Java Mini Project

ImaJedit - A lightweight Java - based Image Editor

Serial Number	Name	Register Number
1	Aathish Sivasubrahmanian	3122215001001
2	Alamelu Kannan	3122215001007
3	Gayathri Venkatesan	3122215001026

Problem Statement

To build a lightweight, cross-platform image editing application capable of performing rudimentary image editing tasks, such as painting colors onto a canvas, applying backdrops, adding text to the image, implementing image and color opacity, converting between image formats etc.

Motivation

Ideally, users should have a simple, lightweight, cross platform image editing application that does not require too much memory or computing power to perform simple operations while still remaining useful and extensible.

In reality, there exists no cross-platform application which allows an end-user to perform basic image editing tasks. Existing software products for image editing are either tied to one platform, or are incredibly complex and unsuitable for basic image editing. Microsoft Paint is restricted to Windows Computers, Preview is restricted to MacOS devices, and GIMP, while being very extensible and powerful, has a steep learning curve and is not suitable for simple image editing.

As a result, users have to rely on cloud based image editors which require an internet connection to function properly. This cannot be guaranteed at all times.

Hence, there exists a need for a cross-platform, simple image editor that is lightweight and robust.

Scope and limitations

By the end of this project, the Java Application will be able to perform the basic image editing tasks outlined in the section containing the List of Proposed Features. These include basic image creation, color painting, with a selection of brushes, and the ability to to fill the background with a color or image.

This project will limit itself to acting as a Raster Image Editor - i.e it will be unable to edit, view or otherwise interact with Vector Images such as SVG Images.

The Image File Format of choice will be restricted to BitMap (.bmp) and Portable Network Graphics (.png).

The application will not acknowledge any kind of pressure sensitivity information (for example) in the event that a drawing tablet is used.

Design of solution

The solution shows the design of a graphical user interface (GUI) for an image editor. The GUI is implemented using the Java Swing library and makes use of object-oriented programming concepts to achieve a modular, reusable and maintainable design.

"MainMenu" class creates a JFrame and sets its layout to be a GridBagLayout. It then creates various UI elements such as buttons, labels and spinners and adds them to the frame using the GridBagConstraints object. These UI elements allow the user to create a new image, specify the width and height of the image, and open an existing image.

"MyCanvas" class provides functionality for displaying and manipulating images. It has two methods for creating a new image canvas: one that takes the width and height of the image as parameters, and another that takes the path of an existing image file as a parameter. The method for creating a new image canvas sets up a JFrame, creates a BufferedImage object and sets it as the content pane of the frame. The method for

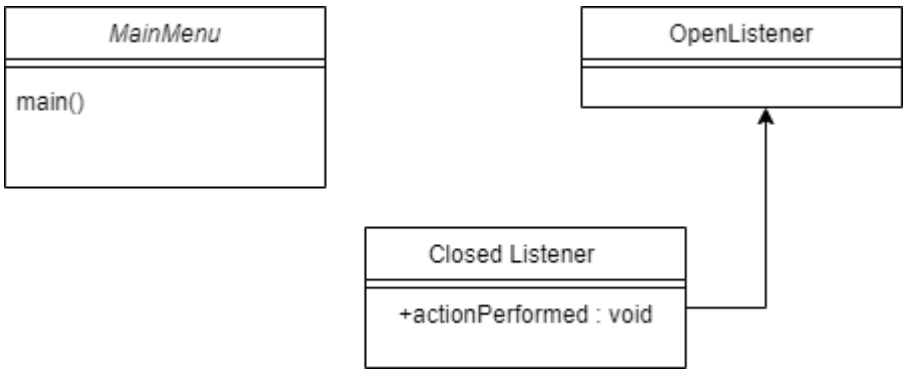
opening an existing image first checks the size of the image and resizes it if it exceeds the maximum allowed size. It then sets up the JFrame and BufferedImage object in the same way as the method for creating a new image canvas.

The "Palette" class extends the JPanel class and overrides its paintComponent method to draw the color palette and the user's drawing on the panel. It also implements the MouseListener interface to handle mouse events such as clicking, dragging and releasing. When the user clicks on the color palette, the changeColor method is called to change the current drawing color. When the user clicks on the drawing area, the setUpDrawingGraphics method is called to set up the graphics context for drawing. As the user drags the mouse, the drawCurve method is called to draw a curve on the panel. The "save" method is called when the user clicks the save button and allows the user to save the image to a file. The "clear" method is called when the user clicks the clear button and clears the drawing from the panel.

