Java Image Editor

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. **Vertical Rotation (Option 1):**
    - **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.*
2. **Horizontal Rotation (Option 2):**
    - **Usage:** With this feature, you can perform a horizontal rotation of the input image. It creates a new image with the left and right halves swapped.
3. **Left Rotation (Option 3):**
    - **Usage:** This feature rotates the input image 90 degrees to the left (counterclockwise). It effectively turns the image on its side.
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. **Print Pixel Values (Option 5):**
    - **Usage:** This feature prints the pixel values of the input image. It provides a numerical value of the image's colors.
6. **Increase Image Brightness (Option 6):**
    - **Usage:** Adjust the brightness of the input image using this feature. You can Specify the percentage by which to increase the brightness.
7. **Convert to Gray Scale (Option 7):**

- **Usage:** This feature converts the input image to Gray Scale, removing color of the image and turns it to shades of gray.

8.Blur Image(option 8):

.Usage:

Adjust the blurness of the input image by using this feature .you can specify the pixel values for blurness of image.

Code Documentation:

1. **`Print pixelvalues`**

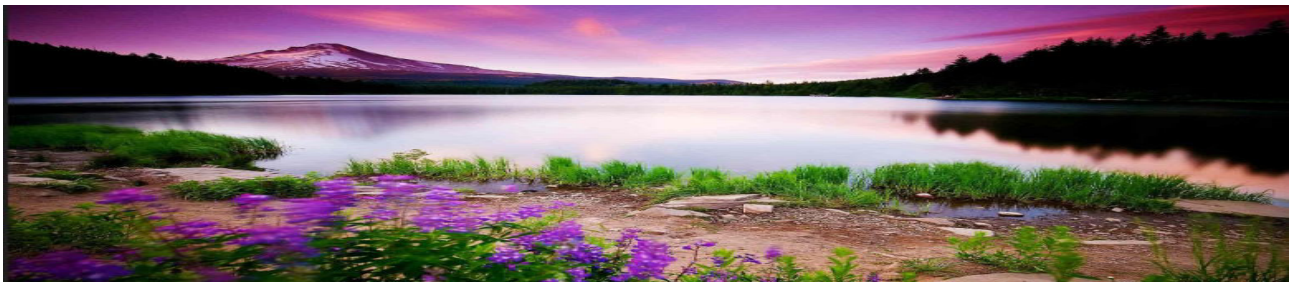
```
// pixels printing

public static void printpixelvalues(BufferedImage inImage) {
    int height = inImage.getHeight();
    int width = inImage.getWidth();
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            System.out.print(inImage.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.

SAMPLE INPUT:



SAMPLE OUTPUT:

```
414400 -12755449 -11372774 -10911196 -12554992 -14528256 -14527232 -12948718 -14265344 -13608190 -11833577 -11571428
 -8284832 -16178688 -15454711 -15848191 -13545699 -14927104 -14730240 -13218805 -14272000 -12430587 -10458080 -11906
 -3754365 -12769280 -10664679 -8231619 -2837875 -3496832 -6655663 -8893139 -11656956 -10011874 -6653867 -5798558 -89
136 -13167616 -12245243 -10335706 -8558268 -11583463 -12769020 -4941711 -3561598 -5272474 -6062760 -7312826 -5472157
7917172 -8905086 -11141022 -9099655 -6072167 -4886372 -6267021 -6133917 -4093836 -4027026 -6000817 -3763336 -4816276
4347 -3894414 -5539239 -5999022 -6788284 -6326710 -3299980 -9220070 -9088488 -7377102 -5074093 -5731000 -7967962 -823
8 -10325478 -12560384 -12953600 -12360960 -11964923 -11964921 -12360192 -12426752 -12097786 -11310577 -9077465 -7895
5473970 -5145259 -5013156 -4881055 -4880539 -5011354 -4880539 -4157847 -4816806 -2381952 -1526641 -4552605 -4223638 -
371 -3167601 -2970991 -3892345 -4484993 -5274509 -5537681 -5077130 -4353407 -3761270 -3563891 -5735060 -5406095 -481
```

2.Convert To GrayScale

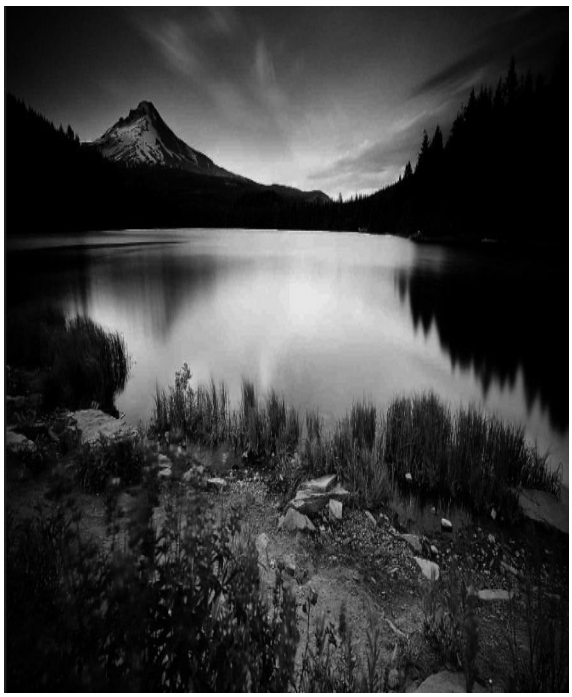
```java
// convert to grey

