CLI-Based Image Editor in Java

Program Documentation

Nymish Kashivishwanath
23BCS10127

Table of Contents

1. Introduction

Purpose and Scope

The Java Image Editor that this document explains was built as part of an assignment given to us in the Computer Programming class. It is a comprehensive and effective way of testing our progress in learning and understanding Java. After one month of learning Java, starting with simple print statements and basic loops, to methods and two-dimensional arrays, this project served as a checkpoint for us to understand how well we understood all the concepts. It allowed us to apply all these concepts and build something that has real-life relevance.

While this project started only as part of our assignments, it has allowed students of Scaler School of Technology to get a glimpse into what professional software development will look like. As each one of us gears up for a rewarding and challenging career in software development, this Image Editor project has given us an idea of what kind of ideas and thought processes are involved in a professional setting. It has also given us a deep insight into Java libraries and the language itself.

2. Project Overview

Project Structure

The project is divided into two major sections. The first one is where all the functions for the different editing operations are defined. The second one is where the main function is defined where we designate different ways of calling each of those functions.

3. Image Editing Features

Supported Operations

The supported operations in this image editor are as follows.

1. Rotate clockwise.
2. Rotate anti-clockwise.
3. Mirror.
4. Convert to greyscale
5. Change brightness
6. Invert colours
7. Blur

How to Use Each Feature?

Using the features is simply done by calling the function by utilising the primitive input-based command line interface that has been created within the main function.

4. Code Explanation

Here’s an explanation of the code.

The imported libraries

```
import java.util.*;
import java.awt.*;
import java.io.*;
import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
```

`**java.util.***`: This is a package in Java that provides various utility classes and data structures like **ArrayList**, **HashMap**, and **LinkedList** to help with common programming tasks.

`**java.awt.***`: AWT stands for Abstract Window Toolkit and is a package in Java for creating graphical user interfaces (GUIs). It provides classes for windows, buttons, and other GUI components.

`**java.io.***`: This package provides classes for input and output operations, including file reading and writing. It includes classes like **FileReader**, **FileWriter**, and **FileInputStream**.

`**javax.imageio.ImageIO**`: This class is part of the Java Image I/O API and is used for reading and writing images in different formats. It provides methods for loading and saving images in formats like JPEG, PNG, and GIF.

`**java.awt.image.BufferedImage**`: This class represents an image in Java and is used for manipulating and processing images. It provides methods for creating, editing and rendering images.

1. The rotate clockwise function

```java
//function 1 rotate clockwise
static BufferedImage rotateClockwise(BufferedImage inputImage){
    int height = inputImage.getHeight();
    int width = inputImage.getWidth();

    BufferedImage outputImage = new BufferedImage(height, width, BufferedImage.TYPE_INT_RGB);

    for(int i = 0; i < width; i++){
        for(int j = 0; j < height; j++){
            Color pixel = new Color(inputImage.getRGB(i,j));
            outputImage.setRGB(j, i, pixel.getRGB());
        }
    }

    outputImage = mirror(outputImage);

    return outputImage;
}
```

This Java code defines a function called `**rotateClockwise**` that rotates an input image 90 degrees clockwise. It starts by determining the height and width of the input image and creates a new **BufferedImage** with swapped dimensions to accommodate the rotated image. The code then iterates through each pixel of the input image, reading its colour and placing it in the corresponding position in the output image with its coordinates swapped (i.e., transposing the image). After transposing, the code calls a separate `mirror` function (not shown in this code snippet) to further adjust the image, likely for intuitive orientation, before returning the rotated and mirrored result. In summary, this function effectively rotates the input image 90 degrees clockwise.

2. The rotate anti-clockwise function

```java
//function 2 rotate anticlockwise
static BufferedImage rotateAntiClockwise(BufferedImage inputImage){
    BufferedImage outputImage = rotateClockwise(inputImage);
    outputImage = rotateClockwise(outputImage);
    outputImage = rotateClockwise(outputImage);
    return outputImage;
}
```

This Java code defines a function named `rotateAntiClockwise` that takes a BufferedImage named `inputImage` as an argument. The function's purpose is to rotate the input image 270 degrees anticlockwise (or equivalently, 90 degrees clockwise) by

repeatedly calling the `rotateClockwise` function three times and returning the result.

