2D Graphics and Multimedia in Android

CONTENTS

2-D Graphics and UI Design are two important aspects in User Interface (UI) design. In this chapter, we will introduce 2-D graphics and some advanced UI design techniques. Main techniques of 2-D graphics include Color, Paint, Path, Canvas, Drawable, and Button Selector. Students will also learn how to create multiple screens, action bars, and custom views on the UI. Moreover, multimedia on Android systems is a functionality increasing your mobile apps'adoptability. In this chapter, we will introduce multimedia in Android and how to add multimedia to our Android app. Three main aspects in multimedia include *Media*, *Audio*, and *Video*.

4.1 INTRODUCTION OF 2 D GRAPHICS TECHNIQUES

Android implements complete 2-D functions in one package, named android.graphics. This package provides various kinds of graphics tools, such as canvas, color filter, point, line, and rectangles. We can use these graphics tools to draw the screen directly. We will introduce some basic knowledge in detail. First of all, we create a new Android application project named ColorTester.

4.1.1 Color

Colors are represented as packed integers, made up of 4 bytes: Alpha, Red, Green, and Blue. Alpha is a measure of transparency, from value 0 to value 255. The value 0 indicates the color is completely transparent. The value 255 indicates the color is completely opaque. Besides alpha, each component ranges between 0 and 255, with 0 meaning no contribution for that component, and 255 meaning 100% contribution.

We can create a half-opaque purple color like: int color1 = Color.argb(127, 255, 0, 255);

Or in XML resource file, like:

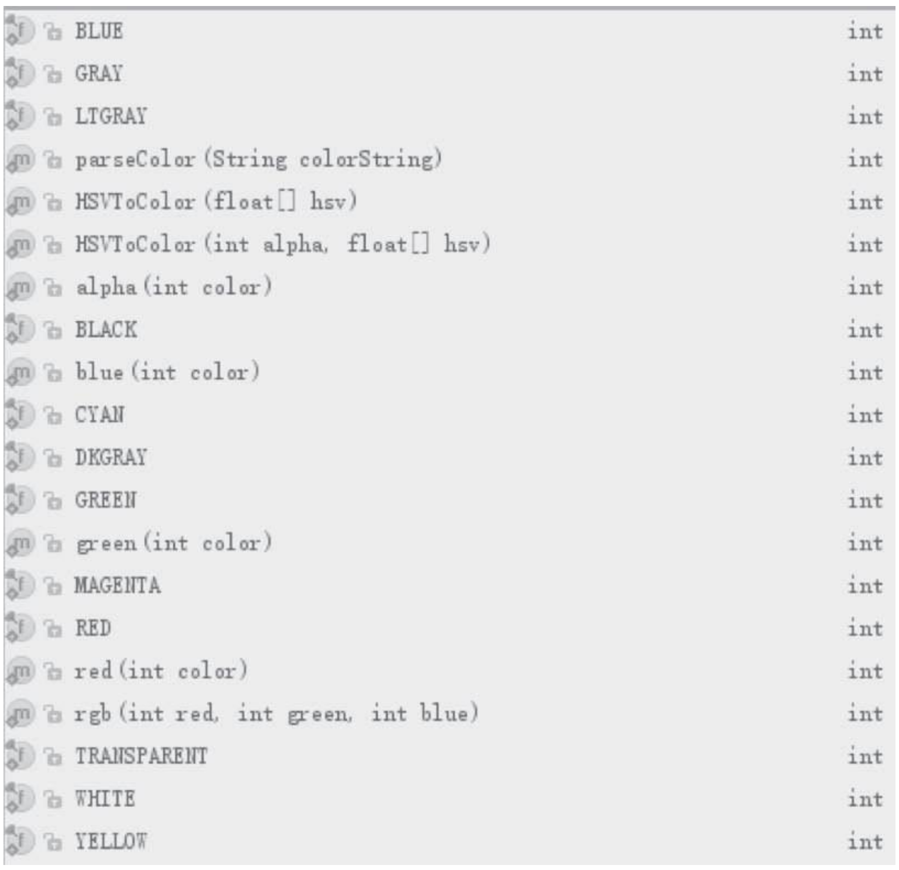
<color name="half_op_purple">#7fff00ff</color>

The colors in Android XML resource files must be formulated as "#" + 6 or 8 bit Hexadecimal number.

