20. An Android Studio Layout Editor ConstraintLayout Tutorial

By far the easiest and most productive way to design a user interface for an Android application is to make use of the Android Studio Layout Editor tool. The goal of this chapter is to provide an overview of how to create a ConstraintLayout-based user interface using this approach. The exercise included in this chapter will also be used as an opportunity to outline the creation of an activity starting with a "bare-bones" Android Studio project.

Having covered the use of the Android Studio Layout Editor, the chapter will also introduce the Layout Inspector tool.

20.1 An Android Studio Layout Editor Tool Example

The first step in this phase of the example is to create a new Android Studio project. Begin, therefore, by launching Android Studio and closing any previously opened projects by selecting the *File -> Close Project* menu option. Within the Android Studio welcome screen click on the *Start a new Android Studio project* quick start option to display the first screen of the new project dialog.

Enter *LayoutSample* into the Application name field and *ebookfrenzy.com* as the Company Domain setting before clicking on the *Next* button and setting the minimum SDK to API 14: Android 4.0 (IceCreamSandwich).

In previous examples, we have requested that Android Studio create a template activity for the project. We will, however, be using this tutorial to learn how to create an entirely new activity and corresponding layout resource file manually, so click *Next* once again and make sure that the *Add No Activity* option is selected before clicking on *Finish* to create the new project.

20.2 Creating a New Activity

Once the project creation process is complete, the Android Studio main window should appear with no tool windows open.

The next step in the project is to create a new activity. This will be a valuable learning exercise since there are many instances in the course of developing

Android applications where new activities need to be created from the ground up.

Begin by displaying the Project tool window using the Alt-1/Cmd-1 keyboard shortcut. Once displayed, unfold the hierarchy by clicking on the right facing arrows next to the entries in the Project window. The objective here is to gain access to the *app -> java -> com.ebookfrenzy.layoutsample* folder in the project hierarchy. Once the package name is visible, right-click on it and select the *New -> Activity -> Empty Activity* menu option as illustrated in Figure 20-1:

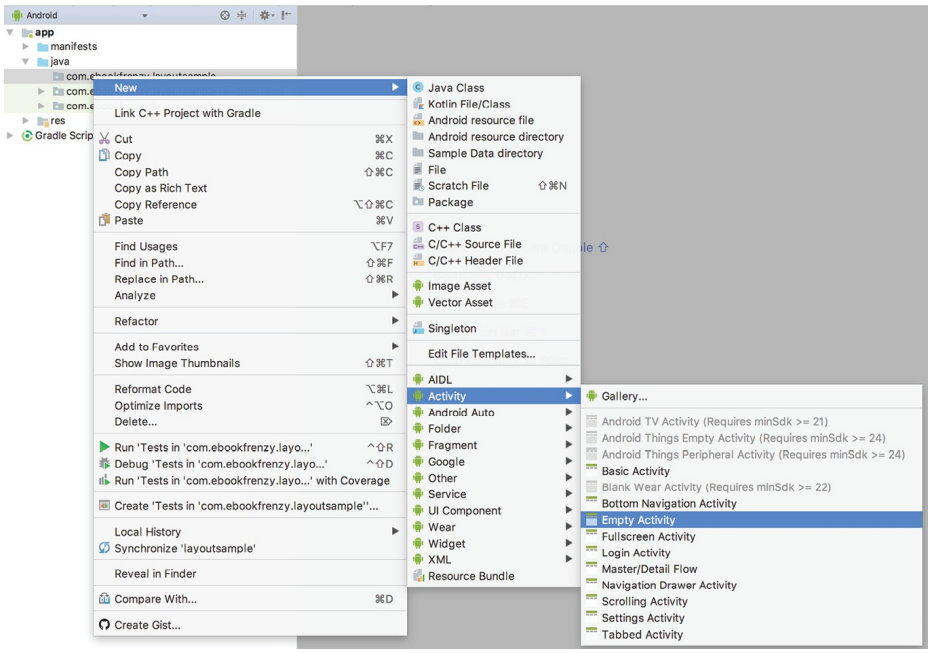

Figure 20-1

In the resulting *New Activity* dialog, name the new activity *LayoutSampleActivity* and the layout *activity_layout_sample*. The activity will, of course, need a layout resource file so make sure that the *Generate Layout File* option is enabled.