3. The mirror function

```java
//function 3 mirror
static BufferedImage mirror(BufferedImage inputImage){
    int height = inputImage.getHeight();
    int width = inputImage.getWidth();

    BufferedImage outputImage = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);

    for(int i = 0; i < height; i++){
        for(int j = 0; j < width/2; j++){
            Color pixel = new Color(inputImage.getRGB(j, i));
            outputImage.setRGB(j,i,inputImage.getRGB(inputImage.getWidth()-1-j , i));
            outputImage.setRGB(inputImage.getWidth()-1-j , i , pixel.getRGB());

        }
    }
    return outputImage;
}
```

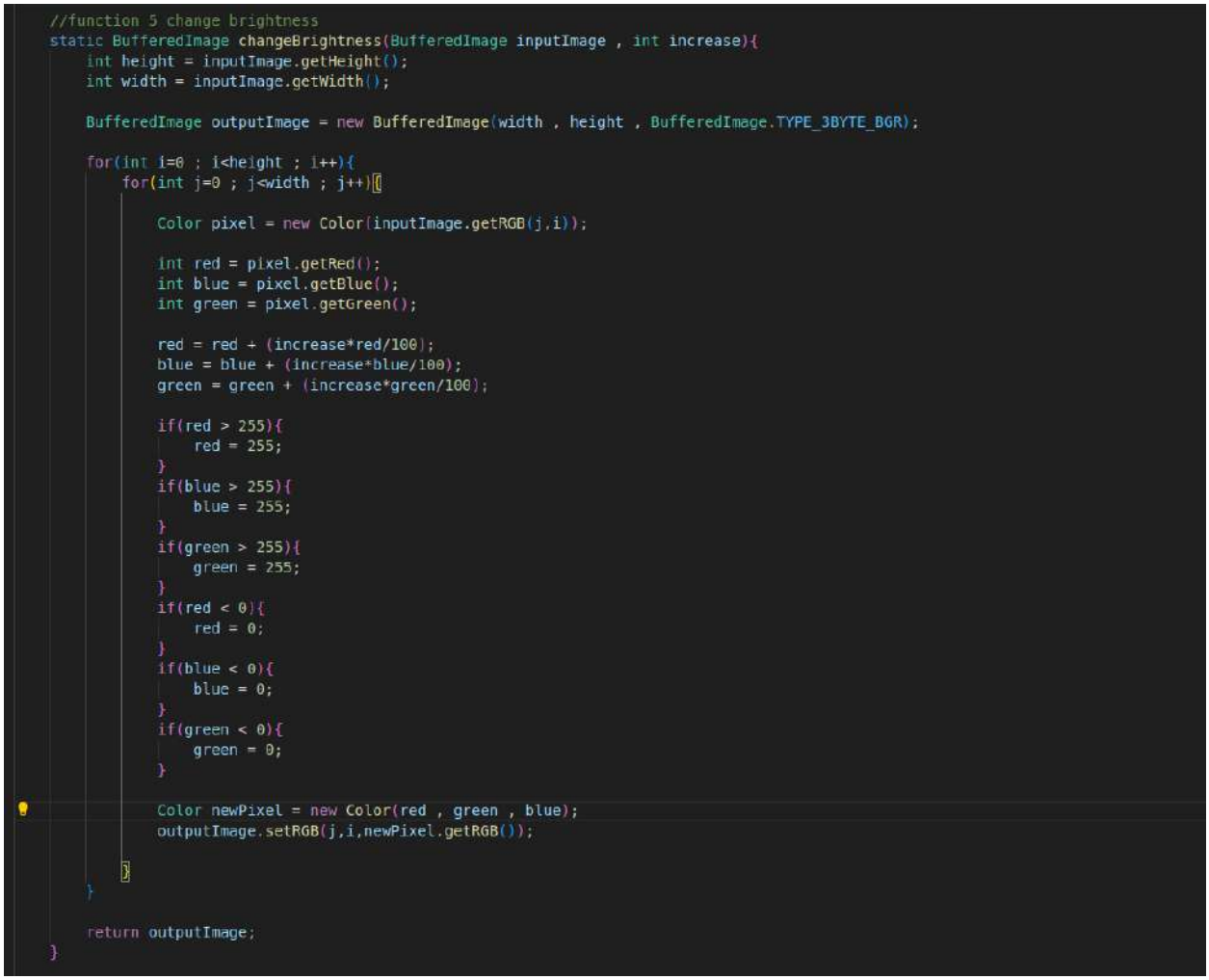
This Java code defines a function called `**mirror**` that takes an input image and creates a horizontally mirrored version of it. It achieves this by iterating through the rows of the input image and swapping the colours of pixels on the left and right sides of each row, effectively reflecting the image along its vertical axis. The mirrored result is stored in a new **BufferedImage** named `**outputImage**`, which is then returned. This function effectively flips the input image horizontally to create a mirrored image.