Furthermore, Android offers some basic colors as constants, as shown in Fig. 4.1. We can use them directly, like:

int color2 = Color.Black;

In Android Studio, we can preview the color we created in XML file, as shown in Fig. 4.2. There are some small squares with the color created in the same line. We can see that the *#ffffffff* is opaque-white, and the *#ff000000* is opaque-black.



BLUE	int
GRAY	int
LTGRAY	int
parseColor (String colorString)	int
HSVToColor (float[] hsv)	int
HSVToColor (int alpha, float[] hsv)	int
alpha (int color)	int
BLACK	int
blue (int color)	int
CYAN	int
DKGRAY	int
GREEN	int
green (int color)	int
MAGENTA	int
RED	int
red (int color)	int
rgb (int red, int green, int blue)	int
TRANSPARENT	int
WHITE	int
YELLOW	int

Figure 4.1 Colors as constants provided by Android.

We can use these color, created in colors.xml by “color/color_name”.

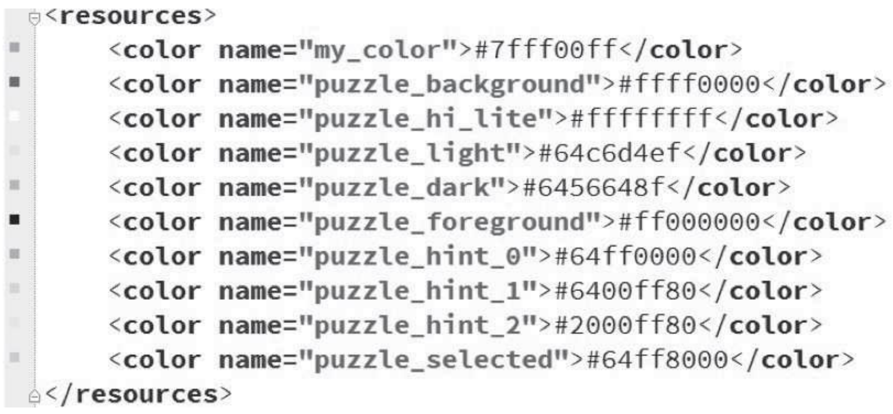
For example, android:background=“color/my_color”.

After we define some colors in the XML file, we can reference them by their names, as we did for strings, or we can use them in Java code like:

`int color3 = getResource().getColor(R.color.my_color);` or

`int color3 = R.color.text_color`

The *getResources()* method returns the *ResourceManager* class for the current activity, and get *getColor()* asks the manager look up a color given a resource ID.

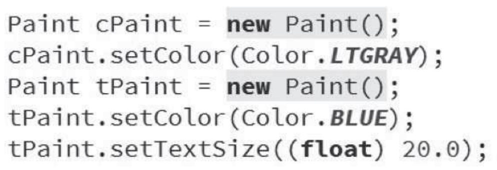


```
<resources>
    <color name="my_color">#7fff00ff</color>
    <color name="puzzle_background">#ffff0000</color>
    <color name="puzzle_hi_lite">#ffffffff</color>
    <color name="puzzle_light">#64c6d4ef</color>
    <color name="puzzle_dark">#6456648f</color>
    <color name="puzzle_foreground">#ff000000</color>
    <color name="puzzle_hint_0">#64ff0000</color>
    <color name="puzzle_hint_1">#6400ff80</color>
    <color name="puzzle_hint_2">#2000ff80</color>
    <color name="puzzle_selected">#64ff8000</color>
</resources>
```

Figure 4.2 Preview of colors in XML files in Android Studio.

4.1.2 Paint

The *Paint* class holds the style and color information on drawing geometries, text, and bitmaps. Before we draw something on the screen, we can set color to a Paint via *setColor()* method.



```
Paint cPaint = new Paint();
cPaint.setColor(Color.LTGRAY);
Paint tPaint = new Paint();
tPaint.setColor(Color.BLUE);
tPaint.setTextSize((float) 20.0);
```

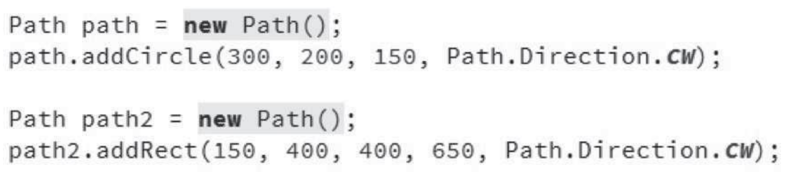
Figure 4.3 Paint class in Android.

