Painting in Swing

Swing's painting system is able to render vector graphics, images, and outline font-based text. Painting is needed in applications when we want to change or enhance an existing widget, or if we are creating a custom widget from scratch To do the painting, we use the painting API provided by the Swing toolkit.

The painting is done within the `paintComponent()` method. In the painting process, we use the `Graphics2D` object.

2D Vector Graphics

There are two different computer graphics. Vector and raster graphics. Raster graphics represents images as a collection of pixels. Vector graphics is the use of geometrical primitives such as points, lines, curves or polygons to represent images. These primitives are created using mathematical equations.

Both types of computer graphics have advantages and disadvantages. The advantages of vector graphics over raster are:

- smaller size
- ability to zoom indefinitely
- moving, scaling, filling or rotating does not degrade the quality of an image

Points

The most simple graphics primitive is point. It is a single dot on the window. There is no method to draw a point in Swing. To draw a point, we use the `drawLine()` method. We use one point twice.

```
package com.zetcode;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Insets;
import java.util.Random;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.SwingUtilities;

class DrawPanel extends JPanel {

    private void doDrawing(Graphics g) {

```

```
        Graphics2D g2d = (Graphics2D) g;

        g2d.setColor(Color.blue);

        for (int i = 0; i <= 1000; i++) {

            Dimension size = getSize();
            Insets insets = getInsets();

            int w = size.width - insets.left - insets.right;
            int h = size.height - insets.top - insets.bottom;

            Random r = new Random();
            int x = Math.abs(r.nextInt()) % w;
            int y = Math.abs(r.nextInt()) % h;
            g2d.drawLine(x, y, x, y);
        }
    }

    @Override
    public void paintComponent(Graphics g) {

        super.paintComponent(g);
        doDrawing(g);
    }
}

public class PointsExample extends JFrame {

    public PointsExample() {
        initUI();
    }

    public final void initUI() {

        DrawPanel dpnl = new DrawPanel();
        add(dpnl);

        setSize(250, 200);
        setTitle("Points");
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public static void main(String[] args) {
```

```
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                PointsExample ex = new PointsExample();
                ex.setVisible(true);
            }
        });
    }
}
```

One point is difficult to observe. Therefore, we will randomly draw 1000 points on the panel surface.

```
class DrawPanel extends JPanel {
```

We are drawing on a custom drawing panel, which is a `JPanel` component. The drawing panel will later be added to a `JFrame` component.

```
@Override
public void paintComponent(Graphics g) {

    super.paintComponent(g);
    doDrawing(g);
}
```

Custom painting is performed inside the `paintComponent()` method, which we override. The `super.paintComponent()` method calls the method of the parent class. It does some necessary work to prepare component for drawing. Actual drawing is delegated to the `doDrawing()` method.

```
Graphics2D g2d = (Graphics2D) g;
```

Painting in Swing is done on the `Graphics2D` object.

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

We will paint our points in blue colour.

```
Dimension size = getSize();
Insets insets = getInsets();
```

The size of the window includes borders and titlebar. We do not paint there.

```
int w =  size.width - insets.left - insets.right;
int h =  size.height - insets.top - insets.bottom;
```

Here we calculate the area, where we will effectively paint our points.

```
Random r = new Random();
int x = Math.abs(r.nextInt()) % w;
int y = Math.abs(r.nextInt()) % h;
```

We get a random number in range of the size of area that we computed above.

```
g2d.drawLine(x, y, x, y);
```

Here we draw the point. As we already said, we use a `drawLine()` method. We specify the same point twice.

