Java Image Editor Documentation

Introduction:

⟶ The Java Image Editor is a command-line type, designed for image editing. This documentation provides a comprehensive guide to its features and usage.

⟶ To use the Java Image Editor, follow these steps:

- Compile the Java code.
- Open a terminal or command prompt.
- Run the tool by executing the file with the command.

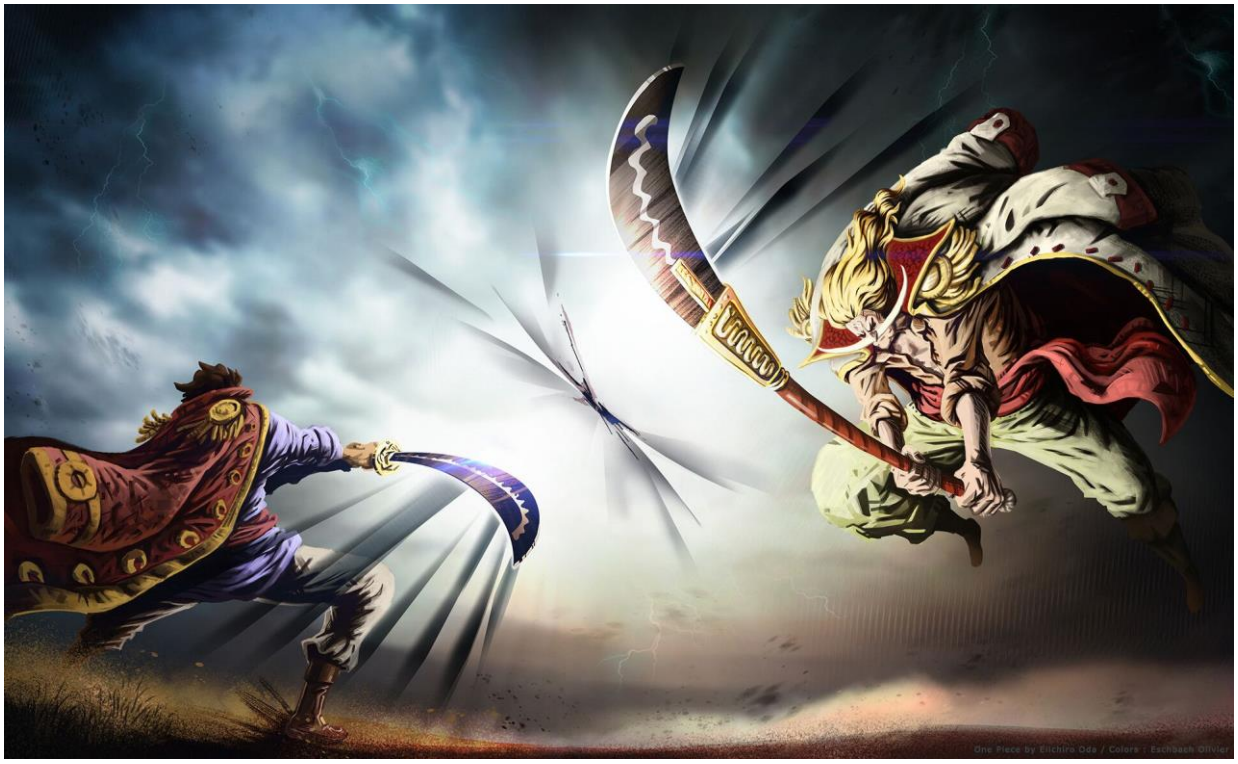
Features:

1. **Print Pixel Values (Option 1):**
    - **Usage:** This feature prints the pixel values of the input image. It provides a numerical value of the image's colors.
2. **Convert to Gray Scale (Option 2):**
    - **Usage:** This feature converts the input image to Gray Scale, removing color of the image and turns it to shades of gray.
3. **Increase Image Brightness (Option 3):**
    - **Usage:** Adjust the brightness of the input image using this feature. You can Specify the percentage by which to increase the brightness.
4. **Right Rotation (Option 4):**
    - **Usage:** The right rotation feature rotates the input image 90 degrees to the right (clockwise). It turns the image in the opposite direction of the left rotation.
5. **Left Rotation (Option 5):**
    - **Usage:** This feature rotates the input image 90 degrees to the left (counterclockwise). It effectively turns the image on its side.
6. **Invert Horizontal (Option 6):**
    - **Usage:** You can perform a horizontal rotation of the input image. It creates a new image with the left and right halves swapped.