4. The convert to greyscale function

```java
//function 4 greyscale
static BufferedImage convertToGreyScale(BufferedImage inputImage){

    int height = inputImage.getHeight();
    int width = inputImage.getWidth();

    BufferedImage outputImage = new BufferedImage(width , height , BufferedImage.TYPE_BYTE_GRAY);

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

        for(int j =0; j < width; j++){

            outputImage.setRGB(j,i, inputImage.getRGB(j,i));

        }
    }
    return outputImage;
}
```

This Java code defines a function called **convertToGreyScale** that takes an input image and converts it into a grayscale version. It accomplishes this by first obtaining the dimensions of the input image, then creating a new **BufferedImage** named **outputImage** with grayscale colour representation. Next, it iterates through all the pixels of the input image, copying each pixel's colour to the corresponding location in the grayscale output image. Essentially, this function converts the input image into grayscale by maintaining the same dimensions and pixel values but changing the colour representation to shades of gray, resulting in a black-and-white version of the original image.

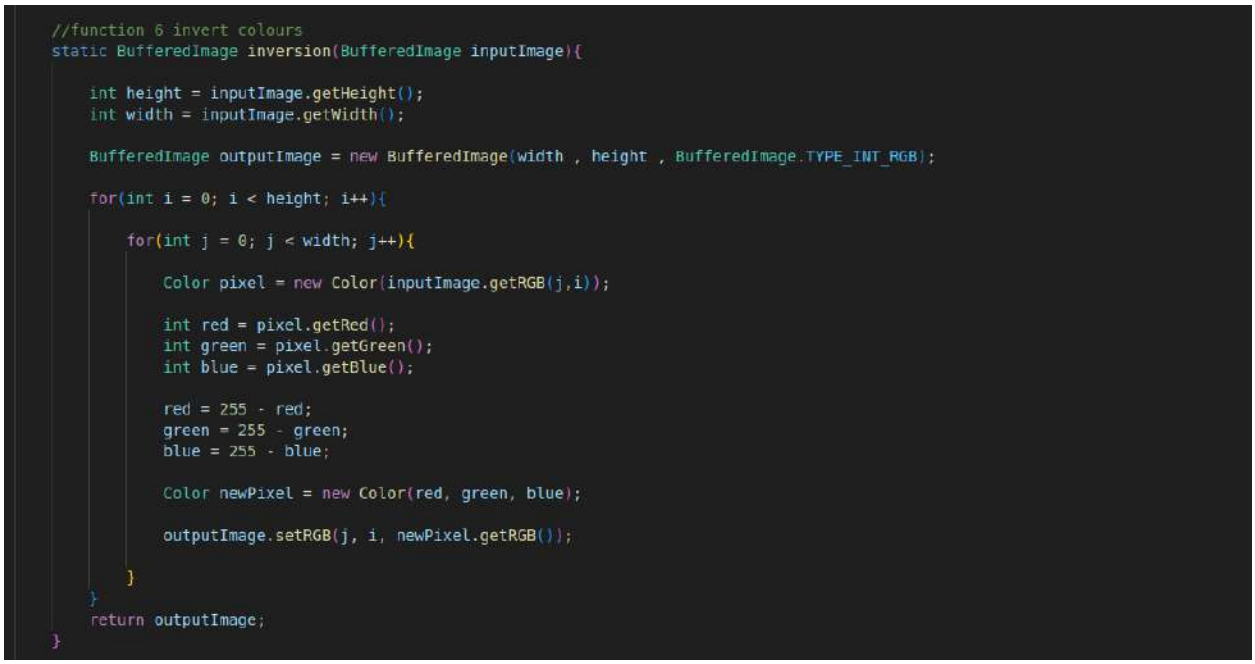
5. The change brightness function



```java
//function 5 change brightness
static BufferedImage changeBrightness(BufferedImage inputImage , int increase){
    int height = inputImage.getHeight();
    int width = inputImage.getWidth();

    BufferedImage outputImage = new BufferedImage(width , height , BufferedImage.TYPE_3BYTE_BGR);

    for(int i=0 ; i<height ; i++){
        for(int j=0 ; j<width ; j++){

            Color pixel = new Color(inputImage.getRGB(j,i));

            int red = pixel.getRed();
            int blue = pixel.getBlue();
            int green = pixel.getGreen();

            red = red + (increase*red/100);
            blue = blue + (increase*blue/100);
            green = green + (increase*green/100);

            if(red > 255){
                red = 255;
            }
            if(blue > 255){
                blue = 255;
            }
            if(green > 255){
                green = 255;
            }
            if(red < 0){
                red = 0;
            }
            if(blue < 0){
                blue = 0;
            }
            if(green < 0){
                green = 0;
            }

            Color newPixel = new Color(red , green , blue);
            outputImage.setRGB(j,i,newPixel.getRGB());

        }
    }

    return outputImage;
}
```

