

# 5

## Graphics, Drawing, and Audio

### Chapter Objectives

In this chapter you will:

- Learn to draw to a canvas.
- Examine the process of adding drawn elements and user interface controls to a layout.
- Understand how to add and manipulate ImageViews programmatically.
- Build frame-by-frame animations.
- Explore a turn-based game containing animation.
- Understand the Android Animate library.
- Explore the MediaPlayer for audio.

### ■ 5.1 Graphics in Android

Graphics are an important part of any Android application. A graphic element can appear as a simple background image, an icon, a component in a sophisticated user interface, or as a moving object with interactive behavior in a game. Before developing an application, it is always a good idea to consider carefully what the graphical demands will be. Varying graphical tasks are often accomplished with varying techniques.

Two general techniques for adding graphics to an Android application are (1) placing an existing image into an ImageView, which can be inflated on the screen, or (2) drawing custom graphics in real time.

Often, ImageViews are populated with resources from drawables; images can also be read from a file on a device or downloaded from a URL. An ImageView graphic can be inflated at any time during the execution of an application. Once placed on the screen, these graphic elements can be manipulated in various ways. In the view hierarchy, parent views are responsible for editing the attributes of their child ImageViews, which can be done dynamically; this means that a graphic object can adjust to a changing environment, such as game play conditions or screen orientation. When the majority

of an application's interface is fixed, it is practical to render the interface in advance, using `ImageViews`, and inflate those images at runtime. For example, a chess game requires multiple `ImageView` graphic elements representing the chessboard and chess pieces. The chessboard element is static and therefore can be added to the XML layout representing the game activity. Chess pieces are placed on the screen during runtime, with their initial positions set according to the player's color selection. As the chess game unfolds, chess pieces are repositioned or removed from the board when an opponent captures them.

Drawing custom graphic elements to be added to an activity is a technique that is generally suited for applications requiring more complex graphic needs. Drawing to a canvas or a `View` object makes use of Android native drawing tools. Custom drawing can be processor intensive and should be optimized.

---

## ■ 5.2 Adding and Manipulating `ImageViews`

---

Perhaps the easiest way to add graphics to an application is by referencing a bitmap file from the project's resources. In earlier chapters, graphics were added during the layout design. In this chapter, we explore the concept of inflating images and modifying them in real time.

Android supports three image file types: PNG, JPG, and GIF. PNG is established as the preferred format for an image file in Android. It is also acceptable to use a JPG. GIF bitmaps should always be avoided.

Bitmap files placed in `res/drawable/` may be automatically optimized with lossless image compression during the build process. For example, a true-color PNG image that does not require more than 256 colors may be converted to an 8-bit PNG with a color palette. This conversion will result in an image of equal quality but is optimized by requiring less memory.

Typically, an image resource is added to the `res/drawable/` folder before it can be referenced in an XML layout file and from an Activity.

The following code segment demonstrates how to build an `ImageView` that uses an image from the drawable directory and add it to the layout.

- Line 6:        A `RelativeLayout` is constructed. The context of the layout is set to the Activity using the `this` argument.
- Line 8:        An `ImageView` object named `imageView` is instantiated.
- Line 10:      A photograph of Mt. Everest, defined by the drawable resource file `photo_of_everest.png`, is set as the content of this `ImageView` object. This segment of code assumes that `photo_of_everest.png` exists in the drawable directory.

- Line 11: The bounds are set to match the Drawable's dimensions. The argument true is used when the ImageView needs to adjust its bounds to preserve the aspect ratio of its drawable.
- Line 13: This sets the top position of this view relative to its parent. This method is meant to be called by the layout system and should not generally be called otherwise, because the property may be changed at any time by the layout.
- Line 15: The ImageView is added to the layout.
- Line 16: mRelativeLayout is set as the view content for the activity of the application. The layout is inflated, adding all top-level views to the activity.

```
1 RelativeLayout mRelativeLayout;
2
3 protected void onCreate(Bundle savedInstanceState) {
4     super.onCreate(savedInstanceState);
5
6     mRelativeLayout = new RelativeLayout (this);
7
8     ImageView imageView = new ImageView(this);
9
10    imageView. setImageDrawable (R.drawable.photo_of_everest);
11    imageView.setAdjustViewBounds(true);
12
13    imageView.setTop(10);
14
15    mRelativeLayout.addView(imageView);
16    setContentView(mRelativeLayout);
17 }
```

The Android resource system can be used to access an application's resources, such as the drawable bitmaps. For example, in some situations, it may be desirable to handle an image resource as a Drawable object. In the following segment of code, the object, `everest`, is assigned the drawable object associated with the resource ID, `R.drawable.photo_of_everest`.

- Line 1: The context of the application is the information about its environment. It allows access to application-specific resources. `getBaseContext()` returns the base context of the application.
- Line 2: `getResources()` returns a resources instance for the application's package.

Lines 3–4: `getDrawable()` returns a drawable object associated with a resource ID, `R.drawable.photo_of_everest`.

```

1 Context mContext = getBaseContext();
2 Resources resource = mContext.getResources();
3 Drawable everest = resource.getDrawable
4     (R.drawable.photo_of_everest);

```

A drawable resource is a general concept for a graphic that can be drawn to the screen. In addition to bitmaps, Android supports XML graphics, which are drawable resources defined in XML.

Consider the XML code segment in the example below. This code defines a graphical shape resource, as shown in Figure 5-1. As a drawable, it is placed in a drawable resource file, `res/drawable/` directory.

Lines 3–4: The root element of the drawable graphic is `shape`. Android provides four primitive shapes: `line`, `oval`, `rectangle`, and `ring`.

Lines 6–8: The oval is outlined in dark red with a stroke width of 2dp.

Lines 10–11: The interior of the oval is filled with a solid color, grey. Additional shape attributes can be applied using the tags `<gradient>`, `<padding>`, `<stroke>`, and `<corners>`.

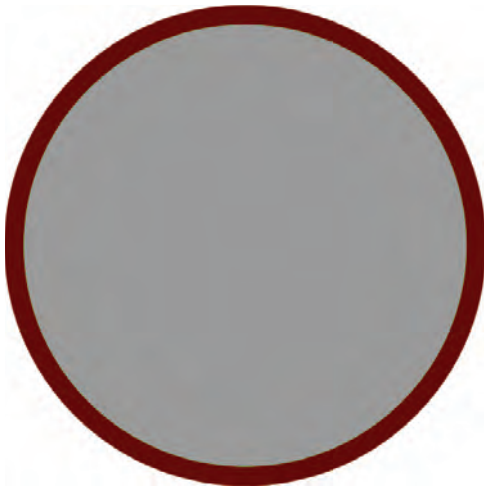
Lines 12–14: The height and width are both set to 50dp, a perfect circle. Figure 5-1 shows the graphic produced by the XML drawable.

`res/drawable/circle.xml`

```

1 <?xml version="1.0" encoding="utf-8"?>
2
3 <shape xmlns:android="http://schemas.android.com/apk/res/android"
4     android:shape="oval">
5
6     <stroke
7         android:width="2dp"
8         android:color="#660000" />
9
10    <solid
11        android:color="#999999" />
12    <size
13        android:height="50dp"
14        android:width="50dp" />
15
16 </shape>

```



**FIGURE 5-1** Graphic created as an XML Drawable.

## ■ Lab Example 5-1: Fibonacci Flower Application

Bitmap graphics play a significant role in much of what we see on smart devices. Creating artwork algorithmically is an interesting endeavor, as well as an efficient method for producing remarkably detailed designs. This lab example explores the dynamic ability of manipulating `ImageView` objects and adding them to a layout.

### Part 1: The Design

The Fibonacci Flower application allows the user to build an artwork with constraints built in. This is an ideal application for illustrating how bitmaps can be placed on the screen and altered programmatically.

The constraint for the artwork is the Golden Ratio; mathematicians, computer scientists, and graphic designers make frequent use of this ratio, which is derived from the Fibonacci series. Leonardi Fibonacci was an Italian mathematician who created the sequence of numbers in which each term is the sum of the two preceding terms. The value of the first term is 1, as is the second term. The third term in the series is computed as  $1 + 1$  (term 1 + term 2). The first 14 terms in the sequence are as follows:

Term 1:	1 First Term
Term 2:	1 Second Term
Term 3:	$2 = 1 + 1$
Term 4:	$3 = 1 + 2$
Term 5:	$5 = 2 + 3$
Term 6:	$8 = 3 + 5$

Term 7:  $13 = 5 + 8$   
Term 8:  $21 = 8 + 13$   
Term 9:  $34 = 13 + 21$   
Term 10:  $55 = 21 + 34$   
Term 11:  $89 = 34 + 55$   
Term 12:  $144 = 55 + 89$   
Term 13:  $233 = 89 + 144$   
Term 14:  $377 = 144 + 233$

An important aspect of this sequence is the ratio produced when dividing successive terms. For example, when the 9<sup>th</sup> term is divided by the 10<sup>th</sup> term, the ratio of 0.61818182 is produced. When the 13<sup>th</sup> term is divided by the 14<sup>th</sup> term, the result is 0.61803279 is produced. As the terms become larger and larger, the ratio between two successive terms converges to approximately 0.6180339. This so-called Golden Ratio is considered to be extraordinary because of its common occurrence in nature.

For this application, the user is provided with two types of flower petals, Pink and Gold, in which to build a Fibonacci flower. The flower can be built one petal at a time by tapping one of the petal buttons, located at the top of the screen. As petals are placed on the screen, they are manipulated, using a Lorenz attractor, to produce a flower that resembles a butterfly. Figure 5-2 shows four screenshots of the application. The first screenshot shows the initial application screen as it appears when it is launched for the first time. The remaining three screenshots show petals added to the screen in a particular formation. The user can use the buttons to choose the petal to add to the flower, while the algorithm will manipulate it and arrange it on the screen.

In addition to the petal selection buttons, a “clear” button is placed on the bottom of the screen to allow users to clear away the petals to begin creating a new flower.

## Part 2: Application Structure and Setup

---

The settings for the application are as follows:

- Application Name: Fibonacci Flower
- Project Name: FibonacciFlower
- Package Name: `com.cornez.fibonacciflower`
- Android Form: Phone and Tablet
- Minimum SDK: API 18: Android 4.3 (Jelly Bean)
- Target SDK: API 21: Android 5.0 (Lollipop)
- Compile with: API 21: Android 5.0 (Lollipop)
- Activity Name: `MyActivity`
- Layout Name: `activity_my`



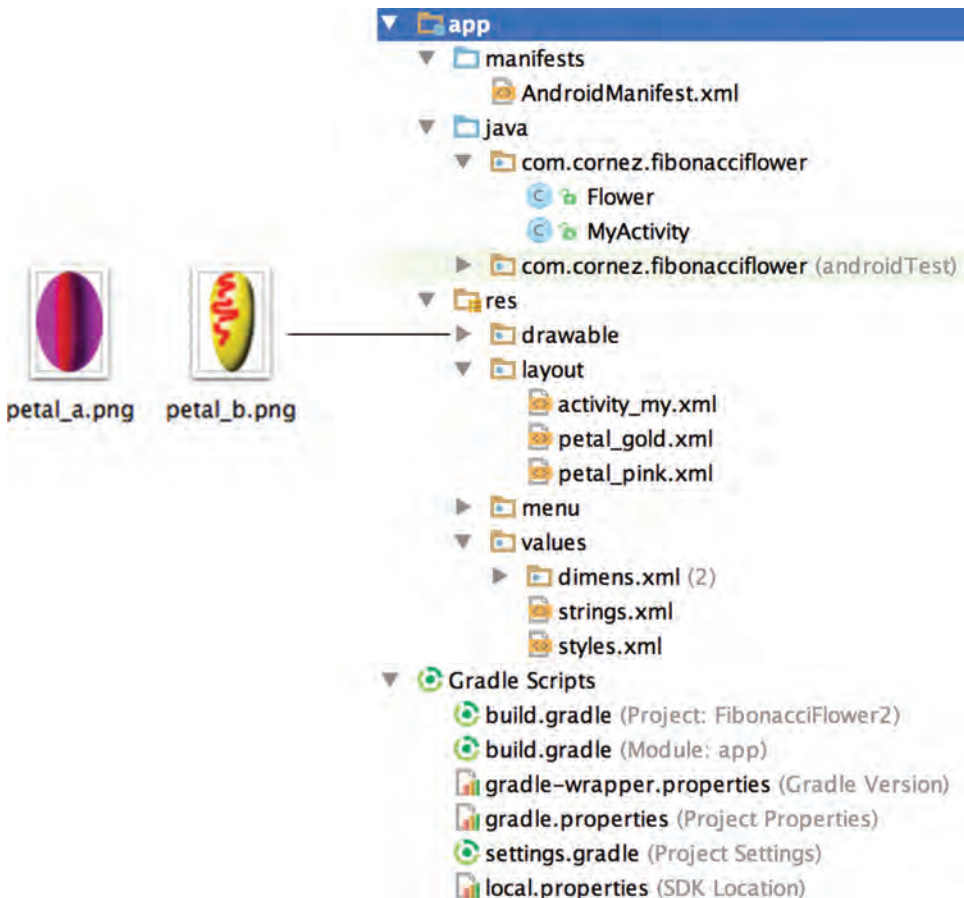
**FIGURE 5-2** The Fibonacci Flower app allows the user to build a flower artwork.



The icon that launches the Fibonacci Flower application is set to the Android default `ic_launcher.png` file. The two drawable bitmap files required by the application are `petal_a.png` and `petal_b.png`. These files can be found in the textbook resources and should be placed in `res/drawable`.

The final project structure is shown in Figure 5-3. The Java source file, `MyActivity`, is the only activity of the application. `Flower.java` is a blueprint for a Fibonacci flower artwork.

Three XML files will be used by the application: (1) `activity_my.xml`, (2) `petal_gold.xml`, and (3) `petal_pink.xml`. `activity_my.xml` is the user interface for `MyActivity`. `petal_gold.xml` and `petal_pink.xml` are also technically layout files; however, their primary function is not as a user interface, but instead as a graphic bitmap container. Each is used to represent a class of petals visually. For example, when a petal graphic is created and added to the screen, it is generated from one of the petal XML files.



**FIGURE 5-3** Project structure for the Fibonacci Flower application.



The orientation of the screen is locked into portrait mode. The screen is presented in a fullscreen configuration with the title bar eliminated. These attributes are set in the `AndroidManifest.xml` file, as shown below:

```

AndroidManifest.xml
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      package="com.cornez.fibonacciflower" >
4
5      <application
6          android:allowBackup="true"
7          android:icon="@drawable/ic_launcher"
8          android:label="@string/app_name"
9          android:theme="@android:style/
10             Theme.Light.NoTitleBar.Fullscreen">
11
12          <activity
13              android:name=".MyActivity"
14              android:screenOrientation="portrait"
15              android:label="@string/app_name" >
16              <intent-filter>
17                  <action android:name="android.intent.action.MAIN" />
18
19                  <category android:name=
20                      "android.intent.category.LAUNCHER" />
21              </intent-filter>
22          </activity>
23
24      </application>
25
26  </manifest>

```

### Part 3: The User Interface

The user interface uses strings for three of the interactive buttons. The `strings.xml` file is shown below with the button labels added:

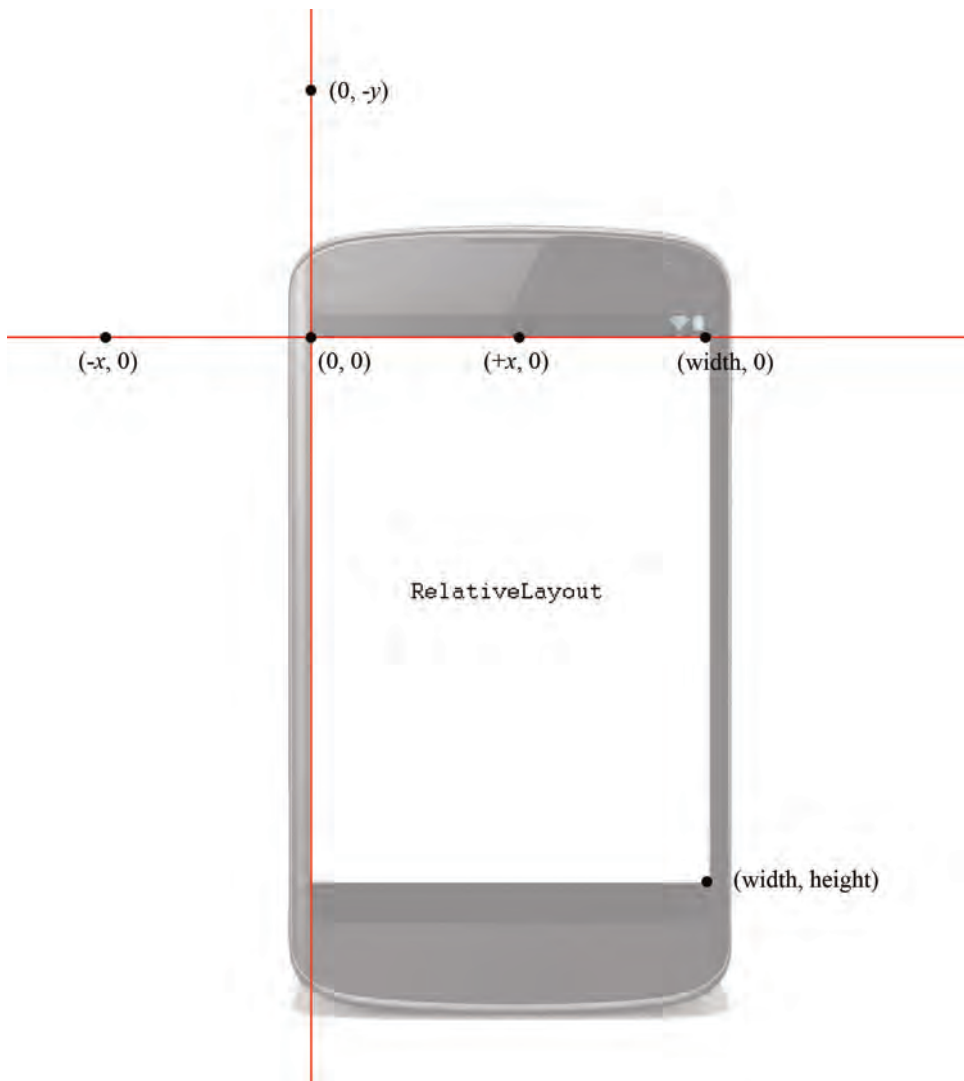
```

strings.xml
1  <?xml version="1.0" encoding="utf-8"?>
2  <resources>
3
4      <string name="app_name">Fibonacci Flower</string>
5      <string name="hello_world">Hello world!</string>
6      <string name="action_settings">Settings</string>
7
8      <!-- BUTTON LABELS -->
9      <string name="pink_btn">Add Pink</string>

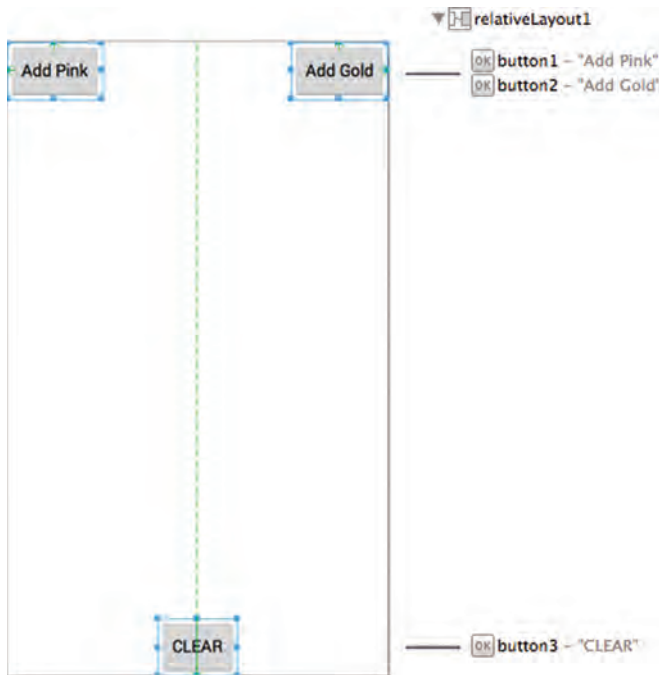
```

```
10 <string name="gold_btn">Add Gold</string>
11 <string name="clear_btn">CLEAR</string>
12
13 </resources>
```

The application is controlled by a single activity, `MyActivity`. `MyActivity` will launch with `activity_my.xml`. The root layout element for `activity_my.xml` is a `RelativeLayout`, which supports a flexible arrangement for the buttons. The coordinate system for a `RelativeLayout` is shown in



**FIGURE 5-4** The coordinate system for a `RelativeLayout`.



**FIGURE 5-5** The layout structure and design of `activity_my.xml`.

Figure 5-4. It is important to understand the coordinate system that is supported by the layout used, because they do not all use the same system.

