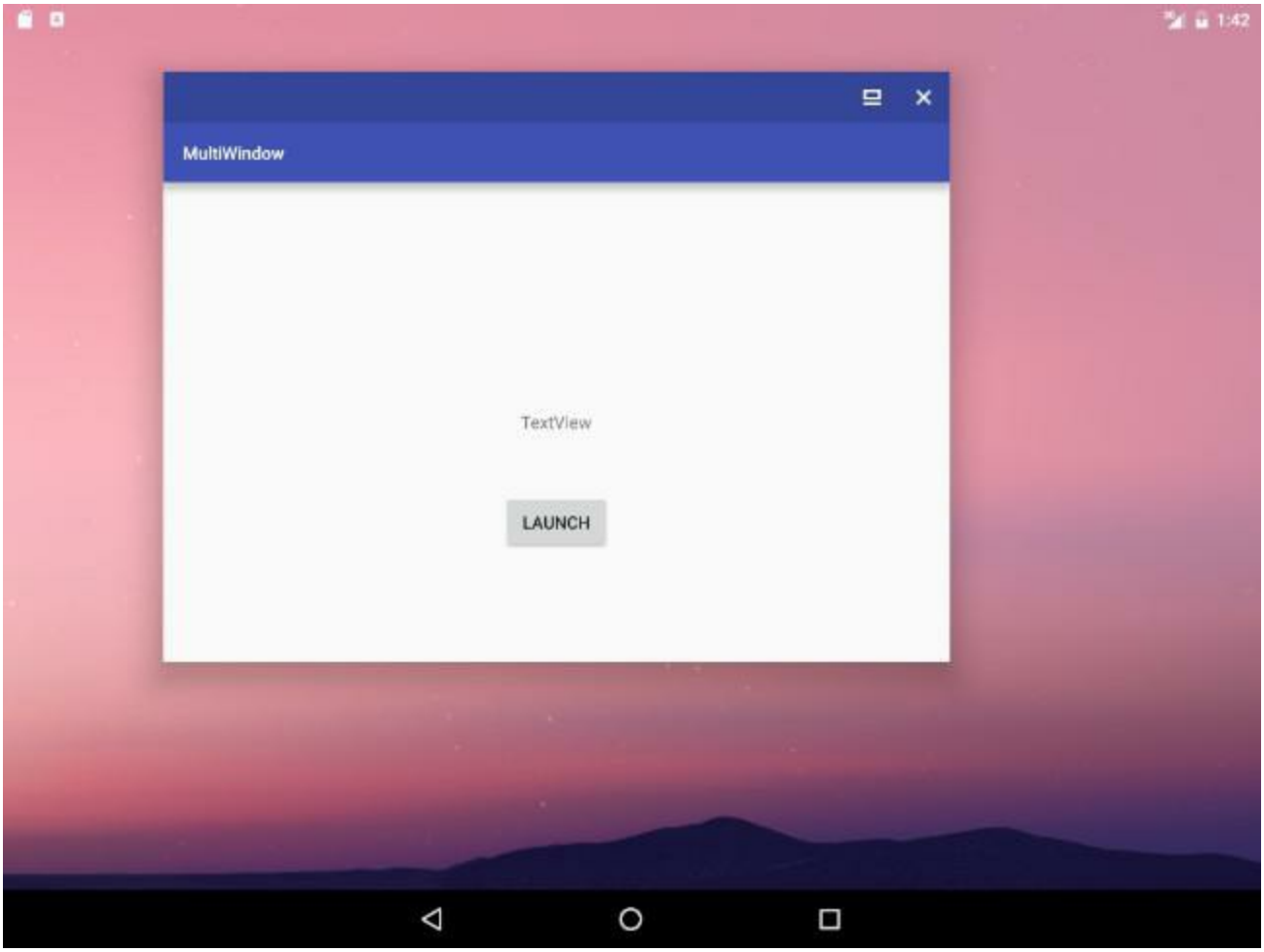


Figure 56-5

Click on the Launch button and note that, once again, the second activity appears in place of the first rather than in a separate window.

In order for the second activity to appear in a different split-screen panel or freeform window, the intent must be launched with the appropriate flags set.

