26. Android Touch and Multi-touch Event Handling

Most Android based devices use a touch screen as the primary interface between user and device. The previous chapter introduced the mechanism by which a touch on the screen translates into an action within a running Android application. There is, however, much more to touch event handling than responding to a single finger tap on a view object. Most Android devices can, for example, detect more than one touch at a time. Nor are touches limited to a single point on the device display. Touches can, of course, be dynamic as the user slides one or more points of contact across the surface of the screen.

Touches can also be interpreted by an application as a *gesture*. Consider, for example, that a horizontal swipe is typically used to turn the page of an eBook, or how a pinching motion can be used to zoom in and out of an image displayed on the screen.

The objective of this chapter is to highlight the handling of touches that involve motion and to explore the concept of intercepting multiple concurrent touches. The topic of identifying distinct gestures will be covered in the next chapter.

26.1 Intercepting Touch Events

Touch events can be intercepted by a view object through the registration of an *onTouchListener* event listener and the implementation of the corresponding *onTouch()* callback method. The following code, for example, ensures that any touches on a ConstraintLayout view instance named *myLayout* result in a call to the *onTouch()* method:

```
myLayout.setOnTouchListener(
       new ConstraintLayout.OnTouchListener() {
              public boolean onTouch(View v, MotionEvent m) {
                     // Perform tasks here

                     return true;
              }
       }
);
```

As indicated in the code example, the *onTouch()* callback is required to return a Boolean value indicating to the Android runtime system whether or not the event should be passed on to other event listeners registered on the same view or discarded. The method is passed both a reference to the view on which the event was triggered and an object of type *MotionEvent*.

26.2 The MotionEvent Object

The MotionEvent object passed through to the *onTouch()* callback method is the key to obtaining information about the event. Information contained within the object includes the location of the touch within the view and the type of action performed. The MotionEvent object is also the key to handling multiple touches.

26.3 Understanding Touch Actions

An important aspect of touch event handling involves being able to identify the type of action performed by the user. The type of action associated with an event can be obtained by making a call to the *getActionMasked()* method of the MotionEvent object which was passed through to the *onTouch()* callback method. When the first touch on a view occurs, the MotionEvent object will contain an action type of ACTION_DOWN together with the coordinates of the touch. When that touch is lifted from the screen, an ACTION_UP event is generated. Any motion of the touch between the ACTION_DOWN and ACTION_UP events will be represented by ACTION_MOVE events.

When more than one touch is performed simultaneously on a view, the touches are referred to as *pointers*. In a multi-touch scenario, pointers begin and end with event actions of type ACTION_POINTER_DOWN and ACTION_POINTER_UP respectively. In order to identify the index of the pointer that triggered the event, the *getActionIndex()* callback method of the MotionEvent object must be called.

26.4 Handling Multiple Touches

The chapter entitled *"An Overview and Example of Android Event Handling"* began exploring event handling within the narrow context of a single touch event. In practice, most Android devices possess the ability to respond to multiple consecutive touches (though it is important to note that the number

of simultaneous touches that can be detected varies depending on the device).

As previously discussed, each touch in a multi-touch situation is considered by the Android framework to be a *pointer*. Each pointer, in turn, is referenced by an *index* value and assigned an *ID*. The current number of pointers can be obtained via a call to the *getPointerCount()* method of the current MotionEvent object. The ID for a pointer at a particular index in the list of current pointers may be obtained via a call to the MotionEvent *getPointerId()* method. For example, the following code excerpt obtains a count of pointers and the ID of the pointer at index 0:

```
public boolean onTouch(View v, MotionEvent m) {
       int pointerCount = m.getPointerCount();
       int pointerId = m.getPointerId(0);
       return true;
}
```

Note that the pointer count will always be greater than or equal to 1 when the *onTouch* listener is triggered (since at least one touch must have occurred for the callback to be triggered).

A touch on a view, particularly one involving motion across the screen, will generate a stream of events before the point of contact with the screen is lifted. As such, it is likely that an application will need to track individual touches over multiple touch events. While the ID of a specific touch gesture will not change from one event to the next, it is important to keep in mind that the index value will change as other touch events come and go. When working with a touch gesture over multiple events, therefore, it is essential that the ID value be used as the touch reference in order to make sure the same touch is being tracked. When calling methods that require an index value, this should be obtained by converting the ID for a touch to the corresponding index value via a call to the *findPointerIndex()* method of the *MotionEvent* object.

26.5 An Example Multi-Touch Application

The example application created in the remainder of this chapter will track up to two touch gestures as they move across a layout view. As the events for each touch are triggered, the coordinates, index and ID for each touch will be displayed on the screen.

Create a new project in Android Studio, entering *MotionEvent* 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 14: Android 4.0 (IceCreamSandwich). Continue to proceed through the screens, requesting the creation of an Empty Activity named *MotionEventActivity* with a corresponding layout file named *activity_motion_event*.