The layout graphic design for the `activity_my.xml` is shown in Figure 5-5. The XML code for `activity_my.xml` is listed as follows:

Line 10:       The `RelativeLayout` root view must be given an id name. This name will be referenced in `MyActivity` as petals are inflated and positioned within the layout.

`activity_my.xml`

```


1  <RelativeLayout
2      xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:tools="http://schemas.android.com/tools"
4      android:layout_width="match_parent"
5      android:layout_height="match_parent"
6      android:paddingLeft="@dimen/activity_horizontal_margin"
7      android:paddingRight="@dimen/activity_horizontal_margin"
8      android:paddingTop="@dimen/activity_vertical_margin"
9      android:paddingBottom="@dimen/activity_vertical_margin"
10     android:id="@+id/relativeLayout1"
11     tools:context=".MyActivity">
12

```

```

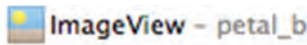
13      <!-- PINK PETAL BUTTON -->
14      <Button
15          android:id="@+id/button1"
16          style="?android:attr/buttonStyleSmall"
17          android:layout_width="wrap_content"
18          android:layout_height="wrap_content"
19          android:layout_alignParentLeft="true"
20          android:layout_alignParentTop="true"
21          android:text="@string/pink_btn" />
22
23      <!-- GOLD PETAL BUTTON -->
24      <Button
25          android:id="@+id/button2"
26          style="?android:attr/buttonStyleSmall"
27          android:layout_width="wrap_content"
28          android:layout_height="wrap_content"
29          android:layout_alignParentRight="true"
30          android:layout_alignParentTop="true"
31          android:text="@string/gold_btn" />
32
33      <!-- CLEAR ALL PETALS BUTTON -->
34      <Button
35          android:id="@+id/button3"
36          style="?android:attr/buttonStyleSmall"
37          android:layout_width="wrap_content"
38          android:layout_height="wrap_content"
39          android:layout_alignParentBottom="true"
40          android:layout_centerHorizontal="true"
41          android:text="@string/clear_btn" />
42
43  </RelativeLayout>

```

 **ImageView - petal\_a**



**FIGURE 5-6** The layout structure for `petal_pink.xml` contains a drawable.



**FIGURE 5-7** The layout structure for `petal_gold.xml`.

The two petal layout files use an `ImageView` as their root element. There are no other views contained within these layouts. Their view structures are shown in Figures 5-6 and 5-7. The XML code for the petal files are listed as follows:

Line 5: The `contentDescription` attribute is not used by Java source code; however, it is customary to add a description for all image elements.

`petal_pink.xml`

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <ImageView xmlns:android="http://schemas.android.com/apk/res/android"
3     android:layout_width="match_parent"
4     android:layout_height="match_parent"
5     android:contentDescription="@string/pink_btn"
6     android:src="@drawable/petal_a" >
7
8 </ImageView>
```

`petal_gold.xml`

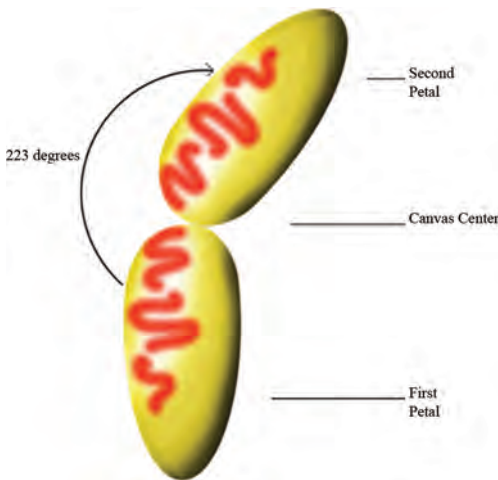
```

1 <?xml version="1.0" encoding="utf-8"?>
2 <ImageView xmlns:android="http://schemas.android.com/apk/res/android"
3     android:layout_width="match_parent"
4     android:layout_height="match_parent"
5     android:contentDescription="@string/gold_btn"
6     android:src="@drawable/petal_b" >
7
8 </ImageView>
```

## Part 4: Source Code for Application

`Flower.java` is the blueprint for a given flower that is seen on the screen. `Flower.java` relies on the Golden Ratio, which is common for giving computer-generated images the appearance of natural harmony. To create a Fibonacci flower for this application, the Golden Ratio is used repeatedly to offset the angle of each petal before it is placed on the screen.

In addition, each petal added to the screen is generated 3% wider and 3% longer than the previous one. The petal's angle of rotation is increased by 360 degrees multiplied by the Golden Ratio. This result for the first two petals placed on the screen is illustrated in Figure 5-8. The second petal is slightly wider and longer and offset at an angle of 233 degrees ( $360 * 0.6180339$ ).



**FIGURE 5-8** A flower petal will rotate at the center of the canvas.

The Java code listing for `Flower.java` appears as follows:

- Lines 5–7: The Golden Ratio and the growth factors are stored as constants.
- Lines 10–16: Each petal within the flower has a set of attributes that will be edited to manipulate the appearance of a petal graphic before it appears on the screen.
- Lines 18–24: The class constructor initializes the first petal of the flower. The data member `degenerate` is set to 1.001. This value will remain constant and will be applied to the angle of each flower petal.
- Lines 76–81: Once a flower petal has been added to the screen, petal values will be updated, which will then be applied to the next petal in the sequence.

Flower.java

```
1 package com.cornez.fibonacciflower;
2
3 public class Flower {
4
5     public final double GOLDEN_RATIO = .618033989;
6     public final double GROW_WIDTH = .03 * GOLDEN_RATIO;
7     public final double GROW_HEIGHT = .03 * GOLDEN_RATIO;
8
9     //PETAL ATTRIBUTES
10    private double angle;
11    private int rotate;
12    private float scaleX;
13    private float scaleY;
14    private int xCenter;
15    private int yCenter;
16    private float degenerate;
17
18    public Flower() {
19        rotate = 0;
20        scaleX = (float) .3;
21        scaleY = (float) .3;
22        degenerate = (float) 1.001;
23        angle = 360 * GOLDEN_RATIO;
24    }
25
26    public void initialize() {
27        //TASK 1: INITIALIZE THE SETTINGS
28        //      FOR THE FIRST FLOWER PETAL
29        rotate = 0;
30        scaleX = (float) .3;
31        scaleY = (float) .3;
32        degenerate = (float) 1.001;
33        angle = 360 * GOLDEN_RATIO;
34    }
35
36    public float getScaleX(){
37        return scaleX;
38    }
39
40    public void setScaleX(float scale){
41        scaleX = scale;
42    }
43    public float getScaleY(){
44        return scaleY;
45    }
46    public void setScaleY(float scale){
47        scaleY = scale;
48    }
49    public void setRotate(int rot) {
```



```
50         rotate = rot;
51     }
52     public int getRotate() {
53         return rotate;
54     }
55     public void set_xCenter(int x){
56         xCenter = x;
57     }
58     public int get_xCenter(){
59         return xCenter;
60     }
61
62     public void set_yCenter(int y){
63         yCenter = y;
64     }
65     public int get_yCenter(){
66         return yCenter;
67     }
68
69     public void setDegenerate(float deg){
70         degenerate = deg;
71     }
72     public void initializeAngle (){
73         angle = 360 * GOLDEN_RATIO;
74     }
75
76     public void updatePetalValues() {
77         rotate += angle;
78         scaleX += scaleX * GROW_WIDTH;
79         scaleY += scaleY * GROW_HEIGHT;
80         angle *= degenerate;
81     }
82 }
```

The Java code listing for `MyActivity.java` appears as follows:

- Line 6: `DisplayMetrics` is a utility public class that is used to access general information describing the display screen. This includes its size, density, and font scaling.
- Line 7: The `LayoutInflater` class is used in `MyActivity` to instantiate a petal layout XML file, which will allow a petal graphic bitmap to appear on the screen.
- Line 15: An `ArrayList` is used to store all petal graphic elements placed on the screen.
- Lines 42–43: `getSystemService()` returns a system-level service for a `LayoutInflater`, which is used to inflate layout resources in this context.

Lines 55–59: The `RelativeLayout` will hold all flower petals. The petals will be positioned in the center of the layout, which is specified as `width/2, height/2`. `getWindowManager()` retrieves the window manager for showing custom windows. `getDefaultDisplay()` returns a default display.

Lines 82–97: A petal `ImageView` object is created based on the text value of the button that was clicked. The new view hierarchy is inflated from the specified petal XML resource.

The visual properties for the `ImageView` are set. `setPivotY()` sets the *y* location of the point around which the petal is rotated and scaled. By default, the pivot point is centered on the object. `setPivotX()` is the same, except the *x* pivot location is set. `setScaleX()` sets the amount that the petal is scaled in *x* around the pivot point, as a proportion of the view's unscaled width. For `setScaleX` and `setScaleY`, a value of 1 means that no scaling is applied.

`setRotation()` sets the degrees that the petal is rotated around the pivot point. Increasing values result in a clockwise rotation.

Line 100: The petal `ImageView` object is added to `relativeLayout`. Properties are applied before the object is added to the layout. The number zero refers to the layer index of the petal being added. By specifying zero, petals will be placed underneath the existing flower petals.

Lines 112–126: Petals are cleared when the user clicks the “clear” button. The `onClick()` handler uses `removeView()` to eliminate the `View` from the `RelativeLayout`. In addition, all petals are cleared from the `ArrayList`.

MyActivity.java

```
1 package com.cornez.fibonacciflower;
2
3 import android.app.Activity;
4 import android.content.Context;
5 import android.os.Bundle;
6 import android.util.DisplayMetrics;
7 import android.view.LayoutInflater;
8 import android.view.Menu;
9 import android.view.MenuItem;
10 import android.view.View;
11 import android.widget.Button;
12 import android.widget.ImageView;
```

```
13 import android.widget.RelativeLayout;
14
15 import java.util.ArrayList;
16
17
18 public class MyActivity extends Activity {
19
20     private ArrayList<ImageView> allPetals;
21     private LayoutInflater layoutInflater;
22
23     private Button pinkBtn;
24     private Button goldBtn;
25     private Button clearBtn;
26     private RelativeLayout relativeLayout;
27
28     Flower myFlower;
29
30     @Override
31     protected void onCreate(Bundle savedInstanceState) {
32         super.onCreate(savedInstanceState);
33         setContentView(R.layout.activity_my);
34
35         myFlower = new Flower();
36         allPetals = new ArrayList<ImageView>();
37
38         //INITIALIZE THE GENERATION OF THE FIBONACCI ARTWORK
39         initialize();
40
41         //CREATE A LAYOUT INFLATER TO ADD PETALS TO RELATIVE LAYOUT
42         layoutInflater = (LayoutInflater)
43             getSystemService(Context.LAYOUT_INFLATER_SERVICE);
44
45         relativeLayout = (RelativeLayout)
46             findViewById(R.id.relativeLayout1);
47         pinkBtn = (Button) findViewById(R.id.button1);
48         goldBtn = (Button) findViewById(R.id.button2);
49         clearBtn = (Button) findViewById(R.id.button3);
50         pinkBtn.setOnClickListener(addPetal);
51         goldBtn.setOnClickListener(addPetal);
52         clearBtn.setOnClickListener(clearPetals);
53
54         //SET THE CENTER COORDINATE
55         DisplayMetrics metrics = new DisplayMetrics();
56         getWindowManager().getDefaultDisplay().getMetrics(metrics);
57         myFlower.set_xCenter(metrics.widthPixels / 2);
58         myFlower.set_yCenter(metrics.heightPixels / 2);
59
60     }
61 }
```

```
62 private void initialize(){
63     //TASK 1: INITIALIZE THE SETTINGS FOR THE FIRST PETAL
64
65     myFlower.setRotate(0);
66     myFlower.setScaleX((float) .3);
67     myFlower.setScaleY((float) .3);
68     myFlower.setDegenerate((float) 1.001);
69     myFlower.initializeAngle();
70
71 }
72
73 //ON CLICK BUTTON HANDLER FOR ADDING A FLOWER PETAL TO THE SCREEN
74 private View.OnClickListener addPetal = new View.OnClickListener() {
75     public void onClick (View view) {
76
77
78         //TASK 1: INSTANTIATE A VIEW TO STORE A PETAL GRAPHIC
79         ImageView petal;
80
81         // INFLATE THE CORRECT PETAL
82         String buttonText = ((Button) view).getText().toString();
83         if (buttonText.equals("Add Pink"))
84             petal = (ImageView)
85                 inflater.inflate(R.layout.petal_pink, null);
86         else
87             petal = (ImageView)
88                 inflater.inflate(R.layout.petal_gold, null);
89
90         //TASK 2: SET THE VISUAL PROPERTIES OF THE PETAL
91         petal.setX(myFlower.get_xCenter());
92         petal.setY(myFlower.get_yCenter());
93         petal.setPivotY(0);
94         petal.setPivotX(100);
95         petal.setScaleX(myFlower.getScaleX());
96         petal.setScaleY(myFlower.getScaleY());
97         petal.setRotation(myFlower.getRotate());
98
99         //TASK 3: PLACE THE INFLATED IMAGEVIEW IN THE MAIN LAYOUT
100         relativeLayout.addView(petal, 0);
101
102         //TASK 4: ADD THE IMAGEVIEW OF THE PETAL TO THE ARRAYLIST
103         allPetals.add(petal);
104
105         //TASK 5: UPDATE THE ANGLE AND SCALE
106         //          FOR THE NEXT PETAL TO BE ADDED
107         myFlower.updatePetalValues();
108     }
109 };
110
```

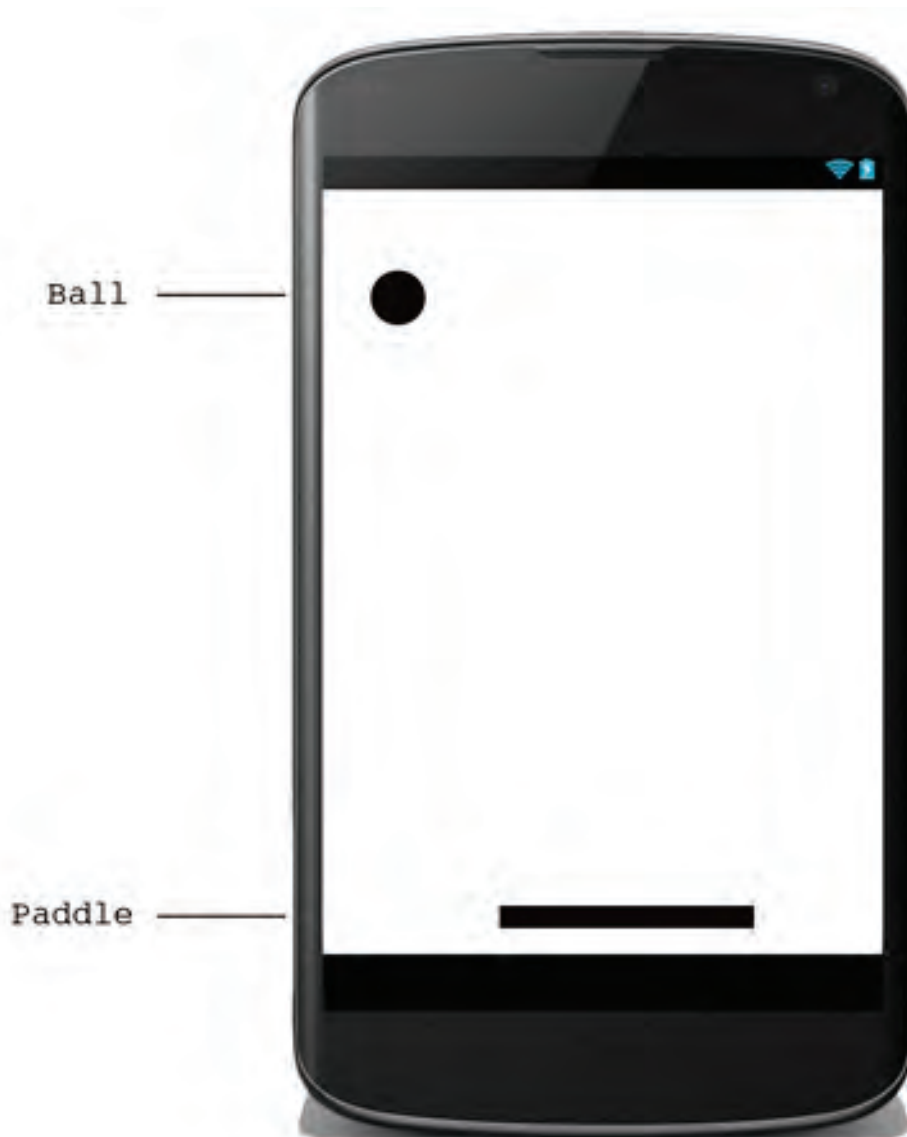
```
111 //ON CLICK BUTTON HANDLER TO CLEAR ALL PETALS ON THE SCREEN
112 private View.OnClickListener clearPetals =
113     new View.OnClickListener() {
114
115         public void onClick (View view) {
116             //TASK 1: REMOVE ALL PETAL IMAGE VIEW FROM THE LAYOUT
117             for (int i = 0; i < allPetals.size(); i++) {
118                 ImageView petal = allPetals.get(i);
119                 relativeLayout.removeView(petal);
120             }
121
122             //TASK 2: CLEAR THE ARRAYLIST AND RESET ALL VARIABLES
123             allPetals.clear();
124             initialize();
125         }
126     };
127
128 @Override
129 public boolean onCreateOptionsMenu(Menu menu) {
130     // Inflate the menu.
131     getMenuInflater().inflate(R.menu.my, menu);
132     return true;
133 }
134
135 @Override
136 public boolean onOptionsItemSelected(MenuItem item) {
137     // Handle action bar item clicks here. The action bar will
138     // automatically handle clicks on the Home/Up button, so long
139     // as you specify a parent activity in AndroidManifest.xml.
140     int id = item.getItemId();
141     if (id == R.id.action_settings) {
142         return true;
143     }
144     return super.onOptionsItemSelected(item);
145 }
146 }
```

### 5.3 Drawing and the Canvas Class

When an application requires specialized drawing, or the control of animated graphics, a Canvas can be used. A Canvas provides an interface to the actual surface upon which graphics will be drawn. The purpose of the canvas is to hold the results of the “draw” calls. By using the Canvas, a drawing is actually performed on an underlying Bitmap, which is placed into the window.

Optimizing an application that requires extensive custom drawing can sometimes be difficult. The use of custom drawing code should be limited to situations where the

content display needs to change dynamically. Consider the game of Pong, as shown in Figure 5-9. Pong can be created as a drawing application, which means it requires custom drawing code to track the movement of the paddle and update the screen as the user drags the paddle back and forth. The application also needs to update the drawing of the two-dimensional ball to reflect its changing position as it bounces off the walls and the paddle.



**FIGURE 5-9** The game of Pong can be created using a custom drawing code.

Drawing custom graphics can be performed in two ways: (1) graphics can be drawn into a `View` object, or (2) they can be drawn directly to a `Canvas`. Drawing to a `View` is a good choice when simple static graphics are needed. Game applications, such as Pong, need dynamic visuals that require the application to regularly redraw itself to a canvas.

The `android.graphics` package provides canvas tools, color filters, points, and rectangles. The `Android Canvas` class has a set of drawing methods, such as `drawBitmap()`, `drawRect()`, and `drawText()`. To draw something, four basic components are needed:

1. A `Bitmap` to hold the pixels.
2. A `Canvas` to host the draw calls (writing into the bitmap).
3. A drawing primitive (e.g., `Rect`, `Path`, text, `Bitmap`).
4. `Paint` (to describe the colors and styles for the drawing).

Consider the following code segment; in this example, a new `Canvas` is created. Before the canvas is instantiated, a `Bitmap`, upon which the drawing will be performed, is first defined. The parameters required for the bitmap are its width, height, and the bitmap configuration.

The bitmap configuration, `ARGB_8888`, establishes that each pixel is stored on 4 bytes. Each channel (RGB and alpha for translucency) is stored with 8 bits of precision (256 possible values.) On Line 2 of the code segment, the canvas is instantiated with the bitmap, which is always required for a new canvas.

