5. Painting in Swing

To write output directly to the surface of a component, you will use one or more drawing methods defined by the AWT, such as drawLine( ) or drawRect( ).

5.1 Painting Fundamentals

The AWT class Component defines a method called paint( ) that is used to draw output directly to the surface of a component. For the most part, paint( ) is not called by your program. Rather, paint( ) is called by the run-time system whenever a component must be rendered. This situation can occur for several reasons. For example, the window in which the component is displayed can be overwritten by another window and then uncovered. Or, the window might be minimized and then restored. The paint( ) method is also called when a program begins running. When writing AWT-based code, an application will override paint( ) when it needs to write output directly to the surface of the component.

Because JComponent inherits Component, all Swing's lightweight components inherit the paint( ) method. However, you will not override it to paint directly to the surface of a component. The reason is that Swing uses a bit more sophisticated approach to painting that involves three distinct methods:

1. paintComponent( ),
2. paintBorder( ), and
3. paintChildren( ).

These methods paint the indicated portion of a component and divide the painting process into its three distinct, logical actions. In a lightweight component, the original AWT method paint( ) simply executes calls to these methods, in the order just shown.

To paint to the surface of a Swing component, you will create a subclass of the component and then override its paintComponent( ) method. This is the method that paints the interior of the component. You will not normally override the other two painting methods. When overriding paintComponent( ), the first thing you must do is call super.paintComponent( ), so that the superclass portion of the painting process takes place. (The only time this is not required is when you are taking complete, manual control over how a component is displayed.) After that, write the output that you want to display. The paintComponent( ) method is shown here:

```
protected void paintComponent(Graphics g)
```

To cause a component to be painted under program control, call repaint( ).The repaint( ) method is defined by Component. Calling it causes the system to call paint( ) as soon as it

ispossible to do so. Because painting is a time-consuming operation, thismechanism allows the run-time system to defer painting momentarily until some higher-priority task has completed. In Swing the call to paint( ) results in a call to paintComponent( ). Therefore, to output to the surface of a component, your program will store the output until paintComponent( ) is called.

5.1 Compute the Paintable Area

When drawing to the surface of a component, you must restrict your output to the area that is inside the border. Although Swing automatically clips any output that will exceed the boundaries of a component, it is still possible to paint into the border, which will then get overwritten when the border is drawn.

To avoid this, you must compute the paintable area of the component.

This is the area defined by the current size of the component minus the space used by the border. Therefore, before you paint to a component, you must obtain the width of the border and then adjust your drawing accordingly. To obtain the border width, call getInsets( ):-

Insets getInsets( )

This method is defined by Container and overridden by JComponent. It returns an Insets object that contains the dimensions of the border. The inset values can be obtained by using these fields:

- int top;
- int bottom;
- int left;
- int right;

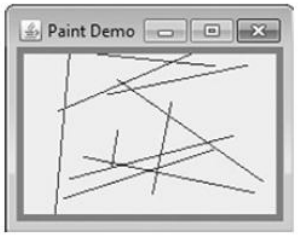
These values are then used to compute the drawing area given the width and the height of the component. You can obtain the width and height of the component by calling getWidth( ) and getHeight( ) on the component. They are shown here:

- int getWidth( )
- int getHeight( )

By subtracting the value of the insets, you can compute the usable width and height of the component.

5.2 A Paint Example

It creates a class called PaintPanel that extends JPanel. The program then uses an object of that class to display lines whose endpoints have been generated randomly.



Sample output from the **PaintPanel** program

```
// Paint lines to a panel.

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.*;

// This class extends JPanel. It overrides
// the paintComponent() method so that random
// lines are plotted in the panel.
class PaintPanel extends JPanel {
  Insets ins; // holds the panel's insets

  Random rand; // used to generate random numbers

  // Construct a panel.
  PaintPanel() {

    // Put a border around the panel.
    setBorder(
      BorderFactory.createLineBorder(Color.RED, 5));

    rand = new Random();
  }
```

```
  // Override the paintComponent() method.
  protected void paintComponent(Graphics g) {
    // Always call the superclass method first.
    super.paintComponent(g);

    int x, y, x2, y2;

    // Get the height and width of the component.
    int height = getHeight();
    int width = getWidth();

    // Get the insets.
    ins = getInsets();

    // Draw ten lines whose endpoints are randomly generated.
    for(int i=0; i < 10; i++) {
      // Obtain random coordinates that define
      // the endpoints of each line.
      x = rand.nextInt(width-ins.left);
      y = rand.nextInt(height-ins.bottom);
      x2 = rand.nextInt(width-ins.left);
      y2 = rand.nextInt(height-ins.bottom);

      // Draw the line.
      g.drawLine(x, y, x2, y2);
    }
  }
}


// Demonstrate painting directly onto a panel.
class PaintDemo {

  JLabel jlab;
  PaintPanel pp;

  PaintDemo() {

    // Create a new JFrame container.
    JFrame jfrm = new JFrame("Paint Demo");

    // Give the frame an initial size.
    jfrm.setSize(200, 150);

    // Terminate the program when the user closes the application.
    jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    // Create the panel that will be painted.
    pp = new PaintPanel();

    // Add the panel to the content pane. Because the default
    // border layout is used, the panel will automatically be
    // sized to fit the center region.
    jfrm.add(pp);
```

```
    // Display the frame.
    jfrm.setVisible(true);
  }

  public static void main(String args[]) {
    // Create the frame on the event dispatching thread.
    SwingUtilities.invokeLater(new Runnable() {
      public void run() {
        new PaintDemo();
      }
    });
  }
}
```

Explanation:-

The PaintPanel class extends JPanel.

To handle painting, PaintPanel overrides the paintComponent( ) method. This enables PaintPanel to write directly to the surface of the component when painting takes place. The size of the panel is not specified because the program uses the default border layout and the panel is added to the center.

This results in the panel being sized to fill the center. If you change the size of the window, the size of the panel will be adjusted accordingly.

The constructor also specifies a 5-pixel wide, red border. This is accomplished by setting the border by using the setBorder( ) method:

void setBorder(Border border)

Border is the Swing interface that encapsulates a border.

You can obtain a border by calling one of the factory methods defined by the BorderFactory class. The one used in the program is createLineBorder( ), which creates a simple line border.

static Border createLineBorder(Color clr, int width)

Here, clr specifies the color of the border and width specifies its width in pixels. Inside the override of paintComponent( ), it first calls super.paintComponent( ). This is necessary to ensure that the component is properly drawn. Next, the width and height of the panel are

obtained along with the insets. These values are used to ensure the lines lie within the drawing area of the panel. The drawing area is the overall width and height of a component less the border width. The computations are designed to work with differently sized PaintPanels and borders. To prove this, try changing the size of the window. The lines will still all lie within the borders of the panel.

The PaintDemo class creates a PaintPanel and then adds the panel to the content pane. When the application is first displayed, the overridden paintComponent( ) method is called, and the lines are drawn. Each time you resize or hide and restore the window, a new set of lines are drawn. In all cases, the lines fall within the paintable area.