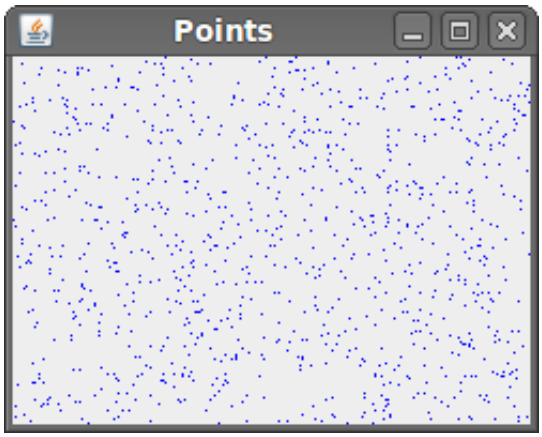

Figure: Points

Lines

A line is a simple graphics primitive. It is drawn using two points.

```
package com.zetcode;

import java.awt.BasicStroke;
import java.awt.Graphics;
import java.awt.Graphics2D;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.SwingUtilities;

class DrawPanel extends JPanel {

    private void doDrawing(Graphics g) {

        Graphics2D g2d = (Graphics2D) g;

        float[] dash1 = {2f, 0f, 2f};
```

```
        float[] dash2 = {1f, 1f, 1f};
        float[] dash3 = {4f, 0f, 2f};
        float[] dash4 = {4f, 4f, 1f};

        g2d.drawLine(20, 40, 250, 40);

        BasicStroke bs1 = new BasicStroke(1, BasicStroke.CAP_BUTT,
                BasicStroke.JOIN_ROUND, 1.0f, dash1, 2f);

        BasicStroke bs2 = new BasicStroke(1, BasicStroke.CAP_BUTT,
                BasicStroke.JOIN_ROUND, 1.0f, dash2, 2f);

        BasicStroke bs3 = new BasicStroke(1, BasicStroke.CAP_BUTT,
                BasicStroke.JOIN_ROUND, 1.0f, dash3, 2f);

        BasicStroke bs4 = new BasicStroke(1, BasicStroke.CAP_BUTT,
                BasicStroke.JOIN_ROUND, 1.0f, dash4, 2f);

        g2d.setStroke(bs1);
        g2d.drawLine(20, 80, 250, 80);

        g2d.setStroke(bs2);
        g2d.drawLine(20, 120, 250, 120);

        g2d.setStroke(bs3);
        g2d.drawLine(20, 160, 250, 160);

        g2d.setStroke(bs4);
        g2d.drawLine(20, 200, 250, 200);
    }

    @Override
    public void paintComponent(Graphics g) {

        super.paintComponent(g);
        doDrawing(g);
    }
}

public class LinesExample extends JFrame {

    public LinesExample() {
        initUI();
    }
```

```
    public final void initUI() {

        DrawPanel dpnl = new DrawPanel();
        add(dpnl);

        setSize(280, 270);
        setTitle("Lines");
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public static void main(String[] args) {

        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                LinesExample ex = new LinesExample();
                ex.setVisible(true);
            }
        });
    }
}
```

In the example, we draw five lines. The first line is drawn using the default values. Other will have a different *stroke*. The stroke is created using the `BasicStroke` class. It defines a basic set of rendering attributes for the outlines of graphics primitives.

```
float[] dash1 = { 2f, 0f, 2f };
```

Here we create a dash that we use in the stroke object.

```
BasicStroke bs1 = new BasicStroke(1, BasicStroke.CAP_BUTT,
    BasicStroke.JOIN_ROUND, 1.0f, dash1, 2f )
```

This code creates a stroke. The stroke defines the line width, end caps, line joins, miter limit, dash and the dash phase.