```
1 Bitmap bitmap = Bitmap.createBitmap(100, 100, Bitmap.Config.ARGB_8888);  
2 Canvas canvas = new Canvas(bitmap);
```

If an application does not require a significant amount of processing or frame-rate speed, such as a turn-based game similar to chess, then a custom `View` component can be created specifically for drawing. Drawing with a `Canvas` on a view is performed with `View.onDraw()`. By using a `View` component, the Android framework will provide a predefined `Canvas` to hold the drawing calls.

This requires an extension of the `View` class and a definition of the `onDraw()` callback method. The Android framework calls the `onDraw()` method to request that the `View` draw itself. Within this method, you can perform calls to draw through the `Canvas`.

The Android framework calls `onDraw()` only as necessary. For example, each time your application is prepared to redraw a canvas, you must request that your `View` be invalidated by calling `invalidate()`. `invalidate()` indicates that the entire view will be drawn again. Android then calls the `onDraw()` method.

A `ShapeDrawable` is a good option when dynamically drawn graphics are needed for an application. Primitive shapes, such as ovals, rectangles, and lines, and simple



styles can be programmatically built easily. Consider the class `CustomView`, implemented in the following code segment. `CustomView` is an extension of the `View` class that draws a primitive oval shape.

- Line 2:       A `ShapeDrawable` is a drawable object that draws only primitive shapes. A `ShapeDrawable` uses a `Shape` object and manages its appearance on the screen. If no `Shape` is specified, the `ShapeDrawable` will default to a rectangle. As we saw previously in Section 5.2 of this chapter, an object can also be defined in an XML file using the `<shape>` tag.
- Line 13:      The paint color for the oval shape is set. The color value `0xff74AC23` is an integer containing alpha, as well as red, green, and blue values.
- Line 14:      `setBounds()` specifies a bounding rectangle for the `Drawable`. This is where the oval shape appears when its `draw()` method is called.

```
1      public class CustomView extends View {  
2          private ShapeDrawable mDrawable;  
3  
4          public CustomView(Context context) {  
5              super(context);  
6  
7              int x = 10;  
8              int y = 10;  
9              int width = 50;  
10             int height = 50;  
11  
12             mDrawable = new ShapeDrawable(new OvalShape());  
13             mDrawable.getPaint().setColor(0xff74AC23);  
14             mDrawable.setBounds(x, y, x + width, y + height);  
15         }  
16  
17         protected void onDraw(Canvas canvas) {  
18             mDrawable.draw(canvas);  
19         }  
20     }
```

A `ShapeDrawable` is an extension of `Drawable`. This is convenient because it can be used anywhere a `Drawable` is expected, such as the background of a `View`.

A drawable can also be built as its own custom `View` and then added to a layout. In the code segment below, the `CustomView` object is set as the layout content in `onCreate()`.

```
1 CustomView mCustomView;  
2  
3 protected void onCreate(Bundle savedInstanceState) {  
4     super.onCreate(savedInstanceState);  
5     mCustomDrawableView = new  
6         CustomView(this);  
7  
8     setContentView(mCustomView);  
9 }
```

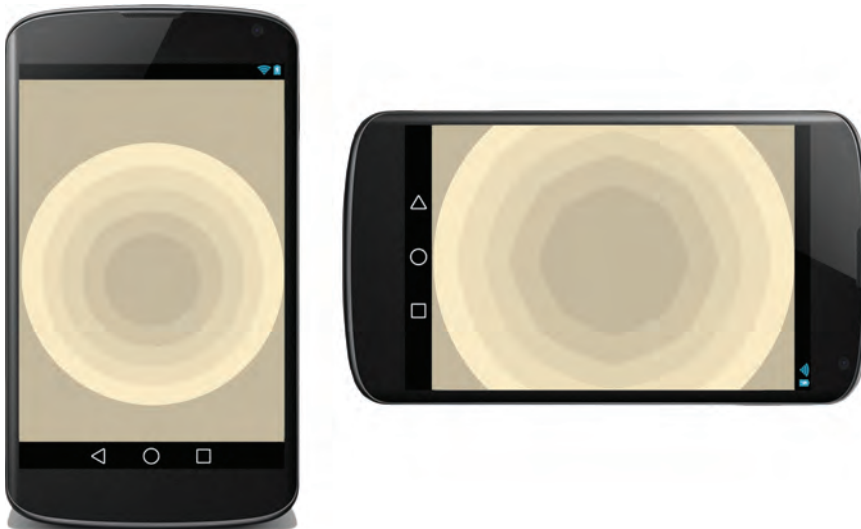
### ■ Lab Example 5-2: Drawing Experiment 1: Bull's-Eye Drawing

This lab example explores the concept of building a custom View, drawing a Bull's Eye within the View, and displaying it on the screen as the layout associated with MyActivity. The Bull's-Eye View is dynamic in the sense that it redraws itself when the user changes orientation on the device. Figure 5-10 shows the Bull's Eye configured for portrait and landscape orientations.

#### Part 1: The Design

The width of the Bull's Eye is set to span the width of the device, regardless of the device's orientation. The drawing is created using five circles, with different sizes, sharing the same center.

The window containing the layout does not feature a title, and it occupies the full screen.



■ **FIGURE 5-10** The Bull's-Eye App running in portrait and landscape orientation.

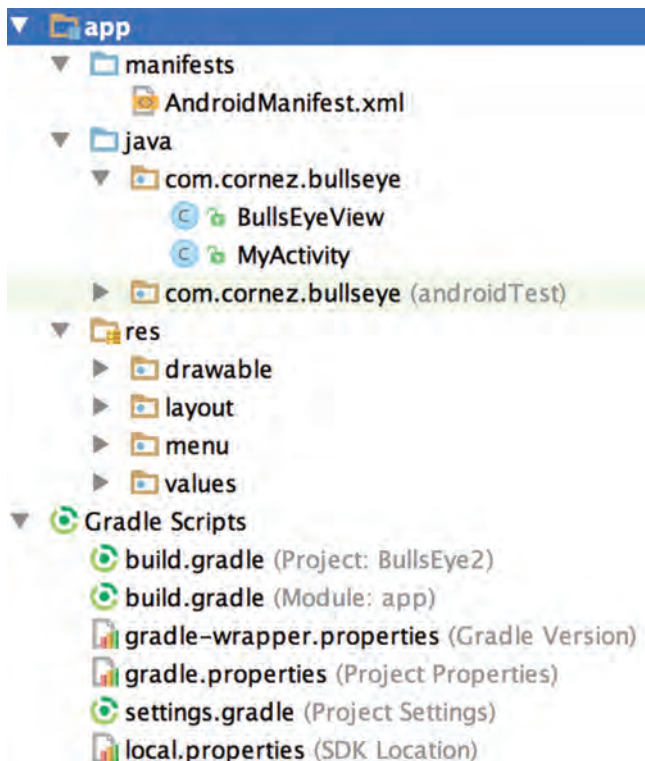
## Part 2: Application Structure and Setup

The application structure is very simple, as this project is intended to be an experiment in drawing rather than an interactive app.

The settings for the application are as follows:

- Application Name: Bulls Eye
- Project Name: BullsEye
- Package Name: `com.cornez.bullseye`
- Android Form: Phone and Tablet
- Minimum SDK: API 18: Android 4.3 (Jelly Bean)
- Target SDK: API 21: Android 5.0 (Lollipop)
- Compile with: API 21: Android 5.0 (Lollipop)
- Activity Name: `MyActivity`

The icon launcher will remain set to the Android default `ic_launcher.png` file. No changes are required. The final project structure resembles the structure shown in Figure 5-11.



**FIGURE 5-11** The Project Structure of the Bull's-Eye application.

MyActivity is the main Activity for the application. BullsEyeView is an extended View class with an implemented onDraw() method. The Bull's-Eye graphic will be drawn into this View.

The AndroidManifest.xml file contains the settings for the main Activity. The android:theme setting is overridden in MyActivity. The XML code for AndroidManifest.xml appears as follows. No layout XML file is used in the Bull's-Eye application. Instead, a layout will be constructed programmatically in Java.

AndroidManifest.xml	
1	<?xml version="1.0" encoding="utf-8"?>
2	<manifest xmlns:android="http://schemas.android.com/apk/res/android"
3	package="com.cornez.bullseye" >
4	
5	<application
6	android:allowBackup="true"
7	android:icon="@drawable/ic_launcher"
8	android:label="@string/app_name"
9	android:theme="@style/AppTheme" >
10	<activity
11	android:name=".MyActivity"
12	android:label="@string/app_name" >
13	<intent-filter>
14	<action android:name="android.intent.action.MAIN" />
15	
16	<category
17	android:name="android.intent.category.LAUNCHER" />
18	</intent-filter>
19	</activity>
20	</application>
21	
22	</manifest>

### Part 3: Source Code for Application

The Activity for the application is MyActivity. As a drawing app experiment, it performs two simple functions: (1) it configures the screen window, and (2) it sets the layout to a drawable. The drawable will be constructed programmatically.

Line 21:       requestWindowFeature () enables specific window features, specifically a “no title” feature. This turns off the title at the top of the screen.

Lines 22–24:   getWindow() retrieves the current window for the running activity. This call is used to access parts of the Window API directly.

A call to `setFlags(int flags, int mask)` sets the behavioral options of a window. The window flag `FLAG_FULLSCREEN` is used to display the Activity in fullscreen. The flags parameter specifies the window flags, and the mask parameter identifies which of the window flag bits to modify. To hide the Status Bar on the device when the application is started, the window flag `FLAG_FULLSCREEN` is passed to both parameters.

The following is a brief list of useful window flags:

- a. `FLAG_FULLSCREEN` hides all screen decorations (such as the status bar) while this window is displayed.
- b. `FLAG_KEEP_SCREEN_ON` keeps the device's screen turned on as long as the window is visible to the user.
- c. `FLAG_SHOW_WALLPAPER` allows the system wallpaper to be shown behind the activity window.
- d. `FLAG_TRANSLUCENT_NAVIGATION` sets the navigation bar to translucent, with minimal system-provided background protection.

Window flags are often required to be set before the first call to `setContentView(View)`.

Line 26: A `BullseyeView` `View` is instantiated. This `View`, containing the bull's-eye graphic, will be bound to `MyActivity` in the next instruction.

Line 27: The activity content is set to the explicit view, `bullseyeView`. This view is placed directly into the activity's view hierarchy. When the user alters the orientation of the device, the activity is restarted. The restart in the activity will force a call to the `onDraw()` method in the `BullseyeView` class.

MyActivity.java

```
1 package com.cornez.bullseye;
2
3 import android.app.Activity;
4 import android.os.Bundle;
5 import android.view.Menu;
6 import android.view.MenuItem;
7 import android.view.Window;
8 import android.view.WindowManager;
9
10
```

```
11 public class MyActivity extends Activity {
12
13     BullsEyeView bullsEyeView;
14
15     @Override
16     protected void onCreate(Bundle savedInstanceState) {
17         super.onCreate(savedInstanceState);
18
19         //WINDOW PROPERTIES ARE SET
20         //THIS ANDROID WINDOW WILL NOT FEATURE A TITLE
21         requestWindowFeature(Window.FEATURE_NO_TITLE);
22         getWindow().setFlags(
23             WindowManager.LayoutParams.FLAG_FULLSCREEN,
24             WindowManager.LayoutParams.FLAG_FULLSCREEN);
25
26         bullsEyeView = new BullsEyeView(this);
27         setContentView(bullsEyeView);
28     }
29
30
31
32     @Override
33     public boolean onCreateOptionsMenu(Menu menu) {
34         // Inflate the menu.
35         getMenuInflater().inflate(R.menu.my, menu);
36         return true;
37     }
38
39     @Override
40     public boolean onOptionsItemSelected(MenuItem item) {
41         // Handle action bar item clicks here. The action bar will
42         // automatically handle clicks on the Home/Up button, so long
43         // as you specify a parent activity in AndroidManifest.xml.
44         int id = item.getItemId();
45         if (id == R.id.action_settings) {
46             return true;
47         }
48         return super.onOptionsItemSelected(item);
49     }
50 }
```

The `BullsEyeView.java` class is the data model for the graphic element. A mix of colors is applied to a `Paint` object, which is used to fill a single circle. After each circle is drawn, paint is mixed again and the process repeats itself.

Line 5:        The `Color` class defines methods for creating and converting color integers. Colors are made up of 4 bytes: alpha, red, green, and blue. Transparency is stored solely in the alpha component, not in the color components. Color values range between 0 and

255: 0 means no contribution for that color, and 255 means 100% contribution. For example, opaque-black would be 0xFF000000 (100% opaque, but no contributions from red, green, or blue).

- Line 6: The `Paint` class holds the style and color information about how to draw geometries, text, and bitmaps.
- Line 7: The `Style` class provides the specifics for how a primitive shape is drawn. For example, it can be filled with color, have a stroke applied, or both. The default is a filled shape.
- Line 10: `BullsEyeView` is extended as a `View`. This allows `BullsEyeView` to support drawing and event handling.
- Lines 22–27: The color values for red, green, and blue are mixed to fill the first circle of the bull's eye. An instance of `Paint` is created so new paint can be applied to newly drawn circles.
- Line 30: `onDraw()` is implemented to perform the drawing needs for this `View`. The canvas represents the background for the drawing.
- Lines 31–32: `getWidth()` returns the width of the current drawing layer, the canvas. The center of the canvas is the width divided by 2 and the height divided by 2.
- Lines 41–59: Five circles will be drawn, each one smaller than the previous one. Each circle is colored with a new mix of red, green, and blue and is drawn on top of the previous circle.

BullsEyeView.java

```
1 package com.cornez.bullseye;
2
3 import android.content.Context;
4 import android.graphics.Canvas;
5 import android.graphics.Color;
6 import android.graphics.Paint;
7 import android.graphics.Paint.Style;
8 import android.view.View;
9
10 public class BullsEyeView extends View {
11
12     private Paint paint;
13
14     //VALUES FOR THE RED, GREEN, AND BLUE VALUES
15     private int redVal;
16     private int greenVal;
17     private int blueVal;
18
19     public BullsEyeView (Context context) {
20         super(context);
21     }
```



```
22         //INITIAL VALUES FOR RED, GREEN, AND BLUE
23         redVal = 248;
24         greenVal = 232;
25         blueVal = 198;
26
27         paint = new Paint();
28     }
29
30     public void onDraw (Canvas canvas) {
31         //INITIALIZE THE CENTER OF THE CANVAS
32         float centerX = canvas.getWidth() / 2;
33         float centerY = canvas.getHeight() / 2;
34
35         //INITIALIZE THE RADIUS FOR THE FIRST RING
36         float radius = canvas.getHeight() / 2;
37
38         //TASK 1: FILL THE ENTIRE CANVAS WITH A BEIGE COLOR
39         canvas.drawRGB(194, 183, 158);
40
41         //TASK 2: DRAW A SET OF FIVE RINGS
42         int ringRed = redVal;
43         int ringGreen = greenVal;
44         int ringBlue = blueVal;
45
46         for (int i = 1; i <= 5; i++) {
47             //DRAW A SINGLE RING
48             paint.setStyle(Style.FILL);
49             paint.setColor(Color.rgb(ringRed, ringGreen, ringBlue));
50             canvas.drawCircle(centerX, centerY, radius, paint);
51
52             //RESET THE COLOR AND SIZE FOR THE NEXT RING
53             ringRed -= 13;
54             ringGreen -= 13;
55             ringBlue -= 13;
56             radius -= 120;
57         }
58     }
59 }
```

## 5.4 Recursive Drawing

Drawing can be applied to an application in varied ways. One area that relies heavily on drawing is the generation of fractal terrain. The terrain in an environment can be a crucial user interface element, particularly in a game application. Fractals are inherent to the development of terrain drawings, mainly because fractal-based terrain is simple to implement and scales well, as fractals are self-similar. Recursive drawing algorithms are often used to produce landscape fractal images.

The math behind recursive drawing algorithms (fractals) can be simple or complex, depending on the requirements of the application. The key concept behind any fractal is self-similarity. An object is said to be self-similar when magnified subsets of the object are identical to the whole and to each other. Landscape terrain falls into the “self-similar” category. For example, a single branch of a tree has the same structure as the tree itself. A recursively generated tree, such as in Figure 5-12, still looks like a tree, regardless of the scale in which it is displayed.



**FIGURE 5-12** Recursive drawing.

The textures of natural objects, such as trees, mountains, and stones, have fractal properties. Almost any item that does not have an absolutely smooth, glassy surface contains bumps, pits, and grooves. The sizes of these features vary by fractal laws; many will be very small, some will be bigger, and a few are relatively large. The distribution of these bumps, pits, and grooves across an object’s surface is not entirely random, but it has a fractal nature.

When drawing fractal elements in an application, realistic-looking texture can be rendered quickly. This has advantages to the alternative, which would create hundreds of still images that would occupy memory on a device.

To be effective on a mobile device, the terrain needs to meet a number of requirements, many of which can be mutually opposing. A terrain should appear to be continuous to the user and must render quickly, yet it must be simplified, where possible, to reduce the load on a low-processing device.

---

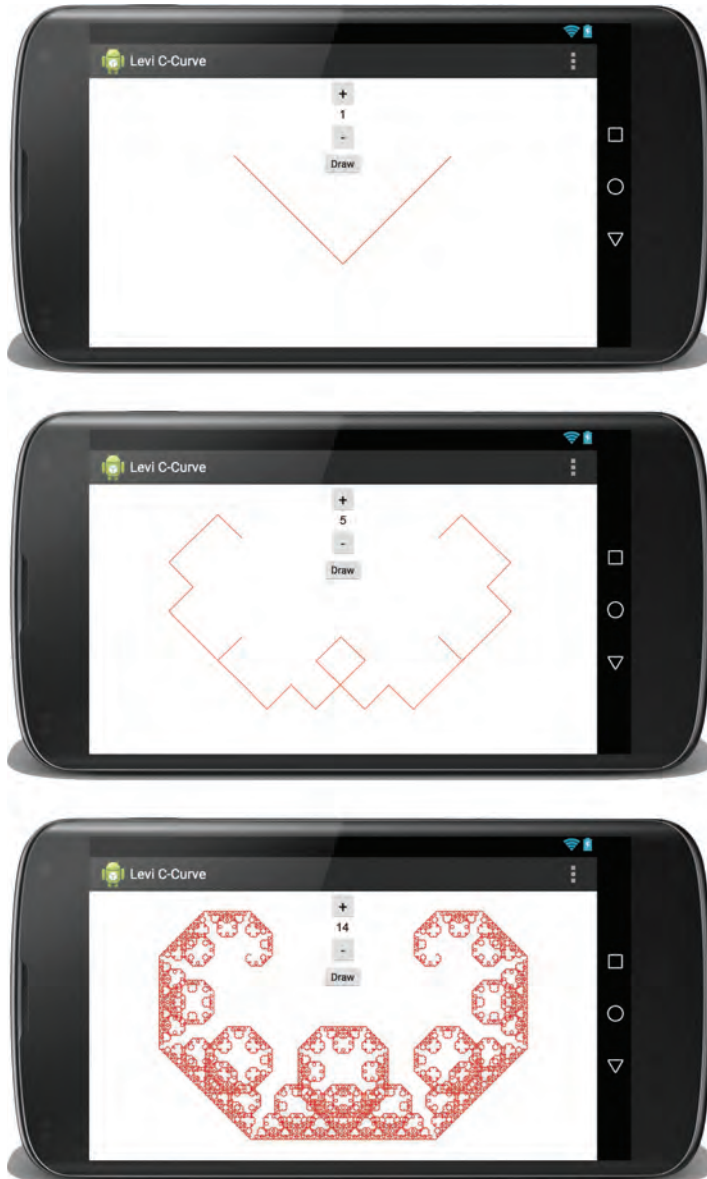
### **Lab Example 5-3: C-Curve Recursive Drawing**

---

This lab experiments with the concept of layering—including drawn elements—onto an existing layout. This is done programmatically, as well as with an XML layout. We will use recursive drawing to construct a well-recognized fractal pattern, known as the C-curve.

## Part 1: The Design

The C-curve fractal is a geometric pattern that can be subdivided into many smaller imitations, that is, self-similar copies of the larger pattern. If you enlarge any small portion of the complete fractal, it will have the same structure as the larger complete work. Figure 5-13 shows several screenshots of the C-curve application.



**FIGURE 5-13** Screenshots of the C-curve application.

When the application launches, the user will see a “number stepper.” This user interface control allows the user to select a C-curve level, which determines the complexity of the final design.

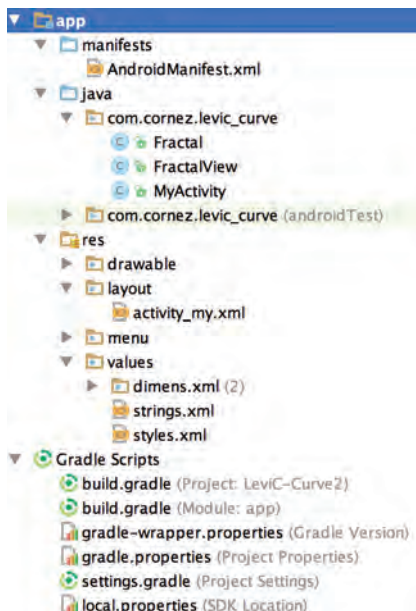
## Part 2: Application Structure and Setup

The settings for the application are as follows:

- Application Name: Levi C-Curve
- Project Name: LeviC\_Curve
- Package Name: `com.cornez.levicurve`
- Android Form: Phone and Tablet
- Minimum SDK: API 18: Android 4.3 (Jelly Bean)
- Target SDK: API 21: Android 5.0 (Lollipop)
- Compile with: API 21: Android 5.0 (Lollipop)
- Activity Name: `MyActivity`
- Layout Name: `activity_my`

The launcher is set to Android’s default `ic_launcher.png` file. No bitmap files are used in this application.

The final project structure, shown in Figure 5-14, contains three source files and one layout file. The C-curve application uses a single activity, `MyActivity.java`, which sets the screen user interface to `activity_my.xml`.



**FIGURE 5-14** Project structure for the Levi C-Curve application.

The theme for the application has been set in the `AndroidManifest` file, along with a landscape screen orientation. The XML code listing for `AndroidManifest.xml` appears as follows:

```

AndroidManifest.xml
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      package="com.cornez.levic_curve" >
4
5      <application
6          android:allowBackup="true"
7          android:icon="@drawable/ic_launcher"
8          android:label="@string/app_name"
9          android:theme="@style/AppTheme" >
10         <activity
11             android:screenOrientation="landscape"
12             android:name=".MyActivity"
13             android:label="@string/app_name" >
14             <intent-filter>
15                 <action android:name="android.intent.action.MAIN" />
16
17                 <category
18                     android:name="android.intent.category.LAUNCHER" />
19             </intent-filter>
20         </activity>
21     </application>
22
23 </manifest>

```

### Part 3: The User Interface Design

A `NumberPicker` widget enables the user to select a number from a predefined range. This lab features a custom-built `NumberPicker`, or rather, a number stepper, which is a control that is used to limit the range of values that a user can input. The number stepper constrains input within the `onClick` handler. By using an increase button or decrease button, the user can choose a level from 0 through 14. As shown in Figure 5-15,



**FIGURE 5-15** A “number stepper” is built, using `Buttons` and a `TextView`.

the decrease button contains a minus sign, and the increase button contains a plus sign. The user does not have direct access to the numeric value that appears between these two buttons.

The `strings.xml` file defines the button labels used by the number stepper. In addition to the number stepper's increment and decrement buttons, a draw button activates the C-curve recursive drawing. The label is defined on Line 9. The XML code for `strings.xml` is listed as follows:

```
strings.xml
1  <?xml version="1.0" encoding="utf-8"?>
2  <resources>
3
4      <string name="app_name">Levi C-Curve</string>
5      <string name="hello_world">Hello world!</string>
6      <string name="action_settings">Settings</string>
7
8      <!--BUTTON LABELS -->
9      <string name="draw_btn">Draw</string>
10     <string name="up_btn">+</string>
11     <string name="down_btn">-</string>
12     <string name="start_level">1</string>
13
14 </resources>
```



**FIGURE 5-16** The layout structure for `activity_my.xml`.

The View hierarchical structure for the layout file `activity_my.xml` is shown in Figure 5-16. The number stepper contains `button1`, `textView1`, and `button2`. The root element in `activity_my.xml` is set to a `RelativeLayout`. The XML code listing for `activity_my.xml` is shown below.

Line 6: It is important that the `RelativeLayout` root is given an identifier name, as this view will be referenced in the main activity of the application.

- Line 15: The method `stepUp()` is the `onClick` event handler for the increment button.
- Line 33: The method `stepDown()` is the `onClick` event handler for the decrement button.
- Line 45: The method `drawFractal()` handles the `onClick` event for the Draw button.

activity\_my.xml

```
1 <RelativeLayout
2   xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:tools="http://schemas.android.com/tools"
4     android:layout_width="match_parent"
5     android:layout_height="match_parent"
6     android:id="@+id/relativeLayout"
7     tools:context=".MyActivity">
8
9     <Button
10      android:id="@+id/button1"
11      android:layout_width="35dp"
12      android:layout_height="35dp"
13      android:layout_alignParentTop="true"
14      android:layout_centerHorizontal="true"
15      android:onClick="stepUp"
16      android:text="@string/up_btn"
17      android:gravity="center" />
18
19     <TextView
20      android:id="@+id/textView1"
21      android:layout_width="wrap_content"
22      android:layout_height="wrap_content"
23      android:layout_below="@+id/button1"
24      android:layout_centerHorizontal="true"
25      android:text="@string/start_level" />
26
27     <Button
28      android:id="@+id/button2"
29      android:layout_width="35dp"
30      android:layout_height="35dp"
31      android:layout_below="@+id/textView1"
32      android:layout_centerHorizontal="true"
33      android:onClick="stepDown"
34      android:text="@string/down_btn"
35      android:gravity="center" />
36
37     <Button
38      android:id="@+id/button3"
39      style="?android:attr/buttonStyleSmall"
```

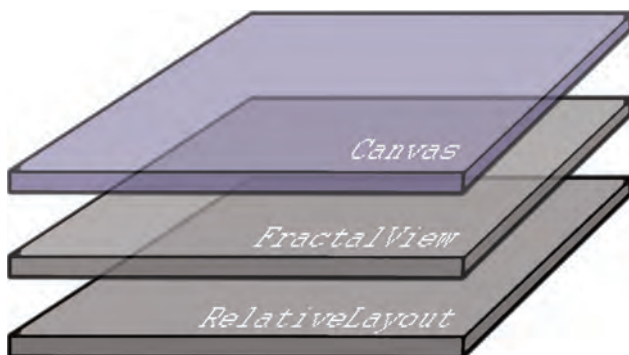


```
40     android:layout_width="wrap_content"
41     android:layout_height="30dp"
42     android:layout_below="@+id/button2"
43     android:layout_centerHorizontal="true"
44     android:text="@string/draw_btn"
45     android:onClick="drawFractal"
46     android:textSize="12sp" />
47
48 </RelativeLayout>
```

## Part 4: Source Code for Application

`MyActivity`, launched when the application is loaded, collects the user's input and initiates the recursive drawing based on the input. `MyActivity` is primarily responsible for arranging the relevant views of the user interface and presenting them in appropriate places on the screen.

- Lines 14–15: `levelSTV` is the `TextView` that is used to display the numeric level as a string to the user. The integer variable `level` is the numeric value. This variable is initialized to one on Line 29.
- Lines 16–17: The user interface is arranged in layers. The main layout is the `RelativeLayout` layer, the root element of `activity_my.xml`. A `FractalView` layer is added to this root element, as shown in Figure 5-17. The `FractalView` object will hold a `Canvas`, the final layer.
- Line 27: The `FractalView` object is added as a child view to the `RelativeLayout`. The zero position is the index position at which the child object has been added.
- Line 30: A reference to the levels `TextView` is assigned to `levelSTV`, which is the numeric text element of the number stepper.



**FIGURE 5-17** Layers applied to the C-curve user interface.

MyActivity.java

```
1 package com.cornez.levic_curve;
2
3 import android.app.Activity;
4 import android.os.Bundle;
5 import android.view.Menu;
6 import android.view.MenuItem;
7 import android.view.View;
8 import android.widget.RelativeLayout;
9 import android.widget.TextView;
10
11
12 public class MyActivity extends Activity {
13
14     private TextView levelsTV;
15     private Integer level;
16     private RelativeLayout relativeLayout;
17     private FractalView fractalView;
18
19     @Override
20     protected void onCreate(Bundle savedInstanceState) {
21         super.onCreate(savedInstanceState);
22         setContentView(R.layout.activity_my);
23
24         relativeLayout = (RelativeLayout)
25             findViewById(R.id.relativeLayout);
26         fractalView = new FractalView(this);
27         relativeLayout.addView(fractalView, 0);
28
29         level = 1;
30         levelsTV = (TextView) findViewById(R.id.textView1);
31     }
32 }
```

Lines 33–36: `drawFractal()` is the event handler for the “draw” button. The fractal level, input by the user in the number stepper, is set. The `invalidate()` method is called to invalidate the entire `fractalView`. This will result in `onDraw(android.graphics.Canvas)` being called, which will draw the lines that make up the C-curve fractal.

Lines 38–51: The method `stepUp()` is the `onClick` event handler for the increment button. The method `stepDown()` is the `onClick` event handler for the decrement button. Both handlers increment/decrement the level variable and update the `TextView` display for the number stepper control.

MyActivity.java (continued)

```
33     public void drawFractal(View view) {
34         fractalView.level = level;
35         fractalView.invalidate();
36     }
37
38     // NUMBERS FOR STEP UP AND STEP DOWN CAN RANGE FROM 1 THROUGH 14
39     public void stepUp(View view) {
40         if (level < 14) {
41             level++;
42             levelsTV.setText(level.toString());
43         }
44     }
45
46     public void stepDown(View view) {
47         if (level > 1) {
48             level--;
49             levelsTV.setText(level.toString());
50         }
51     }
52
53     @Override
54     public boolean onCreateOptionsMenu(Menu menu) {
55         // Inflate the menu.
56         getMenuInflater().inflate(R.menu.my, menu);
57         return true;
58     }
59
60     @Override
61     public boolean onOptionsItemSelected(MenuItem item) {
62         // Handle action bar item clicks here. The action bar will
63         // automatically handle clicks on the Home/Up button, so long
64         // as you specify a parent activity in AndroidManifest.xml.
65         int id = item.getItemId();
66         if (id == R.id.action_settings) {
67             return true;
68         }
69         return super.onOptionsItemSelected(item);
70     }
71 }
```

The FractalView class is a View that occupies a rectangular area on the RelativeLayout in activity\_my.xml. FractalView is responsible for drawing the C-curve fractal.

Lines 8–12: The C-curve fractal is constructed with many line drawings that recursively begin at point  $(x_1, y_1)$  and end at point  $(x_2, y_2)$ .

These line drawings will occur on the canvas provided by this FractalView class.

Line 14: The Fractal class is the fractal generator.

Lines 24–36: The method `onDraw()` initializes the fractal ( $x1$ ,  $y1$ ) and ( $x2$ ,  $y2$ ) settings and then calls `fractal.drawCurve()` to draw the fractal lines to the canvas. `onDraw()` is called by `fractalView.invalidate()` in `MyActivity`.

Lines 25–29: The initial points for the first line in the fractal are set so that the C-curve appears in the center of the screen, no matter the screen size.

Line 32: If a previous recursive drawing appears on the canvas, it is wiped clean by covering the canvas in white paint.

Line 35: When calling `drawCCurve()` to generate the recursive drawing, it must be past the Canvas object provided by FractalView.

FractalView.java

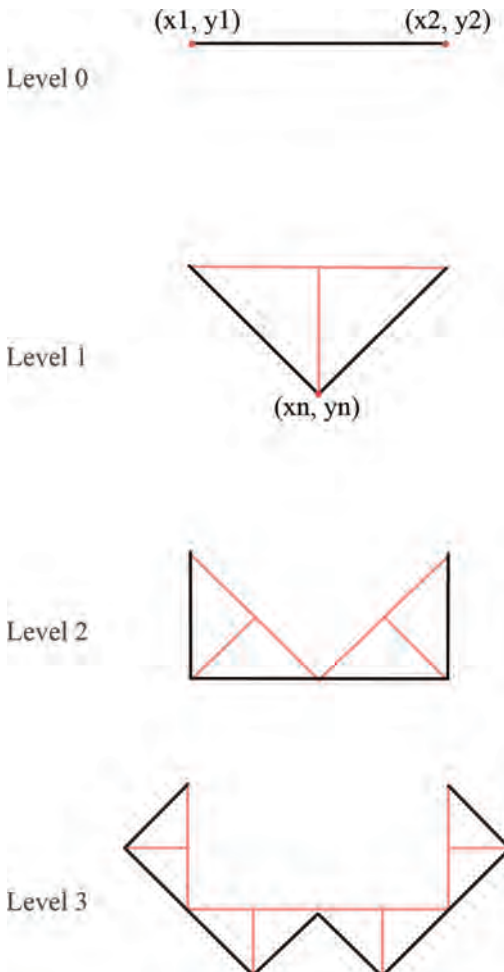
```
1 package com.cornez.levic_curve;
2 import android.content.Context;
3 import android.graphics.Canvas;
4 import android.view.View;
5
6 public class FractalView extends View {
7
8     private float x1;
9     private float y1;
10    private float x2;
11    private float y2;
12    public int level;
13
14    private Fractal fractal;
15
16    public FractalView (Context context) {
17        super (context);
18
19        //CREATE A FRACTAL OBJECT
20        level = 2;
21        fractal = new Fractal();
22    }
23
24    protected void onDraw (Canvas canvas){
25        //TASK 1: GET THE DIMENSIONS OF THE CANVAS
26        x1 = canvas.getWidth() / 3;
27        y1 = canvas.getHeight() / 4;
28        x2 = canvas.getWidth() - x1;
29        y2 = y1;
30    }
```

```

31 //TASK 2: FILL THE CANVAS WITH WHITE PAINT
32 canvas.drawRGB(255, 255, 255);
33
34 //TASK 3: DRAW THE CCURVE
35 fractal.drawCCurve(canvas, x1, y1, x2, y2, level);
36 }
37
38 }

```

In this application, the C-curve is generated recursively. To better understand its construction, the first four levels are illustrated in Figure 5-18. Consider the most primitive level, Level 0. At this level, the fractal starts with a straight line drawn from point  $(x1, y1)$  to the point  $(x2, y2)$ . This basic line always occurs at the primitive state in the recursion.



**FIGURE 5-18** The C-curve from Level 0 to Level 3.

At Level 1 of the fractal, an isosceles right triangle is built using the line at Level 0 as its hypotenuse. The first line is drawn from the point  $(x_1, y_1)$  to the point  $(x_n, y_n)$ . The second line is drawn between points  $(x_n, y_n)$  and  $(x_2, y_2)$ .

The point at  $(x_n, y_n)$  is computed as follows:

$$x_n = (x_1 + x_2)/2 + (y_1 - y_2)/2$$

$$y_n = (x_2 - x_1)/2 + (y_1 + y_2)/2$$

At Level 2, two new lines are used to form the foundation for another right-angled isosceles triangle. At every new level, two new lines from two sides of a respective triangle are constructed to replace an existing line. At Level 0, there are  $2^0$  lines. Level 1 draws  $2^1$  lines, and Level 3 generates  $2^3$  lines. Level  $n$  would be constructed with  $2^n$  lines.

The `Fractal` class is simply the engine that constructs a C-curve. The class contains a recursive method, but it contains no data.

- Lines 8–9: The `Canvas` parameter is required in order to draw to the `FractalView`.
- Lines 17–18: The primitive state for the recursion is Level 0. This results in a single line drawn to the canvas.
- Lines 19–24: The method `drawCCurve()` is called recursively to replace one line with two new lines, based on the computation of an  $x, y$  point in an isosceles right triangle.

Fractal.java

```

1 package com.cornez.levic_curve;
2
3 import android.graphics.Canvas;
4 import android.graphics.Color;
5 import android.graphics.Paint;
6
7 public class Fractal {
8
9     public void drawCCurve (Canvas canvas, float x1,
10         float y1, float x2, float y2, int level){
11
12         //PAINT COLOR IS SET TO RED
13         //LINE STROKE IS SET TO 1
14         Paint paint = new Paint();
15         paint.setColor(Color.rgb(255, 0, 0));
16         paint.setStrokeWidth(1);
17
18         if (level == 0)
19             canvas.drawLine(x1, y1, x2, y2, paint);
20         else {

```

```
21         float xn = (x1 + x2) / 2 + (y1 - y2) / 2;  
22         float yn = (x2 - x1) / 2 + (y1 + y2) / 2;  
23         drawCCurve(canvas, x1, y1, xn, yn, level -1);  
24         drawCCurve(canvas, xn, yn, x2, y2, level -1);  
25     }  
26 }  
27  
28 }
```

## 5.5 Frame-by-Frame and Film Loop Animations

Frame-by-frame animation is a sequence of images that are displayed in rapid succession, one after the other. A film loop is an animated sequence that you can use like a single View object. For example, to create an animation of a character dancing on the stage, a set of bitmaps is placed in the Drawable directory and the images are displayed on the screen within a View. To create a film loop, the sequence of bitmaps that shows the character dancing is created as a View object that can be animated across the screen. When the animation is run, the character dances and moves across the screen at the same time.

Any Drawable subclass that supports the `inflate()` method can be used to create an animation. An animated element can be defined in XML and instantiated in an application's source code. If a Drawable animation uses properties that will change during the runtime of the application, these values can be initialized once the object is instantiated.

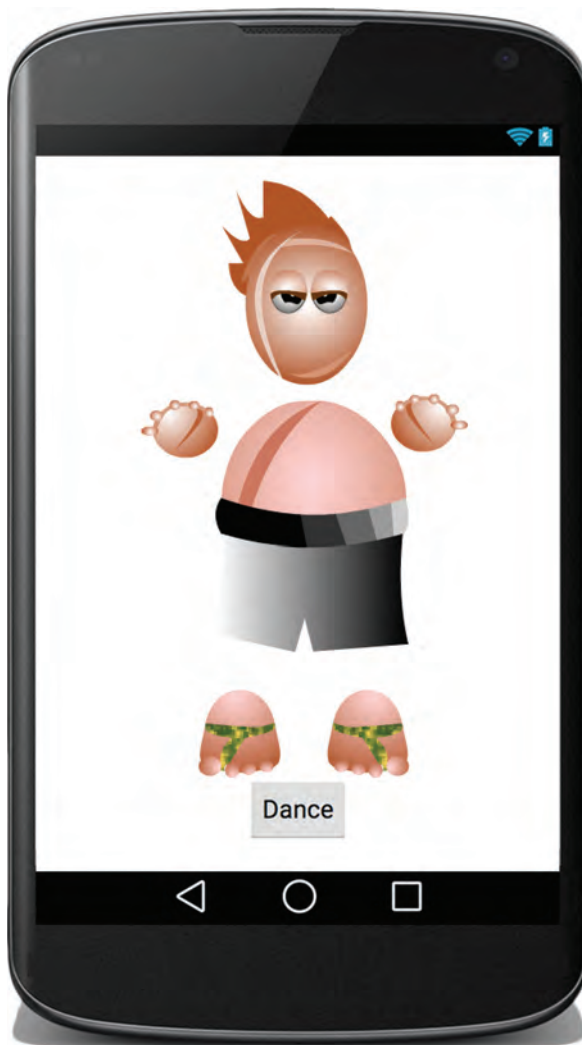
Transition animations are introduced in Chapter 3. Consider the following XML definition for a transition. The XML file, saved as `res/drawable/open_close.xml`, contains two Drawables defined in the `<item>` tag, `image_open`, and `image_close`. Once instantiated, this container of nested Drawables can be set as the content of an `ImageView`. Frame-by-frame animations and film loops can be built in a similar fashion.