public static BufferedImage convertToGrayScale(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

SAMPLE INPUT: **SAMPLE OUTPUT:**



3.Right Tanspose Image

```java
// Right the image

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;
}
```

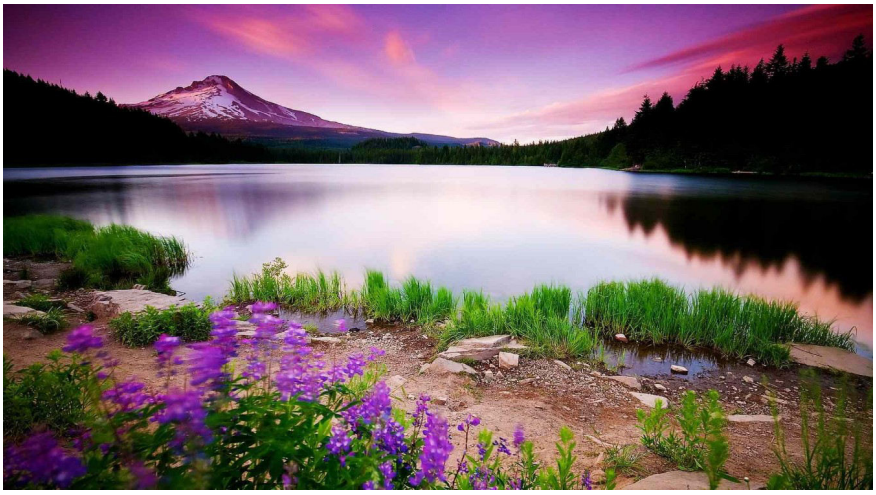
- **Description:** This function performs a right (clockwise) 90-degree rotation of the input image.

Usage

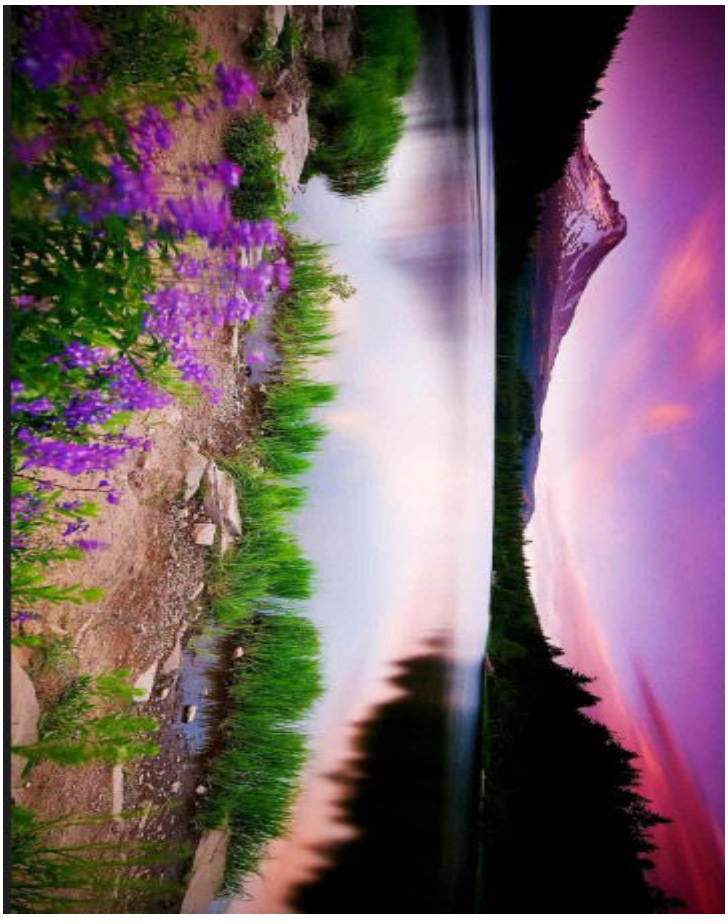
- Create an empty image with exchanged dimensions same height becomes width, and width becomes height for the rotation.
- Iterate through each pixel of the input image.
- For each pixel, copy its color information from the original image.

