Comprehensive Documentation for the Java Paint Application

1. Introduction

This document provides a comprehensive technical and architectural overview of the Java-based "Painter" application. The application is a simple graphical editor built using Java Swing, allowing users to draw various shapes, use freehand tools, and manage their creations through save, open, clear, and undo functionalities.

2. Project Architecture and Design

The application follows a clear separation of concerns, loosely adhering to an **MVC (Model-View-Controller)** pattern, which enhances maintainability and scalability.

Component	Package	Description
Application	paint.app	Contains the main entry point (PaintBrushApp) responsible for initializing the Swing JFrame and the primary UI panel.
User Interface (View/Controller)	paint.ui	Manages the graphical components. PaintPanel sets up the toolbar and layout, acting as the main View. DrawingPanel handles all drawing logic, mouse events, and state management, acting as the core Controller and View component.
Model	paint.model	Defines the data structures for the shapes that can be drawn. It includes an abstract base class (ShapeBase) and concrete implementations for each shape type.
Tools	paint.tools	Contains the ToolType enumeration, which defines the available drawing modes.
Exception	paint.exception	Contains a custom exception class (PaintException), although it is not extensively used in the provided code snippet.

3. Project Hierarchy

The project is organized into logical packages, making the codebase easy to navigate and understand.

```
PaintProject/
```

```
└── Paint/
    └── paint/
        ├── app/
        │   └── PaintBrushApp.java  (Main entry point)
        ├── exception/
        │   └── PaintException.java (Custom exception)
        ├── model/
        │   ├── FreeHandShape.java  (Pencil/Eraser implementation)
        │   ├── LineShape.java
        │   ├── OvalShape.java
        │   ├── RectShape.java
        │   └── ShapeBase.java      (Abstract base class for all shapes)
        ├── tools/
        │   └── ToolType.java       (Enum for drawing modes)
        └── ui/
            ├── DrawingPanel.java   (Core drawing canvas and logic)
```

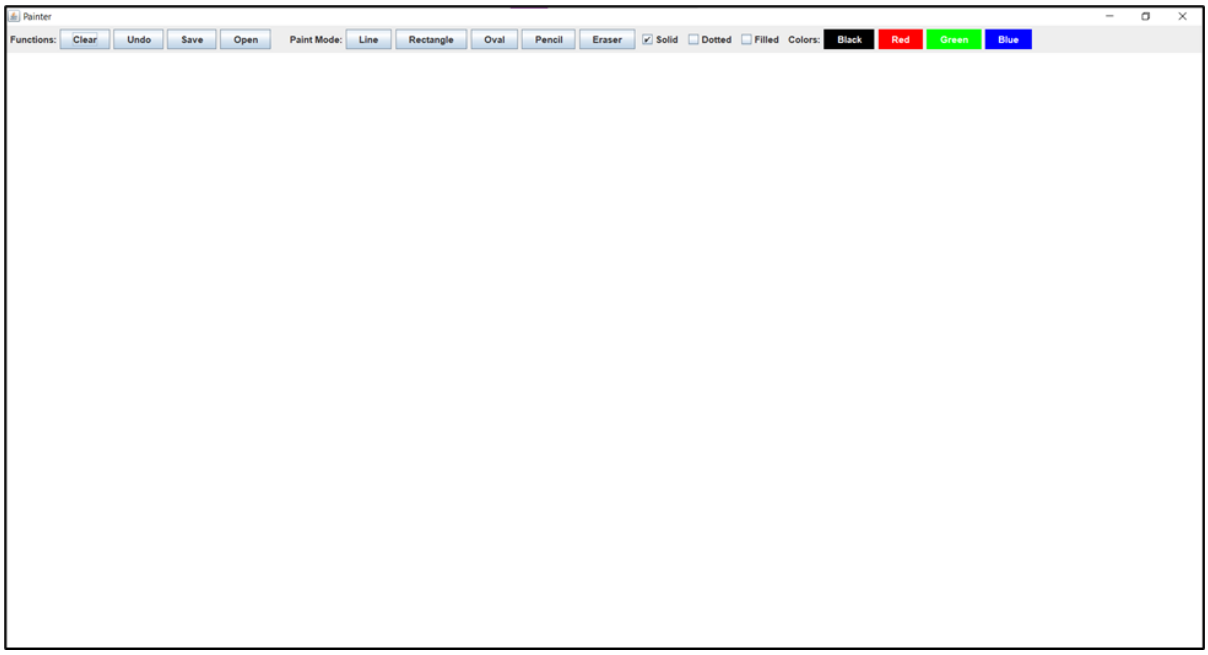
```
            └── PaintPanel.java     (Main panel, sets up toolbar and
layout)
```

4. Graphical User Interface (GUI) Analysis

The application's user interface is simple and functional, consisting of a main drawing area and a toolbar at the top.