## 56.7 Launching the Second Activity in a Different Window

To prevent the second activity from replacing the first activity the *launchIntent()* method needs to be modified to launch the second activity in a different task stack as follows:

```
public void launchIntent(View view) {
    Intent i = new Intent(this, SecondActivity.class);

    i.addFlags(Intent.FLAG_ACTIVITY_LAUNCH_ADJACENT|
            Intent.FLAG_ACTIVITY_MULTIPLE_TASK|
            Intent.FLAG_ACTIVITY_NEW_TASK);

    startActivity(i);
}
```

After making this change, rerun the app, enter split-screen mode and launch the second activity. The second activity should now appear in the panel adjacent to the first activity:

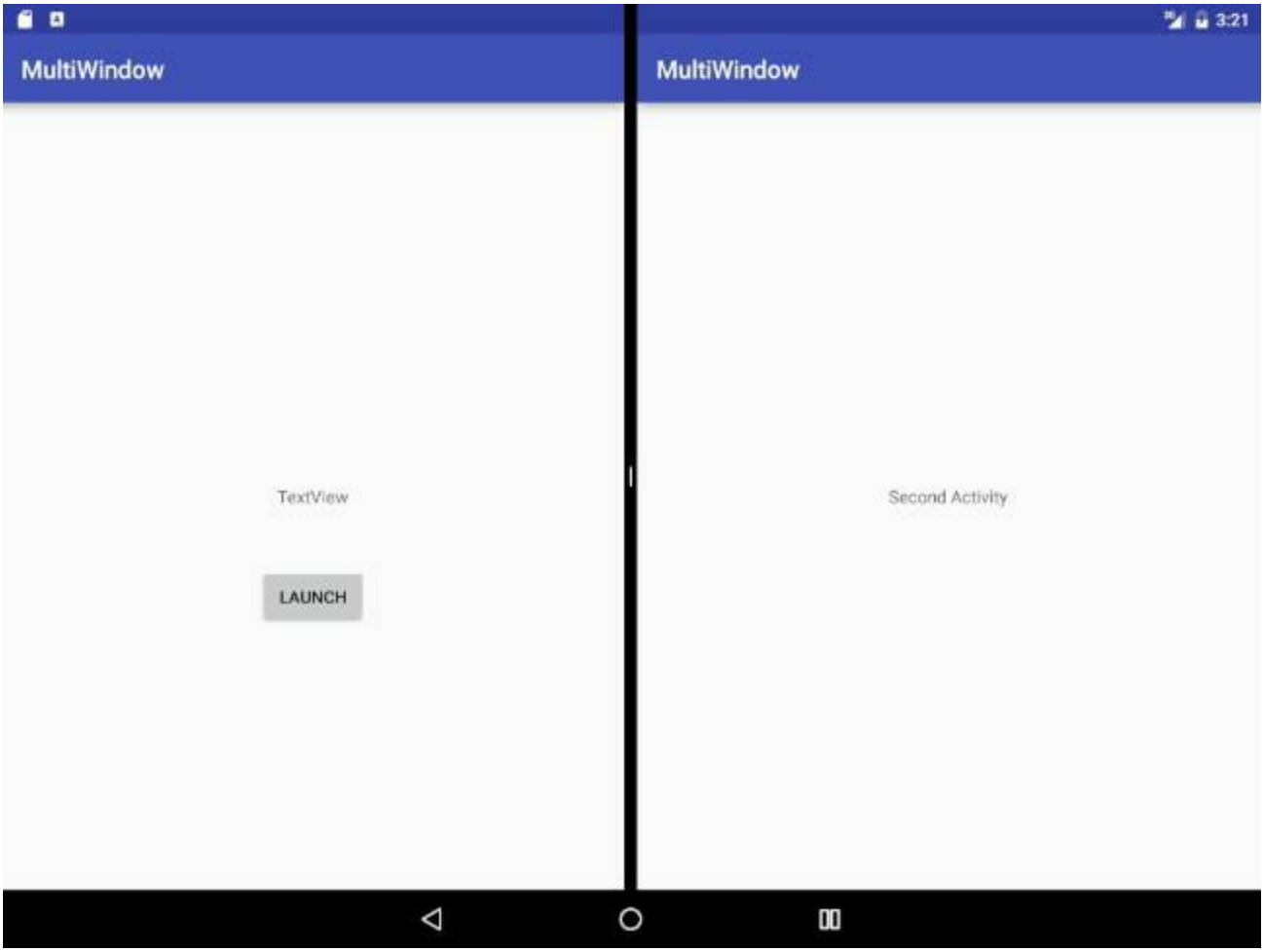


Figure 56-6

Repeat the steps from the previous section to enter freeform mode and verify that the second activity appears in a separate window from the first as shown in Figure 56-7:

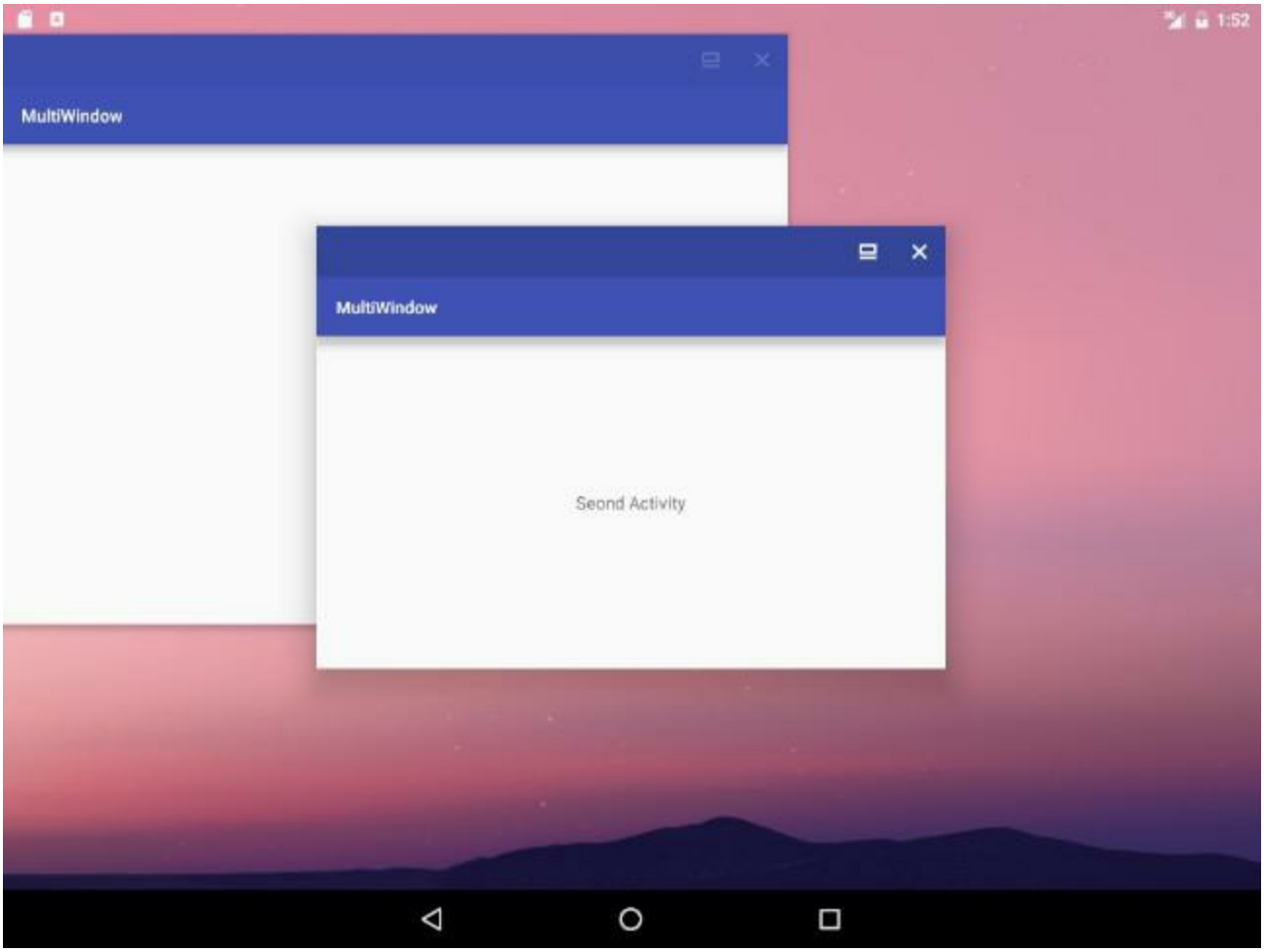


Figure 56-7

## 56.8 Changing the Freeform Window Position and Size

Each time the second activity has launched in a separate window in freeform mode it has appeared in the center of the screen. As a final example, modify the *launchIntent()* method to configure the second activity so that it appears in the top left-hand corner of the screen with dimensions of 100 by 100:

```
package com.ebookfrenzy.multiwindow;

import android.app.ActivityOptions;
import android.content.Intent;
import android.graphics.Rect;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;



import static com.ebookfrenzy.multiwindow.R.id.myTextView;

public class FirstActivity extends AppCompatActivity {

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

    public void launchIntent(View view) {
```

```
        Intent i = new Intent(this, SecondActivity.class);

        i.addFlags(Intent.FLAG_ACTIVITY_LAUNCH_ADJACENT|
                Intent.FLAG_ACTIVITY_MULTIPLE_TASK|
                Intent.FLAG_ACTIVITY_NEW_TASK);

        Rect rect = new Rect(0, 0, 100, 100);

        ActivityOptions options = ActivityOptions.makeBasic();
        ActivityOptions bounds = options.setLaunchBounds(rect);

        startActivity(i, bounds.toBundle());

    }
```

Run the app one last time, enter freeform mode and launch the second activity. Verify that the size and position of the second window matches the specified configuration options.

## 56.9 Summary

This chapter has demonstrated some of the basics of enabling and working with multi-window support within an Android app through the implementation of an example project. In particular, this example has focused on enabling multi-window support, launching a second activity into a new task stack and configuring the size and location of a freeform window.