Click on the *Finish* button to initiate the project creation process.

26.6 Designing the Activity User Interface

The user interface for the application's sole activity is to consist of a ConstraintLayout view containing two TextView objects. Within the Project tool window, navigate to *app* -> *res* -> *layout* and double-click on the *activity_motion_event.xml* layout resource file to load it into the Android Studio Layout Editor tool.

Select and delete the default "Hello World!" TextView widget and then, with autoconnect enabled, drag and drop a new TextView widget so that it is centered horizontally and positioned at the 16dp margin line on the top edge of the layout:

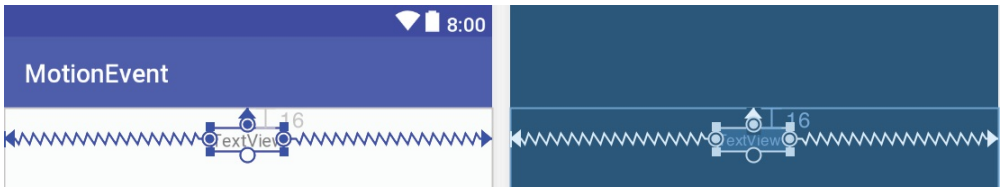

Figure 26-1

Drag a second TextView widget and position and constrain it so that it is distanced by a 32dp margin from the bottom of the first widget:

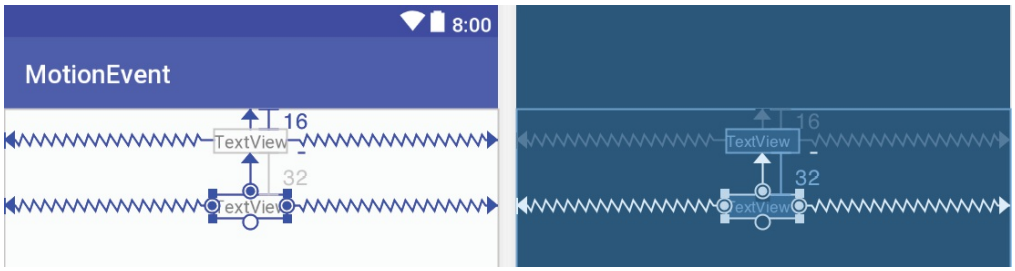

Figure 26-2

Using the Attributes tool window, change the IDs for the TextView widgets to *textView1* and *textView2* respectively. Change the text displayed on the

widgets to read “Touch One Status” and “Touch Two Status” and extract the strings to resources using the warning button in the top right-hand corner of the Layout Editor.

Select the ConstraintLayout entry in the Component Tree and use the Attributes panel to change the ID to *activity_motion_event*.

26.7 Implementing the Touch Event Listener

In order to receive touch event notifications it will be necessary to register a touch listener on the layout view within the *onCreate()* method of the *MotionEventActivity* activity class. Select the *MotionEventActivity.java* tab from the Android Studio editor panel to display the source code. Within the *onCreate()* method, add code to identify the ConstraintLayout view object, register the touch listener and implement code which, in this case, is going to call a second method named *handleTouch()* to which is passed the MotionEvent object:

```
package com.ebookfrenzy.motionevent;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.MotionEvent;
import android.view.View;
import android.support.constraint.ConstraintLayout;
import android.widget.TextView;

public class MotionEventActivity extends AppCompatActivity {

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

        ConstraintLayout myLayout =
         (ConstraintLayout)findViewById(R.id.activity_motion_event);

        myLayout.setOnTouchListener(
                new ConstraintLayout.OnTouchListener() {
                    public boolean onTouch(View v,
                                           MotionEvent m) {
                        handleTouch(m);
                        return true;
```

```
                    }
                }
        );
    }
.
.
.
}
```

The final task before testing the application is to implement the *handleTouch()* method called by the listener. The code for this method reads as follows:

```
void handleTouch(MotionEvent m)
{
            TextView textView1 =
(TextView)findViewById(R.id.textView1);
            TextView textView2 =
(TextView)findViewById(R.id.textView2);

            int pointerCount = m.getPointerCount();

            for (int i = 0; i < pointerCount; i++)
            {
                    int x = (int) m.getX(i);
                    int y = (int) m.getY(i);
                    int id = m.getPointerId(i);
                    int action = m.getActionMasked();
                    int actionIndex = m.getActionIndex();
                    String actionString;


                    switch (action)
                    {
                            case MotionEvent.ACTION_DOWN:
                                    actionString = "DOWN";
                                    break;
                            case MotionEvent.ACTION_UP:
                                    actionString = "UP";
                                    break;
                            case MotionEvent.ACTION_POINTER_DOWN:
                                    actionString = "PNTR DOWN";
                                    break;
                            case MotionEvent.ACTION_POINTER_UP:
```

```
                                actionString = "PNTR UP";
                                break;
                            case MotionEvent.ACTION_MOVE:
                                   actionString = "MOVE";
                                   break;
                            default:
                                   actionString = "";
                    }

                    String touchStatus = "Action: " + actionString + "
Index: " + actionIndex + " ID: " + id + " X: " + x + " Y: " + y;

                    if (id == 0)
                           textView1.setText(touchStatus);
                    else
                           textView2.setText(touchStatus);
             }
}
```

Before compiling and running the application, it is worth taking the time to walk through this code systematically to highlight the tasks that are being performed.

