

Chapter 3: Creating and Displaying the Java Window

Creating and Displaying the Java Window

The rest of the `main()` method is the code that actually creates the window object and displays it. In this case some code displays some of the window's characteristics in BlueJ's terminal window.

The next line in our method,

```
WindowFrame dw = new WindowFrame();
```

creates a `WindowFrame` variable named `dw` and a new `WindowFrame` object for it to refer to. After this statement executes, our window exists, but you can't see it on the display screen. The following line fixes that by setting the window's `visible` property to `true` so the window displays on-screen:

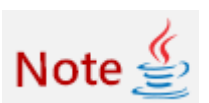
```
dw.setVisible(true);
```

That completes the code to set up and display a window in Java. I could stop here, and the window would display just fine. But I've added a few lines of code that tell us a bit about our window:

```
Dimension dim = dw.getSize();
Insets ins = dw.getInsets();
System.out.println("Window Size is " + dim.width + " by " + dim.height);
System.out.println("Window top inset is " + ins.top);
System.out.println("Window bottom inset is " + ins.bottom);
System.out.println("Window left inset is " + ins.left);
System.out.println("Window right inset is " + ins.right);
```

The first line creates a new `Dimension` variable and uses the `Frame` `getSize()` method to get a `Dimension` object for it. Java's `Dimension` class describes the size of a GUI component, giving us its width and height in pixels.

What's a Pixel?



A *pixel*, or picture element, is one of the tiny dots that make up the display surface of your screen. We'll be discussing about pixels, their colors, and their coordinates a lot in this lesson and in Lesson 12.

The next line creates a new `Insets` variable and object using the `getInsets()` method. The `Insets` class describes the borders of a GUI component, giving us the depth of the borders on all four sides.

These classes are two of a few Java classes that have public variables in them. The Dimension's width and length fields tell us the size of a GUI component. Inset's top, bottom, left, and right fields tell us how deep the border of the component is on those sides. In the terminal window that I showed you in Chapter 2, I used these objects and their variables to display the size of the frame and the depths of the title bar and border.

That was a lot of explanation for a simple program that just shows us a window. But now we'll start putting something into that window.

A Window With Something In It

Let's start this topic with an image of the window we're building. Then I'll show you the code I used to create it.

Text equivalent start.

Java frame with content drawn in various colors:

The window's width is 384.

The window's height is 262.

The window's left inset is 0.

The window's right inset is 0.

The window's top inset is 0.

The window's bottom inset is 0.

Text equivalent stop.

As you can see, I've drawn a blue rectangle just inside the borders of the window and some smaller rectangles inside it with text similar to what the first example displayed in the terminal window. Creating this window involved two classes instead of one.

Here's the code for the first class—the class with our main() method. It looks like the code for our first, empty, window.

```
import javax.swing.JFrame;
import javax.swing.SwingUtilities;

public class DrawingWindow extends JFrame {

    public DrawingWindow() {
        setTitle("Drawing Window");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        add(new DrawingSurface());

        setSize(400, 300);
        setLocationRelativeTo(null); // centers window in display screen
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                DrawingWindow dw = new DrawingWindow();
                dw.setVisible(true);
            }
        });
    }
}
```

Notice the line in the middle of the constructor that calls `add(new DrawingSurface())`. Our frame's `add()` method is what lets us add a visible component to our frame. What we're adding to it in this case is a new `DrawingSurface` object. And that brings me to our second class for this project, `DrawingSurface`, which will do some drawing and display it in the window:

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

class DrawingSurface extends JPanel {
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2d = (Graphics2D) g;
        Dimension size = getSize();
        Insets insets = getInsets();
        int w = size.width - insets.left - insets.right;
        int h = size.height - insets.top - insets.bottom;

        g2d.setColor(Color.white);
        g2d.fillRect(0,0,w,h);

        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);

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

        g2d.setColor(Color.black);
        g2d.drawRect(5, 35, 185, 20);
        g2d.drawString("The window's height is " + size.height + ".", 10, 50);

        g2d.setColor(Color.gray);
        g2d.drawRect(5, 65, 185, 20);
        g2d.drawString("The window's left inset is " + insets.left + ".", 10, 80);

        g2d.setColor(Color.green);
        g2d.drawRect(5, 95, 195, 20);
        g2d.drawString("The window's right inset is " + insets.right + ".", 10, 110);

        g2d.setColor(Color.orange);
```

```

        g2d.drawRect(5, 125, 185, 20);
        g2d.drawString("The window's top inset is " + insets.top + ".", 10, 140);

        g2d.setColor(Color.pink);
        g2d.drawRect(5, 155, 210, 20);
        g2d.drawString("The window's bottom inset is " + insets.bottom + ".", 10,
170);
    }
}