7. **Vertical Rotation (Option 7):**

   Usage: ***This feature allows you to perform a vertical rotation of the input image. It creates a new image with the top and bottom halves swapped***

Image:



Code Documentation:

1. **Print Pixel value:-**

```
public static void printPixelValue(BufferedImage inputImage)
    int height = inputImage.getHeight();
    int width = inputImage.getWidth();
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            System.out.print(inputImage.getRGB(j, i));
        }
    }
}
```

- **Description:** This function iterates through the pixel values of the input image and prints them .It takes an input image as its parameter and prints the pixel values row by row.

Usage:

- The function begins by determining the height and width of the input image.
- Then it iterates through each row and column of the image.
- For each pixel in the image, it retrieves the RGB (Red, Green, Blue) values.
- These RGB values are printed.

Output:

```
12-16247274-16773875-16577266-16577266-16511986-16446193-16315377-16315377-16315377-16315377-16116964-15722206-16116964-1624855
0-15590620-15787999-16314343-15985378-16116964-15590620-15787999-16182757-15985378-15919585-15985378-15722206-15853792-15787999
-16248550-16314343-15787999-15722206-15985378-15853792-15985378-15722206-16445929-15985378-15853792-16116964-15459034-15853792-
15853281-16182246-16248552-15919587-15656415-15722208-15919587-15985380-15985893-16380651-16314858-15788514-15526112-15920870-1
6184042-16118249-16446960-15723237-15657444-15986409-15986409-16183788-16381167-16117995-15986409-16052202-16117995-16183788-16
249581-16183788-16117995-16117995-16052202-16117995-16183788-16249581-16249581-16183788-16052202-15986409-16315374-16381167-163
81167-16249581-16117995-16052202-16052202-16117995-16183788-16315374-16315374-16117995-15986409-16183788-16315374-16249581-1631
5374-16249581-16183788-16117995-16117995-16183788-16249581-16315374-16315376-16249583-16117997-16117997-16183790-16249583-16249
583-16183790-16249583-16183790-16183790-16249583-16315376-16381169-16315376-16249583-16315374-16183788-16117995-16117995-162495
81-16249581-16249581-16183788-16052202-16315374-16381167-16249581-16249581-16381167-16315374-16117995-16117995-16183788-1624958
1-16249581-16249581-16249581-16249581-16183788-16249581-16249581-16249581-16249581-16183788-16183788-16183788-16183788-16315374
```

2. Convert To Gray Scale:-

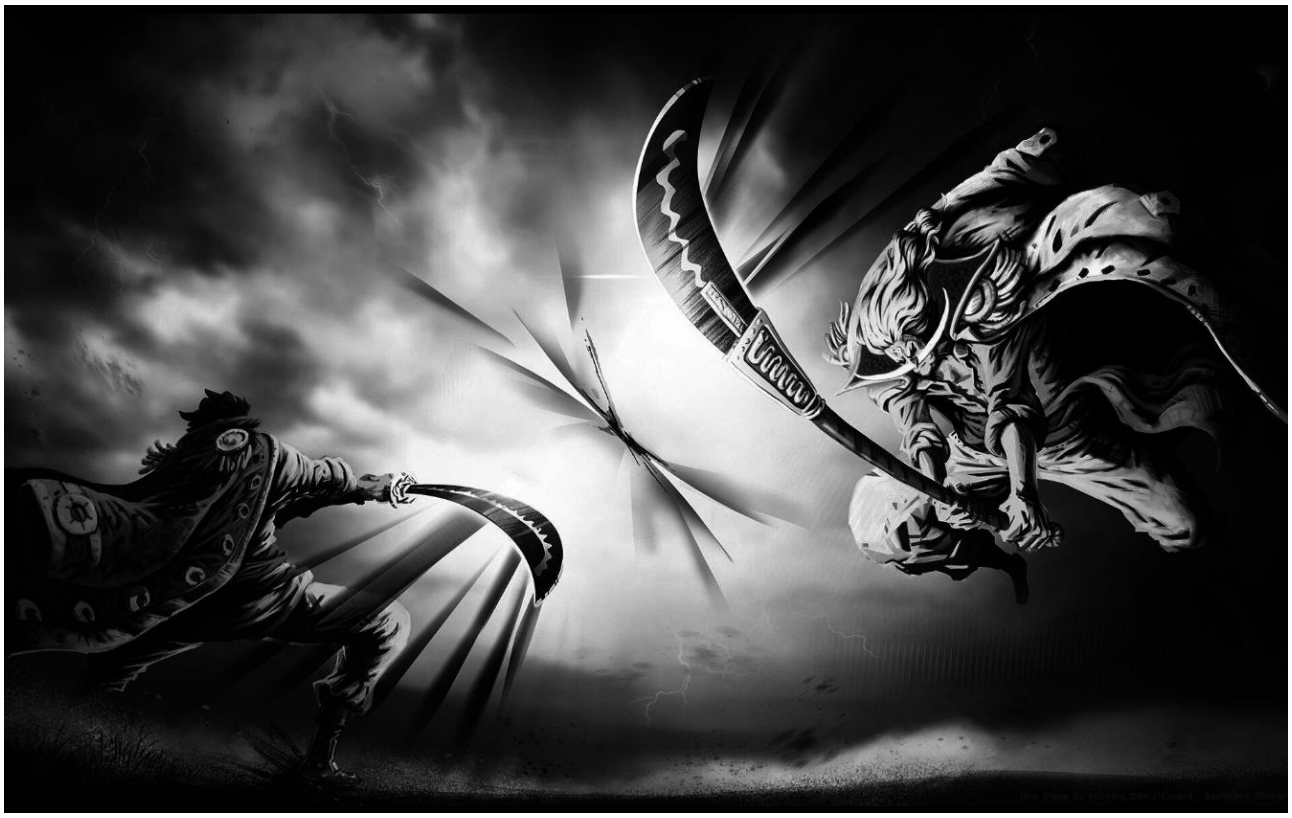
```java
public static BufferedImage grey(BufferedImage inputImage) {
    int height = inputImage.getHeight();
    int width = inputImage.getWidth();
    BufferedImage outputImage = new BufferedImage(width, height, BufferedImage.TYPE_BYTE_GRAY);
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            outputImage.setRGB(j, i, inputImage.getRGB(j, i));
        }
    }
    return outputImage;
}
```