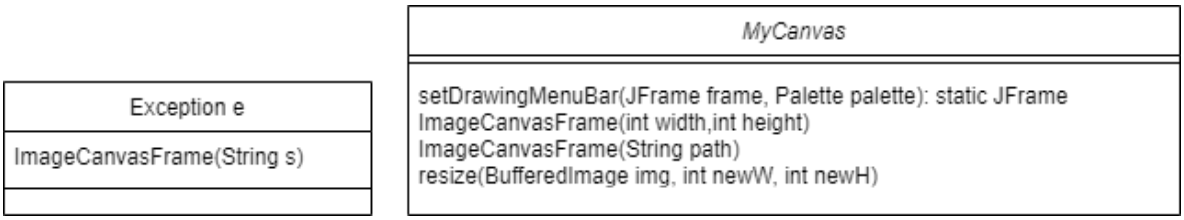
Overall, the design of the solution is well-structured and follows good coding practices such as the use of constants, proper indentation and white space, descriptive and meaningful names, comments, object-oriented programming concepts, and proper error handling. It also makes use of the Swing library to provide a user-friendly GUI for the image editor.

The class diagram is as seen below for each class:

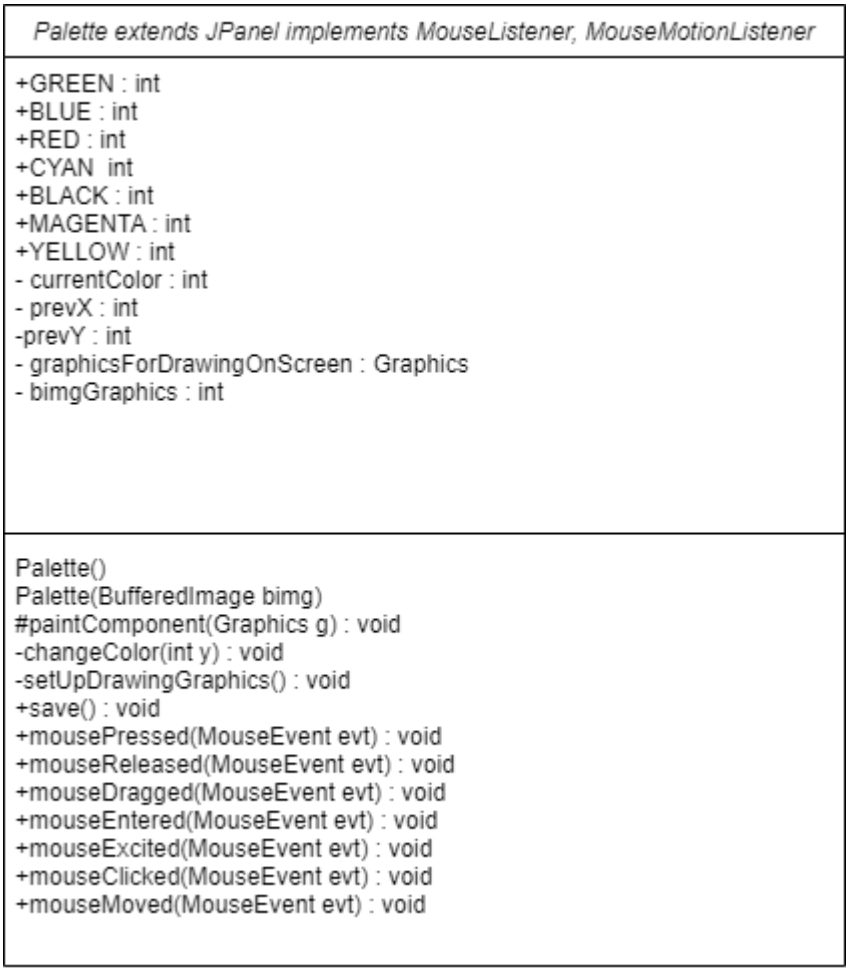
MainMenu:



MyCanvas:



Palette:



Exploring design alternatives

One alternate design for the above codes could be to use the JavaFX library instead of the Swing library for implementing the GUI. The JavaFX library provides newer and more advanced features for creating modern, visually appealing and interactive GUIs. Instead of using the JFrame, JPanel and other Swing components, we can use the Stage, Scene and other JavaFX components. The event handling can be implemented using event handlers and lambda expressions.

Another alternate design for the above codes could be to separate the image manipulation functionality from the GUI code. We can create a separate class or set of classes for handling the creation, loading, saving, and manipulation of images. This can be achieved using the Java Advanced Imaging (JAI) library or other image processing libraries. The GUI code can then use these image manipulation classes to display and manipulate the images. This separation of concerns would make the code more modular and easier to maintain and extend.