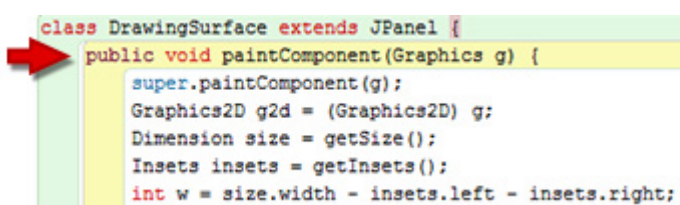
```

This may look like a lot of code, but it's repetitive. So let's look at what we've got.

The first line after the imports declares our class as a subclass of JPanel. So our class will inherit all JPanel's properties and methods. JPanel is Swing's primary class for holding other display objects, and it's going to provide the "surface" or "canvas" where we'll draw.

JPanel has a method named `paintComponent()`, which we'll *override* in order to do our drawing. Overriding a method means we'll replace the method we inherit with the one we'll write. It's a common practice when we want our class to do things a little differently than the class we're inheriting from. It's especially common to add to it, and that's what we'll do.

The first thing about the `paintComponent()` method I want to mention is its parameter, a Graphics object named `g`. We get that object when we call the `paintComponent()` method. We'll use this object to do our drawing, so it's sort of our paintbrush.



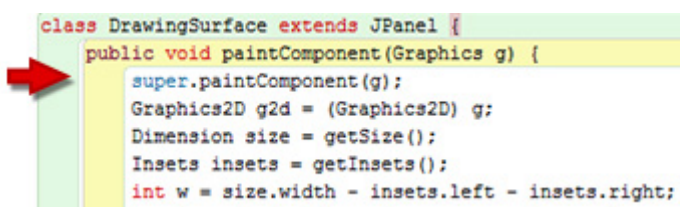
```

import javax.swing.*;
import java.awt.*;
import java.awt.geom.*;

class DrawingSurface extends JPanel {
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2d = (Graphics2D) g;
        Dimension size = getSize();
        Insets insets = getInsets();
        int w = size.width - insets.left - insets.right;
    }
}

```

The next line is another new one: "`super.paintComponent(g)`", which calls the method in our parent class that we're overriding. It goes to our class's superclass, which is JPanel, and runs its `paintComponent()` method.



```

import javax.swing.*;
import java.awt.*;
import java.awt.geom.*;

class DrawingSurface extends JPanel {
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2d = (Graphics2D) g;
        Dimension size = getSize();
        Insets insets = getInsets();
        int w = size.width - insets.left - insets.right;
    }
}

```

Why would we want to do that? We're essentially using it to clear, or erase, the panel and paint an opaque background. The default JPanel is an empty space, and calling its `paintComponent()` method clears out the whole thing. This will come in handy once we get to animation, where each time we paint the window we need to erase the previous contents. The rest of our version of the method will be adding things to it.

The next line is another line to copy just the way it is. It *casts* our Graphics object into a Graphics2D object. Casting is Java's process for changing one data type into another, as long as Java knows how to make the transition.

```
class DrawingSurface extends JPanel {  
    public void paintComponent(Graphics g) {  
        super.paintComponent(g);  
        Graphics2D g2d = (Graphics2D) g;  
        Dimension size = getSize();  
        Insets insets = getInsets();  
        int w = size.width - insets.left - insets.right;  
    }  
}
```

To maintain backward compatibility so that programs written in previous versions of Java will still run, the parameter for this method has to be a Graphics object. But it's actually been updated to a newer version of the class, a subclass of Graphics named Graphics2D. So the object we receive in our method is actually a Graphics2D object wrapped in a Graphics "shell." When we cast it, we're telling Java we really know it's a Graphics2D object, and that's how we want to use it.

The next four lines repeat what we did in the first program to get the height and width of our window. This time we're getting the width and height of our DrawingSurface space. Now we know how much room we have to work with. And we're finally ready to draw!

We'll do all our drawing by calling methods in the Graphics2D class. The first two calls paint a white background. I did that because windows in different operating systems have different default background colors, and I want to be consistent across platforms.

The first call, "g2d.setColor(Color.white)", sets the color we want to draw in. Java's Color class has a group of predefined colors set up in it. The color white is one of them, and we'll see others as we go along. (You can see the complete list in the API documentation for the Color class.)

The next line, "g2d.fillRect(0,0,w,h)", calls the method that draws a filled rectangle with the color we just set, white. The first two arguments in the call are the x-coordinate and y-coordinate of the top-left corner of the rectangle. That corner will be at the origin of the window, the point (0,0).

Understanding Java's Coordinates



In computer graphics, the origin of the window, position (0,0), is the top-left corner. As we move to the right, the x-coordinate increases, just as it does in the Cartesian coordinate systems you probably learned in high school algebra. But the y-coordinate in graphics increases as we go down the window, which is different from the graphing coordinates in algebra. And Java measures coordinates in pixels rather than inches or centimeters.

In our window, then, the top-left corner is the point (0,0). The top-right corner is (w-1,0), where w is the width of the window. The bottom-left corner is (0,h-1), where h is the height of the window. And the bottom-right corner is (w-1,h-1).

The third argument is the width of the rectangle, and since w is the width of the window, it's as wide as our window. The fourth argument is the height of the rectangle, and since h is the height of the window, the rectangle is also as high as our window. So that call to fillRect() draws a filled white rectangle that becomes the background for our drawing.

In the next chapter we'll review the rest of the code from that window and see how to draw other items.

