BASIC GRAPHICS DRAWING TOOL

A PROJECT REPORT

Submitted by

PRASHANTH REDDY C

NOVEMBER - 2024

ABSTRACT

The Basic Graphics Drawing Tool is a Java-based GUI application designed to enable users to create and manipulate graphical elements like shapes, lines, and text on a canvas. Built using Java's Swing and AWT libraries, the application provides an interactive interface for drawing circles, rectangles, lines, and custom text. Users can choose colors, clear the canvas, and dynamically adjust the shape's appearance. This project serves as an excellent introduction to graphics programming, event handling, and GUI design in Java..

INTRODUCTION

Drawing tools are fundamental applications in creative and technical fields, enabling users to visually represent ideas and designs. This project focuses on implementing a simple yet powerful graphics application using Java. By leveraging the AWT and Swing libraries, the application offers functionalities to draw shapes, handle user interactions, and customize graphical elements. It demonstrates key concepts of graphics programming, including canvas rendering, shape drawing, and event-driven programming, making it a valuable educational tool.

OBJECTIVE

The main objectives of this project are:

- To create a user-friendly interface for drawing shapes and text.
- To implement core graphics functionalities such as shape drawing, color selection, and canvas clearing.
- To demonstrate the use of Java's AWT and Swing libraries for developing interactive applications.

LITERATURE REVIEW

Graphics programming in Java is typically done using the Abstract Window Toolkit (AWT) and Swing libraries. AWT provides low-level access to the system's graphical resources, while Swing builds upon AWT to offer more advanced components. Previous implementations of graphics tools in Java highlight the importance of proper event handling, component rendering, and user interaction design. However, many basic implementations lack dynamic shape manipulation and a user-friendly interface. This project addresses these gaps by introducing a clean UI and comprehensive functionality.

SYSTEM DESIGN AND IMPLEMENTATION

Architecture

The application consists of:

1. Canvas Component: A drawing area where graphical elements are rendered.
2. Toolbar: Buttons and controls for selecting shapes, colors, and clearing the canvas.
3. Event Handling: Listeners for mouse actions and button clicks to handle user inputs.

Components

1. Shapes and Text:
    - Lines, rectangles, circles, and custom text.
2. Color Selection:
    - Dynamic color picker for customizing the drawing color.
3. Canvas Management:
    - Clearing the canvas and resetting states.

CODE

GraphicsDrawingTool.java

```
import javax.swing.*;

public class GraphicsDrawingTool {

    public static void main(String[] args) {
        SwingUtilities.invokeLater(DrawingFrame::new);
    }
}
```

DrawingFrame.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.ArrayList;
class DrawingFrame extends JFrame {
    private final DrawingCanvas canvas;
    private final JComboBox<String> shapeSelector;
    private final JButton colorPicker;
    private final JTextField textInput;
    private Color currentColor = Color.BLACK;

    public DrawingFrame() {
        setTitle("Basic Graphics Drawing Tool");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(800, 600);
        setLayout(new BorderLayout());

        // Canvas
        canvas = new DrawingCanvas();
        add(canvas, BorderLayout.CENTER);

        // Controls Panel
        JPanel controls = new JPanel();
        controls.setLayout(new FlowLayout(FlowLayout.LEFT));

        // Shape Selector
        shapeSelector = new JComboBox<>(new String[]{"Line", "Rectangle", "Circle", "Text"});
        controls.add(new JLabel("Shape:"));
        controls.add(shapeSelector);
```

```
        // Color Picker
        colorPicker = new JButton("Pick Color");
        colorPicker.addActionListener(e -> pickColor());
        controls.add(colorPicker);

        // Text Input
        textInput = new JTextField(15);
        controls.add(new JLabel("Text:"));
        controls.add(textInput);