As shown in Fig. 4.3, we create two *Paint*s, which are *cPaint* to draw a circle and *tPaint* to draw text. We set the color of the circle as light gray and the color of text as blue. Beside colors, we also can set other attributes to Paint class, such as the *TextSize*.

4.1.3 Path

The *Path* class encapsulates multiple contour geometric paths, such as lines, rectangles, circles, and curves. Fig. 4.4 is an example that defines a circular path and a rectangle path.

The second line defines a circle, whose center is at position x=300, y=200, with a radius of 150 pixels. The fourth line defines a rectangle whose left top point is at position x=150, y=400, and right bottom point is at position x=400, y=650. The *Path.Direction.CW* indicates that the shape will be drawn clockwise. The other direction is CCW, which indicates counter-clockwise.



```
Path path = new Path();
path.addCircle(300, 200, 150, Path.Direction.CW);

Path path2 = new Path();
path2.addRect(150, 400, 400, 650, Path.Direction.CW);
```

Figure 4.4 Create two *Path* object and add details to them.

4.1.4 Canvas

To draw something, we need to prepare four basic components, including a *Bitmap* to hold the pixels, a *Canvas* to host the draw call, a drawing primitive, and a *Paint*. The *Bitmap* is the place where to draw something, and the Canvas is used to hold the "draw" calls. A drawing primitive can be a Rect, a Circle, a Path, a Text, and a Bitmap.

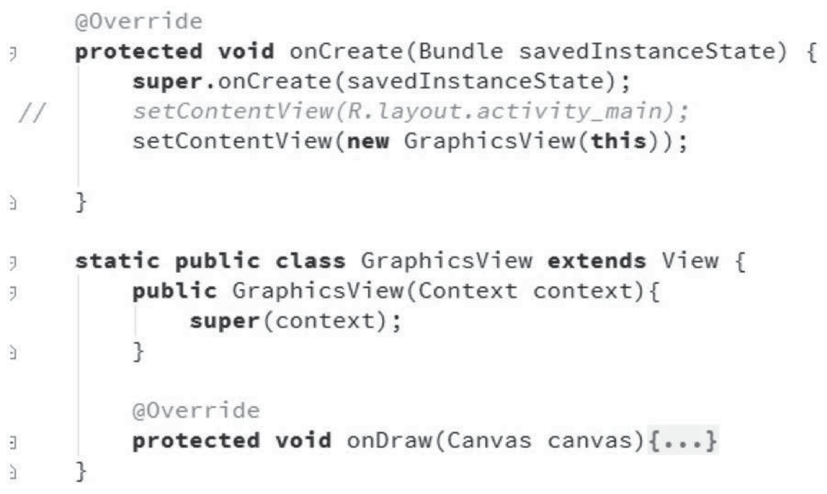
In Android, a display screen is taken up by an Activity, which hosts a *View*, which in turn hosts a Canvas. We can draw on the canvas by overriding the *View.onDraw()* method. A Canvas object is the only parameter to *onDraw()* method. We create a new activity, which contains a view called *GraphicsView*, but not the layout.xml, as shown in Fig. 4.8.

In Fig. 4.5, we comment the original code and set the content view of this activity to some layout.xml, and set it to some new view we created, which is *GraphicsView*.

Let's review the two methods of designing Android apps. There are two methods to design Android apps, which are procedural and declarative. The "setContentView(R.layout.activity_main)" is a typical example of declarative, which is described all objects in the activity using XML files. The "setContentView(new GraphicsView(this))" is a typical example of a procedural, which means writing Java code to create and manipulate all the user interface objects

[56].

This new class, *GraphicsView*, extends the class *View*. The *on- Draw()* method is over- rider and used to implement the function of drawing. Fig. 4.6 shows the details of the *onDraw()* method. We use Paint with different colors to draw a Path on the View via calling *onDraw(Canvas)* method.



```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
//      setContentView(R.layout.activity_main);
        setContentView(new GraphicsView(this));

    }

    static public class GraphicsView extends View {
        public GraphicsView(Context context){
            super(context);
        }

        @Override
        protected void onDraw(Canvas canvas){...}
    }
```

Figure 4.5 A new activity whose contentView is the view created our- selves but not layout.xml.

Meanwhile, we have another choice to create a Canvas, as shown in Fig. 4.8. In Fig. 4.8, we create a Bitmap that is a square whose size is 100*100 and will use it as the argument of Canvas. Then we can use this canvas as the same as the one offered in the onDraw() method.