- **Description:** This function converts the input color image into grayscale.

Usage:

- The function first determines the height and width of the input color image.
- Then it creates an empty grayscale image.
- It iterates through each pixel in the input image, row by row and column by column.
- And it converts into grey
- This process continues for all pixels in the input image, resulting in a grayscale image

Output:-



3. Image brightness:-

- **Description:** This function adjusts the brightness of the input image based on the specified percentage increase or decrease

```
public static BufferedImage brightness(BufferedImage inputImage, int A) {
    int height = inputImage.getHeight();
    int width = inputImage.getWidth();
    BufferedImage brit = new BufferedImage(width, height, BufferedImage.TYPE_3BYTE_BGR);
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            Color output = new Color(inputImage.getRGB(j, i));
            int green = output.getGreen();
            int blue = output.getBlue();
            int red = output.getRed();
            red += (red * A) / 100;
            green += (green * A) / 100;
            blue += (blue * A) / 100;
            if (red > 255)
                red = 255;
            if (green > 255)
                green = 255;
            if (blue > 255)
                blue = 255;
            if (red < 0)
                red = 0;
            if (blue < 0)
                blue = 0;
            if (green < 0)
                green = 0;
            Color newoutput = new Color(red, green, blue);
            brit.setRGB(j, i, newoutput.getRGB());
        }
    }
    return brit;
```

how code works:

- A new mage is created with the same width and height as the original image.
- A nested loop is used to iterate through each row and column of the original image.
- In the inner loop, the following steps are performed:
    - A Color object is created from the pixel value at the current row and column.
    - The red, green, and blue values of the Color object are increased by the specified percentage.
    - A new Color object is created with the increased red, green, and blue values.

- The setRGB() method is used to set the pixel value at the current row and column of the new image to the value of the new Color object.

- The new image is returned.