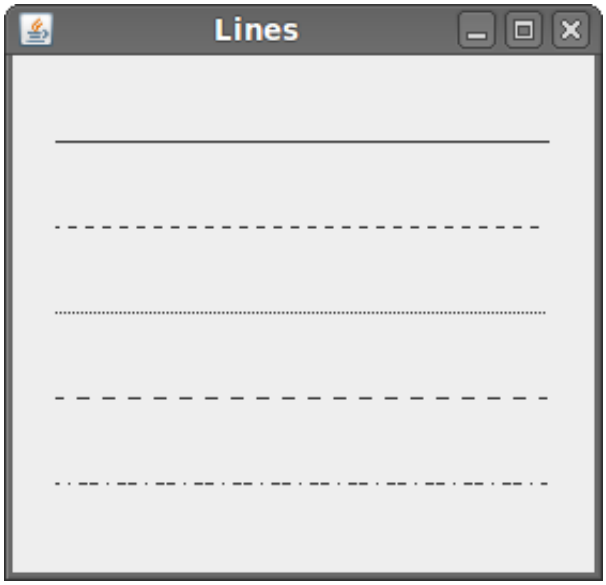

Figure: Lines

Rectangles

To draw rectangles, we use the `drawRect()` method. To fill rectangles with the current color, we use the `fillRect()` method.

```
package com.zetcode;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.SwingUtilities;

class DrawPanel extends JPanel {

    private void doDrawing(Graphics g) {

        Graphics2D g2d = (Graphics2D) g;

        g2d.setColor(new Color(212, 212, 212));
        g2d.drawRect(10, 15, 90, 60);
        g2d.drawRect(130, 15, 90, 60);
        g2d.drawRect(250, 15, 90, 60);
        g2d.drawRect(10, 105, 90, 60);
        g2d.drawRect(130, 105, 90, 60);
        g2d.drawRect(250, 105, 90, 60);
        g2d.drawRect(10, 195, 90, 60);
        g2d.drawRect(130, 195, 90, 60);
        g2d.drawRect(250, 195, 90, 60);
```

```
        g2d.setColor(new Color(125, 167, 116));
        g2d.fillRect(10, 15, 90, 60);

        g2d.setColor(new Color(42, 179, 231));
        g2d.fillRect(130, 15, 90, 60);

        g2d.setColor(new Color(70, 67, 123));
        g2d.fillRect(250, 15, 90, 60);

        g2d.setColor(new Color(130, 100, 84));
        g2d.fillRect(10, 105, 90, 60);

        g2d.setColor(new Color(252, 211, 61));
        g2d.fillRect(130, 105, 90, 60);

        g2d.setColor(new Color(241, 98, 69));
        g2d.fillRect(250, 105, 90, 60);

        g2d.setColor(new Color(217, 146, 54));
        g2d.fillRect(10, 195, 90, 60);

        g2d.setColor(new Color(63, 121, 186));
        g2d.fillRect(130, 195, 90, 60);

        g2d.setColor(new Color(31, 21, 1));
        g2d.fillRect(250, 195, 90, 60);
    }

    @Override
    public void paintComponent(Graphics g) {

        super.paintComponent(g);
        doDrawing(g);
    }
}

public class RectanglesExample extends JFrame {

    public RectanglesExample() {
        initUI();
    }

    public final void initUI() {
```

```
        DrawPanel dpnl = new DrawPanel();
        add(dpnl);

        setSize(360, 300);
        setTitle("Rectangles");
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public static void main(String[] args) {

        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                RectanglesExample ex = new RectanglesExample();
                ex.setVisible(true);
            }
        });
    }
}
```

In the example we draw nine coloured rectangles.

```
g2d.setColor(new Color(212, 212, 212));
g2d.drawRect(10, 15, 90, 60);
...
```

We set the colour of the outline of the rectangle to a soft gray colour, so that it does not interfere with the fill colour. To draw the outline of the rectangle, we use the `drawRect()` method. The first two parameters are the x and y values. The third and fourth are width and height.

```
g2d.fillRect(10, 15, 90, 60);
```

To fill the rectangle with a colour, we use the `fillRect()` method.