Identification of process and modules

This solution for the above problem statement utilizes the following three modules: The MainMenu Module, the MyCanvas Module, and the Palette Module. These three will be explained below:

The MainMenu Module

It creates a graphical user interface (GUI) for an image editing application called "ImaJedit". The GUI is built using the Swing library and the layout is managed using the GridBagLayout manager. The GUI consists of several components such as buttons, labels, spinners (for inputting numerical values), and a file chooser.
The code begins by importing several classes from various Java packages such as java.io, java.awt, and javax.swing. These classes are needed for various purposes such

as reading and writing files, creating and customizing graphical components, and managing the layout of the GUI.

Next, a new JFrame object is created and given the title "ImaJedit". The JFrame class represents a top-level window with a title and a border. The JFrame object is then configured to close the application when the frame is closed by the user. The layout of the JFrame is then set to be a GridBagLayout object, which is a flexible layout manager that aligns components vertically and horizontally in a grid.
A GridBagConstraints object is then created to specify the constraints for adding components to the grid layout. The constraints are set to fill the horizontal space, anchor the components to the center, give the components a weight of 1.0 in both the x and y directions (which determines how much the component expands or shrinks when the frame is resized), and set the x and y positions to (0,0) in the grid. The constraints object is also given a default vertical padding of 20 pixels and no horizontal padding.
Next, a "Create" button is created and added to the JFrame using the constraints object. Then, a "Width" label and a "Height" label are added to the JFrame, followed by two spinners for inputting the width and height values. The spinners are created using SpinnerNumberModel objects, which allow the user to select a numerical value from a list of predefined values. The constraints for each component are modified as needed to specify its position in the grid.

After the spinners, a "File" button and a "Save As" button are added to the JFrame. These buttons are used for opening and saving image files, respectively. The "File" button is accompanied by a JFileChooser object, which allows the user to browse and select a file. The file chooser is configured to only accept image files with certain file extensions using a FileNameExtensionFilter object.
Finally, several ActionListener objects are added to the buttons to specify their behavior when clicked. The "Create" button creates a new image with the specified width and height and opens it in a new window. The "File" button opens the selected image file in a new window. The "Save As" button allows the user to save the current image to a file.
The code then ends with the main method, which simply calls the static main method of the MainMenu class and passes it an array of String arguments. This is the entry point of the application and it simply sets the frame to be visible to the user.

The MyCanvas Module

This code creates a simple image editing application with a graphical user interface (GUI) using the Java programming language. The program allows the user to open an image file, draw on it using a pencil tool, increase or decrease the size of the pencil, change the resolution of the image, and save the edited image.

The main class of this code is MyCanvas, which contains several static methods that are used to create and manage the GUI elements of the program.

The setDrawingMenuBar() method is used to create the menu bar for the application. It takes a JFrame object, which represents the main window of the program, and a Palette object, which is the panel on which the user can draw. The method creates a new JMenuBar object and sets it as the menu bar for the frame. It then creates two JMenu objects, one for the pencil tool and one for the resolution settings. The pencil menu has two JMenuItem objects, one for increasing the size of the pencil and one for decreasing it. The resolution menu has two JMenuItem objects, one for increasing the blur of the image and one for filling the entire canvas with a solid color.

The Blur JMenuItem uses a ConvolveOp object, which is a filter that can be applied to an image, to apply a blur effect to the image. It takes the current image, represented by the bimg field of the Palette object, and creates a new BufferedImage object with the same dimensions. It then applies a kernel, which is a 3x3 matrix of values that define the blur effect, to the new image using the ConvolveOp object. The filtered image is then drawn on the canvas using the graphicsForDrawingOnScreen field of the Palette object.

The FillCanvas JMenuItem uses the setUpDrawingGraphics() method of the Palette object to set up the graphics context for the canvas and fill the entire canvas with a solid color using the fillRect() method of the bimgGraphics field.

The resize() method is used to change the size of an image. It takes a BufferedImage object, representing the image, and two integers representing the new width and height of the image. It creates a new Image object with the specified dimensions and draws it on a new BufferedImage object, which is returned as the result.

The ImageCanvasFrame() method is used to create the main window of the program. It takes two integer parameters representing the width and height of the window. The method creates a new JFrame object and sets the title and layout of the frame. It then creates a new Palette object and adds it to the frame. The method then calls the setDrawingMenuBar() method to create the menu bar and sets the size of the frame to the specified width and height. Finally, the method makes the frame visible to the user.

Overall, this code creates a simple image editing application that allows the user to open an image file, draw on it using a pencil tool, increase or decrease the size of the pencil, change the resolution of the image, and save the edited image. The program uses the basic features of the Java GUI library to create a graphical interface and respond to user input. Additionally, it also makes use of Java's image processing classes to perform some basic image editing operations.

The Palette Module

This is a Java program that creates a simple paint application. The program creates a panel on which the user can draw lines by dragging the mouse. The panel also contains a palette of colors that the user can choose from, as well as a button to save the image.