In order for an application to be able to run on a device it needs to have an activity designated as the *launcher activity*. Without a launcher activity, the operating system will not know which activity to start up when the application first launches and the application will fail to start. Since this example only has one activity, it needs to be designated as the launcher activity for the application so make sure that the *Launcher Activity* option is enabled before clicking on the *Finish* button.

At this point Android Studio should have added two files to the project. The Java source code file for the activity should be located in the *app -> java -> com.ebookfrenzy.layoutsample* folder.

In addition, the XML layout file for the user interface should have been created in the *app -> res -> layout* folder. Note that the Empty Activity template was chosen for this activity so the layout is contained entirely within the *activity_layout_sample.xml* file and there is no separate content layout file.

Finally, the new activity should have been added to the *AndroidManifest.xml* file and designated as the launcher activity. The manifest file can be found in the project window under the *app -> manifests* folder and should contain the following XML:

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

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

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

</manifest>
```

20.3 Preparing the Layout Editor Environment

Locate and double-click on the *activity_layout_sample.xml* layout file located in the *app -> res -> layout* folder to load it into the Layout Editor tool. Since the purpose of this tutorial is to gain experience with the use of constraints,

turn off the Autoconnect feature using the button located in the Layout Editor toolbar. Once disabled, the button will appear with a line through it as is the case in Figure 20-2:

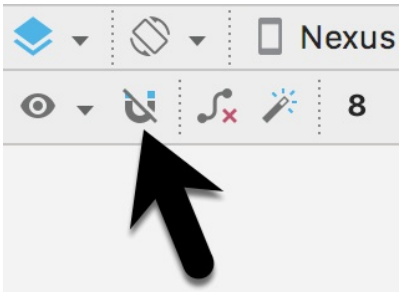

Figure 20-2