```
1 <?xml version="1.0" encoding="utf-8"?>  
2  
3 <transition xmlns:android="http://schemas.android.com/apk/res/android">  
4     <item android:drawable="@drawable/image_open"></item>  
5     <item android:drawable="@drawable/image_close"></item>  
6 </transition>
```

Animations in Android are Drawable animations. They require a set of drawable items, such as the items used in the above transition example, `res/drawable/open_close.xml`.

A frame-by-frame animation is prepared as a series of timeline frames to be displayed in an ordered sequence with a time delay. Frame and film loop animations fall under the category of View animations because the ordered sequence of images appears within an `ImageView`.

To understand Drawable animation, consider the simple Dance application shown in Figure 5-19. When the application is launched for the first time, a still image of a cartoon character appears on the screen. Below the character is a button that can activate the frame-by-frame animation. The animation relies on a sequence of dance moves that form a complete dance when it is displayed as a running loop.



**FIGURE 5-19** Animation that begins with an `onClick` event.



The sequence of images comprising the dance moves are placed into the application's drawables folders. The bitmap images shown in Figure 5-20 were used to create the Dance animation. When giving images file names, consider using a numeric indicator for the order in which they will appear.

A Drawable animation relies on an ordered list specifying how the images will appear; this ordered list is defined as an XML file. The duration of an individual image on the screen is also specified in this animation resource file. For example, in the animation resource XML code below, the root element is `<animation-list>`. This root element is a container for a set of eight images that constitutes a dancing film loop. Each prebuilt image is located in the drawable directory, such as `drawable/dancer1`. The duration value for a frame image is an integer value (100 milliseconds) that indicates how long the item appears on the screen before it is replaced by the next image in the sequence.

The XML code for a Drawable dancing animation, `res/anim/dance_animation.xml`, appears as follows. The `res/anim` directory is created specifically for Drawable animations.

```
1 <animation-list
2 xmlns:android="http://schemas.android.com/apk/res/android"
3   android:oneshot="false" >
4
5   <item
6     android:drawable="@drawable/dancer1"
7     android:duration="100"/>
8   <item
9     android:drawable="@drawable/dancer2"
10    android:duration="100"/>
11  <item
12    android:drawable="@drawable/dancer3"
13    android:duration="100"/>
14  <item
15    android:drawable="@drawable/dancer4"
16    android:duration="100"/>
17  <item
18    android:drawable="@drawable/dancer5"
19    android:duration="100"/>
20  <item
21    android:drawable="@drawable/dancer6"
22    android:duration="100"/>
23  <item
24    android:drawable="@drawable/dancer7"
25    android:duration="100"/>
26  <item
27    android:drawable="@drawable/dancer8"
28    android:duration="100"/>
29
30 </animation-list>
```

An `ImageView` is the container used for holding the animation element. In the following code segment, an `ImageView`, `dancerView`, is referenced from the main activity layout.

- Lines 4–5: The Drawable animation `res/anim/dance_animation.xml` is set as the background resource for the `dancerView`. A Drawable object is then created, based on this background.
- Line 7: The animation is started.

```

1  ImageView dancerView = (ImageView) findViewById(R.id.imageView1);
2
3  dancerView.setBackgroundResource(R.drawable.dance_animation);
4  AnimationDrawable danceAnimation = (AnimationDrawable)
5      dancerView.getBackground();
6
7  danceAnimation.start();

```



**FIGURE 5-20** Bitmap images, numbered sequentially, create the Dance animation.

Film loops and frame-by-frame animations are suited only to cases where you have either already created the frame images or you plan to implement them as drawables. For more detailed control over animations, property or tween animations are best. These types of animations, both of which are more complex, are discussed in Chapter 6.

In many cases, an animation is intended to enhance interaction with UI items such as buttons, rather than to be a standalone component. Animations can provide the user with helpful visual cues. These animations should not intrude on a given operation or distract from it.

---

## ■ Lab Example 5-4: Animated Maze Chase

---

In this lab, we explore the use of a frame-by-frame animation that is programmatically added to the main activity layout. In addition, the main activity layout includes a drawn canvas. Interactions for this application feature both of these elements. Maze Chase is not a complete game in which a user competes against an opponent; instead, this is a lab exercise that examines how to combine a drawing algorithm with animation and interactivity.

### Part 1: The Design

---

The user who launches the application is presented with a graphic image of a “perfect” maze. A different maze is generated each time the application launches. A perfect maze is one in which exactly one path exists between any two given cells, such as the one shown in the application screenshot in Figure 5-21.

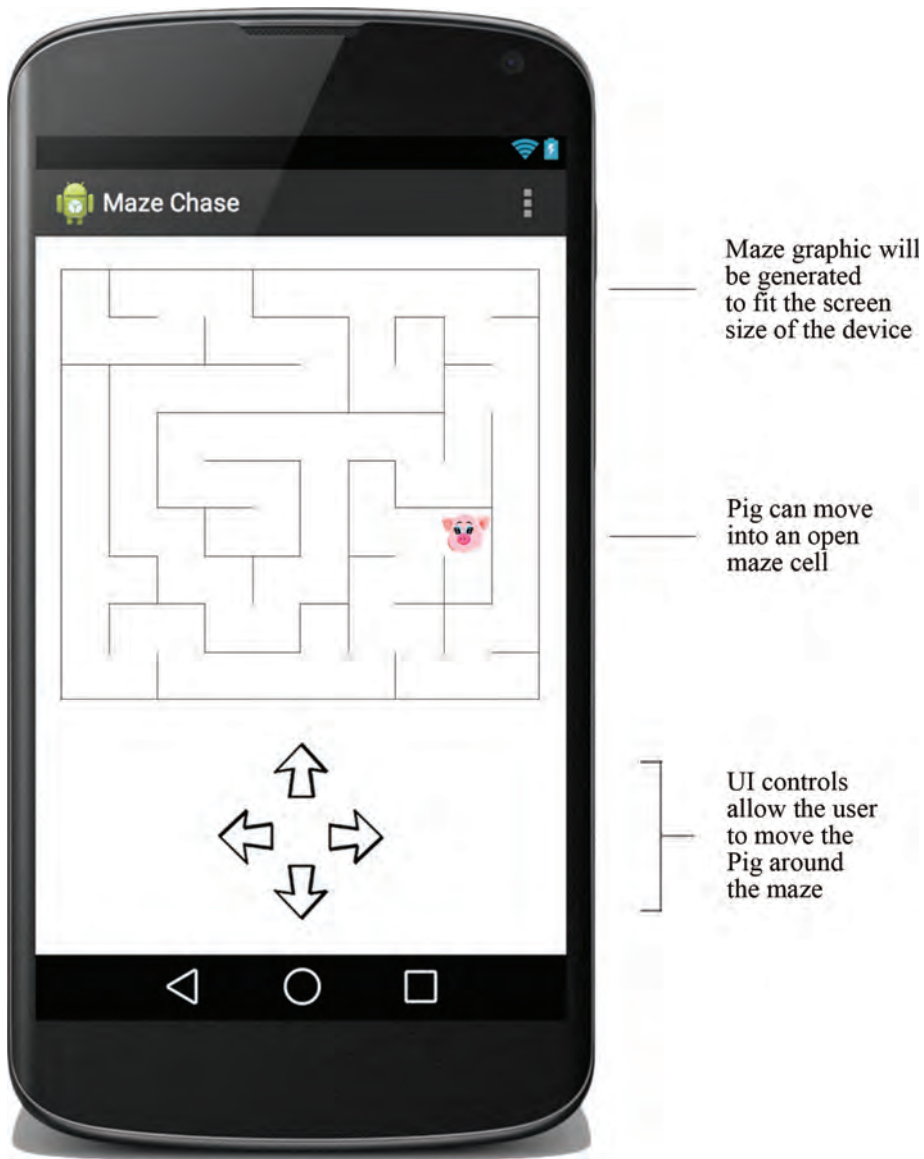
A film loop animation of a pig is positioned in a maze cell. The pig is animated with nuanced facial movements. The control buttons are provided to move the pig up, down, left, or right into an open cell. If a cell is walled off, the pig will remain in its cell.

### Part 2: Application Structure and Setup

---

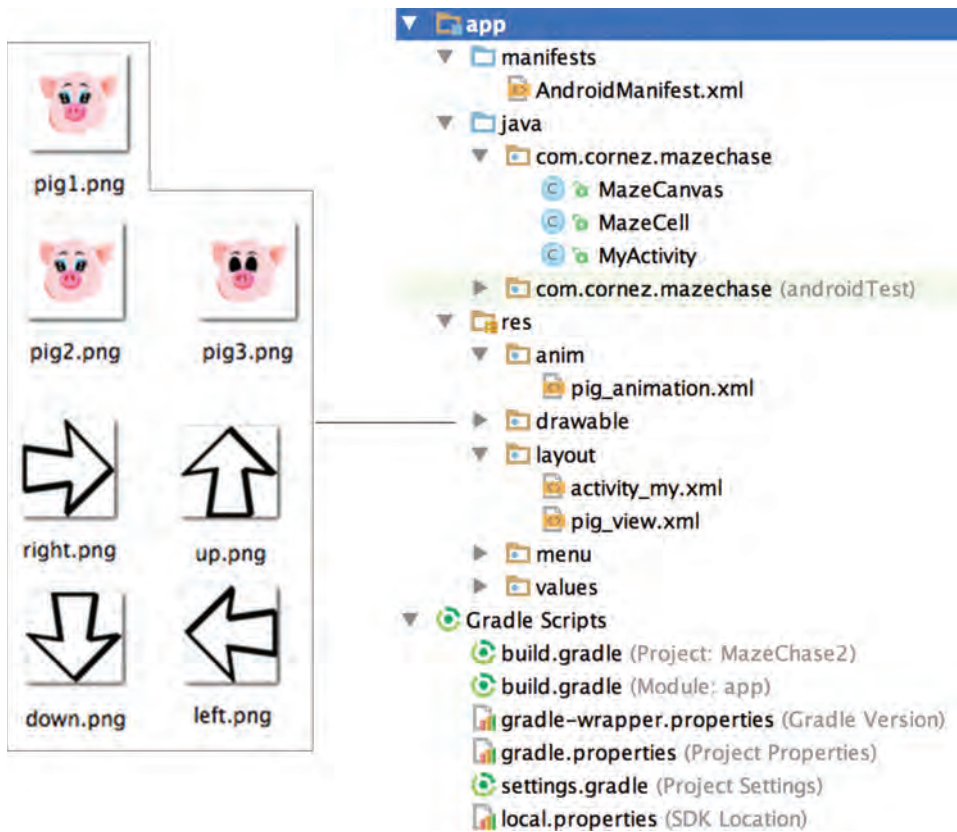
The settings for the Maze Chase application are as follows:

- Application Name: Maze Chase
- Project Name: MazeChase
- Package Name: com.cornez.mazechase
- Android Form: Phone and Tablet
- Minimum SDK: API 18: Android 4.3 (Jelly Bean)
- Target SDK: API 21: Android 5.0 (Lollipop)
- Compile with: API 21: Android 5.0 (Lollipop)
- Activity Name: MyActivity
- Layout Name: activity\_my



**FIGURE 5-21** An animated pig moves in a “perfect” maze.

The icon launcher for the application is set to Android’s default `ic_launcher.png` file. Seven additional bitmap files, used for the pig animation and the control buttons, are added to the drawable folders. Figure 5-22 shows the final project structure for the Maze Chase application. This application is driven by a single activity, `MyActivity`. The layout associated with `MyActivity` is `activity_my.xml`. Additional files are used for a drawing canvas, animation, and the data model for a maze cell.



**FIGURE 5-22** Project structure for the Maze Chase application.

The orientation of the screen is locked into portrait mode and the main activity is set to `MyActivity`. The XML code listing for `AndroidManifest.xml` appears as follows:

```

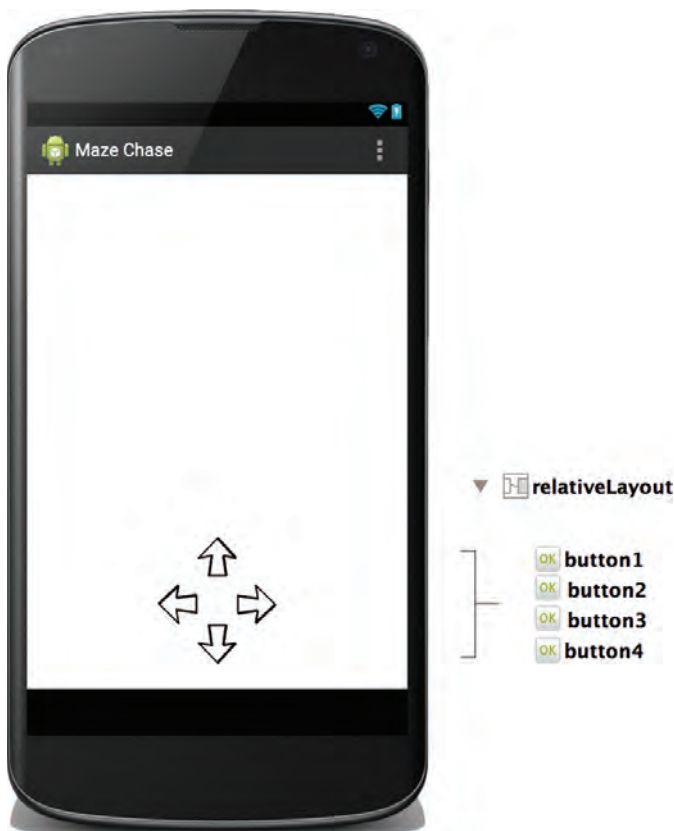
AndroidManifest.xml
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      package="com.cornez.mazechase" >
4
5      <application
6          android:allowBackup="true"
7          android:icon="@drawable/ic_launcher"
8          android:label="@string/app_name"
9          android:theme="@style/AppTheme" >
10         <activity
11             android:name=".MyActivity"
12             android:screenOrientation="portrait"
13             android:label="@string/app_name" >

```

```
14         <intent-filter>
15             <action android:name="android.intent.action.MAIN" />
16
17             <category
18                 android:name="android.intent.category.LAUNCHER" />
19         </intent-filter>
20     </activity>
21 </application>
22
23 </manifest>
```

### Part 3: The User Interface for the Application

The application's user interface is made up entirely of images. Figure 5-23 shows the control buttons of the user interface and the RelativeLayout root View. In the final user interface, a canvas for the maze drawing will be layered on top of the RelativeLayout, along with an animation.



**FIGURE 5-23** The activity\_my.xml layout structure.

Given that the user interface consists of graphics and an animation, it appears that text strings will not be needed. Nevertheless, a well-constructed graphical application requires content descriptions for the image elements, such as the control buttons and the animated pig that moves around the maze. The XML code for `strings.xml` includes the descriptions for the graphic component.

`strings.xml`

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <resources>
3
4      <string name="app_name">Maze Chase</string>
5      <string name="hello_world">Hello world!</string>
6      <string name="action_settings">Settings</string>
7
8      <string name="up_button">Up button</string>
9      <string name="down_button">Down button</string>
10     <string name="left_button">Left button</string>
11     <string name="right_button">Right button</string>
12     <string name="pig">Pig</string>
13
14 </resources>

```

The file that provides the layout elements for `MyActivity` is `activity_my.xml`. It contains a `RelativeLayout` root element. Its view hierarchical structure is shown in Figure 5-23. The XML code for this layout is shown below. `onClick` event handlers have been identified for each of the control buttons. An id has been applied to the root element, `@+id/relativeLayout`, which will be used by `MyActivity.java` to add a canvas layer for drawing a perfect maze.

`activity_my.xml`

```

1  <RelativeLayout
2  xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:tools="http://schemas.android.com/tools"
4      android:layout_width="match_parent"
5      android:layout_height="match_parent"
6
7      android:id="@+id/relativeLayout"
8      tools:context=".MyActivity">
9
10     <Button
11         android:id="@+id/button1"
12         style="?android:attr/buttonStyleSmall"
13         android:layout_width="40dp"
14         android:layout_height="40dp"
15         android:layout_alignParentBottom="true"
16         android:layout_centerHorizontal="true"

```

```

17         android:layout_marginBottom="113dp"
18         android:background="@drawable/up"
19         android:onClick="goUp" />
20
21     <Button
22         android:id="@+id/button2"
23         style="?android:attr/buttonStyleSmall"
24         android:layout_width="40dp"
25         android:layout_height="40dp"
26         android:layout_alignTop="@+id/button1"
27         android:layout_marginTop="48dp"
28         android:layout_toLeftOf="@+id/button1"
29         android:background="@drawable/left"
30         android:onClick="goLeft" />
31
32     <Button
33         android:id="@+id/button3"
34         style="?android:attr/buttonStyleSmall"
35         android:layout_width="40dp"
36         android:layout_height="40dp"
37         android:layout_alignTop="@+id/button2"
38         android:layout_toRightOf="@+id/button1"
39         android:background="@drawable/right"
40         android:onClick="goRight" />
41
42     <Button
43         android:id="@+id/button4"
44         style="?android:attr/buttonStyleSmall"
45         android:layout_width="40dp"
46         android:layout_height="40dp"
47         android:layout_below="@+id/button2"
48         android:layout_centerHorizontal="true"
49         android:background="@drawable/down"
50         android:onClick="goDown" />
51
52
53 </RelativeLayout>

```

## Part 4: Animation Resources

The animated component, the pig, takes the form of a film loop. The pig appears as an idle animation; it does not perform an action. The idle animation, featuring wiggling ears and blinking eyes, is constructed using three bitmap images. The bitmap images shown in Figure 5-24 contain very nuanced changes, but when they are placed as frames in a film loop, the character becomes vibrant. Small changes such as these can have an enormous impact in a game application.



The bitmap images shown in Figure 5-24 are all PNG files. A small border around each image has been left bare, making that portion of the image transparent. This allows the lines in the maze to show through when the pig is placed in a cell that is the same size as the pig.



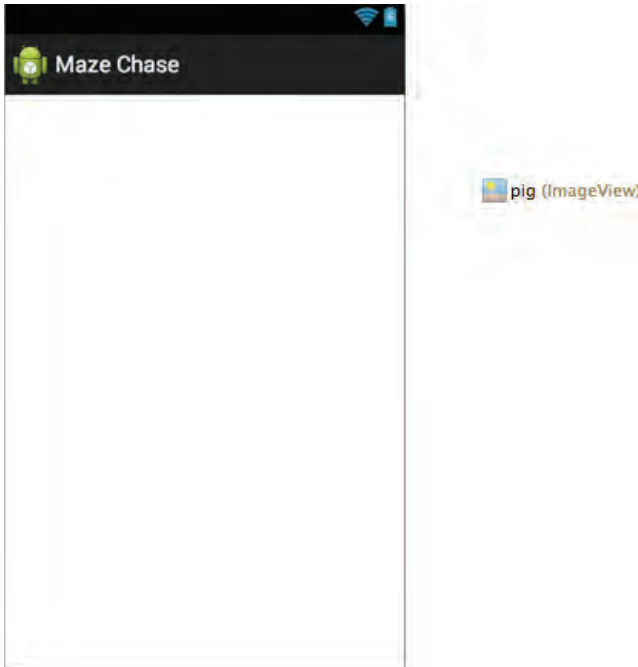
**FIGURE 5-24** Bitmaps representing frames in the pig animation.