Output:-

80% brightness

4. Right Transpose Image:-

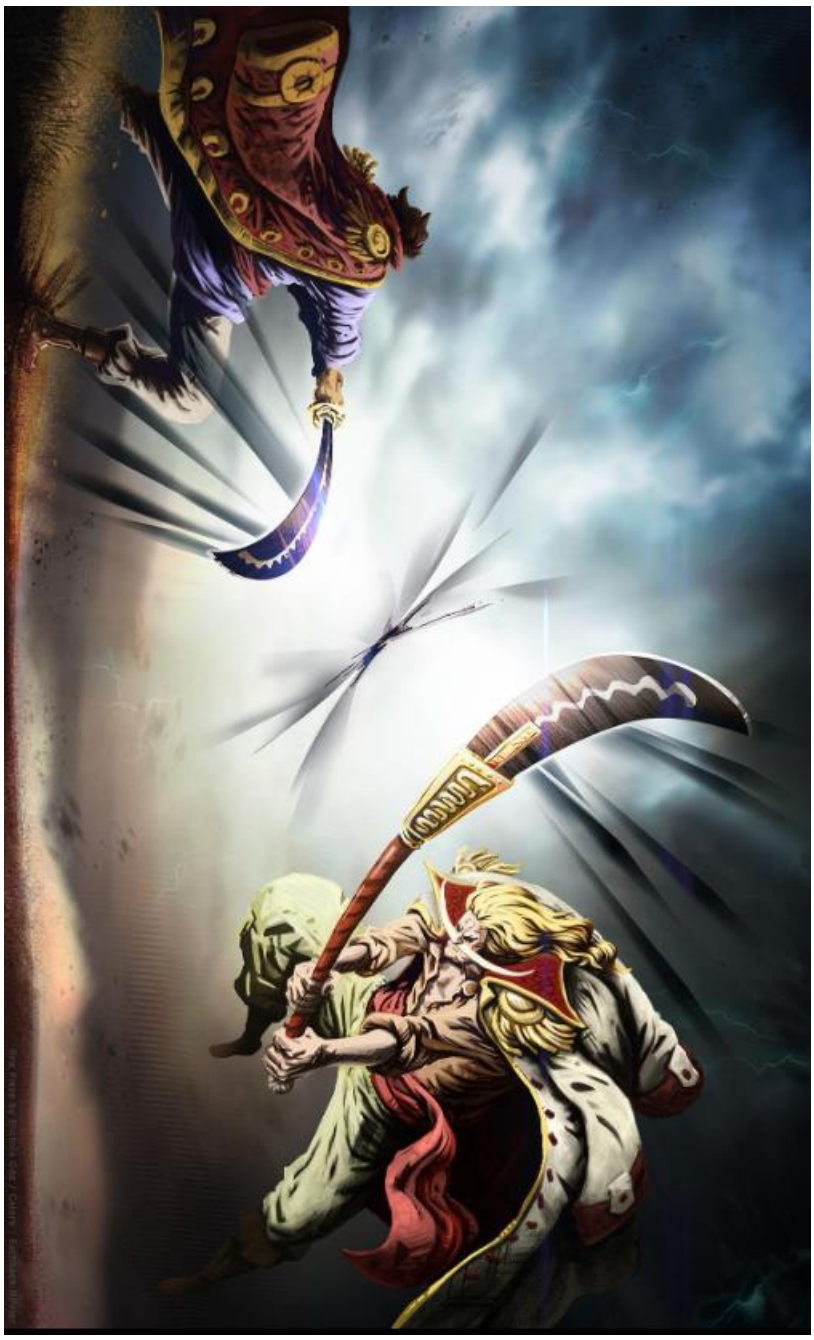
```java
public static BufferedImage Righttransposeimage(BufferedImage inImage) {
    int height = inImage.getHeight();
    int width = inImage.getWidth();
    BufferedImage rotatedImage = new BufferedImage(height, width, BufferedImage.TYPE_INT_RGB);
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            rotatedImage.setRGB(i, j, inImage.getRGB(j, i));
        }
    }
    int index = rotatedImage.getWidth() - 1;
    for (int i = 0; i < rotatedImage.getHeight(); i++) {
        for (int j = 0; j < rotatedImage.getWidth() / 2; j++) {
            Color temp = new Color(rotatedImage.getRGB(j, i));
            rotatedImage.setRGB(j, i, rotatedImage.getRGB(index - j, i));
            rotatedImage.setRGB(index - j, i, temp.getRGB());

        }
    }
    return rotatedImage;
}
```

- **Description:** This function performs a right (clockwise) 90-degree rotation of the input image.

Usage:

- Create an empty image with exchanged dimensions same height becomes width, and width becomes height for the rotation.
- Iterate through each pixel of the input image.
- For each pixel, copy its color information from the original image.
- Place the copied color in the rotated image, but in a position that's rotated by 90 degrees clockwise
- Do the rotation by changing pixels horizontally within each row.
- Repeat this process for all pixels in the input image.
- The resulting rotated image is returned as the output.