The user interface design will also make use of the ImageView object to display an image. Before proceeding, this image should be added to the project ready for use later in the chapter. This file is named *galaxys6.png* and can be found in the *project_icons* folder of the sample code download available from the following URL:

http://www.ebookfrenzy.com/retail/androidstudio30/index.php

Locate this image in the file system navigator for your operating system and copy the image file. Right-click on the *app* -> *res* -> *drawable* entry in the Project tool window and select Paste from the menu to add the file to the folder. When the copy dialog appears, click on OK to accept the default settings.

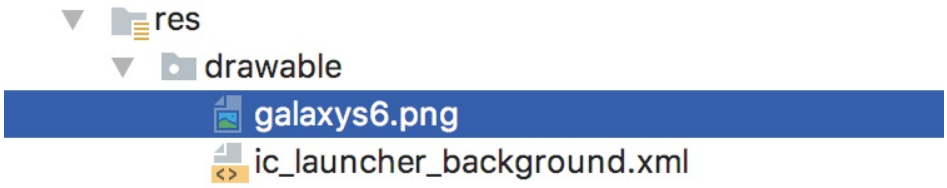

Figure 20-3

20.4 Adding the Widgets to the User Interface

From within the *Images* palette category, drag an ImageView object into the center of the display view. Note that horizontal and vertical dashed lines appear indicating the center axes of the display. When centered, release the mouse button to drop the view into position. Once placed within the layout, the Resources dialog will appear seeking the image to be displayed within the view. In the search bar located at the top of the dialog, enter "galaxy" to locate the *galaxys6.png* resource as illustrated in Figure 20-4.

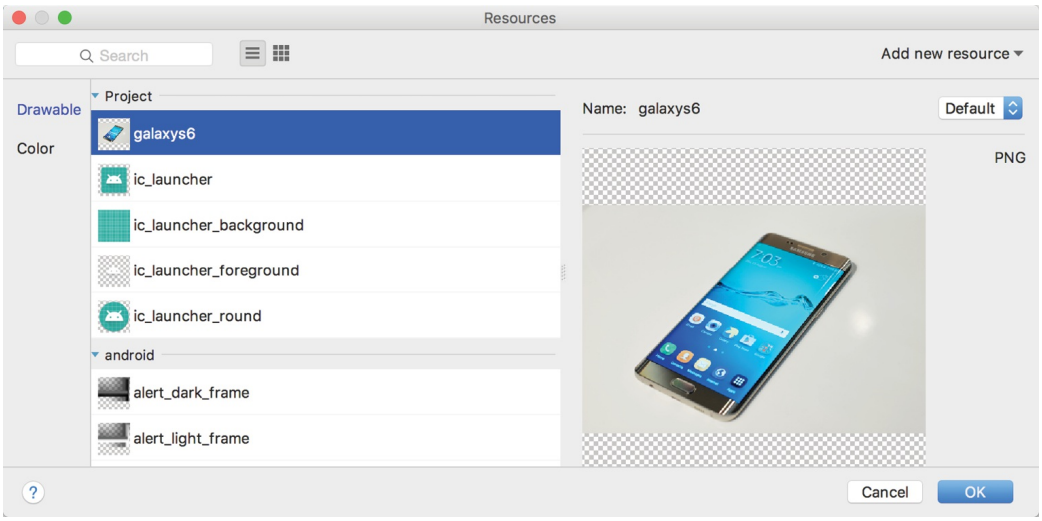

Figure 20-4

Select the image and click on OK to assign it to the ImageView object. If necessary, adjust the size of the ImageView using the resize handles and reposition it in the center of the layout. At this point the layout should match Figure 20-5:

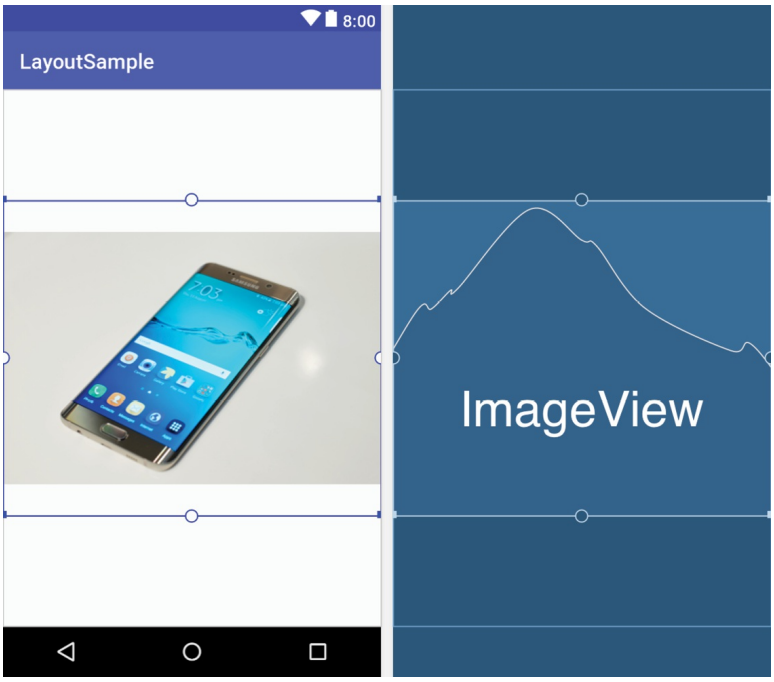

Figure 20-5

Click and drag a TextView object from the Text section of the palette and position it so that it appears above the ImageView as illustrated in Figure 20-6.

Using the Attributes panel, change the *textSize* property to 24sp, the

textAlignment setting to center and the text to "Samsung Galaxy S6".

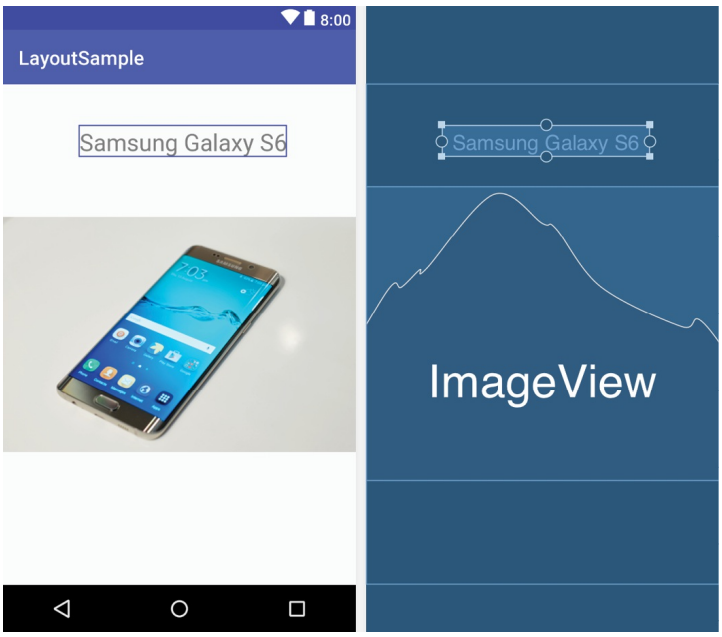

Figure 20-6

