33. An Android Transition Tutorial using beginDelayedTransition

The previous chapter, entitled *"Animating User Interfaces with the Android Transitions Framework"*, provided an introduction to the animation of user interfaces using the Android Transitions framework. This chapter uses a tutorial based approach to demonstrate Android transitions in action using the *beginDelayedTransition()* method of the TransitionManager class.

The next chapter will create a more complex example that uses layout files and transition resource files to animate the transition from one scene to another within an application.

33.1 Creating the Android Studio TransitionDemo Project

Create a new project in Android Studio, entering *TransitionDemo* 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 19: Android 4.4 (KitKat). Continue to proceed through the screens, requesting the creation of an Empty Activity named *TransitionDemoActivity* with a layout resource file named *activity_transition_demo*.

33.2 Preparing the Project Files

The first example transition animation will be implemented through the use of the *beginDelayedTransition()* method of the TransitionManager class. If Android Studio does not automatically load the file, locate and double-click on the *app -> res -> layout -> activity_transition_demo.xml* file in the Project tool window panel to load it into the Layout Editor tool.

Switch the Layout Editor to Design mode, delete the "Hello World!" TextView, drag a Button from the Widget section of the Layout Editor palette and position it in the top left-hand corner of the device screen layout. Once positioned, select the button and use the Attributes tool window to specify an ID value of *myButton*.

Select the ConstraintLayout entry in the Component Tree tool window and use the Attributes window to set the ID to *myLayout*.

33.3 Implementing beginDelayedTransition Animation

The objective for the initial phase of this tutorial is to implement a touch handler so that when the user taps on the layout view the button view moves to the lower right-hand corner of the screen.

Open the *TransitionDemoActivity.java* file (located in the Project tool window under *app -> java -> com.ebookfrenzy.transitiondemo*) and modify the *onCreate()* method to implement the onTouch handler:

```
package com.ebookfrenzy.transitiondemo;

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

public class TransitionDemoActivity extends AppCompatActivity {

    ConstraintLayout myLayout;

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

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

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

The above code simply sets up a touch listener on the ConstraintLayout container and configures it to call a method named *handleTouch()* when a touch is detected. The next task, therefore, is to implement the *handleTouch()* method as follows:

```
public void handleTouch() {
    Button button = (Button) findViewById(R.id.myButton);

    button.setMinimumWidth(500);
    button.setMinimumHeight(350);

    ConstraintSet set = new ConstraintSet();

    set.connect(R.id.myButton, ConstraintSet.BOTTOM,
          ConstraintSet.PARENT_ID, ConstraintSet.BOTTOM, 0);

    set.connect(R.id.myButton, ConstraintSet.RIGHT,
          ConstraintSet.PARENT_ID, ConstraintSet.RIGHT, 0);

    set.constrainWidth(R.id.myButton, ConstraintSet.WRAP_CONTENT);

    set.applyTo(myLayout);
}
```

This method obtains a reference to the button view in the user interface layout and sets new minimum height and width attributes so that the button increases in size.

A ConstraintSet object is then created and configured with constraints that will position the button in the lower right-hand corner of the parent layout. This constraint set is then applied to the layout.

Test the code so far by compiling and running the application. Once launched, touch the background (not the button) and note that the button moves and resizes as illustrated in Figure 33-1:

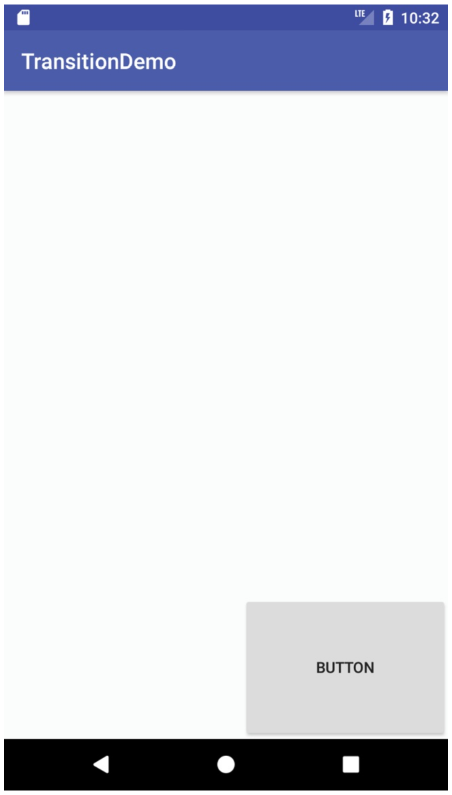

Figure 33-1

Although the layout changes took effect, they did so instantly and without any form of animation. This is where the call to the *beginDelayedTransition()* method of the TransitionManager class comes in. All that is needed to add animation to this layout change is the addition of a single line of code before the layout changes are implemented. Remaining within the *TransitionDemoActivity.java* file, modify the code as follows:

```
.
.
import android.transition.TransitionManager;
.
.
    public void handleTouch() {
        Button button = (Button) findViewById(R.id.myButton);

        TransitionManager.beginDelayedTransition(myLayout);

        ConstraintSet set = new ConstraintSet();

        button.setMinimumWidth(500);
        button.setMinimumHeight(350);
        set.connect(R.id.myButton, ConstraintSet.BOTTOM,
              ConstraintSet.PARENT_ID, ConstraintSet.BOTTOM, 0);

        set.connect(R.id.myButton, ConstraintSet.RIGHT,
              ConstraintSet.PARENT_ID, ConstraintSet.RIGHT, 0);

        set.constrainWidth(R.id.myButton,
ConstraintSet.WRAP_CONTENT);

        set.applyTo(myLayout);
    }
}
.
.
.
}
```

Compile and run the application once again and note that the transition is now animated.

33.4 Customizing the Transition

The final task in this example is to modify the changeBounds transition so that it is performed over a longer duration and incorporates a bounce effect when the view reaches its new screen location. This involves the creation of a Transition instance with appropriate duration interpolator settings which is, in turn, passed through as an argument to the *beginDelayedTransition()* method:

```
.
.
import android.transition.ChangeBounds;
import android.transition.Transition;
import android.view.animation.BounceInterpolator;
.
.
public void handleTouch() {
    Button button = (Button) findViewById(R.id.myButton);

    Transition changeBounds = new ChangeBounds();
    changeBounds.setDuration(3000);
    changeBounds.setInterpolator(new BounceInterpolator());
    TransitionManager.beginDelayedTransition(myLayout,
            changeBounds);

    TransitionManager.beginDelayedTransition(myLayout);

    ConstraintSet set = new ConstraintSet();

    button.setMinimumWidth(500);
    button.setMinimumHeight(350);

    set.connect(R.id.myButton, ConstraintSet.BOTTOM,
           ConstraintSet.PARENT_ID, ConstraintSet.BOTTOM, 0);

    set.connect(R.id.myButton, ConstraintSet.RIGHT,
           ConstraintSet.PARENT_ID, ConstraintSet.RIGHT, 0);

    set.constrainWidth(R.id.myButton, ConstraintSet.WRAP_CONTENT);

    set.applyTo(myLayout);
}
```

When the application is now executed, the animation will slow to match the new duration setting and the button will bounce on arrival at the bottom right-hand corner of the display.

33.5 Summary

The most basic form of transition animation involves the use of the *beginDelayedTransition()* method of the TransitionManager class. Once called, any changes in size and position of the views in the next user interface rendering frame, and within a defined view group, will be animated using the specified transitions. This chapter has worked through a simple Android Studio example that demonstrates the use of this approach to implementing transitions.