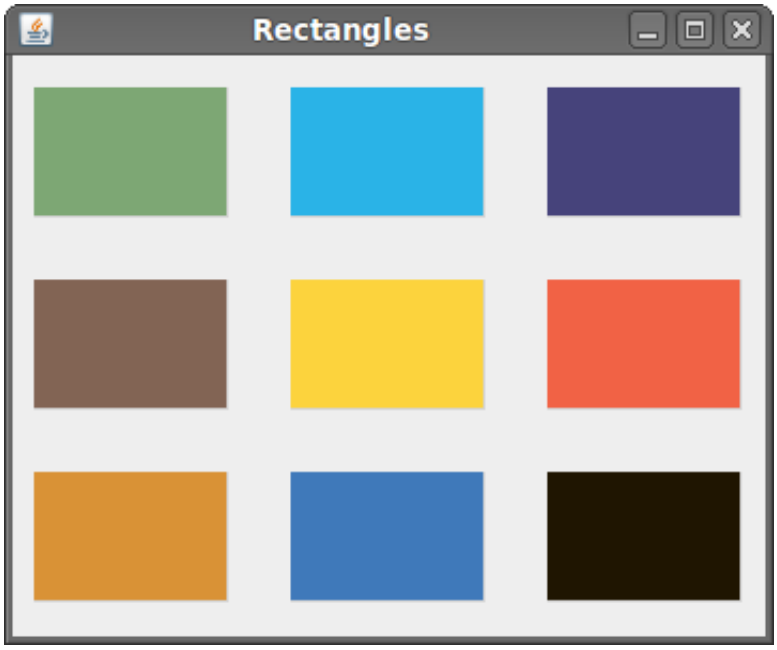

Figure: Rectangles

Textures

A texture is a bitmap image applied to a shape. To work with textures in Java 2D, we use the `TexturePaint` class.

```
package com.zetcode;

import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Rectangle;
import java.awt.TexturePaint;
import java.awt.image.BufferedImage;

import javax.swing.JFrame;
import javax.swing.JPanel;

import java.io.IOException;
import java.util.logging.Logger;
import java.util.logging.Level;
import javax.imageio.ImageIO;
import javax.swing.SwingUtilities;

class DrawPanel extends JPanel {

    BufferedImage slate;
    BufferedImage java;
    BufferedImage pane;
    TexturePaint slatetp;
    TexturePaint javatp;
```

```
    TexturePaint panetp;

    public DrawPanel() {
        loadImages();
    }

    private void loadImages() {

        try {

            slate = ImageIO.read(this.getClass().getResource("slate.png"));
            java = ImageIO.read(this.getClass().getResource("java.png"));
            pane = ImageIO.read(this.getClass().getResource("pane.png"));

        } catch (IOException ex) {

            Logger.getLogger(Textures.class.getName()).log(Level.SEVERE,
                    null, ex);
        }
    }

    @Override
    public void paintComponent(Graphics g) {
        super.paintComponent(g);

        doDrawing(g);
    }

    private void doDrawing(Graphics g) {
        Graphics2D g2d = (Graphics2D) g;

        slatetp = new TexturePaint(slate, new Rectangle(0, 0, 90, 60));
        javatp = new TexturePaint(java, new Rectangle(0, 0, 90, 60));
        panetp = new TexturePaint(pane, new Rectangle(0, 0, 90, 60));

        g2d.setPaint(slatetp);
        g2d.fillRect(10, 15, 90, 60);

        g2d.setPaint(javatp);
        g2d.fillRect(130, 15, 90, 60);

        g2d.setPaint(panetp);
        g2d.fillRect(250, 15, 90, 60);
    }
}
```

```
public class Textures extends JFrame {

    public Textures() {
        initUI();
    }

    public final void initUI() {

        DrawPanel dpnl = new DrawPanel();
        add(dpnl);

        setSize(360, 120);
        setTitle("Textures");
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public static void main(String[] args) {

        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                Textures ex = new Textures();
                ex.setVisible(true);
            }
        });
    }
}
```

In the code example, we fill three rectangles with three different textures.

```
slate = ImageIO.read(this.getClass().getResource("slate.png"));
```

Here we read the image into the buffered image.

```
slatetp = new TexturePaint(slate, new Rectangle(0, 0, 90, 60));
```

We create a `TexturePaint` class out of the buffered image.

```
g2d.setPaint(slatetp);
g2d.fillRect(10, 15, 90, 60);
```

We fill a rectangle with a texture.