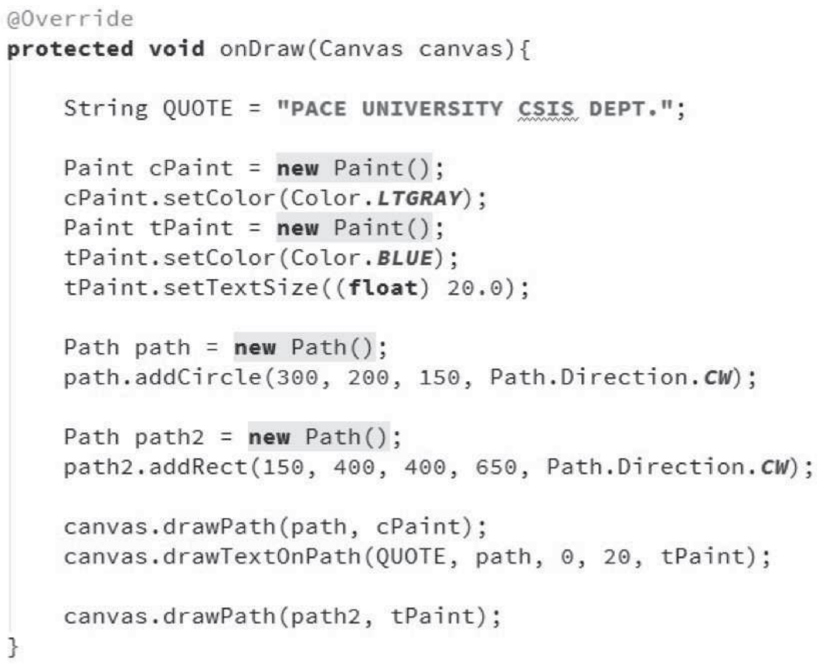
4.1.5 Drawable

Android.graphics.drawable provides classes to manage a variety of visual elements, which are intended for display, such as bitmaps and gradients. We can combine drawables with other graphics, or we can use them in UI widgets, such as the background for a button. Android offers following types of drawables:

Bitmap: A bitmap graphic file (.png, .jpg, or .gif).

Nine-Patch: A PNG file with stretchable regions to allow image resizing based on content (.9.png).

Layer: A *Drawable* that manages an array of other *Drawables*. These are drawn in array order, so the element with the largest index is be drawn on top.



```
@Override
protected void onDraw(Canvas canvas){

    String QUOTE = "PACE UNIVERSITY CSIS DEPT.";

    Paint cPaint = new Paint();
    cPaint.setColor(Color.LTGRAY);
    Paint tPaint = new Paint();
    tPaint.setColor(Color.BLUE);
    tPaint.setTextSize((float) 20.0);

    Path path = new Path();
    path.addCircle(300, 200, 150, Path.Direction.CW);

    Path path2 = new Path();
    path2.addRect(150, 400, 400, 650, Path.Direction.CW);

    canvas.drawPath(path, cPaint);
    canvas.drawTextOnPath(QUOTE, path, 0, 20, tPaint);

    canvas.drawPath(path2, tPaint);
}
```

Figure 4.6 The “onDraw()” method that draws a circle and a rectangle.

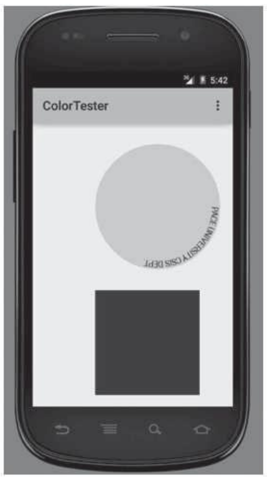
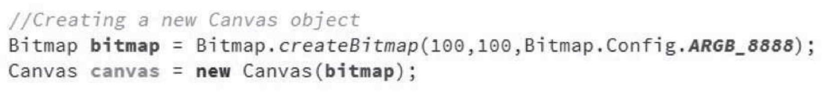


Figure 4.7 Running result of GraphicsView.



```
//Creating a new Canvas object
Bitmap bitmap = Bitmap.createBitmap(100,100,Bitmap.Config.ARGB_8888);
Canvas canvas = new Canvas(bitmap);
```

Figure 4.8 Use Bitmap to create a new Canvas.

State: An XML file that references different bitmap graphics for different states (for example, to use a different image when a button is pressed).

Level: An XML file that defines a drawable that manages a num- ber of alternate *Drawables*, each assigned a maximum numerical value. Creates a *LevelListDrawable*.