The code starts by importing several classes from the Java standard library, including the classes for the Abstract Window Toolkit (AWT) and the Swing GUI library. The ImageIO and File classes are also imported to support reading and writing image files.

The main class in the program is called "Palette" and it is a subclass of the JPanel class, which is a basic container for a GUI interface. The class implements two interfaces, MouseListener and MouseMotionListener, which allow it to respond to mouse events such as clicks and drags.

The class contains several fields including the current color that is selected by the user, and some fields that are used when the user is drawing a curve by dragging the mouse.

The class also has a constructor that initializes the panel with white background and adds itself as a listener for mouse events.

The class also has a paintComponent() method which is called automatically when the panel needs to be redrawn. This method first calls the super class's paintComponent() method, to make sure that the panel is properly initialized. Then, it checks if the bimg is not null and draws the bimg on the panel otherwise it sets the background color to white.

The method then draws a 3-pixel gray border around the panel and a 56-pixel wide gray rectangle along the right edge of the panel. The color palette and the "Save" button will be drawn on top of this rectangle.

Then, the method draws the "Save" button as a white rectangle in the lower right corner of the panel, allowing for a 3-pixel border, and writes "SAVE" on top of it.

After that, the method draws the seven color rectangles on the panel, with colors Black, Red, Green, Blue, Cyan, Magenta, and Yellow. These rectangles are located at the right side of the panel, and the user can click on one of them to select a color to draw with.

The class also contains several methods that respond to mouse events, such as mousePressed(), mouseDragged(), and mouseReleased(). These methods are called by the system when the user interacts with the mouse while the program is running.

The mousePressed() method is called when the user presses a mouse button. It sets the previous position of the mouse, and sets a flag to indicate that the user is currently drawing. The mouseDragged() method is called when the user moves the mouse while a button is pressed. This method draws a line between the previous position of the mouse and the current position, using the current color. The mouseReleased() method is called when the user releases the mouse button, and it sets the flag to indicate that the user is no longer drawing.

Overall, this code creates a simple paint application that allows the user to draw lines on a panel, and choose colors from a palette. The program uses the basic features of the Java GUI library to create a graphical interface and respond to user input.

OOPs concepts used

The following Object Oriented Programming Concepts can be observed within the solution:

Inheritance: In the second code, the class "Palette" extends the class "MyCanvas", which means that "Palette" inherits the properties and methods of "MyCanvas".

Polymorphism: The second code has two methods with the same name "ImageCanvasFrame" but with different number and types of arguments. This is an example of method overloading, which is a form of polymorphism.

Encapsulation: The variables and methods in the "MyCanvas" and "Palette" classes are encapsulated, as they are declared private and can only be accessed through the public methods of the class.

Abstraction: The class "MyCanvas" has a method "resize" which takes a "BufferedImage" object and two integers as arguments and returns a "BufferedImage" object. This method does not reveal the implementation details of how the resizing is done, but only provides the user with the necessary abstraction to use it.

Interfaces: The code makes use of several interfaces such as "ActionListener", "MouseListener" and "MouseMotionListener". These interfaces define a set of methods that a class must implement if it wants to implement the interface.

Overriding: The "Palette" class overrides the "paint" method of the "MyCanvas" class. This means that the "paint" method in "Palette" has the same name, return type and arguments as the "paint" method in "MyCanvas", but has a different implementation.

Use of Lambda expressions: In the method setDrawingMenuBar(), lambda expressions are used and this is also an example of functional programming, which is a paradigm of OOP.

Best Practices

The following Best Practices can be observed:

Use of constants: The code makes use of constants such as "CONSTANTS.MAX_IMG_SIZE" which helps to improve the readability of the code and also makes it easier to maintain.

Use of proper indentation and white space: The code uses proper indentation and white space, which helps to improve the readability and understandability of the code.

Use of descriptive and meaningful names: The variables, methods and classes in the code have descriptive and meaningful names, which helps to improve the readability and understandability of the code.

Use of comments: The code uses comments to explain the purpose and functioning of the various parts of the code, which helps to improve the understandability of the code.

Use of object-oriented programming concepts: The code makes use of various object-oriented programming concepts such as inheritance, polymorphism, encapsulation and abstraction, which helps to improve the modularity, reusability and maintainability of the code.

Use of proper error handling: The code uses try-catch blocks to handle exceptions and prevent the program from crashing, which helps to improve the reliability of the code.

Use of proper file handling: The code uses the "JFileChooser" class to handle the saving and loading of image files, which helps to improve the reliability and robustness of the code.
Use of Versioning of Code: Multiple versions of working code were made before the final product.