The frame-by-frame animation is defined in XML. Each bitmap image is placed in a particular order in an animation list container, specifically the `<animation-list>`. The duration attribute is used to set the frame rate. For example, each pig image is displayed on the screen for the duration of 200 milliseconds. The XML code for `pig_animation.xml` is shown as follows. This file will be placed in `res/anim`.

```
pig_animation.xml
1  <?xml version="1.0" encoding="utf-8"?>
2
3  <animation-list
4  xmlns:android="http://schemas.android.com/apk/res/android"
5      android:oneshot="false">
6
7      <item
8          android:drawable="@drawable/pig1"
9          android:duration="200" />
10     <item
11         android:drawable="@drawable/pig2"
12         android:duration="200" />
13     <item
14         android:drawable="@drawable/pig3"
15         android:duration="200" />
16
17 </animation-list>
```

## Part 5: The Animated Film Loop

The `<animation-list>` in `pig_animation.xml` is used to define the frame structure of the animation. To create the actual film loop, an `ImageView` is used to hold the frame-by-frame structure. The film loop for the pig animation is stored as an independent XML file within `res/layout`. This file, named `pig_view.xml`, is structured as shown in Figure 5-25.



**FIGURE 5-25** The `pig_view.xml` layout file contains an `ImageView`.

As a container for the pig animation, `pig_view.xml` requires an `ImageView` as its root element. For this lab exercise, the XML code does not assign an image source, leaving the `View` empty, as shown in Figure 5-25. An identifier value, `@id/pig`, will be used to access this `View` from the Java source code, which will assign it the animation. The XML code for `pig_view.xml` is listed as follows:

`pig_view.xml`

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <ImageView xmlns:android="http://schemas.android.com/apk/res/android"
3     android:layout_width="match_parent"
4     android:layout_height="match_parent"
5     android:id="@+id/pig" >
6
7 </ImageView>
```

## Part 6: Source Code for Application

---

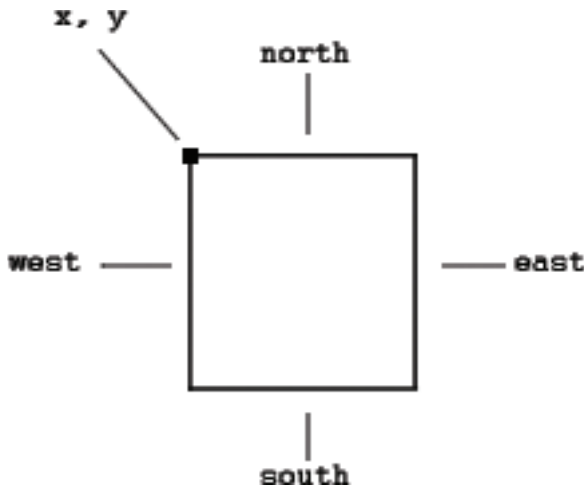
`MyActivity.java` drives the application.

- Line 5: `AnimationDrawable` is a `Drawable` container. This class allows the code to instantiate a frame-by-frame animation.
- Line 17: `pig` is the `ImageView` that stores the animated film loop of the pig.
- Lines 19–20: The (x, y) location of the pig element on the screen is stored in `xPos` and `yPos`. For simplicity, the location is defined in `MyActivity`, rather than in a separate class.
- Line 21: `maze` is a canvas for drawing, as well as the data that are used to render the drawing. The data are used to determine whether or not the pig is allowed to move into an adjacent cell.
- Lines 23, 31: Each cell in the maze can be uniquely identified by a number. The pig can be tracked by its `cellId`, which is initialized to 22: the 11<sup>th</sup> column in the 11<sup>th</sup> row.
- Lines 34–38: A perfect maze is drawn on a canvas, which is then added to the `RelativeLayout`.
- Lines 40–42: Prior to placing an existing `ImageView` onto the `RelativeLayout`, a `LayoutInflater` must be instantiated to perform the work.
- Lines 44–47: An instance of the pig film loop animation is inflated. This object's background is set to the pig frame-by-frame animation.
- Lines 50–52: An `AnimationDrawable` object is used to create the frame-by-frame animations that have been defined by the series of pig `Drawable` objects, which was set as the `View` object's background. The film loop animation is started.
- Lines 54–58: The film loop is running, but the user cannot see it because it has not yet been added to the screen. The position and scale of the pig `ImageView` are set, and the view is added to the `RelativeLayout` at index 1. By adding the `ImageView` at index 1, it will be layered above the drawing canvas containing the maze and the controls.

`MyActivity.java`

```
1 package com.cornez.mazechase;
2
3 import android.app.Activity;
4 import android.content.Context;
5 import android.graphics.drawable.AnimationDrawable;
6 import android.os.Bundle;
7 import android.view.LayoutInflater;
8 import android.view.Menu;
9 import android.view.MenuItem;
```

```
10 import android.view.View;
11 import android.widget.ImageView;
12 import android.widget.RelativeLayout;
13
14
15 public class MyActivity extends Activity {
16     private RelativeLayout relativeLayout;
17     private ImageView pig;
18     private LayoutInflater layoutInflater;
19     private float xPos;
20     private float yPos;
21     private MazeCanvas maze;
22
23     private int cellId;
24
25     @Override
26     protected void onCreate(Bundle savedInstanceState) {
27         super.onCreate(savedInstanceState);
28         setContentView(R.layout.activity_my);
29         xPos = 10;
30         yPos = 10;
31         cellId = 22;
32
33
34         // CONSTRUCT THE MAZE AND ADD IT TO THE RELATIVE LAYOUT
35         maze = new MazeCanvas(this);
36         relativeLayout = (RelativeLayout)
37             findViewById(R.id.relativeLayout);
38         relativeLayout.addView(maze, 0);
39
40         // CREATE A LAYOUT INFLATER
41         layoutInflater = (LayoutInflater)
42             getSystemService(Context.LAYOUT_INFLATER_SERVICE);
43
44         // SET THE BACKGROUND OF THE IMAGEVIEW TO THE PIG ANIMATION
45         pig = (ImageView) layoutInflater.inflate(
46             R.layout.pig_view, null);
47         pig.setBackgroundResource(R.anim.pig_animation);
48
49         // CREATE AN ANIMATION DRAWABLE OBJECT BASED ON THIS BACKGROUND
50         AnimationDrawable pigAnimate = (AnimationDrawable)
51             pig.getBackground();
52         pigAnimate.start();
53
54         pig.setX(xPos);
55         pig.setY(yPos);
56         pig.setScaleX(.15f);
57         pig.setScaleY(.15f);
58         relativeLayout.addView(pig, 1);
59
60     }
```



**FIGURE 5-26** The model for a MazeCell object.

The user moves the pig, using the arrow buttons provided by the user interface. The maze structure controls the pig's movement within the maze; in particular, the attributes of the maze cells control the movement. Four walls, as shown in Figure 5-26, characterize a maze cell: north, south, east, and west. If an adjacent wall to the pig is open, the pig may enter the cell. Lines 61–91 of `MyActivity.java` are the implementation of the `onClick` handlers that control the pig's movement into a given cell.

`MyActivity.java` (continued)

```

61 public void goUp(View view) {
62     if (maze.board[cellId].north == false){
63         yPos -= 100;
64         pig.setY(yPos);
65         cellId -= maze.COLS;
66     }
67 }
68
69 public void goLeft(View view) {
70     if (maze.board[cellId].west == false){
71         xPos -= 100;
72         pig.setX(xPos);
73         cellId--;
74     }
75 }
76
77 public void goRight(View view) {
78     if (maze.board[cellId].east == false){
79         xPos += 100;
80         pig.setX(xPos);
81         cellId++;

```

```
82     }
83 }
84
85 public void goDown(View view) {
86     if (maze.board[cellId].south == false){
87         yPos += 100;
88         pig.setY(yPos);
89         cellId += maze.COLS;
90     }
91 }
92
93
94 @Override
95 public boolean onCreateOptionsMenu(Menu menu) {
96     // Inflate the menu.
97     getMenuInflater().inflate(R.menu.my, menu);
98     return true;
99 }
100
101 @Override
102 public boolean onOptionsItemSelected(MenuItem item) {
103     // Handle action bar item clicks here. The action bar will
104     // automatically handle clicks on the Home/Up button, so long
105     // as you specify a parent activity in AndroidManifest.xml.
106     int id = item.getItemId();
107     if (id == R.id.action_settings) {
108         return true;
109     }
110     return super.onOptionsItemSelected(item);
111 }
112 }
```

The definition of the maze cell is represented by the `MazeCell.java` class. The Java code for this class is as follows. All the cell data members have been made public for quick and simple access.

MazeCell.java

```
1 package com.cornez.mazechase;
2 public class MazeCell {
3     public int x;
4     public int y;
5     public int id;
6     public boolean visited;
7     public boolean north;
8     public boolean south;
9     public boolean east;
10    public boolean west;
11 }
```

```
12 //NEW CELLS ARE INSTANTIATED WITH ALL WALLS INTACT
13 public MazeCell (int xPos, int yPos, int cellId){
14     x = xPos;
15     y = yPos;
16     visited = false;
17     north = true;
18     south = true;
19     east = true;
20     west = true;
21 }
22 }
```

## Part 7: Constructing a Perfect Maze

In computer science terms, a perfect maze is characterized as a minimal spanning tree over a set of cells. The task of carving out a path from one cell to the next is based on the concept of a depth-first-search, which uses a stack data structure.

Creating a perfect maze involves building the maze cell by cell while making sure that no loops exist and that no cell ends up isolated.

Lines 27–79: The maze for this application will be instantiated from the MazeCanvas class. Each maze cell will be drawn row by row.

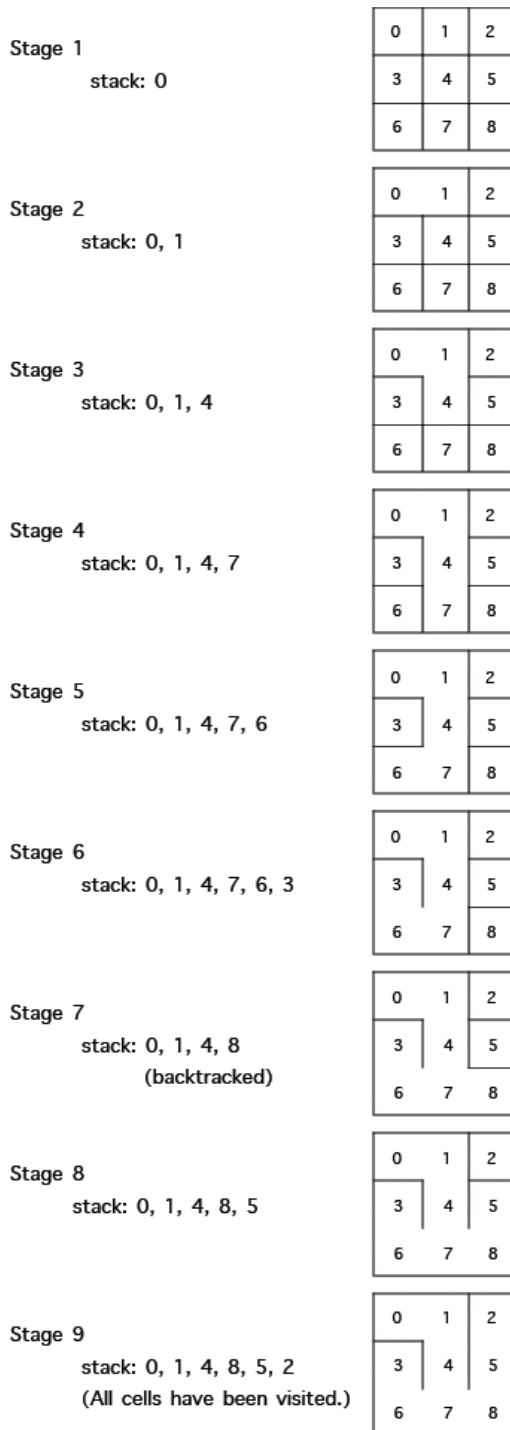
Lines 81–161: The `backtracker()` method belongs to a class of algorithms that share a common operational goal:

1. Begin with an array of maze cells with all of the walls intact.
2. Choose a starting cell.
3. Repeatedly select a random adjacent cell in the maze—one that has been unvisited—and open the wall between the two. Continue to do this until every cell has been visited and a wall has been eliminated during the visit.

The resulting maze contains no circular paths; every cell is connected to every other cell by exactly one path. The `backtracker()` method uses an iterative loop and a stack to carve out its paths.

The process of creating a perfect maze, such as the one shown in Figure 5-21, is done in stages. In Stage 1, the maze is built containing a two-dimensional array representing the maze cells. As shown in the first image of Figure 5-27, all the walls are initially intact. An empty stack is also constructed. The stack will be used to ensure that no loops exist in the path and that no cells end up isolated.

Stage 1 is the beginning of the backtracker procedure. The starting cell, 0, is selected and placed on the stack. Cell 0 cannot be visited again. At each stage in the process, a random, unvisited cell is chosen from the possible adjacent cells. Once the cell has been selected, it is tagged as visited, pushed onto the stack, and its adjacent wall is eliminated.



I FIGURE 5-27



In Stage 7, no unvisited cells are adjacent to cell 3. At this point, the stack is used for backtracking to a cell that has unvisited neighbors. Finally, at Stage 9, the process is complete once all the cells have been visited.

MazeCanvas.java

```
1 package com.cornez.mazechase;
2
3 import android.content.Context;
4 import android.graphics.Canvas;
5 import android.graphics.Color;
6 import android.graphics.Paint;
7 import android.view.View;
8
9 import java.util.Stack;
10
11 public class MazeCanvas extends View {
12
13     //MAZE DIMENSIONS
14     public final int COLS = 10;
15     public final int ROWS = 9;
16     final int N_CELLS = COLS * ROWS;
17     final int SIZE = 100;
18     final int OFFSET = 100;
19
20
21     //ARRAY OF MAZE CELLS
22     public MazeCell [] board;
23
24     private Paint paint;
25
26
27     public MazeCanvas (Context context){
28
29         super(context);
30
31         //TASK 1: DECLARE A MAZE ARRAY OF SIZE N_CELLS TO HOLD THE CELLS
32         board = new MazeCell[N_CELLS];
33
34         //TASK 2: INSTANTIATE CELL OBJECTS FOR EACH CELL IN THE MAZE
35         int cellId = 0;
36         for (int r = 0; r < ROWS; r++){
37             for (int c = 0; c < COLS; c++){
38                 //STEP 1: GENERATE A MAZE CELL WITH THE X, Y AND CELL ID
39                 int x = c * SIZE + OFFSET;
40                 int y = r * SIZE + OFFSET;
41                 MazeCell cell = new MazeCell(x, y, cellId);
42
43                 //STEP 2: PLACE THE CELL IN THE MAZE
44                 board[cellId] = cell;
```

```

45         cellId++;
46     }
47 }
48
49 //TASK 3: SET THE PAINT FOR THE MAZE
50 paint = new Paint();
51 paint.setColor(Color.BLACK);
52 paint.setStrokeWidth(2.0f);
53
54 //TASK 4: USE A BACKTRACKER METHOD TO BREAK DOWN THE WALLS
55 backtrackMaze();
56 }
57
58
59
60 public void onDraw(Canvas canvas){
61     //TASK 1: FILL THE CANVAS WITH WHITE PAINT
62     canvas.drawRGB(255, 255, 255);
63
64     //TASK 2: DRAW THE LINES FOR EVERY CELL
65     for (int i = 0; i < N_CELLS; i++){
66         int x = board[i].x;
67         int y = board[i].y;
68
69         if (board[i].north)
70             canvas.drawLine(x, y, x+SIZE, y, paint);
71         if (board[i].south)
72             canvas.drawLine(x, y+SIZE, x+SIZE, y+SIZE, paint);
73         if (board[i].east)
74             canvas.drawLine(x+SIZE, y, x+SIZE, y+SIZE, paint);
75         if (board[i].west)
76             canvas.drawLine(x, y, x, y+SIZE, paint);
77     }
78
79 }
80
81 public void backtrackMaze() {
82     // TASK 1: CREATE THE BACKTRACKER VARIABLES AND INITIALIZE THEM
83     Stack<Integer> stack = new Stack<Integer>();
84     int top;
85
86     // TASK 2: VISIT THE FIRST CELL AND PUSH IT ONTO THE STACK
87     int visitedCells = 1; // COUNTS HOW MANY CELLS HAVE BEEN VISITED
88     int cellID = 0; // THE FIRST CELL IN THE MAZE
89     board[cellID].visited = true;
90     stack.push(cellID);
91
92     // TASK 3: BACKTRACK UNTIL ALL THE CELLS HAVE BEEN VISITED
93     while (visitedCells < N_CELLS) {
94         //STEP 1: WHICH WALLS CAN BE TAKEN DOWN FOR A GIVEN CELL?

```

```
95 String possibleWalls = "";
96 if (board[cellID].north == true && cellID >= COLS) {
97     if (!board[cellID - COLS].visited) {
98         possibleWalls += "N";
99     }
100 }
101 if (board[cellID].west == true && cellID % COLS != 0) {
102     if (!board[cellID - 1].visited) {
103         possibleWalls += "W";
104     }
105 }
106 if (board[cellID].east == true &&
107     cellID % COLS != COLS - 1) {
108     if (!board[cellID + 1].visited) {
109         possibleWalls += "E";
110     }
111 }
112 if (board[cellID].south == true &&
113     cellID < COLS * ROWS - COLS) {
114     if (!board[cellID + COLS].visited) {
115         possibleWalls += "S";
116     }
117 }
118
119 //STEP 2: RANDOMLY SELECT A RANDOM WALL
120 //      FROM AVAILABLE WALLS
121 if (possibleWalls.length() > 0) {
122     int index = Math.round((int)(Math.random()
123         *possibleWalls.length()));
124     char randomWall = possibleWalls.charAt(index);
125
126     switch (randomWall) {
127         case 'N':
128             board[cellID].north = false;
129             board[cellID - COLS].south = false;
130             cellID -= COLS;
131             break;
132         case 'S':
133             board[cellID].south = false;
134             board[cellID + COLS].north = false;
135             cellID += COLS;
136             break;
137         case 'E':
138             board[cellID].east = false;
139             board[cellID + 1].west = false;
140             cellID++;
141             break;
142         case 'W':
143             board[cellID].west = false;
144             board[cellID - 1].east = false;
```

```

145         cellID--;
146     }
147     board[cellID].visited = true;
148     stack.push(cellID);
149     visitedCells++;
150
151 }
152 //IF THERE ARE NO WALLS TO BUST DOWN,
153 //BACKTRACK BY GRABBING THE TOP OF THE STACK
154 else {
155     top = stack.pop();
156     if (top == cellID){
157         cellID = stack.pop();
158         stack.push(cellID);
159     }
160 }
161 }
162 }
163 }
164 }

```

## ■ 5.6 Animate Library

Beginning with KitKat, Android 4.4, the transitions framework allows the definition of scene animations. These are typically view hierarchies that describe how to animate or custom transform a scene based on specific properties, such as layout bounds, or visibility. There is also an abstraction for an animation that can be applied to Views, Surfaces, or other objects.

To generate simple tweened animations, Android provides a package called `android.view.animation`. The following are a collection of classes that support basic animations:

AlphaAnimation:	Animates the changing transparency of an object
RotateAnimation:	Rotates animation of an object
ScaleAnimation:	Animates the scaling of an object
TranslateAnimation:	Moves an object

Unlike frame-by-frame animations, these classes control the specific View property of an object on display on the screen. The four standard animation attributes used when generating tweens are as follows:

1. `android:startOffset`: The start time (in milliseconds) of a transformation, where 0 is the start time of the root animation set.
2. `android:duration`: The duration (in milliseconds) of a transformation.

3. `android:fillAfter`: Whether you want the transformation you apply to continue after the duration of the transformation has expired. If false, the original value will immediately be applied when the transformation is done. Suppose, for example, you want to make a dot move down and then right in an “L” shape. If this value is not true, at the end of the down motion, the text box will immediately jump back to the top before moving right.
4. `android:fillBefore`: True if you want this transformation to be applied at time zero, regardless of your start time value (you will probably never need this).

Consider the following segment of code. This code assumes that an `ImageView` named `imageView` has been inflated and placed within the layout for the running activity. The animation, `alpha`, controls the transparency of `imageView` so that it animates from full view to complete transparency. `setFillAfter()` is used to apply the transformation after the animation ends. If `setfillAfter()` is passed a true argument, the final transformation will persist once the animation is completed.