This Java code defines a function named `**changeBrightness**` that adjusts the brightness of an input image. It takes two parameters: `**inputImage**`, the image to be modified, and `increase`, an integer value representing the amount of brightness change (positive for increase, negative for decrease). The code first determines the dimensions of the input image and creates an output image of the same size. It then iterates through each pixel of the input image, extracting its red, blue, and green colour components. The brightness change is applied to these components by scaling them based on the `**increase**` value. The code also ensures that the adjusted colour components stay within the valid range of 0 to 255 by clamping them. Finally, a new colour is created using the adjusted components, and this colour is set for the corresponding pixel in the output image. This function effectively brightens or darkens the input image based on the `**increase**` parameter, allowing for dynamic adjustments to the image's brightness.

6. The invert colours function



```java
//function 6 invert colours
static BufferedImage inversion(BufferedImage inputImage){

    int height = inputImage.getHeight();
    int width = inputImage.getWidth();

    BufferedImage outputImage = new BufferedImage(width , height , BufferedImage.TYPE_INT_RGB);

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

        for(int j = 0; j < width; j++){

            Color pixel = new Color(inputImage.getRGB(j,i));

            int red = pixel.getRed();
            int green = pixel.getGreen();
            int blue = pixel.getBlue();

            red = 255 - red;
            green = 255 - green;
            blue = 255 - blue;

            Color newPixel = new Color(red, green, blue);

            outputImage.setRGB(j, i, newPixel.getRGB());

        }
    }
    return outputImage;
}
```

This Java code defines a function named `inversion` that inverts

the colours of an input image. It starts by obtaining the height and width of the input image and creates a new **BufferedImage** with the same dimensions. Then, it iterates through each pixel in the input image, extracting its red, green, and blue colour components. The code inverts these colour components by subtracting each component's value from 255, effectively creating a negative image. It then constructs a new colour using the inverted components and sets it as the colour of the corresponding pixel in the output image. This function effectively transforms the input image into its colour-negative counterpart, where colours are inverted.