Output:-



5. Left Transpose Image:-

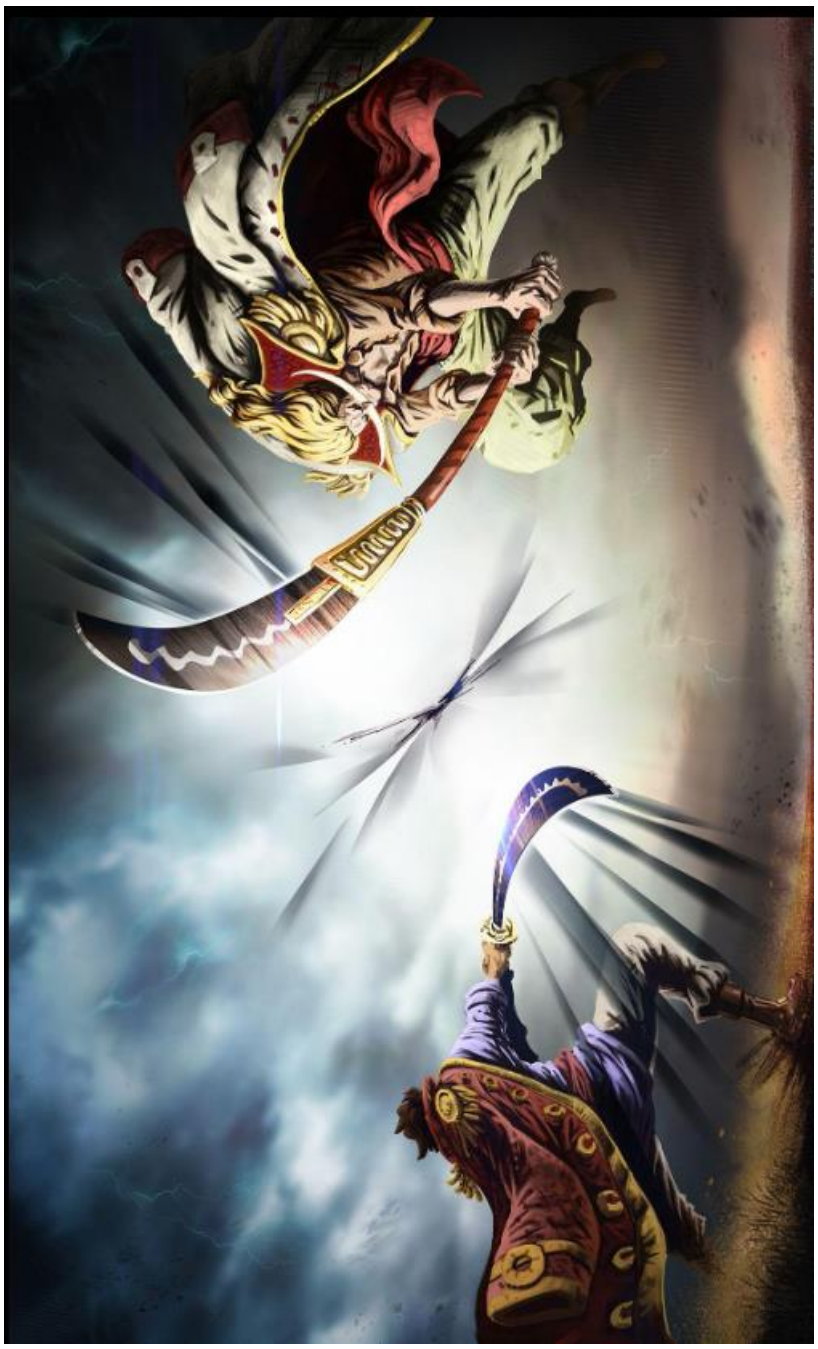
```java
public static BufferedImage Lefttransposeimage(BufferedImage inImage) {
    int height = inImage.getHeight();
    int width = inImage.getWidth();
    BufferedImage rotatedImage = new BufferedImage(height, width, BufferedImage.TYPE_INT_RGB);
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            rotatedImage.setRGB(i, j, inImage.getRGB(j, i));
        }
    }
    int index = rotatedImage.getHeight() - 1;
    for (int j = 0; j < rotatedImage.getWidth(); j++) {
        for (int i = 0; i < rotatedImage.getHeight() / 2; i++) {
            Color temp = new Color(rotatedImage.getRGB(j, i));
            rotatedImage.setRGB(j, i, rotatedImage.getRGB(j, index - i));
            rotatedImage.setRGB(j, index - i, temp.getRGB());

        }
    }

    return rotatedImage;
}
```

- **Description:** This function performs a left 90-degree rotation of the input image.

- **Usage:**

- Create an empty image with exchanged dimensions same height becomes width, and width becomes height for the rotation.
- Iterate through each pixel of the input image.
- For each pixel, copy its color information from the original image.
- Place the copied color in the rotated image, but in a position that's rotated by 90 degrees counterclockwise.
- Do the rotation by changing pixels vertically within each column.
- Repeat this process for all pixels in the input image.
- The resulting rotated image is returned as the output.