The code begins by obtaining references to the two TextView objects in the user interface and identifying how many pointers are currently active on the view:

```
TextView textView1 = (TextView)findViewById(R.id.textView1);
TextView textView2 = (TextView)findViewById(R.id.textView2);

int pointerCount = m.getPointerCount();
```

Next, the *pointerCount* variable is used to initiate a *for* loop which performs a set of tasks for each active pointer. The first few lines of the loop obtain the X and Y coordinates of the touch together with the corresponding event ID, action type and action index. Lastly, a string variable is declared:

```
for (int i = 0; i < pointerCount; i++)
{
                    int x = (int) m.getX(i);
                    int y = (int) m.getY(i);
                    int id = m.getPointerId(i);
                    int action = m.getActionMasked();
                    int actionIndex = m.getActionIndex();
                    String actionString;
```

Since action types equate to integer values, a *switch* statement is used to convert the action type to a more meaningful string value, which is stored in the previously declared *actionString* variable:

```
switch (action)
       {
            case MotionEvent.ACTION_DOWN:
                 actionString = "DOWN";
                 break;
            case MotionEvent.ACTION_UP:
                 actionString = "UP";
                 break;
            case MotionEvent.ACTION_POINTER_DOWN:
                 actionString = "PNTR DOWN";
                 break;
            case MotionEvent.ACTION_POINTER_UP:
                 actionString = "PNTR UP";
                 break;
            case MotionEvent.ACTION_MOVE:
                 actionString = "MOVE";
                 break;
                 default:
                 actionString = "";
        }
```

Finally, the string message is constructed using the *actionString* value, the action index, touch ID and X and Y coordinates. The ID value is then used to decide whether the string should be displayed on the first or second TextView object:

```
String touchStatus = "Action: " + actionString + " Index: "
         + actionIndex + " ID: " + id + " X: " + x + " Y: " + y;

                  if (id == 0)
                         textView1.setText(touchStatus);
                  else
                         textView2.setText(touchStatus);
```

26.8 Running the Example Application

Compile and run the application and, once launched, experiment with single and multiple touches on the screen and note that the text views update to reflect the events as illustrated in Figure 26-3. When running on an emulator, multiple touches may be simulated by holding down the Ctrl (Cmd on

macOS) key while clicking the mouse button:

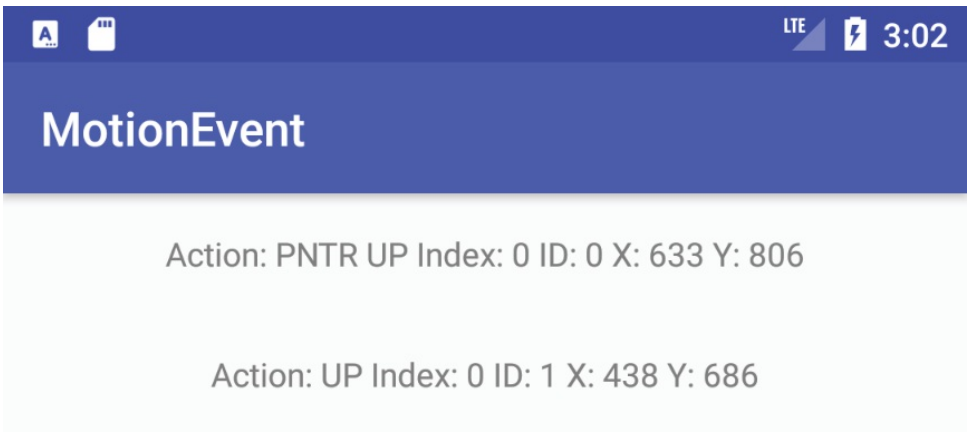

Figure 26-3

26.9 Summary

Activities receive notifications of touch events by registering an onTouchListener event listener and implementing the *onTouch()* callback method which, in turn, is passed a MotionEvent object when called by the Android runtime. This object contains information about the touch such as the type of touch event, the coordinates of the touch and a count of the number of touches currently in contact with the view.

When multiple touches are involved, each point of contact is referred to as a pointer with each assigned an index and an ID. While the index of a touch can change from one event to another, the ID will remain unchanged until the touch ends.

This chapter has worked through the creation of an example Android application designed to display the coordinates and action type of up to two simultaneous touches on a device display.

Having covered touches in general, the next chapter (entitled *"Detecting Common Gestures using the Android Gesture Detector Class"*) will look further at touch screen event handling through the implementation of gesture recognition.