7. The blur function

```
//function 7 blur
static BufferedImage blurr(BufferedImage inputImage , int pixelCount){

    int height = inputImage.getHeight();
    int width = inputImage.getWidth();

    BufferedImage outputImage = new BufferedImage(width , height , BufferedImage.TYPE_INT_RGB);

    int rowStart = 0;
    int rowEnd = pixelCount-1;

    while(rowEnd < height){

        int columnStart = 0;
        int columnEnd = pixelCount-1 ;

        while(columnEnd < width){

            int sumRed = 0;
            int sumGreen = 0;
            int sumBlue = 0;
```

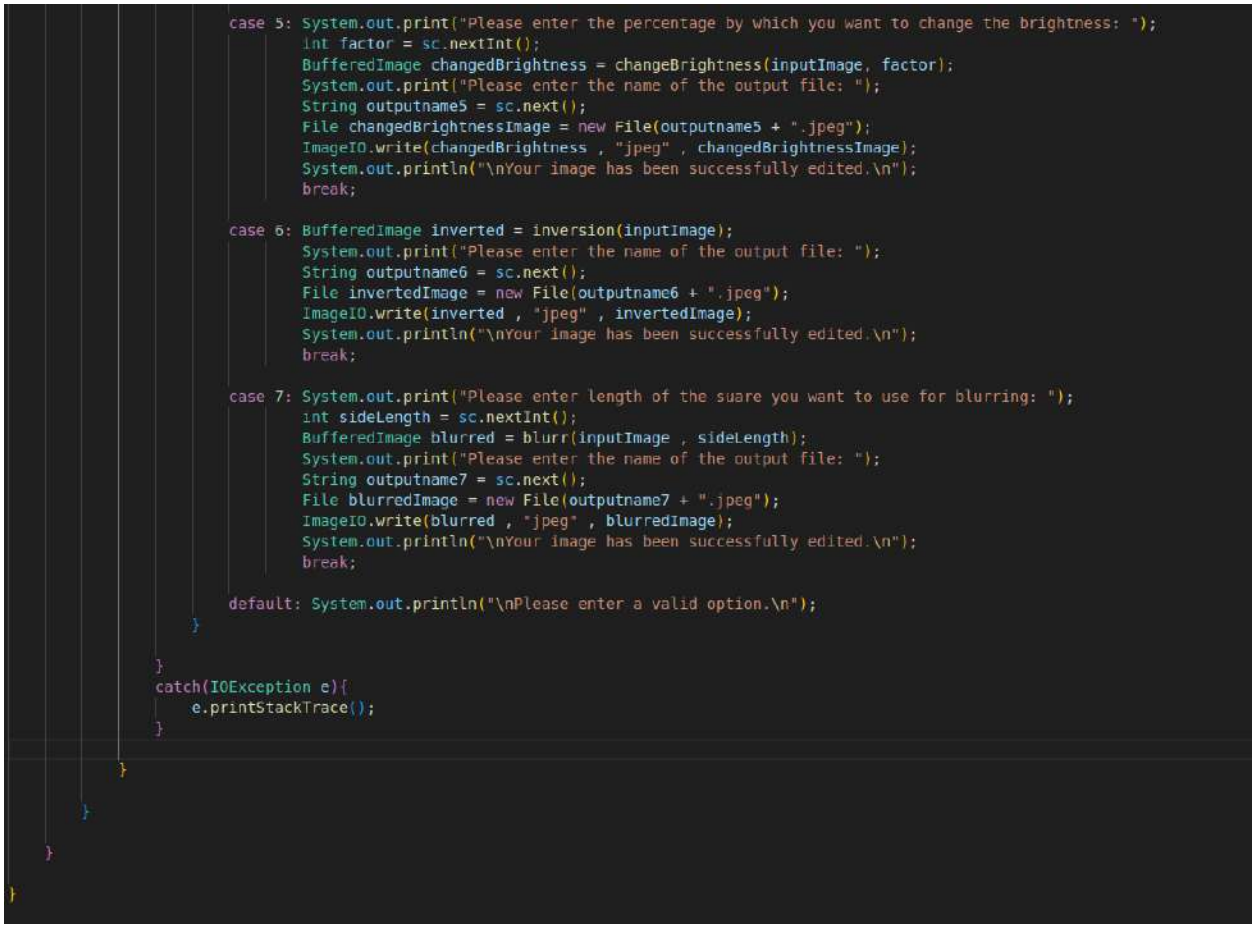
```
            for(int i = rowStart; i <= rowEnd; i++){
                for(int j = columnStart; j <= columnEnd; j++){

                    Color pixel = new Color(inputImage.getRGB(j,i));

                    sumRed += pixel.getRed();
                    sumBlue += pixel.getBlue();
                    sumGreen += pixel.getGreen();

                }
            }

            int avgRed = sumRed/(pixelCount*pixelCount);
            int avgBlue = sumBlue/(pixelCount*pixelCount);
            int avgGreen = sumGreen/(pixelCount*pixelCount);

            Color newPixel = new Color(avgRed , avgGreen , avgBlue);

            for(int i = rowStart ; i <= rowEnd ; i++){
                for(int j = columnStart; j <= columnEnd; j++){
                    outputImage.setRGB(j, i, newPixel.getRGB() );
                }
            }
```