Output:-



6. Horizontally Invert:-

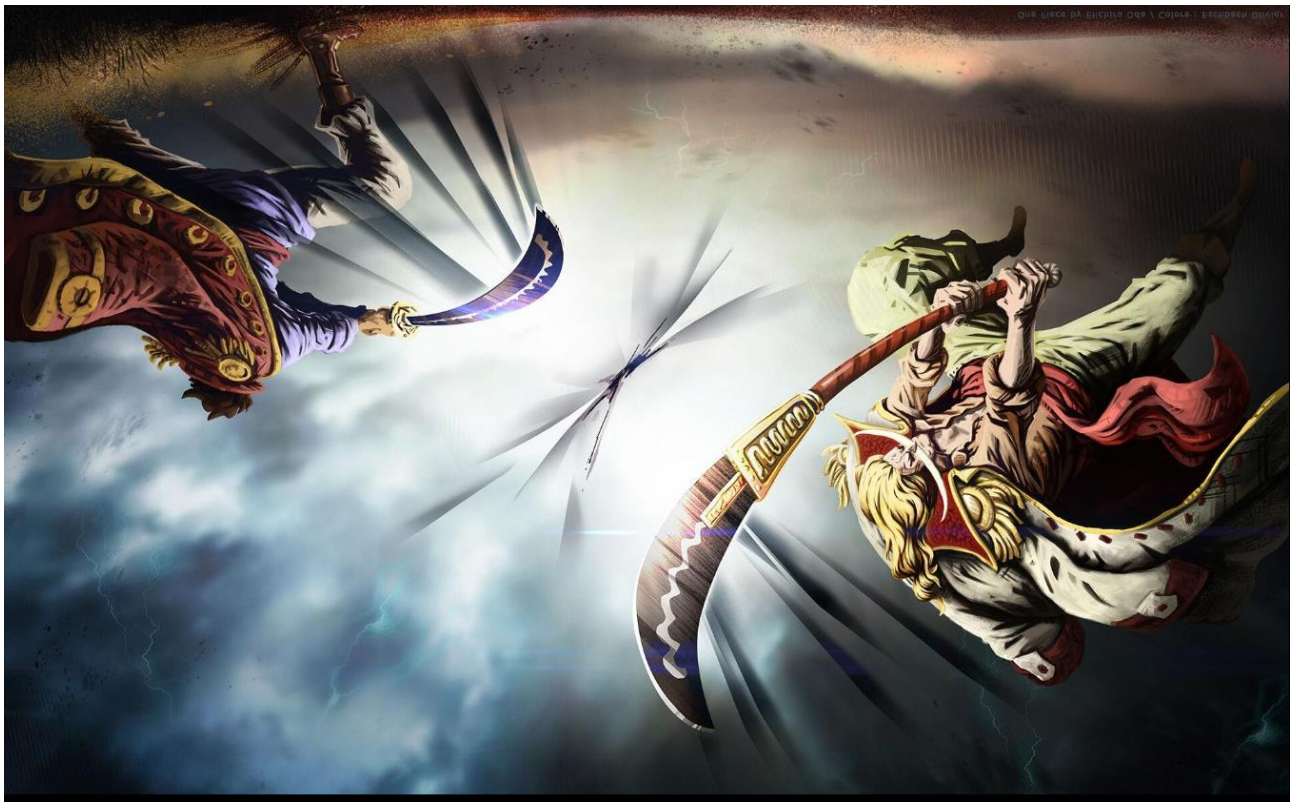
```
public static BufferedImage invertHorizontally(BufferedImage InputImage){
    int height = InputImage.getHeight();
    int width= InputImage.getWidth();
    BufferedImage outputImage= new BufferedImage(width,height,BufferedImage.TYPE_3BYTE_BGR);
    for(int j=0;j<width;j++){
        for(int i=0;i<height;i++){
            outputImage.setRGB(j,i,InputImage.getRGB(j,height-i-1));
        }
    }
    return outputImage;
}
```

- **Description:** This function horizontally inverts (flips) the input image.

How code works :

- The height and width variables are initialized to the height and width of the original image.
- A new image is created with the same width and height as the original image.
- A nested loop is used to iterate through each row and column of the original image.
- In the inner loop, the setRGB() method is used to set the pixel value of the corresponding column in the new image to the pixel value of the column at the opposite end of the original image.
- The new image is returned.

