Chapter 4: Drawing in Our Window

Drawing in Our Window

The next four lines draw the blue line near the edges of the window. Here are those lines again:

```
        g2d.setColor(Color.blue);
        g2d.drawLine(2, 2, 2, h-3);
        g2d.drawLine(2, h-3, w-3, h-3);
        g2d.drawLine(w-3, h-3, w-3, 2);
        g2d.drawLine(w-3, 2, 2, 2);
```

The first line changes the drawing color to blue. The next four lines use the drawLine() method to draw four lines around the edges of the window, two pixels away from the edges. The Graphics2D drawLine() method will draw a line between any two points.

The first two arguments are the x-coordinate and y-coordinate of one end of the line; the third and fourth arguments are the coordinates of the other end of the line. So the first call to drawLine() above draws a vertical line down the left edge of the window, two pixels away from the edge, between the points (2,2) and (2,h-3).

The second call to drawLine() draws a horizontal line along the bottom edge of the window between points (2,h-3) and (w-3, h-3).

The third call draws another vertical line from (w-3,h-3) up along the right edge to (w-3,2).

And the last call draws another horizontal line along the top, from (w-3,2) back to the top-left corner at (2,2). After that call, we have a blue box around the edge of the window. (I could have drawn it with one call to drawRect(), but I wanted to discuss about drawing lines first.)

Here are the next three lines in the code:

```
        g2d.setColor(Color.red);
        g2d.drawRect(5, 5, 180, 20);
        g2d.drawString("The window's width is " + size.width + ".", 10, 20);
```

These lines change the color to red and draw a red rectangle from the point (5,5) to (185,25), a width of 180 and a height of 20. The drawRect() method is just like the fillRect() that we used for the background except that it draws the outline of a rectangle instead of a filled rectangle.

The third line, the call to drawString(), is the way we can paint text on the screen. This line uses the default font; we'll discuss in a bit about how to use other fonts.

The first argument in the call is the string we want to display, and the two numeric arguments are the coordinates of the left end of the string's *baseline*. A font's baseline is like the line you learned to print on in grade school. It's the red line at the bottom edge of the letters in the image below. If you'd like more information on font measurements, you can refer to the Supplementary Material.

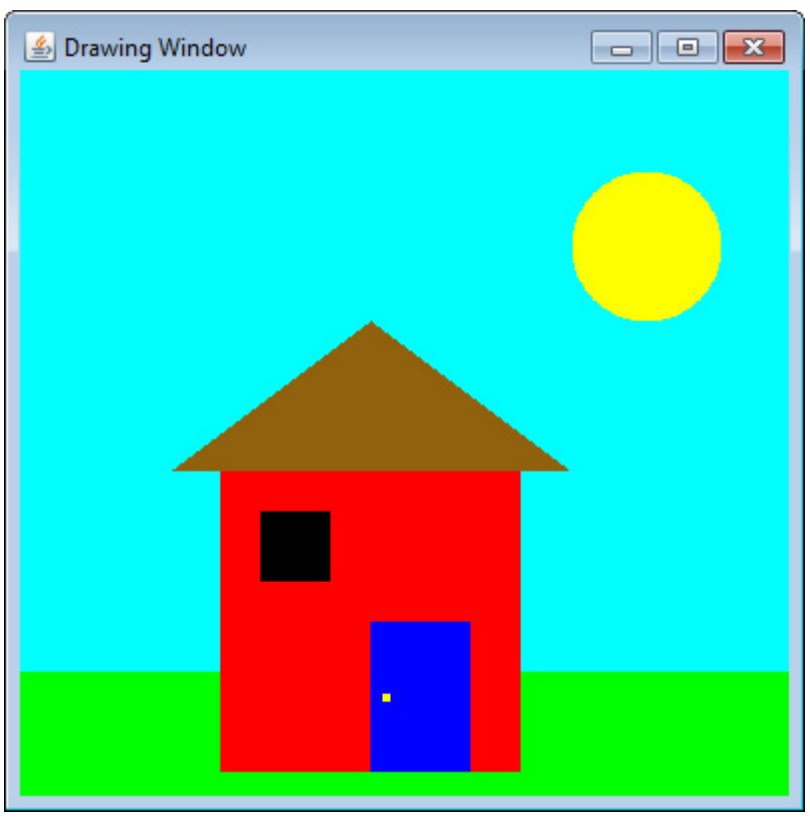
Font measurements, including the baseline