GUI Sample

The provided GUI sample illustrates the main components of the application:



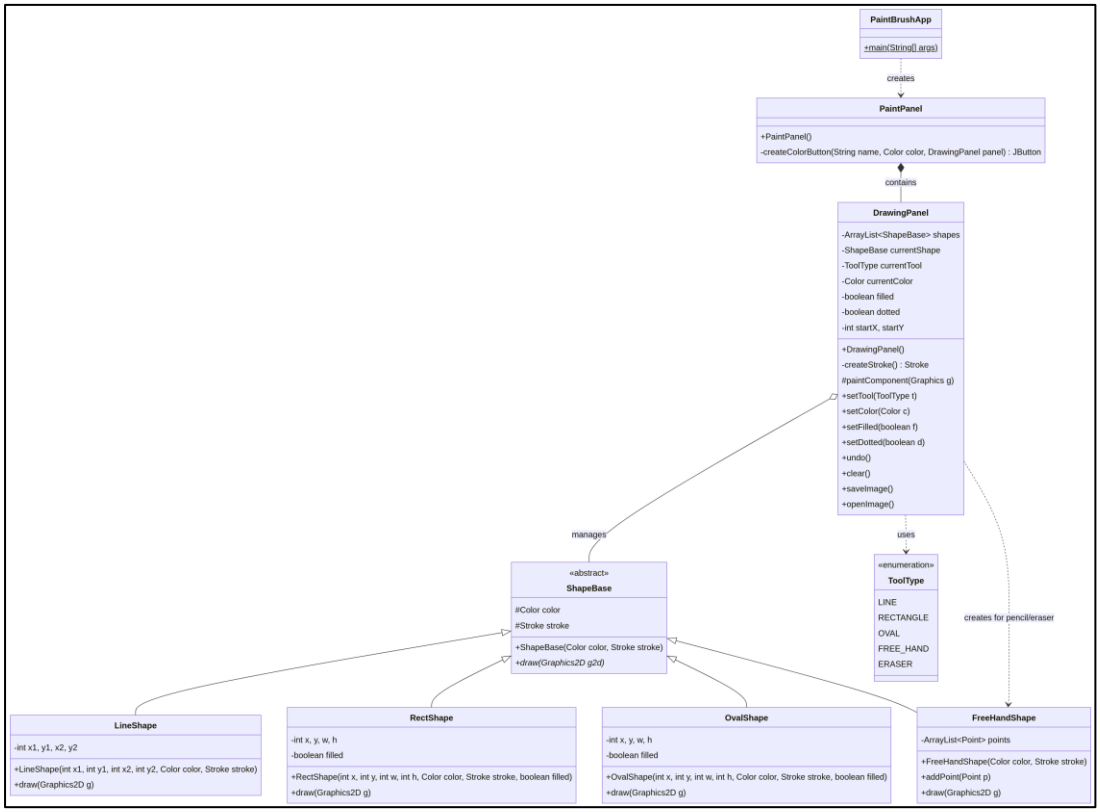
Toolbar Components

The toolbar is logically divided into three main sections:

Section	Components	Functionality
Functions	Clear, Undo, Save, Open	Provides file and state management capabilities. Clear removes all shapes, Undo removes the last drawn shape, and Save/Open handle image persistence using ImageIO.
Paint Mode	Line, Rectangle, Oval, Pencil, Eraser	Allows the user to select the type of drawing tool. This selection is managed by the ToolType enumeration.
Options & Colors	Solid/Dotted/Filled Checkboxes, Color Buttons (Black, Red, Green, Blue)	Controls the appearance of the drawn shapes. The Solid/Dotted options control the Stroke style, and Filled controls whether shapes are drawn as outlines or solid objects.

5. Class Diagram (UML)

The following class diagram illustrates the relationships and structure of the core classes within the application.