        // Clear Button
        JButton clearButton = new JButton("Clear Canvas");
        clearButton.addActionListener(e -> canvas.clearCanvas());
        controls.add(clearButton);
        add(controls, BorderLayout.NORTH);
        setVisible(true);
    }

    private void pickColor() {
        currentColor = JColorChooser.showDialog(this, "Pick a Color", currentColor);
        if (currentColor != null) {
            colorPicker.setBackground(currentColor);
        }
    }

    public String getSelectedShape() {
        return (String) shapeSelector.getSelectedItem();
    }

    public Color getCurrentColor() {
        return currentColor;
    }

    public String getInputText() {
        return textInput.getText();
    }

    class DrawingCanvas extends JPanel {
        private final ArrayList<Drawable> drawables = new ArrayList<>();
        private Point startPoint;
        public DrawingCanvas() {
            setBackground(Color.WHITE);
            addMouseListener(new MouseAdapter() {
                @Override
                public void mousePressed(MouseEvent e) {
                    startPoint = e.getPoint();
                }
```

```
            @Override
            public void mouseReleased(MouseEvent e) {
                String shape = getSelectedShape();
                if (shape.equals("Text")) {
                    String text = getInputText();
                    if (!text.isEmpty()) {
                        drawables.add(new DrawableText(startPoint, text, getCurrentColor()));
                    }
                } else {
                    Point endPoint = e.getPoint();
                    switch (shape) {
                        case "Line" -> drawables.add(new DrawableLine(startPoint, endPoint,
getCurrentColor()));
                        case "Rectangle" -> drawables.add(new DrawableRectangle(startPoint, endPoint,
getCurrentColor()));
                        case "Circle" -> drawables.add(new DrawableCircle(startPoint, endPoint,
getCurrentColor()));
                    }
                }
                repaint();
            }
        });
    }
    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        for (Drawable drawable : drawables) {
            drawable.draw(g);
        }
    }

    public void clearCanvas() {
        drawables.clear();
        repaint();
    }
  }
}
```

Drawable.java

```
import java.awt.*;

interface Drawable {
  void draw(Graphics g);
}
```

DrawableCircle.java

```
import java.awt.*;

class DrawableCircle implements Drawable {
    private final Point startPoint;
    private final Point endPoint;
    private final Color color;
    public DrawableCircle(Point startPoint, Point endPoint, Color color) {
        this.startPoint = startPoint;
        this.endPoint = endPoint;
        this.color = color;
    }
    @Override
    public void draw(Graphics g) {
        g.setColor(color);
        int x = Math.min(startPoint.x, endPoint.x);
        int y = Math.min(startPoint.y, endPoint.y);
        int diameter = Math.max(Math.abs(startPoint.x - endPoint.x), Math.abs(startPoint.y - endPoint.y));
        g.drawOval(x, y, diameter, diameter);
    }
}
```

DrawableLine.java

```
import java.awt.*;

class DrawableLine implements Drawable {
    private final Point startPoint;
    private final Point endPoint;
    private final Color color;
    public DrawableLine(Point startPoint, Point endPoint, Color color) {
        this.startPoint = startPoint;
        this.endPoint = endPoint;
        this.color = color;
    }

    @Override
    public void draw(Graphics g) {
        g.setColor(color);
        g.drawLine(startPoint.x, startPoint.y, endPoint.x, endPoint.y);
    }
}
```

DrawableRectangle.java

```
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Point;

class DrawableRectangle implements Drawable {
  private final Point startPoint;
  private final Point endPoint;
  private final Color color;
  public DrawableRectangle(Point var1, Point var2, Color var3) {
    this.startPoint = var1;
    this.endPoint = var2;
    this.color = var3;
  }