Output:-



7. Vertically Invert:-

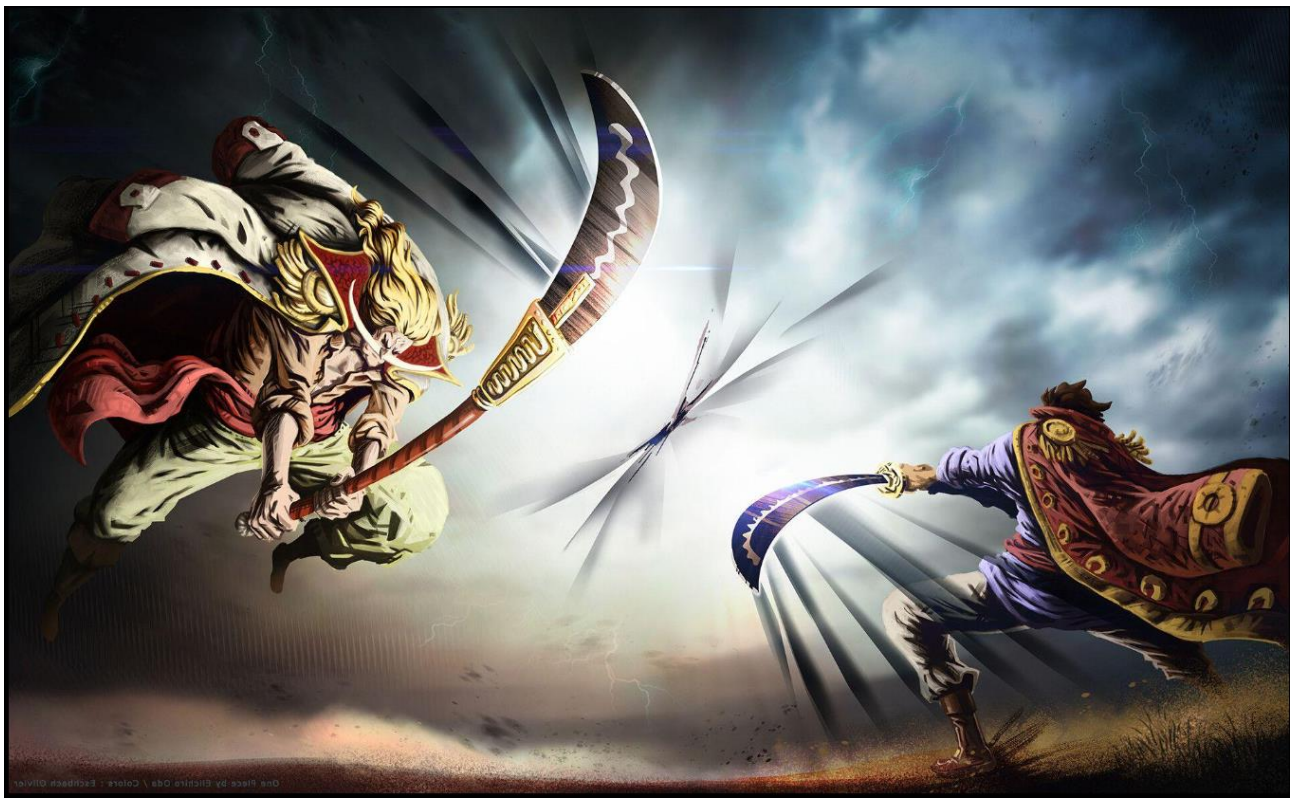
```
public static BufferedImage invertVertically(BufferedImage InputImage){
    int height = InputImage.getHeight();
    int width= InputImage.getWidth();
    BufferedImage outputImage=new BufferedImage(width,height,BufferedImage.TYPE_3BYTE_BGR);
    for(int i=0;i<height;i++){
        for(int j=0;j<width;j++){
            outputImage.setRGB(j,i,InputImage.getRGB(width-j-1,i));
        }
    }
    return outputImage;
}
```

- **Description:** This function vertically inverts (flips) the input image.

How code works:

- The height and width variables are initialized to the height and width of the original image.
- A new image is created with the same width and height as the original image.
- A nested loop is used to iterate through each column and row of the original image.
- In the inner loop, the setRGB() method is used to set the pixel value of the corresponding row in the new image to the pixel value of the row at the opposite end of the original image.
- The new image is returned.