Next, add three Button widgets along the bottom of the layout and set the text attributes of these views to "Buy Now", "Pricing" and "Details". The completed layout should now match Figure 20-7:

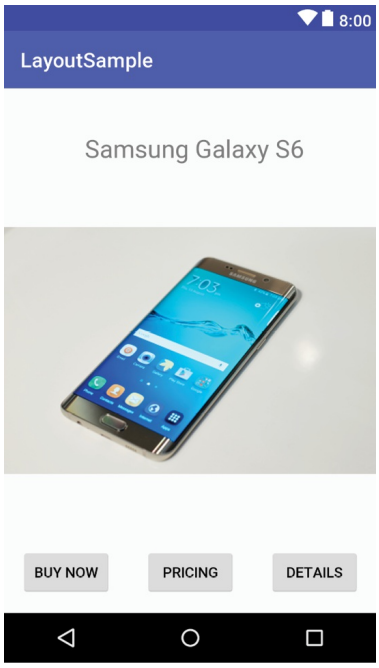

Figure 20-7

At this point, the widgets are not sufficiently constrained for the layout engine to be able to position and size the widgets at runtime. Were the app to

run now, all of the widgets would be positioned in the top left-hand corner of the display.

With the widgets added to the layout, use the device rotation button located in the Layout Editor toolbar (indicated by the arrow in Figure 20-8) to view the user interface in landscape orientation:

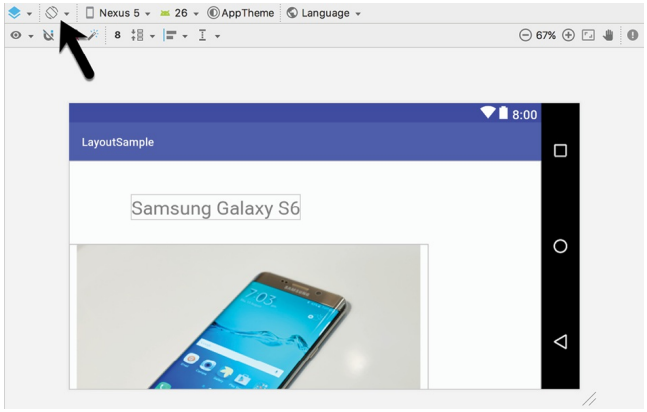

Figure 20-8

The absence of constraints results in a layout that fails to adapt to the change in device orientation, leaving the content off center and with part of the image and all three buttons positioned beyond the viewable area of the screen. Clearly some work still needs to be done to make this into a responsive user interface.

20.5 Adding the Constraints

Constraints are the key to creating layouts that can adapt to device orientation changes and different screen sizes. Begin by rotating the layout back to portrait orientation and selecting the TextView widget located above the ImageView. With the widget selected, establish constraints from the left, right and top sides of the TextView to the corresponding sides of the parent ConstraintLayout as shown in Figure 20-9:

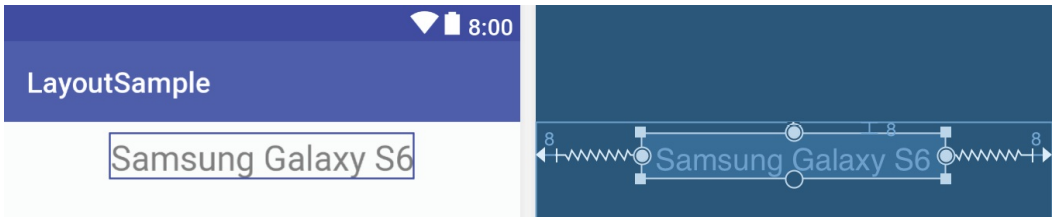

Figure 20-9

With the TextView widget constrained, select the ImageView instance and establish opposing constraints on the left and right-hand sides with each

connected to the corresponding sides of the parent layout. Next, establish a constraint connection from the top of the ImageView to the bottom of the TextView and from the bottom of the ImageView to the top of the center Button widget. If necessary, click and drag the ImageView so that it is still positioned in the vertical center of the layout.