  public void draw(Graphics var1) {
    var1.setColor(this.color);
    int var2 = Math.min(this.startPoint.x, this.endPoint.x);
    int var3 = Math.min(this.startPoint.y, this.endPoint.y);
    int var4 = Math.abs(this.startPoint.x - this.endPoint.x);
    int var5 = Math.abs(this.startPoint.y - this.endPoint.y);
    var1.drawRect(var2, var3, var4, var5);
  }
}
```

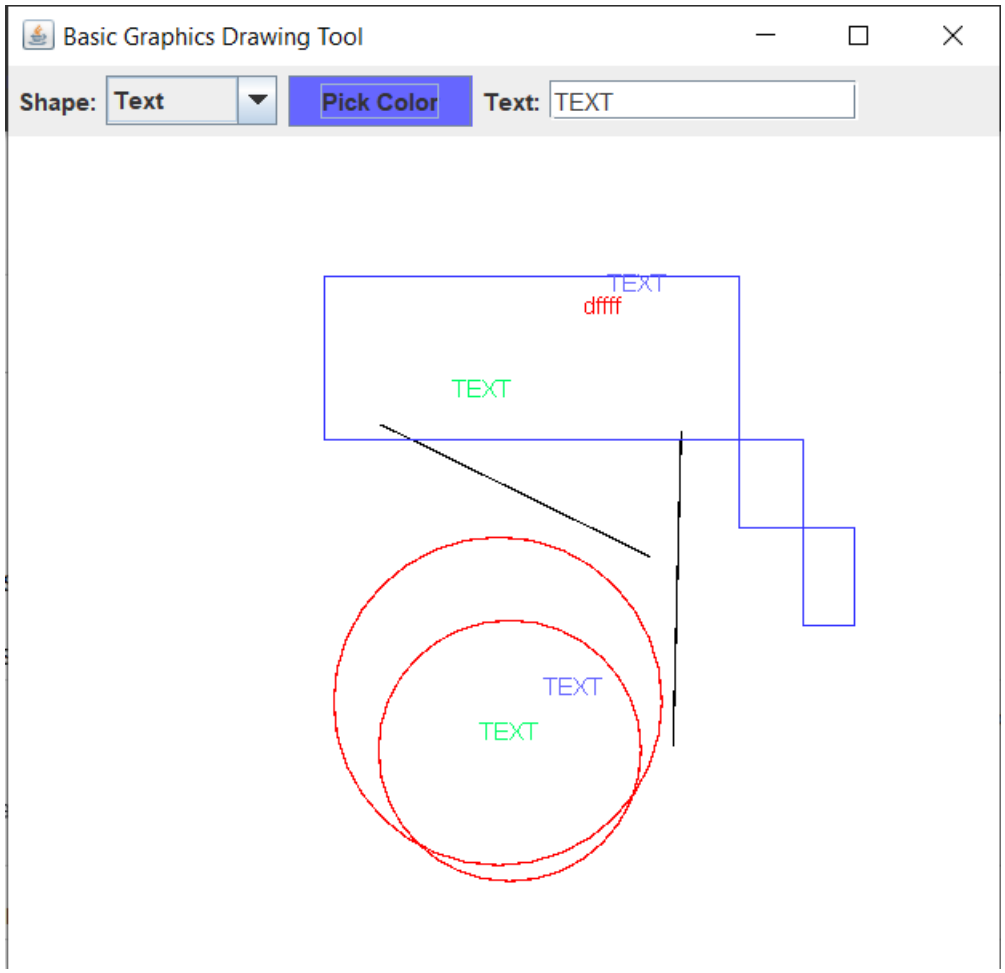
DrawableText.java

```
import java.awt.*;

class DrawableText implements Drawable {
   private final Point position;
   private final String text;
   private final Color color;
   public DrawableText(Point position, String text, Color color) {
      this.position = position;
      this.text = text;
      this.color = color;
   }
   @Override
   public void draw(Graphics g) {
      g.setColor(color);
      g.drawString(text, position.x, position.y);
   }
}
```

RESULTS

Screenshots:



CONCLUSION

The Basic Graphics Drawing Tool successfully demonstrates the use of Java's AWT and Swing libraries for creating interactive graphical applications. By incorporating a clean UI and essential drawing functionalities, the application serves as both a creative tool and an educational resource for learning graphics programming.

REFERENCES

[1] Oracle Documentation on Java.

[2] Effective Java, 3rd Edition by Joshua Bloch

[3] Java: The Complete Reference by Herbert Schildt