Transition: An XML file that defines a drawable that can cross-fade between two drawable resources.

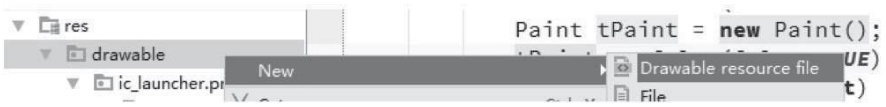
Inset Drawable: An XML file that defines a drawable that insets another drawable by a specified distance. This is useful when a View needs a background drawable that is smaller than the View's actual bounds.

Clip: An XML file that defines a drawable that clips another *Draw- able* based on this *Drawable's* current level value.

Scale: An XML file that defines a drawable that changes the size of another *Drawable* based on its current level value.

Shape: An XML file that defines a geometric shape, including colors and gradients.

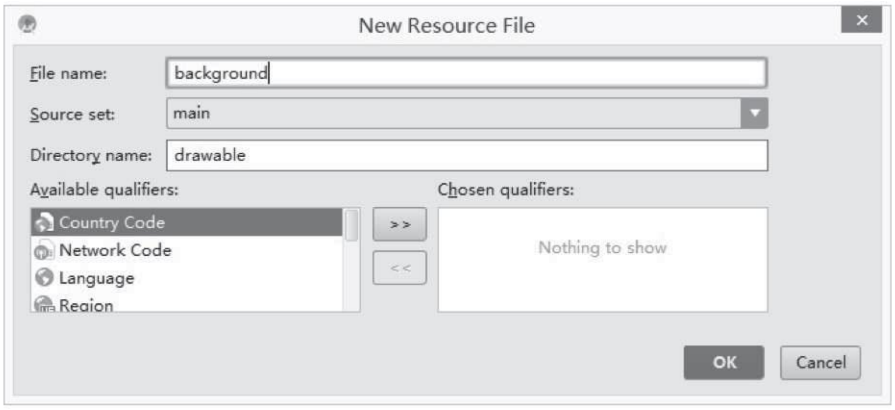
A drawable resource is a general concept for a graphic that can be drawn to the screen and that can be retrieved with *Application Programming Interface* (API). Now we will add a gradient background to our ColorTester. We create a drawable resource file in res\drawable, and then create a *Shape* inside the background.xml file, as shown in Fig. 4.9 and Fig. 4.10.



```
Paint tPaint = new Paint();
```

Figure 4.9 The first step of creating a new Drawable resource file.

As shown in Fig. 4.11, we define a gradient from the start color to the end color. The angle indicates the direction of the gradient, and it must be the extract times 45. When the angle = 0, the sequence is from left to right. When the angle = 90, the sequence is from bottom to top. When the angle = 180, the sequence is from right to left. When the angle = 270, the sequence is from top to bottom.


New Resource File

File name: background

Source set: main

Directory name: drawable

Available qualifiers:

Chosen qualifiers:

Figure 4.10 The second step of creating a new Drawable resource file.