Output:-



8. Blur Image:-

```
public static BufferedImage Blur(BufferedImage input, int pixels) {
    int height = input.getHeight();
    int width = input.getWidth();
    BufferedImage outputImage = new BufferedImage(width, height, BufferedImage.TYPE_3BYTE_BGR);

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

            int red = 0;
            int green = 0;
            int blue = 0;

            for (int k = i * pixels; k < i * pixels + pixels; k++) {
                for (int l = j * pixels; l < j * pixels + pixels; l++) {
                    Color pixel = new Color(input.getRGB(l, k));
                    red += pixel.getRed();
                    blue += pixel.getBlue();
                    green += pixel.getGreen();
                }
            }
            int finalRed = red / (pixels * pixels);
            int finalGreen = green / (pixels * pixels);
            int finalBlue = blue / (pixels * pixels);
```

```
                for (int k = i * pixels; k < i * pixels + pixels; k++) {
                    for (int l = j * pixels; l < j * pixels + pixels; l++) {
                        Color newPixel = new Color(finalRed, finalGreen, finalBlue);
                        outputImage.setRGB(l, k, newPixel.getRGB());
                    }
                }
            }
        }

        return outputImage;
    }
```

.Description: This function adjusts the blurness of the input image based on the specified pixel values for increase or decrease\

how code works:

• Declare the function blur(), which takes two arguments: the input image and the number of pixels to blur the image by.

• Get the height and width of the input image.

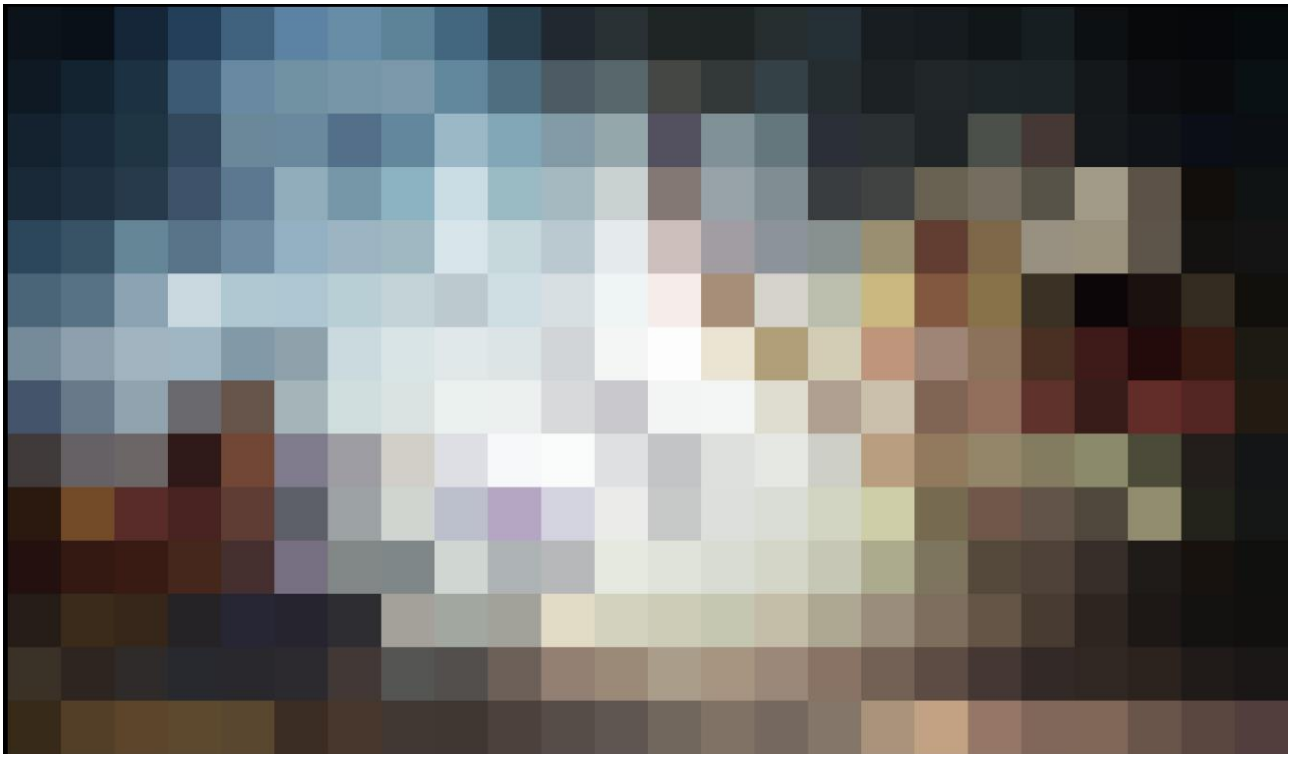
• Create a new output image with the same dimensions as the input image.

• Iterate through the input image, pixel by pixel.

• For each pixel, average the RGB values of its neighborhood and store the average value in the corresponding pixel of the output image.

Output:-

80% blur;



Import Used:-

java.io.File;

java.io.IOException;

java.awt.image.BufferedImage;

javax.imageio.ImageIO;

java.awt.*;

java.util.*;

java.lang.*;