With the ImageView still selected, use the Inspector in the attributes panel to change the top and bottom margins on the ImageView to 24 and 8 respectively and to change both the widget height and width dimension properties to *match_constraint* so that the widget will resize to match the constraints. These settings will allow the layout engine to enlarge and reduce the size of the ImageView when necessary to accommodate layout changes:

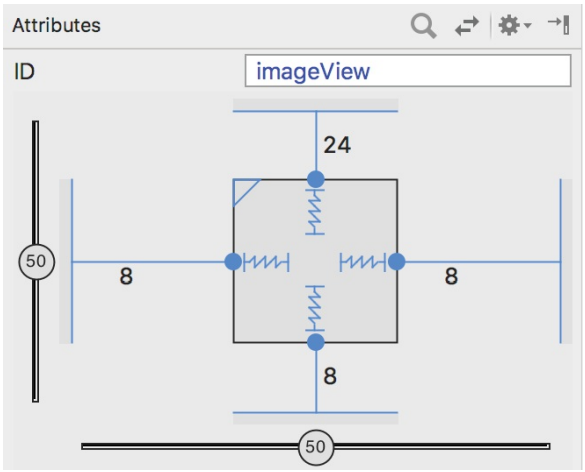

Figure 20-10

Figure 20-11, shows the currently implemented constraints for the ImageView in relation to the other elements in the layout:

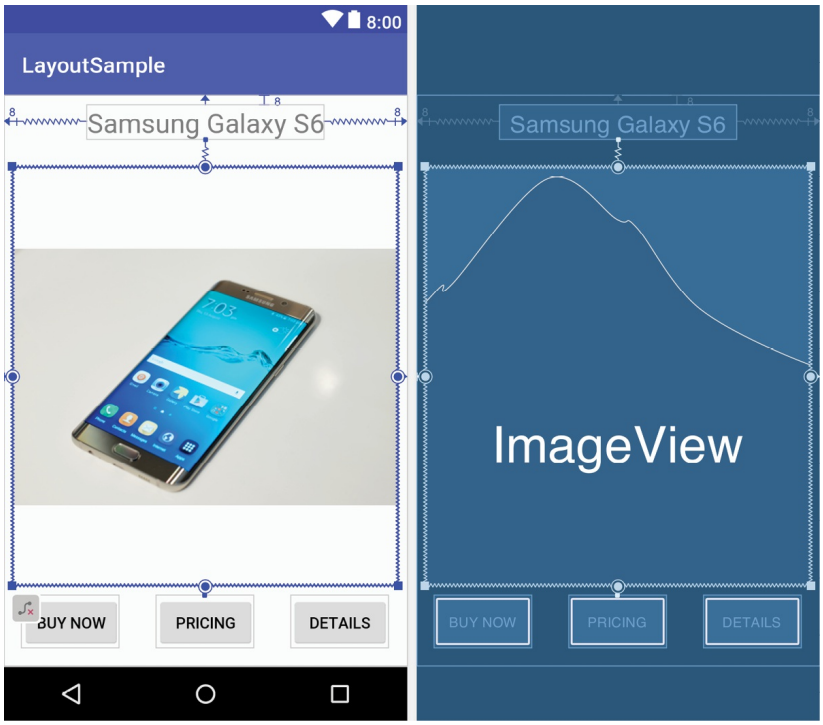

Figure 20-11

The final task is to add constraints to the three Button widgets. For this example, the buttons will be placed in a chain. Begin by turning on Autoconnect within the Layout Editor by clicking the toolbar button highlighted in Figure 20-2.

Next, click on the Buy Now button and then shift-click on the other two buttons so that all three are selected. Right-click on the Buy Now button and select the *Chain -> Create Horizontal Chain* menu option from the resulting menu. By default, the chain will be displayed using the spread style which is the correct behavior for this example.

Finally, establish a constraint between the bottom of the Buy Now button and the bottom of the layout. Repeat this step for the remaining buttons.

On completion of these steps the buttons should be constrained as outlined in Figure 20-12:

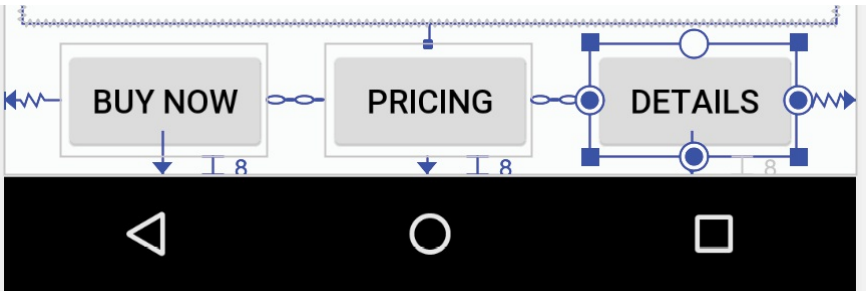

Figure 20-12

20.6 Testing the Layout

With the constraints added to the layout, rotate the screen into landscape orientation and verify that the layout adapts to accommodate the new screen dimensions.

While the Layout Editor tool provides a useful visual environment in which to design user interface layouts, when it comes to testing there is no substitute for testing the running app. Launch the app on a physical Android device or emulator session and verify that the user interface reflects the layout created in the Layout Editor. Figure 20-13, for example, shows the running app in landscape orientation:

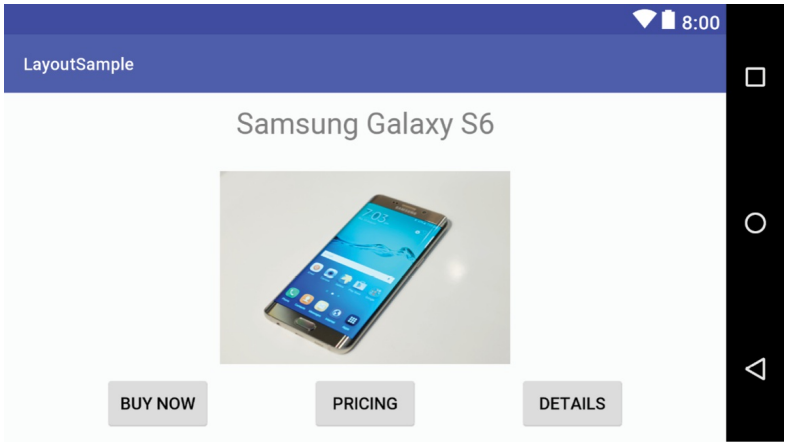

Figure 20-13

The very simple user interface design is now complete. Designing a more complex user interface layout is a continuation of the steps outlined above. Simply drag and drop views onto the display, position, constrain and set properties as needed.

20.7 Using the Layout Inspector

The hierarchy of components that make up a user interface layout may be viewed at any time using the Layout Inspector tool. In order to access this information the app must be running on a device or emulator. Once the app is running, select the *Tools -> Android -> Layout Inspector* menu option followed by the process to be inspected.

Once the inspector loads, the left most panel (A) shows the hierarchy of components that make up the user interface layout. The center panel (B) shows a visual representation of the layout design. Clicking on a widget in the visual layout will cause that item to highlight in the hierarchy list making it easy to find where a visual component is situated relative to the overall layout hierarchy.

Finally, the rightmost panel (marked C in Figure 20-14) contains all of the property settings for the currently selected component, allowing for in-depth analysis of the component's internal configuration.

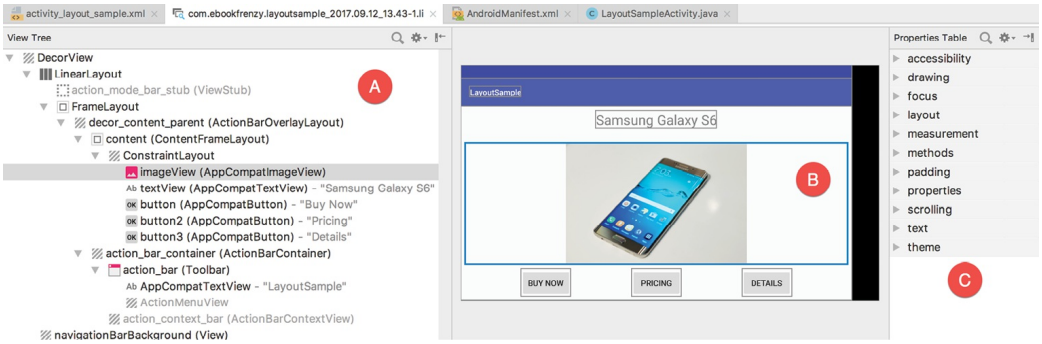

Figure 20-14

20.8 Summary

The Layout Editor tool in Android Studio has been tightly integrated with the ConstraintLayout class. This chapter has worked through the creation of an example user interface intended to outline the ways in which a ConstraintLayout-based user interface can be implemented using the Layout Editor tool in terms of adding widgets and setting constraints. This chapter also introduced the Layout Inspector tool which is useful for analyzing the structural composition of a user interface layout.