The net result of those three lines of code is to draw a red rectangle with a line of red text in it giving us our width in pixels. Note that the width this time is different from the full width of the window. This is the width of what Java calls our DrawingSurface, which in this case is the interior dimensions of the window.

The rest of the program is five more repetitions of these three lines of code, using different colors, sizes of rectangles, and text strings.

And that's the basic process for drawing in a Java window. But as you might suspect, we can't stop there. We can use a lot more drawing tools in the Graphics2D class.

For example, take a look at this image. A kindergartner could draw it, but it shows some of the other drawing methods that Java makes available.



Kindergarten art of a house and sun

This image consists of five rectangles, two ovals (the doorknob is an oval even though the screen resolution doesn't look like it), and a triangle.

The Basics of Geometry

Note

> For those of us doing our best to forget geometry, a *polygon* is a two-dimensional shape with three or more straight sides. That includes triangles (three-sided), rectangles (four-sided), pentagons (five-sided), and so on.
>
> Java defines polygons as a sequence of points that form the corners of the polygon, with each point joined to the next with a line to form the sides. The final point joins back to the first to form the final side.

The DrawingWindow class for this example is identical to that from the previous example except for the name of the second class it refers to (DrawingArt instead of DrawingSurface) and its size (400 × 400 rather than 400 × 300). So I'm going to focus on the code for the second class, DrawingArt, which is here:

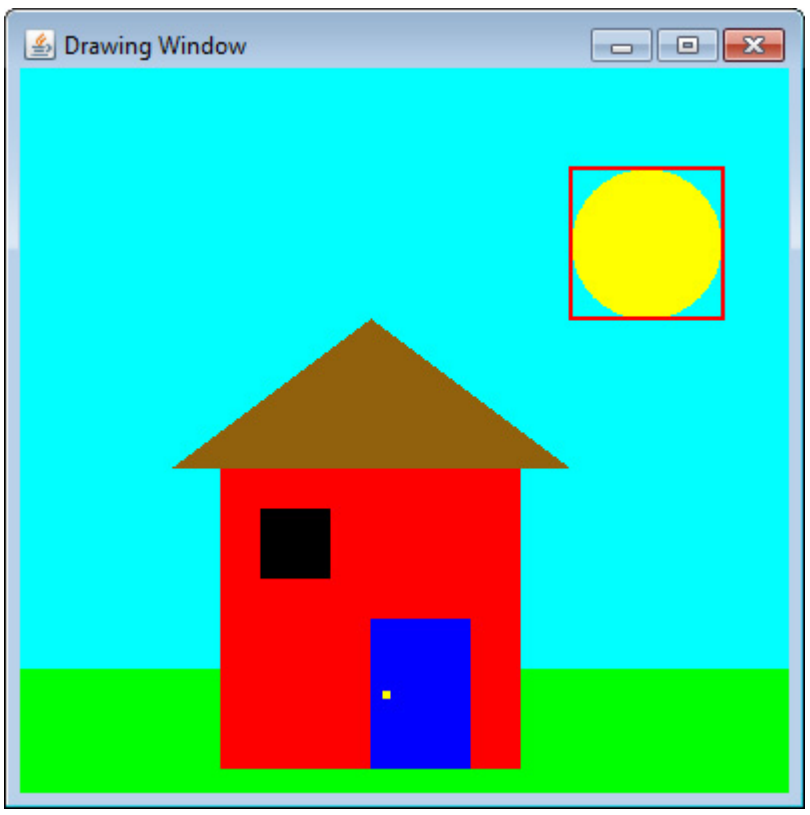
```
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Polygon;
import javax.swing.JPanel;

class DrawingArt extends JPanel {
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2d = (Graphics2D) g;

        // draw a light blue sky
        g2d.setColor(Color.cyan);
        g2d.fillRect(0, 0, getWidth(), getHeight());

        // draw green grass
        g2d.setColor(Color.green);
        g2d.fillRect(0, 300, getWidth(), getHeight()-300);

        // draw the red wall
        g2d.setColor(Color.red);
        g2d.fillRect(100, 200, 150, 150);

        // draw the window
        g2d.setColor(Color.black);
        g2d.fillRect(120, 220, 35, 35);

        // draw the door
        g2d.setColor(Color.blue);
        g2d.fillRect(175, 275, 50, 75);

        // draw the sun
        g2d.setColor(Color.yellow);
        g2d.fillOval(275, 50, 75, 75);

        // draw the doorknob
        g2d.setColor(Color.yellow);
        g2d.fillOval(180, 310, 5, 5);

        // draw the roof
        Polygon p = new Polygon();
        p.addPoint(75,200);
```

```
        p.addPoint(175,125);
        p.addPoint(275,200);
        g2d.setColor(new Color(145,97,13));
        g2d.fillPolygon(p);
    }
}
```