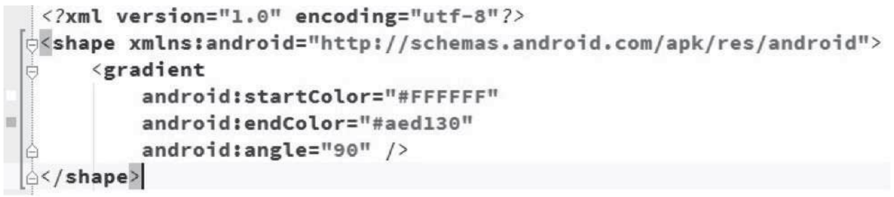

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android">
    <gradient
        android:startColor="#FFFFFF"
        android:endColor="#aed130"
        android:angle="90" />
</shape>
```

Figure 4.11 Shape Drawable.

Then add one attribute into the RelativeLayout in activity_main .xml as “android:background:@drawable/background”. The running result is shown in Fig. 4.12.

Besides gradient, there are some other common attributes that can be added into a shape, including stroke, corners, and padding. We add them into the shape of background.xml, and set the color of the stroke is red, the width of the dash is 10dp, etc. The attributes and the running result are shown in Fig. 4.13. From Fig. 4.13, we can see that the background is stroked by a red dash, and every corner has a round edge.

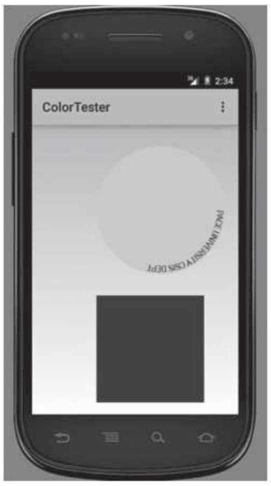
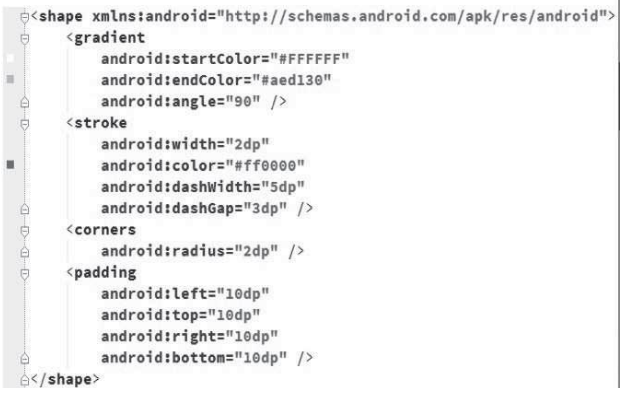


Figure 4.12 Gradient background.



```xml
<shape xmlns:android="http://schemas.android.com/apk/res/android">
    <gradient
        android:startColor="#FFFFFF"
        android:endColor="#aed130"
        android:angle="90" />
    <stroke
        android:width="2dp"
        android:color="#ff0000"
        android:dashWidth="5dp"
        android:dashGap="3dp" />
    <corners
        android:radius="2dp" />
    <padding
        android:left="10dp"
        android:top="10dp"
        android:right="10dp"
        android:bottom="10dp" />
</shape>
```

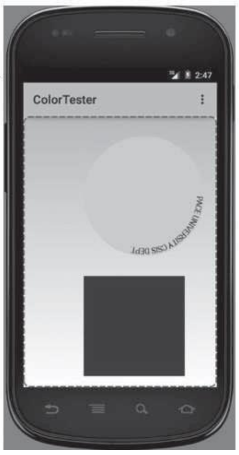
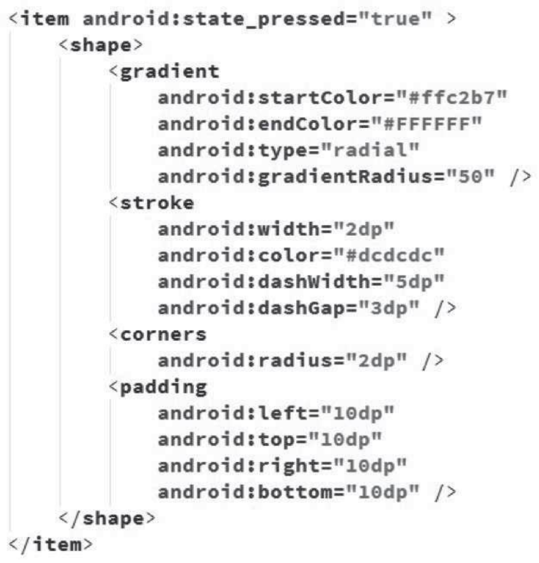


Figure 4.13 Stroke, Corners, and Padding Drawables

4.1.6 Button Selector

We want to set different colors to buttons when they are at differ- ent states. We set the default color of a button as light purple, and the color when it is pressed is light orange. As we introduced in the previous section, we need to create a drawable resource file to implement this function. Thus, we create a new drawable resource file named "button_selection", and between the <selector> and < /selector> add two items. The first one is the pressed state, which indicates that the button is pressed, as shown in Fig. 4.14. The second one is the default state, as shown in Fig. 4.15.



```
<item android:state_pressed="true" >
    <shape>
        <gradient
            android:startColor="#ffc2b7"
            android:endColor="#FFFFFF"
            android:type="radial"
            android:gradientRadius="50" />
        <stroke
            android:width="2dp"
            android:color="#dcdcdc"
            android:dashWidth="5dp"
            android:dashGap="3dp" />
        <corners
            android:radius="2dp" />
        <padding
            android:left="10dp"
            android:top="10dp"
            android:right="10dp"
            android:bottom="10dp" />
    </shape>
</item>
```

Figure 4.14 Default state of button.

Then, we add one attribute to all the three buttons as follows: android:background="@drawable/button_selector".

The running result is shown in Fig. 4.16.