```
1 AlphaAnimation alpha = new AlphaAnimation(1.0f, 0.0f);
2 alpha.setDuration(3000);
3 alpha.setFillAfter(true);
4 imageView.startAnimation(alpha);
```

In the case of rotation, movement will take place in the *X-Y* plane. The center point of rotation can be specified as an *x, y* coordinate, where (0, 0) is the top left point. When not specified, the default point of rotation is set as (0, 0). The following segment of code rotates the `imageView` from a starting point of 0 degrees and an ending point of 90 degrees. Line 11 sets the acceleration curve for the animation, which defaults to a linear interpolation.

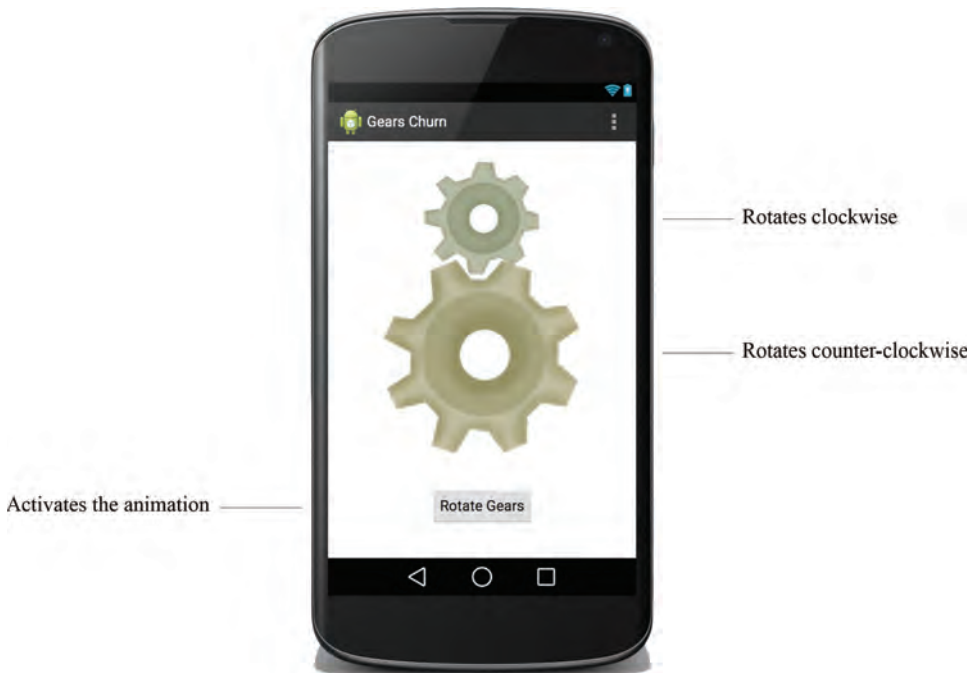
```
1 RotateAnimation mRotate = new RotateAnimation(
2     0,
3     90,
4     Animation.RELATIVE_TO_SELF,
5     0.5f,
6     Animation.RELATIVE_TO_SELF,
7     0.5f);
8 mRotate.setDuration(200);
9 mRotate.setFillAfter(true);
10
11 mRotate.setInterpolator(new AccelerateInterpolator());
12
13 imageView.startAnimation(mRotate);
```

### ■ Lab Example 5-5: Gears Churning Basic Rotating Animation

The Android `android.view.animation` package provides classes that handle tween animations. This lab example explores the creation of a simple animation that involves a tweened element, such as rotation. A tween animation can perform a series of simple transformations in regard to position, size, rotation, and so on.

#### Part 1: The Design

The Gear Churn application contains graphics, specifically two gears, that are animated. When the user taps the button labeled “Rotate Gears,” the wheels of the gears slowly churn, as shown in Figure 5-28. Tween settings are applied so that the gears move in sync: The smallest gear rotates clockwise while the larger gear rotates counterclockwise. The tween performs the animation with a starting point and an ending point, so that the rotations are not performed continuously and eventually come to an end.



■ FIGURE 5-28

## Part 2: Application Structure and Setup

---

The settings for the application are as follows:

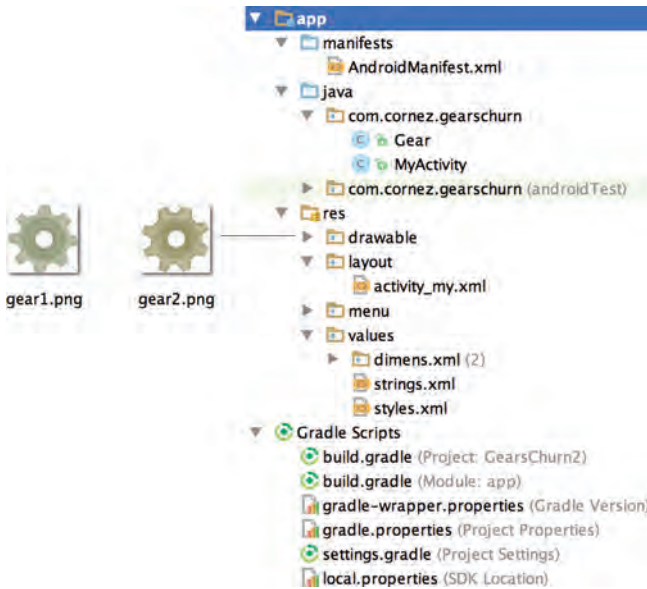
- Application Name: Gears Churn
- Project Name: GearsChurn
- Package Name: com.cornez.gearschurn
- Android Form: Phone and Tablet
- Minimum SDK: API 18: Android 4.3 (Jelly Bean)
- Target SDK: API 21: Android 5.0 (Lollipop)
- Compile with: API 21: Android 5.0 (Lollipop)
- Activity Name: MyActivity
- Layout Name: activity\_my

The `AndroidManifest.xml` file sets the launch icon to the Android default `ic_launcher.png` file. The main activity of the application is `MyActivity` and its associated user interface is `activity_my.xml`.

`AndroidManifest.xml`

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3     package="com.cornez.gearschurn" >
4
5     <application
6         android:allowBackup="true"
7         android:icon="@drawable/ic_launcher"
8         android:label="@string/app_name"
9         android:theme="@style/AppTheme" >
10         <activity
11             android:name=".MyActivity"
12             android:screenOrientation="portrait"
13             android:label="@string/app_name" >
14             <intent-filter>
15                 <action android:name="android.intent.action.MAIN" />
16
17                 <category
18                     android:name="android.intent.category.LAUNCHER" />
19             </intent-filter>
20         </activity>
21     </application>
22
23 </manifest>
24
```

Two bitmap images, `gear1.png` and `gear2.png`, are added to the `Drawables` directory. The final project structure for this application appears in Figure 5-29. Gear class will be used as the data model for each gear.



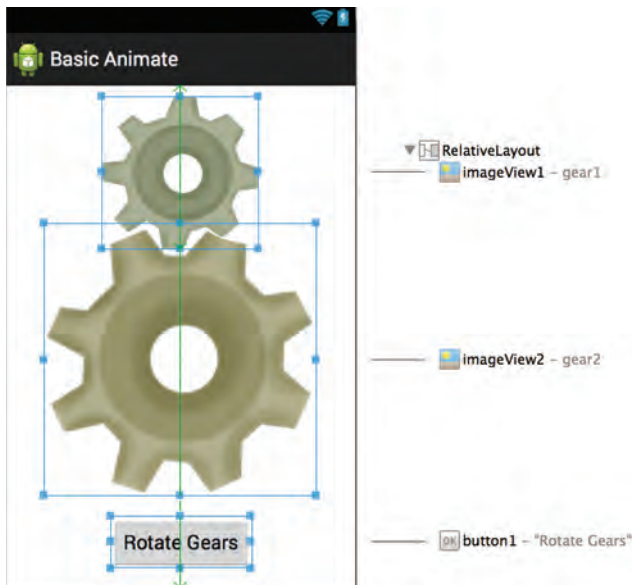
**FIGURE 5-29** The Project Structure of the Gears Churn application.

### Part 3: The User Interface

The user interface relies on `strings.xml` and the two drawable images: `gear1` and `gear2`. Figure 5-30 shows the hierarchical View structure of the graphic elements and the button, which constitute the application's layout design. The added string values are used for image descriptions and the button label that appears on the screen. The XML code for `strings.xml` appears as follows:

```
strings.xml
1  <?xml version="1.0" encoding="utf-8"?>
2  <resources>
3
4      <string name="app_name">Gears Churn</string>
5      <string name="hello_world">Hello world!</string>
6      <string name="action_settings">Settings</string>
7
8      <string name="gear_large">Large gear</string>
9      <string name="gear_small">Small gear</string>
10     <string name="rotate_btn">Rotate Gears</string>
11
12 </resources>
```





**FIGURE 5-30** The layout design for `activity_my.xml`.

The layout design for `activity_my.xml` requires strict adherence to the placement of the gears. The teeth of the gears must fit together to provide an animation that appears to churn at the correct angles. The first gear is placed 10dps from the top of the screen. The larger gear has a width and height of 250dps and is positioned at -25dps below the smaller gear. This overlap allows the teeth of the smaller gear to fit correctly into the gap provided by the larger gear.

The button View is assigned an event handler named `animateGears()`. The button has been given a unique identifier name; however, this is a convention of `RelativeLayouts`, rather than a requirement. Nevertheless, the `ImageViews` that hold the gear bitmaps both require identifier names. This will allow `MyActivity.java` to be accessed and for the gears to be customized with an animation. The names that have been generically assigned to the gears are `imageView1` and `imageView2`.

The XML code for `activity_my.xml` appears as follows:

```
activity_my.xml
1 <RelativeLayout
2   xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:tools="http://schemas.android.com/tools"
4     android:layout_width="match_parent"
5     android:layout_height="match_parent"
6     android:paddingLeft="@dimen/activity_horizontal_margin"
7     android:paddingRight="@dimen/activity_horizontal_margin"
8     android:paddingTop="@dimen/activity_vertical_margin"
9     android:paddingBottom="@dimen/activity_vertical_margin"
```

```
10     tools:context=".MyActivity">
11
12
13     <ImageView
14         android:layout_width="wrap_content"
15         android:layout_height="wrap_content"
16         android:id="@+id/imageView1"
17         android:layout_alignParentTop="true"
18         android:layout_centerHorizontal="true"
19         android:layout_marginTop="10dp"
20         android:src="@drawable/gear1"
21         android:contentDescription="@string/gear_small" />
22
23     <ImageView
24         android:layout_width="250dp"
25         android:layout_height="250dp"
26         android:id="@+id/imageView2"
27         android:layout_below="@+id/imageView1"
28         android:layout_centerHorizontal="true"
29         android:layout_marginTop="-25dp"
30         android:src="@drawable/gear2"
31         android:contentDescription="@string/gear_large" />
32
33     <Button
34         android:layout_width="wrap_content"
35         android:layout_height="wrap_content"
36         android:text="@string/rotate_btn"
37         android:id="@+id/button"
38         android:layout_below="@+id/imageView2"
39         android:layout_centerHorizontal="true"
40         android:layout_marginTop="38dp"
41         android:onClick="animateGears"/>
42
43
44 </RelativeLayout>
```

## Part 4: Source Code for Application

As the class that represents a gear's data, the Gear class has two data members: `mStartDegree` and `mEndDegree`. These data members are used to customize the tween animation. For example, an idle gear has an initial starting degree before it begins to rotate. Once the animation begins, the gear rotates until it reaches an end degree, at which point it stops and becomes idle again. The Java code listing for `Gear.java` appears as follows:

Gear.java

```
1 package com.cornez.gearschurn;
2
3 public class Gear {
4
5     private int mStartDegree;
6     private int mEndDegree;
7
8     public Gear(){
9         mStartDegree = 0;
10        mEndDegree = 0;
11    }
12
13    public void setStartDegree(int startDegree){
14        mStartDegree = startDegree;
15    }
16    public int getStartDegree(){
17        return mStartDegree;
18    }
19    public void setEndDegree(int endDegree){
20        mEndDegree = endDegree;
21    }
22    public int getEndDegree(){
23        return mEndDegree;
24    }
25 }
```

MyActivity.java is the controller of the application, and it is the sole activity. The Java code listing for this source file appears as follows:

- Lines 8–9: The imported Animation class provides the animation behavior that can be applied to the gears. The RotationAnimation class is an extension of Animation. Both classes are required for the rotation tween used in this application.
- Lines 15–16: The two ImageView objects holding the gear bitmaps need to be referenced in order to be animated. The small gear is referenced by gear1Img and the large gear is referenced by gear2Img. The references are assigned on Lines 30 and 31.
- Lines 18–19: Two Gear objects are declared. gear1 and gear2 hold the tween information for the ImageViews named gear1Img and gear2Img, respectively. gear1 and gear2 are instantiated on Lines 38 and 43.

Lines 36–46: The idle positions for the gears are set. The starting angle is zero for both gears. Because gear1 will move in a clockwise rotation, its ending angle will be 360 degrees. The second gear, gear2, will move counterclockwise, and therefore, its ending angle will be  $-360$  degrees.

MyActivity.java

```
1 package com.cornez.gearschurn;
2
3 import android.app.Activity;
4 import android.os.Bundle;
5 import android.view.Menu;
6 import android.view.MenuItem;
7 import android.view.View;
8 import android.view.animation.Animation;
9 import android.view.animation.RotateAnimation;
10 import android.widget.ImageView;
11
12
13 public class MyActivity extends Activity {
14
15     private ImageView gear1Img;
16     private ImageView gear2Img;
17
18     private Gear gear1;
19     private Gear gear2;
20
21     private float currentDegree;
22     private float degree;
23
24
25     @Override
26     protected void onCreate(Bundle savedInstanceState) {
27         super.onCreate(savedInstanceState);
28         setContentView(R.layout.activity_my);
29
30         gear1Img = (ImageView) findViewById(R.id.imageView1);
31         gear2Img = (ImageView) findViewById(R.id.imageView2);
32
33         initializeGears();
34     }
35
36     private void initializeGears(){
37         //GEAR 1 WILL MOVE IN A CLOCKWISE DIRECTION
38         gear1 = new Gear();
39         gear1.setStartDegree(0);
40         gear1.setEndDegree(360);
41     }
```

```

42 //GEAR 1 WILL MOVE IN A COUNTER-CLOCKWISE DIRECTION
43 gear2 = new Gear();
44 gear2.setStartDegree(0);
45 gear2.setEndDegree(-360);
46 }

```

Lines 50–56: `RotateAnimation` is a tween animation that controls the rotation of an object. This rotation takes place in the  $x, y$  plane. The parameters for this animation are:

`fromDegrees`: Rotation offset to apply at the start of the animation.

`toDegrees`: Rotation offset to apply at the end of the animation.

`pivotXType`: Specifies how `pivotXValue` should be interpreted. There are three possible options: (1) `Animation.ABSOLUTE`, (2) `Animation.RELATIVE_TO_SELF`, or (3) `Animation.RELATIVE_TO_PARENT`.

`pivotXValue`: The  $x$  coordinate of the point about which the object is being rotated, specified as an absolute number where 0 is the left edge.

`pivotYType`: Specifies how `pivotYValue` should be interpreted.

`pivotYValue`: The  $y$  coordinate of the point about which the object is being rotated, specified as an absolute number where 0 is the top edge.

Line 57: `setDuration()` sets how long the animation should last. It should be noted that duration cannot be negative.

Line 59: `setFillAfter()` set to true. This means the transformation that this animation performs will persist when it is finished.

MyActivity.java (continued)

```

47 public void animateGears(View view){
48     final int DELAY = 1000;
49
50     RotateAnimation ral = new RotateAnimation(
51         gear1.getStartDegree(),
52         gear1.getEndDegree(),
53         Animation.RELATIVE_TO_SELF,
54         0.5f,

```

```
55             Animation.RELATIVE_TO_SELF,
56             0.5f);
57         ra1.setDuration(DELAY);
58         ra1.setFillAfter(true);
59         gear1Img.startAnimation(ra1);
60
61         RotateAnimation ra2 = new RotateAnimation(
62             gear2.getStartDegree(),
63             gear2.getEndDegree(),
64             Animation.RELATIVE_TO_SELF,
65             0.5f,
66             Animation.RELATIVE_TO_SELF,
67             0.5f);
68         ra2.setDuration(DELAY);
69         ra2.setFillAfter(true);
70         gear2Img.startAnimation(ra2);
71     }
72
73     @Override
74     public boolean onCreateOptionsMenu(Menu menu) {
75         // Inflate the menu.
76         getMenuInflater().inflate(R.menu.my, menu);
77         return true;
78     }
79
80     @Override
81     public boolean onOptionsItemSelected(MenuItem item) {
82         // Handle action bar item clicks here. The action bar will
83         // automatically handle clicks on the Home/Up button, so long
84         // as you specify a parent activity in AndroidManifest.xml.
85         int id = item.getItemId();
86         if (id == R.id.action_settings) {
87             return true;
88         }
89         return super.onOptionsItemSelected(item);
90     }
91 }
```

---

## 5.7 Audio

A basic understanding of audio file formats and conversions is required when developing applications that use audio media. An audio file is characterized by its file format (audio container) and its data format (audio encoding). The data format of an audio file refers to the content and how it has been encoded. For example, WAV is a file format that can contain audio that is encoded in PCM. PCM describes the technique used to

**TABLE 5-1** Android supports these common data formats.

AAC	<p>AAC stands for “Advanced Audio Coding,” and it was designed to be the successor of MP3. As you would guess, it compresses the original sound, resulting in disk savings but lower quality. The loss of quality is not always noticeable, however, depending on how low you set the bit rate. In practice, AAC usually does better compression than MP3, especially at bit rates below 128kbit/s).</p> <p>The supported file formats for AAC are:</p> <ul style="list-style-type: none"> <li>3GPP (.3gp)</li> <li>MPEG-4 (.mp4, .m4a)</li> <li>ADTS</li> <li>MPEG-TS</li> </ul>
HE-AAC	<p>HE-AAC is a superset of AAC, where the HE stands for “high efficiency.” HE-AAC is optimized for low-bit-rate audio, such as streaming audio.</p> <p>The supported file formats for HE-AAC are the same as AAC.</p>
AMR	<p>AMR stands for “Adaptive Multi-Rate” and is another encoding optimized for speech, featuring very low bit rates.</p> <p>The supported file format for AMR is 3GPP (.3gp).</p>
MP3	<p>The format we all know and love: MP3. MP3 is still a very popular format after all of these years, and it is supported by the iPhone.</p> <p>MP3 supports its own file format.</p>
PCM	<p>This stands for linear “Pulse Code Modulation,” and it describes the technique used to convert analog sound data into a digital format, or in simple terms, into uncompressed data. Because the data are uncompressed, PCM is the fastest to play and is the preferred encoding for audio on an Android device when space is not an issue.</p> <p>The supported file format for PCM is WAV.</p>
FLAC	<p>FLAC stands for “Free Lossless Audio Codec.” This audio format is similar to MP3, but it is lossless, meaning that audio is compressed in FLAC without any loss in quality. FLAC is nonproprietary and has an open-source reference implementation.</p> <p>FLAC supports its own file format.</p>

convert analog sound data into a digital format. Table 5-1 lists common data formats, or audio encoding, supported by Android.

You can play audio in an Android application in several ways. Android uses two APIs for this purpose: `SoundPool` and `MediaPlayer`.

### 5.7.1 `SoundPool`

`SoundPool` provides an easy way to play short audio files, which is particularly useful for audio alerts and simple game sounds (such as making a “click” when moving a game piece).