The first few lines of code are the same as in our previous example except for the class name. Then we have five pairs of code lines that use fillRect() to draw five rectangles: a blue one for the sky, a green one for grass, a red one for a wall, a black one for a window, and a blue one for a door.

Then there are two pairs of lines that use fillOval(), a method that draws filled ovals, to draw two yellow circles for the sun and the doorknob.

In Java, you position ovals and circles using a *bounding rectangle*, which is the smallest rectangle that can hold the oval. The four arguments that draw an oval work just the same as for a rectangle, but instead of drawing the rectangle, Java draws the oval that fits into the rectangle. For example, the following image shows the sun from our kindergarten art along with its bounding rectangle.



The last six lines of code draw the triangle for the roof. Java doesn't have a method to draw triangles, but it has one for polygons, which will work just fine.

Creating a polygon is easy. The first of those six lines of code creates a Polygon object named p. The next three lines add three points that define the three corners of the triangle. The first point uses the same y-coordinate as the top of the wall (200) and an x-coordinate just a little to the left of the wall (75). The second point defines the peak of the roof. It's at the midpoint of the wall (x is 175) and a little above it (y is 125). The third point is to the right of the wall (x is 275), and it's even with the top of the wall (y is 200). That defines the shape of the roof.

The next line of code defines the roof's color. I wanted a brown roof, and the Color class doesn't have a predefined brown shade. So I pulled up an online color chart at Colorpicker (http://www.colorpicker.com) and found a shade of brown I liked. The chart showed me the numbers to use to create it.

The three numbers I used for my Color constructor define the shades of red, green, and blue for the color—in other words, its *RGB values*. Those values can be any integer between 0 and 255. Using the RGB definition, I can define 256^3 colors, or 16,777,216 colors. You can figure out just about any color you want using a color chart, and then you can use the values from the chart to create your color.

The last line of the DrawingArt class draws the roof, which completes our picture.

There are a lot more methods in the Graphics2D class, and more inherited from the Graphics class, that you can use for drawing images. And my students have far more imagination when it comes to using them than I do! Here's a short list of the ones I've found useful and their purposes:

- drawArc() and fillArc() draw partial ovals.
- drawImage() displays an image that's been loaded from a JPEG, BMP, or PNG file.
- drawRoundRect() and fillRoundRect() draw rounded rectangles.
- setStroke() sets line thickness and patterns of dashed and dotted lines.

There's more information about two-dimensional graphics in the Supplementary Material.

What did you think of our exploration of Java art? Will you use colors and designs in your own projects?

Please join me in Chapter 5 so we can finish Lesson 11.