Interactive Drawing Tool

A Java-based Drawing Application

AP Computer Science A Project

May 30, 2025

Demo

Project Overview

- A Java Swing-based drawing application
- Provides multiple drawing tools (line, rectangle, circle, text, free drawing)
- Supports layer management for complex drawings
- Includes selection and manipulation of shapes
- Features undo/redo functionality
- Allows saving and loading drawings
- Implements zoom and pan capabilities

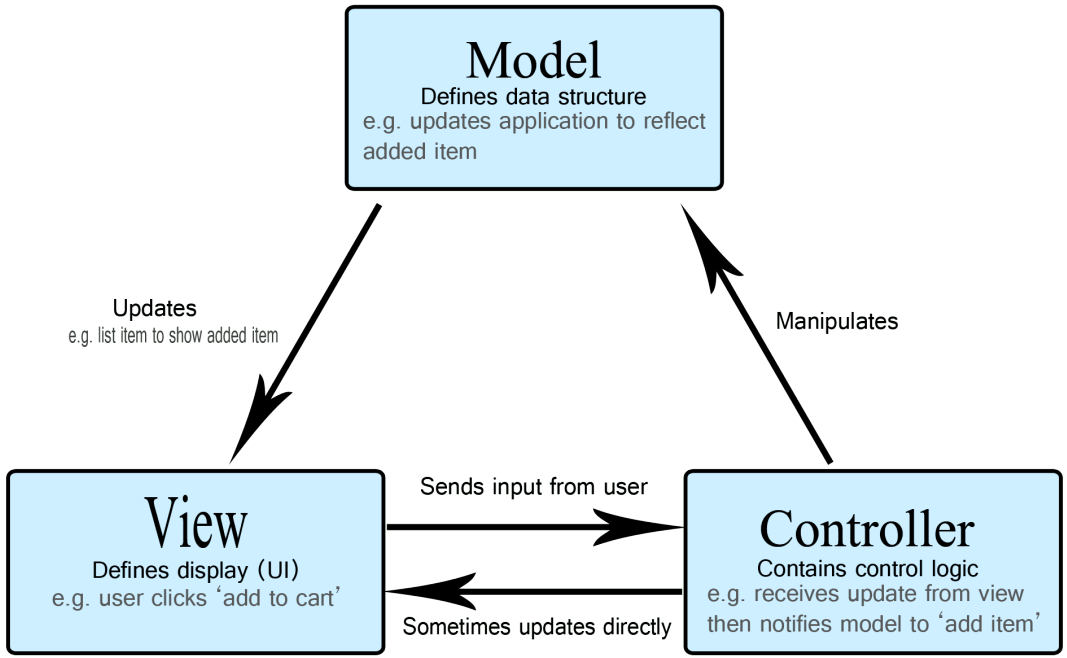
Design Principles

Object-Oriented Design

- Inheritance hierarchy for shapes
- Encapsulation of functionality
- Polymorphism for drawing operations

Design Patterns

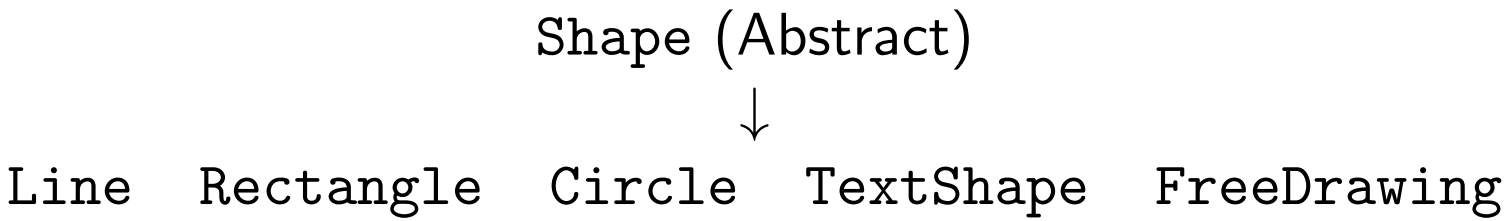
- Model-View-Controller pattern
- Command pattern for undo/redo
- Observer pattern for UI updates
- Strategy pattern for drawing tools



Architecture Overview

Component	**Purpose**
DrawingApp	Main application window and UI
DrawingPanel	Canvas for drawing and interaction
Shape	Abstract base class for all shapes
Layer	Container for organizing shapes
LayerPanel	UI for managing layers

Class Hierarchy



- `Shape`: Common properties and behaviors
- Specific shape classes: Implement drawing logic
- `Layer`: Contains and manages multiple shapes
- `DrawingPanel`: Manages layers and user interaction

Core Component: Shape

- Abstract base class for all drawable elements
- Defines common properties:
  - Coordinates (x1, y1, x2, y2)
  - Color and stroke width
  - Fill state (filled or outline)
  - Selection state
- Provides methods for:
  - Selection and manipulation
  - Hit detection
  - Drawing selection handles

```
public abstract class Shape implements Serializable {
    protected Color color;
    protected int x1, y1, x2, y2;
    protected boolean filled;
    protected boolean selected;

    public abstract void draw(Graphics g);
    public boolean containsPoint(int x, int y) { ... }
```