It can repeat sounds and play several sounds simultaneously. Typically, sound files played with `SoundPool` should not exceed 1 MB.

Examine the following code segment:

- Lines 1–4: A `SoundPool` object is instantiated using three arguments: (1) `maxStreams`, the maximum number of simultaneous streams allowed for this `SoundPool` object; (2) the `streamType`, specified as `AudioManager.STREAM_MUSIC`. This is the audio stream type as described in `AudioManager`. Android supports different audio streams for different purposes. Game applications normally use `STREAM_MUSIC`; and (3) `srcQuality`, which specifies the sample-rate converter quality. Currently this has no effect. Zero is used for the default.
- Line 6: The sound `R.raw.explosion` is loaded. The directory for sound files is `res/raw`. Note that the extension is dropped. For example, if you want to load a sound from the raw resource file “`explosion.mp3`,” you would specify `R.raw.explosion` as the resource ID. You cannot have both an `explosion.wav` and an `explosion.mp3` in the `res/raw` directory. `load()` returns a nonzero `streamID` if successful, zero if it fails. The `streamID` can be used to further control playback.
- Lines 8–12: Attributes for the explosion sound are set. The values for left volume and right volume can range from 0.0 to 1.0, with 1.0 representing the maximum volume. Priority refers to the stream priority. Zero is the lowest priority. The loop attribute refers to how many times the sound will loop. For example, zero means no loop; it will play once and end. A value of `-1` forces a forever loop. The last attribute is `rate`, the playback rate (pitch). By specifying a 1.0, the playback will occur at a normal rate. This value for `rate` can range from .5 to 2.0. A value of 2.0 means playback twice as fast, and a value of 0.5 means playback at half speed.
- Lines 14–19: The explosion sound, specified by its `soundID`, is played. Calling `play()` may cause another sound to stop playing if the maximum number of active streams is exceeded. Otherwise, the sound will be layered over the sound currently playing.

```
1  SoundPool soundPool = new SoundPool(  
2      maxStreams,  
3      AudioManager.STREAM_MUSIC,  
4      0);  
5  
6  int soundId = soundPool.load(this, R.raw.explosion, 1);  
7
```



```
8    float leftVolume = 1;
9    float rightVolume = 1
10   int priority = 1;
11   int loop = 0;
12   float rate = 1.0f;
13
14   soundPool.play(soundId,
15                 leftVolume,
16                 rightVolume,
17                 priority,
18                 loop,
19                 rate);
```

SoundPool has many advantages. You can play several sounds at once (using a different SoundPool for each sound), and you can play sounds even when your app is in the background. However, SoundPool can be extremely slow for large raw files. If a large sound has not fully loaded, there may be a noticeable delay when it is triggered.

### 5.7.2 MediaPlayer

MediaPlayer provides the resources for handling media playback. For example, an application can use MediaPlayer to create an interface between the user and a music file. The interface may include playback controls for interacting with playback components and for sending notification as the playback elapses.

For applications that require the retrieval of audio files located on the device, the ContentResolver class is used to access these files, the MediaPlayer class is used to play audio, and the MediaController class is used to control playback. The MediaPlayer class can be used to control playback of both audio and video files and streams.

Android supports a variety of common media types, allowing the integration of audio, video, and images into an application. Audio files used with MediaPlayer are often stand-alone files in the filesystem, or they arrive from a data stream over a network connection. As in SoundPool, audio files can also be stored in the application's resources (raw resources).

The code segment below illustrates how a raw audio resource of an explosion is played using MediaPlayer. mediaController is a view containing controls for a MediaPlayer. Typically this view contains buttons such as “Play/Pause,” “Rewind,” “Fast Forward,” and a progress slider.

MediaController takes care of synchronizing the controls with the state of the MediaPlayer. The way to use this class is to instantiate it programmatically. The MediaController creates a default set of controls and puts them in a floating window. For the control window to appear, a setAnchorView() must be specified.

```
1 MediaPlayer mediaPlayer = new MediaPlayer();
2 MediaController mediaController = new MediaController(this);
3
4 mediaController.setMediaPlayer(this);
5 mediaController.setAnchorView(findViewById(R.id.playback_view));
6
7 try {
8     mediaPlayer.setDataSource(R.raw.explosion);
9     mediaPlayer.prepare();
10    mediaPlayer.start();
11 } catch (IOException e) {
12     Log.e(TAG, "Could not open the audio file for playback.", e);
13 }
```

The view that acts as the anchor for the playback control window will disappear if it has been left idle for three seconds. By using an `onTouchEvent()` shown in the following code segment, the playback controls will reappear when the user touches the anchor view.

```
1
2 @Override
3 public boolean onTouchEvent(MotionEvent event) {
4     mediaController.show();
5     return false;
6 }
```

## ■ Lab Example 5-6: Simple Jukebox Sound Effects

This lab explores a simple implementation of sound effects, played with the `SoundPool` and `MediaPlayer`. As Figure 5-31 illustrates, `SoundPool` is used to play short sound bursts, such as a brief bell clang or a quick clang on a gong. `MediaPlayer` is used for longer sounds, such as a drum solo.

### Part 1: The Design

The Sound Jukebox application contains a collection of sound effects. The users of this application can create sound punctuations to enhance speech. For example, a scary description of an event might be followed by a “spooky cry” sound effect.

Sounds will be preloaded or configured when the application launches for the first time. Sounds using `SoundPool` can result in very slow loads. Often, a preloaded `SoundPool` audio will continue to load for a delayed period.



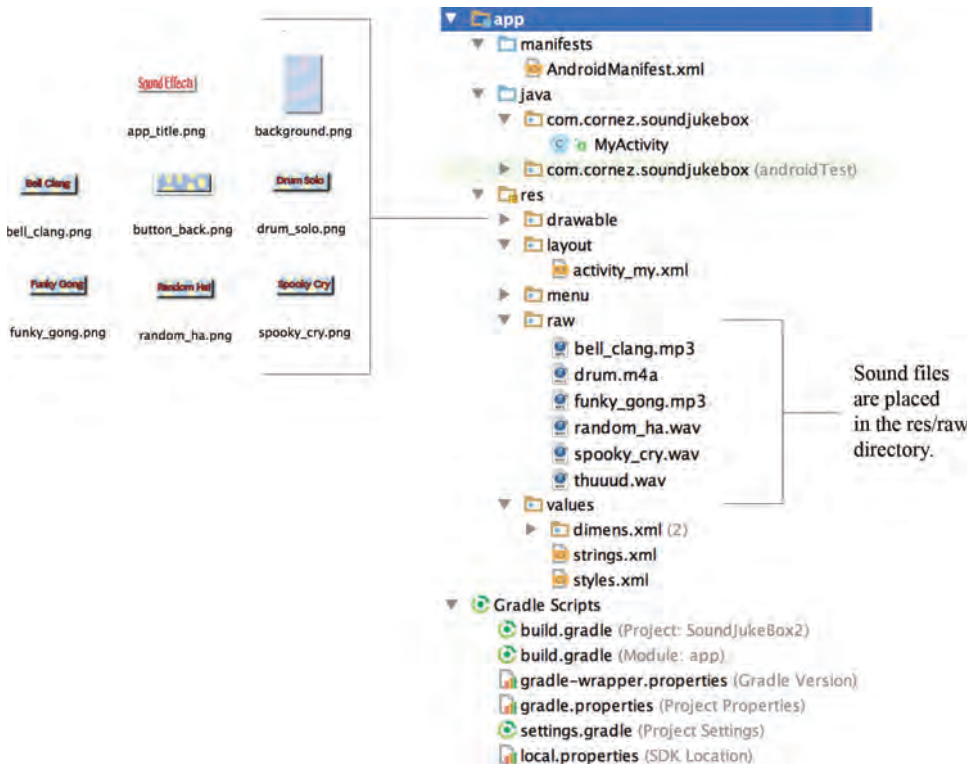
**FIGURE 5-31** The Sound Jukebox application.

## Part 2: Application Structure and Setup

The settings for the application are as follows:

- Application Name: Sound Jukebox
- Project Name: SoundJukebox
- Package Name: `com.cornez.soundjukebox`
- Android Form: Phone and Tablet
- Minimum SDK: API 18: Android 4.3 (Jelly Bean)
- Target SDK: API 21: Android 5.0 (Lollipop)
- Compile with: API 21: Android 5.0 (Lollipop)
- Activity Name: `MyActivity`
- Layout Name: `activity_my`

The final project structure for the Sound Jukebox application is shown in Figure 5-32. The audio files, WAVs, and MP3 are not Drawables and should not be placed in a drawable folder. A separate directory, `res/raw`, is used to store all audio files. This directory must be created. During the build process, the `R.java` class automatically stores the generated identifiers for these files. The launch icon for the application is the default Android icon.



**FIGURE 5-32** Project structure for the Sound Jukebox application.

The orientation of the screen is locked into portrait mode, and a fullscreen is used. The code listing for `AndroidManifest.xml` is shown as follows. The main activity of the application is `MainActivity.java`.

```
AndroidManifest.xml
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3     package="com.cornez.soundjukebox" >
4
5     <application
6         android:allowBackup="true"
7         android:icon="@drawable/ic_launcher"
```

```

8      android:label="@string/app_name"
9      android:theme="@android:style/
10          Theme.Holo.NoActionBar.Fullscreen" >
11      <activity
12          android:name=".MyActivity"
13          android:screenOrientation="portrait"
14          android:label="@string/app_name" >
15          <intent-filter>
16              <action android:name="android.intent.action.MAIN" />
17
18              <category
19                  android:name="android.intent.category.LAUNCHER" />
20          </intent-filter>
21      </activity>
22  </application>
23
24 </manifest>

```

### Part 3: Value Resources and the User Interface

The user interface consists of buttons that allow the user to activate a sound effect. For visual interest, `ImageButton` widgets are used for this purpose. `ImageButton` widgets are treated as `ImageViews` in the sense that they should be assigned a content description. `strings.xml`, listed below, stores the content descriptions.



**FIGURE 5-33** The user interface for the Sound Jukebox Application.

strings.xml

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3
4     <string name="app_name">Sound Jukebox</string>
5     <string name="hello_world">Hello world!</string>
6     <string name="action_settings">Settings</string>
7
8     <!-- SOUND EFFECTS -->
9     <string name="bell_clang">Bell Clang</string>
10    <string name="funky_gong">Funky Gong</string>
11    <string name="random_ha">Random Ha</string>
12    <string name="spooky_cry">Spooky Cry</string>
13    <string name="drum_solo">Drum Solo</string>
14
15 </resources>
```

The layout associated with the application activity is `activity_my.xml`, shown in Figure 5-33. The code listing for this layout is displayed as follows:

activity\_my.xml

```
1 <RelativeLayout
2   xmlns:android="http://schemas.android.com/apk/res/android"
3   xmlns:tools="http://schemas.android.com/tools"
4   android:layout_width="match_parent"
5   android:layout_height="match_parent"
6   android:id="@+id/relativeLayout"
7   tools:context=".MyActivity"
8   android:background="@drawable/background">
9
10
11    <ImageView
12        android:layout_width="wrap_content"
13        android:layout_height="wrap_content"
14        android:id="@+id/imageView"
15        android:layout_alignParentTop="true"
16        android:layout_marginTop="20dp"
17        android:layout_centerHorizontal="true"
18        android:src="@drawable/app_title"
19        android:contentDescription="@string/app_name" />
20
21    <ImageButton
22        android:layout_width="wrap_content"
23        android:layout_height="wrap_content"
24        android:id="@+id/imageButton1"
25        android:layout_below="@+id/imageView"
26        android:layout_centerHorizontal="true"
27        android:layout_marginTop="20dp"
28        android:src="@drawable/bell_clang"
```

```

29         android:contentDescription="@string/bell_clang" />
30
31     <ImageButton
32         android:layout_width="wrap_content"
33         android:layout_height="wrap_content"
34         android:id="@+id/imageButton2"
35         android:layout_below="@+id/imageButton1"
36         android:layout_centerHorizontal="true"
37         android:layout_marginTop="5dp"
38         android:src="@drawable/funky_gong"
39         android:contentDescription="@string/funky_gong" />
40
41     <ImageButton
42         android:layout_width="wrap_content"
43         android:layout_height="wrap_content"
44         android:id="@+id/imageButton3"
45         android:layout_below="@+id/imageButton2"
46         android:layout_centerHorizontal="true"
47         android:layout_marginTop="5dp"
48         android:src="@drawable/random_ha"
49         android:contentDescription="@string/random_ha" />
50
51     <ImageButton
52         android:layout_width="wrap_content"
53         android:layout_height="wrap_content"
54         android:id="@+id/imageButton4"
55         android:layout_below="@+id/imageButton3"
56         android:layout_centerHorizontal="true"
57         android:layout_marginTop="5dp"
58         android:src="@drawable/spooky_cry"
59         android:contentDescription="@string/spooky_cry" />
60
61     <ImageButton
62         android:layout_width="wrap_content"
63         android:layout_height="wrap_content"
64         android:id="@+id/imageButton5"
65         android:layout_below="@+id/imageButton4"
66         android:layout_centerHorizontal="true"
67         android:layout_marginTop="5dp"
68         android:src="@drawable/drum_solo"
69         android:contentDescription="@string/drum_solo" />
70 </RelativeLayout>

```

## Part 4: Source Code for Application

To simplify the implementation of audio playback, this application uses no data models. An adapter for a scrolling list of data would be implemented to construct Sound Jukebox into a sophisticated application. The code listing for the application's activity, `MyActivity.java`, is listed as follows. A unique object references each

ImageButton element. An `onClick` listener event is applied to each button, which triggers the handler `playSoundEffect()`.

Unlike a normal array of integers, there can be gaps in the indices. Therefore, we use `SparseIntArray` to map integers to integers. `SparseIntArray` is memory efficient because it avoids auto-boxing keys and values, and its data structure does not rely on an extra entry object for each mapping, as compared to `HashMap`.

The `SparseIntArray` container keeps its mappings in an array data structure, using a binary search to find keys. The implementation is not intended to be appropriate for data structures that may contain large numbers of items. It is generally slower than a traditional `HashMap`, since lookups require a binary search, and adds and removes require inserting and deleting entries in the array. For containers holding up to hundreds of items, the performance difference—less than 50%—is not significant. It is possible to iterate over the items in this container using `keyAt()` and `valueAt()`. Iterating over the keys using `keyAt()` with ascending values of the index returns the keys in ascending order, or the values corresponding to the keys in ascending order in the case of `valueAt()`. On Lines 46–49, `put(int key, int value)` is used to add a mapping from the specified key to the specified sound file.

MyActivity.java

```
1 package com.cornez.soundjukebox;
2
3 import android.app.Activity;
4 import android.media.AudioManager;
5 import android.media.MediaPlayer;
6 import android.media.SoundPool;
7 import android.os.Bundle;
8 import android.util.SparseIntArray;
9 import android.view.Menu;
10 import android.view.MenuItem;
11 import android.view.View;
12 import android.widget.ImageButton;
13 import android.widget.MediaController;
14 import android.widget.RelativeLayout;
15 import android.widget.Toast;
16
17 public class MyActivity extends Activity {
18
19     private ImageButton bellClangBtn;
20     private ImageButton funkyGongBtn;
21     private ImageButton spookyCryBtn;
22     private ImageButton randomHaBtn;
23     private ImageButton drumSoloBtn;
24
25     private SoundPool soundPool;
26     private SparseIntArray soundMap;
27 }
```



```
28 private MediaPlayer mMediaPlayer;
29 private MediaController mMediaController;
30
31 @Override
32 protected void onCreate(Bundle savedInstanceState) {
33     super.onCreate(savedInstanceState);
34     setContentView(R.layout.activity_my);
35
36     configureSounds();
37     initializeJukeBoxBtns();
38 }
39
40 private void configureSounds() {
41
42     // CONFIGURE THE SOUNDS USE IN THE JUKEBOX
43     // PRE-LOAD THE FIRST FOUR SOUNDS
44     soundPool = new SoundPool(1, AudioManager.STREAM_MUSIC, 0);
45     soundMap = new SparseIntArray(4);
46     soundMap.put(1, soundPool.load(this, R.raw.bell_clang, 1));
47     soundMap.put(2, soundPool.load(this, R.raw.funky_gong, 1));
48     soundMap.put(3, soundPool.load(this, R.raw.spooky_cry, 1));
49     soundMap.put(4, soundPool.load(this, R.raw.random_ha, 1));
50
51     // FIFTH SOUND WILL BE PLAYED IN MEDIA PLAYER
52     mMediaPlayer = MediaPlayer.create(this, R.raw.drum);
53     mMediaController = new MediaController(this);
54     mMediaController.setEnabled(true);
55 }
56
57 private void initializeJukeBoxBtns() {
58     // SET REFERENCES TO THE SOUND EFFECT BUTTONS ON THE LAYOUT
59
60     bellClangBtn = (ImageButton) findViewById(R.id.imageButton1);
61     funkyGongBtn = (ImageButton) findViewById(R.id.imageButton2);
62     spookyCryBtn = (ImageButton) findViewById(R.id.imageButton3);
63     randomHaBtn = (ImageButton) findViewById(R.id.imageButton4);
64     drumSoloBtn = (ImageButton) findViewById(R.id.imageButton5);
65
66     // REGISTER LISTENER EVENTS FOR THE BUTTONS ON THE LAYOUT
67     bellClangBtn.setOnClickListener(playSoundEffect);
68     funkyGongBtn.setOnClickListener(playSoundEffect);
69     spookyCryBtn.setOnClickListener(playSoundEffect);
70     randomHaBtn.setOnClickListener(playSoundEffect);
71     drumSoloBtn.setOnClickListener(playSoundEffect);
72 }
73
74 private View.OnClickListener playSoundEffect = new
75     View.OnClickListener() {
76     public void onClick(View btn) {
77
```

```
78         // IDENTIFY THE SOUND TO BE PLAYED
79         String soundName = (String) btn.getContentDescription();
80
81         // PLAY THE SOUND
82         if (soundName.contentEquals("Bell Clang")) {
83             soundPool.play(1, 1, 1, 1, 0, 1.0f);
84         }
85         else if (soundName.contentEquals("Funky Gong")) {
86             soundPool.play(2, 1, 1, 1, 0, 1.0f);
87         }
88         else if (soundName.contentEquals("Random Ha"))
89             soundPool.play(3, 1, 1, 1, 0, 1.0f);
90         else if (soundName.contentEquals("Spooky Cry"))
91             soundPool.play(4, 1, 1, 1, 0, 1.0f);
92         else if (soundName.contentEquals("Drum Solo")) {
93             mMediaController.show();
94             mMediaPlayer.start();
95         }
96     }
97 };
98
99 @Override
100 public boolean onCreateOptionsMenu(Menu menu) {
101     // Inflate the menu.
102     getMenuInflater().inflate(R.menu.my, menu);
103     return true;
104 }
105
106 @Override
107 public boolean onOptionsItemSelected(MenuItem item) {
108     // Handle action bar item clicks here. The action bar will
109     // automatically handle clicks on the Home/Up button, so long
110     // as you specify a parent activity in AndroidManifest.xml.
111     int id = item.getItemId();
112     if (id == R.id.action_settings) {
113         return true;
114     }
115     return super.onOptionsItemSelected(item);
116 }
117 }
```

---

## Exercises

---

- 5.1 Describe the steps for creating an XML graphic element to be stored as a drawable resource file.
- 5.2 Explain when `onInflate()` and `onActivityCreated()` are called.
- 5.3 Describe the coordinate system used by a `RelativeLayout`. How does this compare with the coordinate system used in a `FrameView`?
- 5.4 List the properties that must be set for an `ImageView` that will result in the rotation around a pivot point.
- 5.5 Write a segment of code to remove an `ImageView` object from the screen during runtime.
- 5.6 What is the relationship between a `Bitmap` and a `Canvas`?
- 5.7 Briefly explain the purpose of `requestWindowFeature()`?
- 5.8 Create a frame-by-frame animation. Describe the steps that are required.
- 5.9 What parameters are required for `RotateAnimation()`?
- 5.10 Briefly describe the purpose of `SoundPool` and `MediaPlayer`.