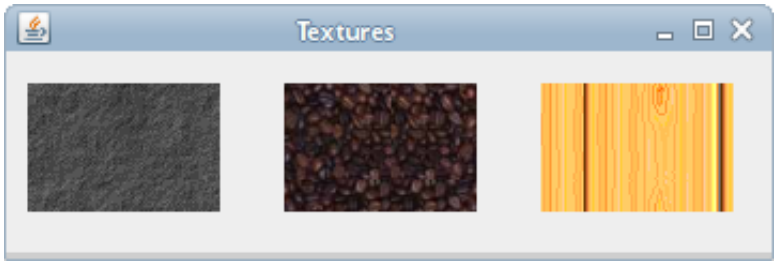

Figure: Textures

Gradients

In computer graphics, gradient is a smooth blending of shades from light to dark or from one colour to another. In 2D drawing programs and paint programs, gradients are used to create colourful backgrounds and special effects as well as to simulate lights and shadows. (answers.com)

```
package com.zetcode;

import java.awt.Color;
import java.awt.GradientPaint;
import java.awt.Graphics;
import java.awt.Graphics2D;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.SwingUtilities;

class DrawPanel extends JPanel {

    private void doDrawing(Graphics g) {

        Graphics2D g2d = (Graphics2D) g;

        GradientPaint gp1 = new GradientPaint(5, 5,
                Color.red, 20, 20, Color.black, true);

        g2d.setPaint(gp1);
        g2d.fillRect(20, 20, 300, 40);

        GradientPaint gp2 = new GradientPaint(5, 25,
                Color.yellow, 20, 2, Color.black, true);

        g2d.setPaint(gp2);
        g2d.fillRect(20, 80, 300, 40);

        GradientPaint gp3 = new GradientPaint(5, 25,
                Color.green, 2, 2, Color.black, true);
```

```
        g2d.setPaint(gp3);
        g2d.fillRect(20, 140, 300, 40);

        GradientPaint gp4 = new GradientPaint(25, 25,
                Color.blue, 15, 25, Color.black, true);

        g2d.setPaint(gp4);
        g2d.fillRect(20, 200, 300, 40);

        GradientPaint gp5 = new GradientPaint(0, 0,
                Color.orange, 0, 20, Color.black, true);

        g2d.setPaint(gp5);
        g2d.fillRect(20, 260, 300, 40);
    }

    @Override
    public void paintComponent(Graphics g) {
        super.paintComponent(g);

        doDrawing(g);
    }
}

public class GradientsExample extends JFrame {

    public GradientsExample() {
        initUI();
    }

    public final void initUI() {

        DrawPanel dpnl = new DrawPanel();
        add(dpnl);

        setSize(350, 350);
        setTitle("Gradients");
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public static void main(String[] args) {

        SwingUtilities.invokeLater(new Runnable() {
            @Override
```

```
            public void run() {
                GradientsExample ex = new GradientsExample();
                ex.setVisible(true);
            }
        });
    }
}
```

Our code example presents five rectangles with gradients.

```
GradientPaint gp4 = new GradientPaint(25, 25,
    Color.blue, 15, 25, Color.black, true);
```

To work with gradients, we use Java Swing's `GradientPaint` class. By manipulating the colour values and the starting end ending points, we can get different results.

```
g2d.setPaint(gp5);
```

The gradient is activated calling the `setPaint()` method.