This Java code defines a function called `**blurr**` that applies a blur effect to an input image. It takes two parameters: `**inputImage**`, the image to be blurred, and `**pixelCount**`, which determines the degree of blur. The code first determines the height and width of the input image and creates a new **BufferedImage** with the same dimensions. It then divides the image into square regions of size `**pixelCount**` by `**pixelCount**`, iterating through each region. For each region, it calculates the average colour value by summing the colour components of all the pixels within that region and then dividing by the total number of pixels. This average colour is used to set the colour of all the pixels in that region in the output image. The process repeats for all regions, effectively blurring the input image. The larger the `**pixelCount**`, the greater the blur effect.

The ‘main()’ function

```
Run | Debug
public static void main (String agrs[]){

    try (Scanner sc = new Scanner(System.in)) {
        while (true){

            System.out.println("********** JAVA IMAGE EDITOR **********");
            System.out.println("");
            System.out.println("Please enter the path of the image you want to edit");
            System.out.println("(You can enter just the name and extension of the file if it is in the same directory): ");

            String location = sc.next();

            File inputFile = new File(location);

            System.out.println("1. Rotate clockwise");
            System.out.println("2. Rotate anticlockwise");
            System.out.println("3. Mirror");
            System.out.println("4. Convert to greyscale");
            System.out.println("5. Change brightness");
            System.out.println("6. Invert colours");
            System.out.println("7. Blur");
            System.out.println("0. EXIT");
            System.out.println("");
            System.out.print("Please choose the operation you want to perform on the image: ");

            int choice = sc.nextInt();

            if (choice == 0){
                System.out.println("Exit successful.");
                break;
            }
```

```
            try{

                BufferedImage inputImage = ImageIO.read(inputFile);

                switch(choice){

                    case 1: BufferedImage rotatedClockwise = rotateClockwise(inputImage);
                            System.out.print("Please enter the name of the output file: ");
                            String outputname1 = sc.next();
                            File rotatedClockwiseImage = new File(outputname1 + ".jpeg");
                            ImageIO.write(rotatedClockwise , "jpeg" , rotatedClockwiseImage);
                            System.out.println("\nYour image has been successfully edited.\n");
                            break;

                    case 2: BufferedImage rotatedAntiClockwise = rotateAntiClockwise(inputImage);
                            System.out.print("Please enter the name of the output file: ");
                            String outputname2 = sc.next();
                            File rotatedAntiClockwiseImage = new File(outputname2 + ".jpeg");
                            ImageIO.write(rotatedAntiClockwise , "jpeg" , rotatedAntiClockwiseImage);
                            System.out.println("\nYour image has been successfully edited.\n");
                            break;

                    case 3: BufferedImage mirrored = mirror(inputImage);
                            System.out.print("Please enter the name of the output file: ");
                            String outputname3 = sc.n  java.io.File.File(String pathname)
                            File mirroredImage = new File(outputname3 + ".jpeg");
                            ImageIO.write(mirrored , "jpeg" , mirroredImage);