- 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.

SAMPLE INPUT:



SAMPLE OUTPUT:



4.Left Transpose Image:

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

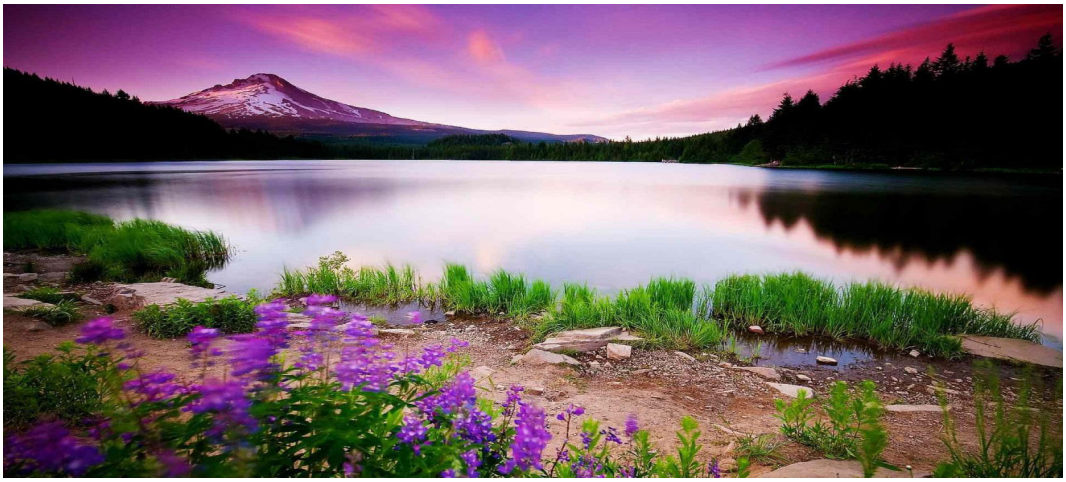
```java
// Left the image

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;
}
```

How code works

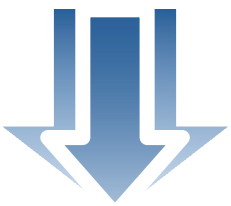
Create an empty image with exchanged dimensions same height becomes width, and width becomes height for the rotation.Iterate through each pixel of the input image.