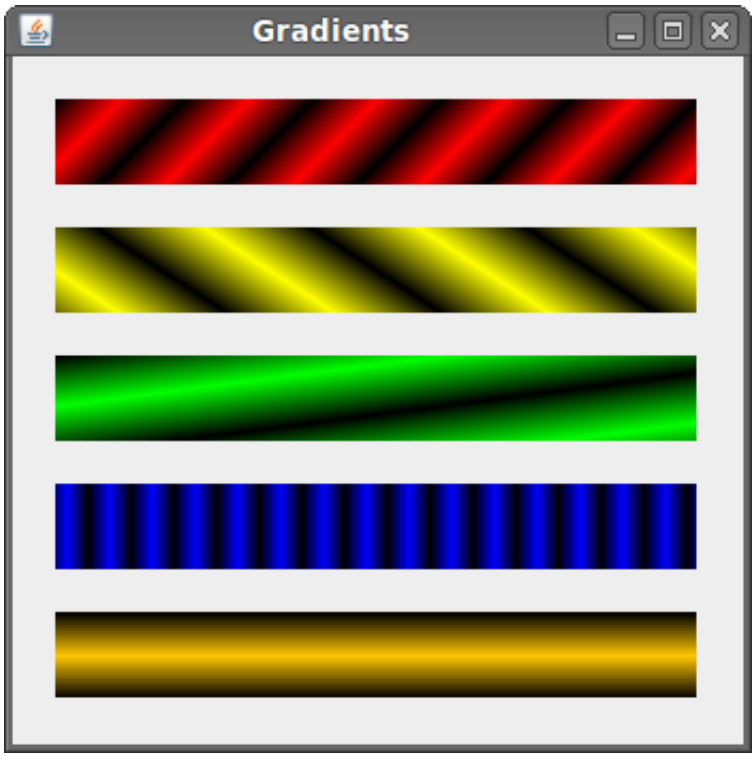

Figure: Gradients

Drawing text

Drawing is done with the `drawString()` method. We specify the string we want to draw and the position of the text on the window area.

```
package com.zetcode;
```

```
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.SwingUtilities;

class DrawPanel extends JPanel {

    private void doDrawing(Graphics g) {

        Graphics2D g2d = (Graphics2D) g;

        RenderingHints rh = new RenderingHints(
                RenderingHints.KEY_ANTIALIASING,
                RenderingHints.VALUE_ANTIALIAS_ON);

        rh.put(RenderingHints.KEY_RENDERING,
                RenderingHints.VALUE_RENDER_QUALITY);

        g2d.setRenderingHints(rh);

        Font font = new Font("URW Chancery L", Font.BOLD, 21);
        g2d.setFont(font);

        g2d.drawString("Not marble, nor the gilded monuments", 20, 30);
        g2d.drawString("Of princes, shall outlive this powerful rhyme;", 20,
60);
        g2d.drawString("But you shall shine more bright in these contents",
                20, 90);
        g2d.drawString("Than unswept stone, besmear'd with sluttish time.",
                20, 120);
        g2d.drawString("When wasteful war shall statues overturn,", 20,
150);
        g2d.drawString("And broils root out the work of masonry,", 20, 180);
        g2d.drawString("Nor Mars his sword, nor war's quick "
                + "fire shall burn", 20, 210);
        g2d.drawString("The living record of your memory.", 20, 240);
        g2d.drawString("'Gainst death, and all oblivious enmity", 20, 270);
        g2d.drawString("Shall you pace forth; your praise shall still "
                + "find room", 20, 300);
        g2d.drawString("Even in the eyes of all posterity", 20, 330);
        g2d.drawString("That wear this world out to the ending doom.", 20,
```

```
360);
        g2d.drawString("So, till the judgment that yourself arise,", 20,
390);
        g2d.drawString("You live in this, and dwell in lovers' eyes.", 20,
420);
    }

    @Override
    public void paintComponent(Graphics g) {

        super.paintComponent(g);
        doDrawing(g);
    }
}

public class DrawingText extends JFrame {

    public DrawingText() {
        initUI();
    }

    public final void initUI() {

        DrawPanel dpnl = new DrawPanel();
        add(dpnl);

        setSize(500, 470);
        setTitle("Sonnet55");
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public static void main(String[] args) {

        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                DrawingText ex = new DrawingText();
                ex.setVisible(true);
            }
        });
    }
}
```

In our example, we draw a sonnet on the panel component.

```
RenderingHints rh = new RenderingHints(
    RenderingHints.KEY_ANTIALIASING,
    RenderingHints.VALUE_ANTIALIAS_ON);

rh.put(RenderingHints.KEY_RENDERING,
    RenderingHints.VALUE_RENDER_QUALITY);

g2d.setRenderingHints(rh);
```

This code is to make our text look better. We apply a technique called *antialiasing*.

```
Font font = new Font("URW Chancery L", Font.BOLD, 21);
g2d.setFont(font);
```

We choose a specific font for our text.

```
g2d.drawString("Not marble, nor the gilded monuments", 20, 30);
```

This is the code that draws the text.

Images

On of the most important capabililies of a toolkit is the ability to display images. An image is an array of pixels. Each pixel represents a colour at a given position. We can use components like `JLabel` to display an image, or we can draw it using the *Java 2D API*.

```
package com.zetcode;

import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Image;
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.SwingUtilities;

class DrawPanel extends JPanel {

    Image img;

    public DrawPanel() {

        loadImage();
```

```
        Dimension dm = new Dimension(img.getWidth(null),
img.getHeight(null));
        setPreferredSize(dm);
    }

    private void loadImage() {
        img = new ImageIcon("slanec.png").getImage();
    }

    private void doDrawing(Graphics g) {

        Graphics2D g2d = (Graphics2D) g;

        g2d.drawImage(img, 0, 0, null);
    }

    @Override
    public void paintComponent(Graphics g) {

        super.paintComponent(g);
        doDrawing(g);
    }
}

public class ImageExample extends JFrame {

    public ImageExample() {
        initUI();
    }

    public final void initUI() {

        DrawPanel dpnl = new DrawPanel();
        add(dpnl);

        setTitle("Image");
        pack();
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public static void main(String[] args) {

        SwingUtilities.invokeLater(new Runnable() {
            @Override
```

```
            public void run() {
                ImageExample ex = new ImageExample();
                ex.setVisible(true);
            }
        });
    }
}
```

This example will draw an image on the panel. The image will fit the `JFrame` window.

```
public DrawPanel() {

    loadImage();
    Dimension dm = new Dimension(img.getWidth(null), img.getHeight(null));
    setPreferredSize(dm);
}
```

In the constructor of the `DrawPanel` class, we call the `loadImage()` method. We determine the image dimensions and set the preffered size of the panel component. This will together with the `pack()` method display the image to fit exactly the window.

```
private void loadImage() {
    img = new ImageIcon("slanec.png").getImage();
}
```

The method loads an image from the disk. We use the `ImageIcon` class. This class simplyfies the work with the images in Java Swing.

```
g2d.drawImage(this.img, 0, 0, null);
```

The image is drawn using the `drawImage()` method.

In this chapter, we did some painting.