6. Detailed Class Descriptions

6.1. Model Classes

The shape model is built around the abstract ShapeBase class, which enforces a common interface for all drawable objects.

Class	Key Attributes	Key Methods	Description
ShapeBase	#color (Color), #stroke (Stroke)	draw(Graphics2D g2d) (abstract)	The abstract base class. It stores the common properties (color and stroke) and defines the contract for drawing.
LineShape	-x1, -y1, -x2, -y2	draw()	Implements drawing a straight line segment.
RectShape	-x, -y, -w, -h, -filled	draw()	Implements drawing a rectangle. It includes logic to handle the filled state and ensures coordinates are correctly calculated regardless of the drag direction.
OvalShape	-x, -y, -w, -h, -filled	draw()	Implements drawing an oval, similar to RectShape in its coordinate handling and filled state.
FreeHandShape	-points (ArrayList<Point>)	addPoint(Point p), draw()	Used for the **Pencil** and **Eraser** tools. It stores a list of points and draws a series of connected line segments between them. The **Eraser** is implemented by setting the color to white.

6.2. UI and Application Classes

Class	Key Attributes	Key Methods	Description
PaintBrushApp	N/A	main(String[] args)	The application's entry point. It initializes the main JFrame and adds the PaintPanel to it, setting the application to full-screen mode.
PaintPanel	N/A	PaintPanel(), createColorButton()	The main container panel. It uses a BorderLayout to place the toolbar (FlowLayout in BorderLayout.NORTH) and the DrawingPanel (BorderLayout.CENTER). It sets up all the buttons and checkboxes and links their actions to methods in DrawingPanel.
DrawingPanel	-shapes (ArrayList<ShapeBase>), -currentTool, -currentColor, -filled, -dotted	mousePressed(), mouseDragged(), mouseReleased(), paintComponent(), undo(), clear(), saveImage(), openImage()	The core drawing canvas. It extends JPanel and uses a MouseAdapter to capture user input. It manages the list of drawn shapes (shapes) and is responsible for: 1. **Shape Creation**: Instantiating a new shape object on mouseReleased (or mousePressed for freehand). 2. **Redrawing**: Overriding paintComponent to iterate through the shapes list and call draw() on each one. 3. **State Management**: Providing setters for the current tool, color, and style options. 4. **File I/O**: Implementing saveImage and openImage using ImageIO and JFileChooser.

7. Key Functionality Details

Drawing and State Management

The DrawingPanel is central to the application's functionality.

1. **Shape Storage**: All completed shapes are stored in an ArrayList<ShapeBase> shapes.
2. **Drawing Process**:
    - When the mouse is pressed, the starting coordinates (startX, startY) are recorded. For **Pencil** and **Eraser** (which are FreeHandShapes), the shape is created immediately and added to the shapes list.
    - When the mouse is dragged, if the current shape is a FreeHandShape, new points are continuously added to it, and repaint() is called to update the canvas in real-time.
    - When the mouse is released, for tools like **Line**, **Rectangle**, and **Oval**, the final shape object is instantiated using the start and end coordinates, and then added to the shapes list.
3. **Redrawing**: The paintComponent(Graphics g) method is overridden to ensure that every shape in the shapes list is redrawn whenever the panel needs to be repainted (e.g., after a new shape is added, or after an undo operation).

Undo and Clear

- **undo()**: This method simply removes the last element from the shapes list (shapes.remove(shapes.size() - 1)) and calls repaint(), effectively removing the most recently drawn object from the canvas.
- **clear()**: This method empties the entire shapes list (shapes.clear()) and calls repaint(), resulting in a blank canvas.

Save and Open

The saveImage() and openImage() methods handle image persistence.

- **saveImage()**: A BufferedImage is created with the panel's current dimensions. The panel's content is drawn onto this image buffer using paint(img.getGraphics()). A JFileChooser is used to prompt the user for a save location, and ImageIO.write() saves the image as a PNG file.
- **openImage()**: A JFileChooser is used to select an image file. ImageIO.read() loads the image into a BufferedImage. The image is then drawn directly onto the canvas using g.drawImage(). Note that this implementation draws the image directly to the graphics context but does not convert it into a list of ShapeBase objects, meaning the opened image itself cannot be undone or manipulated as individual shapes.