Core Component: Layer

- Container for organizing shapes
- Manages visibility and z-order
- Provides shape selection functionality
- Enables layer-specific operations

```
public class Layer implements Serializable {
    private String name;
    private ArrayList<Shape> shapes;
    private boolean visible;
    private boolean selected;

    public void draw(Graphics g) {
        if (visible) {
            for (Shape shape : shapes) {
                shape.draw(g);
            }
        }
    }
```

Core Component: DrawingPanel

- Central canvas where drawing happens
- Manages mouse and keyboard interactions
- Creates appropriate shapes based on selected tool
- Implements undo/redo functionality
- Handles selection, moving, and resizing
- Provides zoom and pan capabilities

```
public class DrawingPanel extends JPanel {
    private ArrayList<Layer> layers;
    private Layer currentLayer;
    private Stack<ArrayList<Layer>> undoStack;
    private Stack<ArrayList<Layer>> redoStack;
    private String currentShape;
    private Shape currentDrawing;
    // Mouse listeners, keyboard listeners, etc.
}
```

Core Component: DrawingApp

- Main application window
- Creates all UI components:
  - Menus and toolbars
  - Drawing panel (canvas)
  - Properties panel
  - Layer panel
- Handles file operations
- Connects user actions to drawing functionality

```
public class DrawingApp extends JFrame {
    private DrawingPanel drawingPanel;
    private JTextField textField;
    private JMenuBar menuBar;
    private JToolBar toolBar;
    private JToolBar propertiesBar;
    // UI creation methods, event handlers, etc.
}
```

Drawing Process

1. User selects a drawing tool (e.g., Rectangle)
2. User presses mouse button on canvas
   - `DrawingPanel` creates new shape instance
   - Initial coordinates are set
3. User drags mouse
   - Shape's end coordinates are updated
   - Canvas is repainted to show the shape being drawn
4. User releases mouse button
   - Shape is finalized and added to current layer
   - Current state is saved for undo functionality

Selection and Manipulation

Selection Process

1. User clicks on canvas
2. `DrawingPanel` checks all shapes from top to bottom
3. First shape containing the click point is selected
4. Selection handles are displayed

Manipulation

- **Moving:** Drag selected shape
- **Resizing:** Drag selection handles
- **Deleting:** Press Delete key
- **Reordering:** Layer panel controls

Layer Management

- Layers organize shapes in z-order
- `LayerPanel` provides UI for:
  - Creating new layers
  - Deleting layers
  - Changing visibility
  - Reordering layers
- Drawing process:
  1. `DrawingPanel.paintComponent()` is called
  2. Each visible layer is drawn in order (bottom to top)
  3. Each layer draws its shapes in order

Undo/Redo Implementation

- Uses command pattern with state snapshots
- Each significant action:
  1. Creates a deep copy of all layers
  2. Pushes copy to undo stack
  3. Clears redo stack
- Undo operation:
  1. Pops state from undo stack
  2. Pushes current state to redo stack
  3. Restores popped state
- Redo operation: Reverses the undo process

File Operations

- **Saving Drawings**
  - Renders all visible layers to a BufferedImage
  - Saves image as PNG file
- **Opening Images**
  - Loads PNG image into BufferedImage
  - Creates new ImageShape containing the image
  - Adds shape to current layer
- Uses Java's ImageIO library for file operations

Special Features

Zoom and Pan

- Zoom with mouse wheel or keyboard
- Pan with middle mouse button
- Coordinate transformation between screen and canvas

Font Selection

- Custom JFontChooser dialog
- Preview of selected font
- Font family, style, and size options

Splash Screen

- Professional loading screen
- Fade-in and fade-out animations
- Progress bar simulation

Code Example: Creating a Shape

```
// In DrawingPanel.mousePressed()
switch (currentShape) {
    case "Circle":
        currentDrawing = new Circle(currentColor,
                                    canvasX, canvasY,
                                    canvasX, canvasY,
                                    filled);
        break;
    case "Rectangle":
        currentDrawing = new Rectangle(currentColor,
                                       canvasX, canvasY,
                                       canvasX, canvasY,
                                       filled);
        break;
    case "Line":
        currentDrawing = new Line(currentColor,
                                  canvasX, canvasY,
```

Code Example: Drawing a Shape

```
// In Rectangle.java
public void draw(Graphics g) {
    Graphics2D g2d = (Graphics2D) g;

    // Save original stroke
    Stroke originalStroke = g2d.getStroke();

    // Set stroke width
    g2d.setStroke(new BasicStroke(strokeWidth));

    // Set the color
    g2d.setColor(color);

    // Calculate actual rectangle coordinates
    int x = Math.min(x1, x2);
    int y = Math.min(y1, y2);
    int width = Math.abs(x2 - x1);
```

Future Enhancements

- **Additional Shape Types**
  - Polygon, Star, Arrow, etc.
- **Enhanced Text Editing**
  - In-place text editing
  - Rich text formatting
- **Advanced Layer Features**
  - Layer groups
  - Layer blend modes
- **Selection Improvements**
  - Multiple selection
  - Grouping shapes
- **File Format Support**
  - Native format to preserve layers and editability
  - Export to SVG, PDF, etc.

Thank You!

Questions?