                            System.out.println("\nYour image has been successfully edited.\n");
                            break;

                    case 4: BufferedImage grayScale = convertToGreyScale(inputImage);
                            System.out.print("Please enter the name of the output file: ");
                            String outputname4 = sc.next();
                            File grayScaleImage = new File(outputname4 + ".jpeg");
                            ImageIO.write(grayScale , "jpeg" , grayScaleImage);
                            System.out.println("\nYour image has been successfully edited.\n");
                            break;
```



```
                    case 5: System.out.print("Please enter the percentage by which you want to change the brightness: ");
                            int factor = sc.nextInt();
                            BufferedImage changedBrightness = changeBrightness(inputImage, factor);
                            System.out.print("Please enter the name of the output file: ");
                            String outputname5 = sc.next();
                            File changedBrightnessImage = new File(outputname5 + ".jpeg");
                            ImageIO.write(changedBrightness , "jpeg" , changedBrightnessImage);
                            System.out.println("\nYour image has been successfully edited.\n");
                            break;

                    case 6: BufferedImage inverted = inversion(inputImage);
                            System.out.print("Please enter the name of the output file: ");
                            String outputname6 = sc.next();
                            File invertedImage = new File(outputname6 + ".jpeg");
                            ImageIO.write(inverted , "jpeg" , invertedImage);
                            System.out.println("\nYour image has been successfully edited.\n");
                            break;

                    case 7: System.out.print("Please enter length of the suare you want to use for blurring: ");
                            int sideLength = sc.nextInt();
                            BufferedImage blurred = blurr(inputImage , sideLength);
                            System.out.print("Please enter the name of the output file: ");
                            String outputname7 = sc.next();
                            File blurredImage = new File(outputname7 + ".jpeg");
                            ImageIO.write(blurred , "jpeg" , blurredImage);
                            System.out.println("\nYour image has been successfully edited.\n");
                            break;

                    default: System.out.println("\nPlease enter a valid option.\n");
                }

            }
            catch(IOException e){
                e.printStackTrace();
            }

        }

    }

}
```

This Java `main` function represents an interactive image editing program. Here's a brief explanation:

- It starts by setting up a loop that keeps running until the user decides to exit (`while (true)`).

- Inside the loop, it displays a user-friendly menu for a variety of image editing options, such as rotating, mirroring, converting to grayscale, adjusting brightness, inverting colours, and blurring.

- It prompts the user to enter the path (or just the name and extension if the file is in the same directory) of the image they want to edit.

- It reads the user's choice for the editing operation and performs the selected operation on the input image.

- After performing the operation, it asks the user for the desired output file name for the edited image and saves the edited image as a JPEG file with the chosen name.

- The loop continues, allowing the user to perform more editing operations on different images or the same image repeatedly.

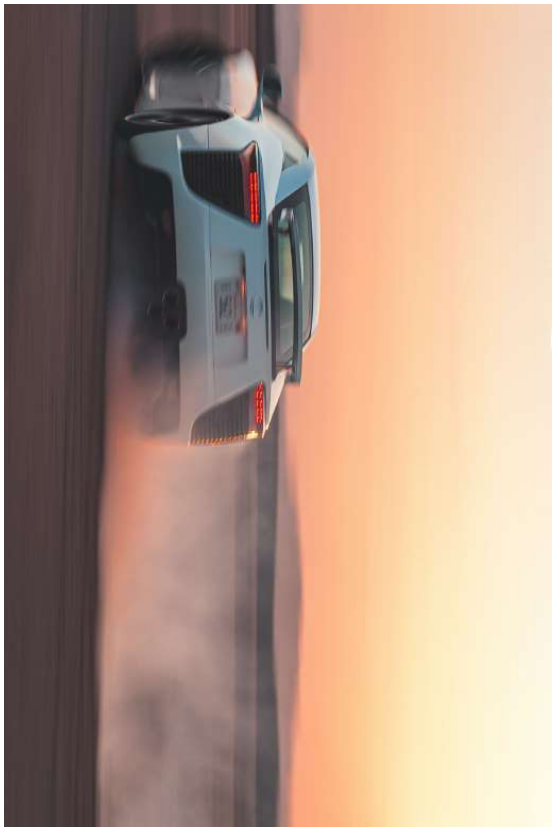
- If the user chooses to exit (by entering 0), the program terminates, displaying an exit message.

- Exception handling is in place to manage potential issues, such as file not found errors or image processing errors, and to print error details if they occur.

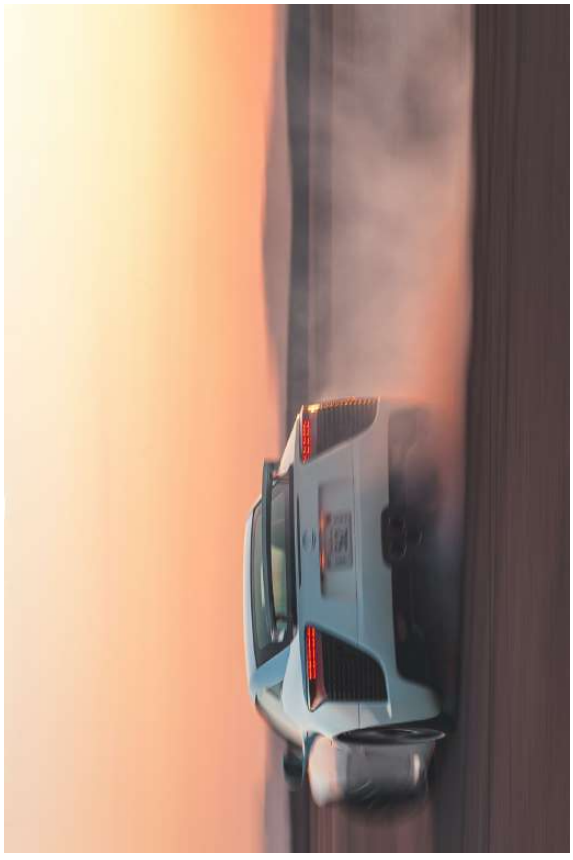
In summary, this `**main**` function provides an interactive command-line interface for image editing, where users can choose from various editing options, apply them to images, and save the edited versions.

5. Example Use Cases

Function 1 - Rotate Clockwise



Function 2 - Rotate Anti-clockwise



Function 3 - Mirror

Function 4 - Convert to Greyscale

Function 5 - Change Brightness (75% increase)

Function 6 – Invert colours

Function 7 - Blur (pixel square size taken - 20)

Provide detailed walkthroughs of basic image editing tasks using your command-line image editor. Include command-line commands and expected results.

6. Future Improvements

While this project has shown a tremendous leap in our understanding and hold over Java as a language and programming itself, there is a huge scope for improvement. One of the major ones is integrating a Graphical User Interface (GUI).

Other potential improvements include -

- Better exception handling
- More options such as circle crop, non-conventional rotate, edge detection, etc.
- Enhanced user interaction features such as sliders for indicating the amount of brightness to be increased/decreased.
- Options to convert the output image to multiple formats.

7. Acknowledgements

I would like to take this opportunity to thank Kshitij Sir, the BSMs and all my friends for helping me complete this project and for being with me through my journey learning Java.