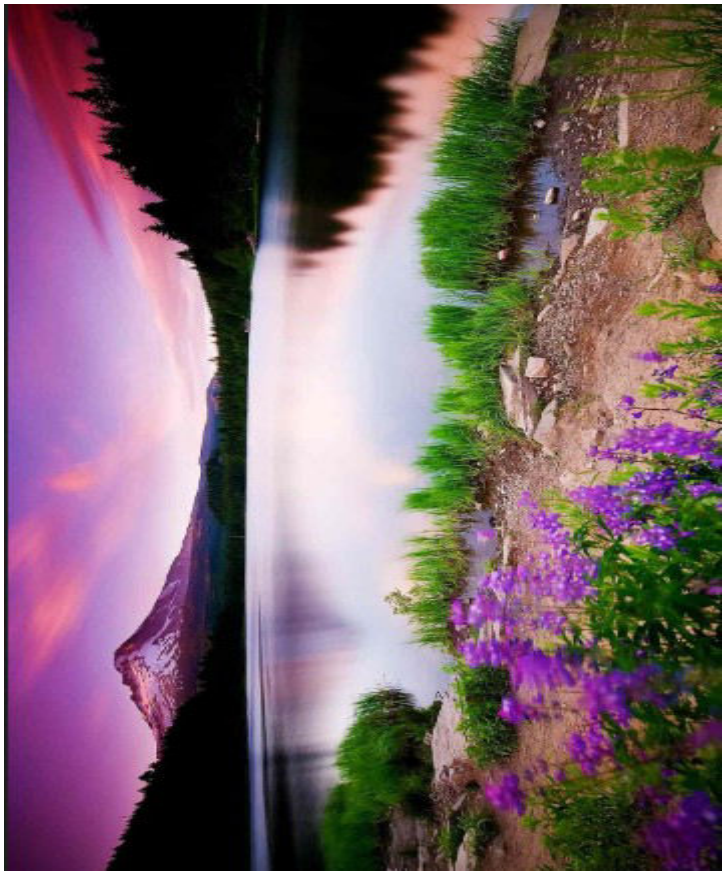
- For each pixel, copy its color information from the original image.
- Place the copied color in the rotated image, but in a position that's rotated by 90 degrees counterclockwise.
- Do the rotation by changing pixels vertically within each column.
- Repeat this process for all pixels in the input image.
- The resulting rotated image is returned as the output.

- SAMPLE INPUT:



SAMPLE OUTPUT:





5. Horizontally Invert

```java
// horizontal image

public static BufferedImage horizontallyinvert(BufferedImage inImage) {
    int height = inImage.getHeight();
    int width = inImage.getWidth();
    BufferedImage HinvertImage = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            HinvertImage.setRGB(width - j - 1, i, inImage.getRGB(j, i));
        }
    }

    return HinvertImage;
}
```

- **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.

SAMPLE INPUT:

SAMPLE OUTPUT:

6.Vertically Invert

```java
// vertical image

public static BufferedImage verticallyinvert(BufferedImage inImage) {
    int height = inImage.getHeight();
    int width = inImage.getWidth();
    BufferedImage VinvertImage = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);
    for (int j = 0; j < width; j++) {
        for (int i = 0; i < height; i++) {

            VinvertImage.setRGB(j, height - i - 1, inImage.getRGB(j, i));
        }
    }

    return VinvertImage;
}
```

- **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.

SAMPLE INPUT:

SAMPLE OUTPUT:

7.Image brightness

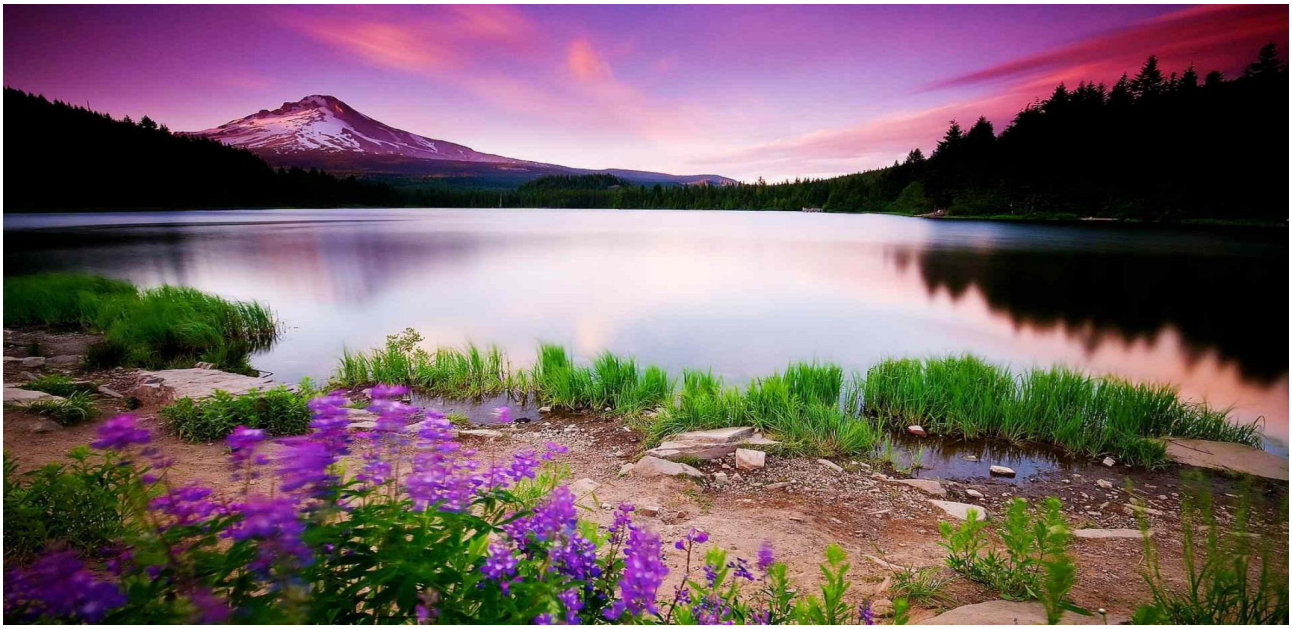
```java
// brightness
public static BufferedImage Ibrightness(BufferedImage inImage, int a) {
    int height = inImage.getHeight();
    int width = inImage.getWidth();
    BufferedImage output = new BufferedImage(width, height, BufferedImage.TYPE_3BYTE_BGR);
    for (int i = 0; i < height; i++) {
  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

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

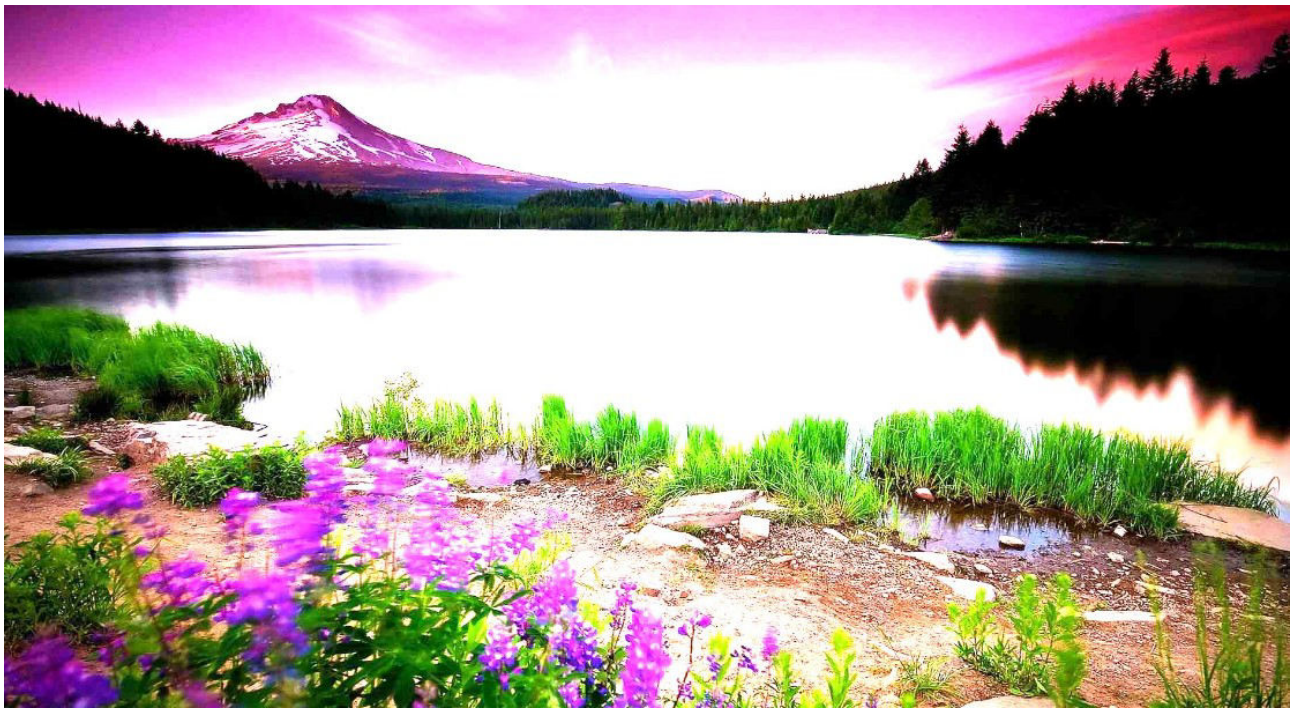
how code works:

- 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.

SAMPLE INPUT:



SAMPLE OUTPUT:



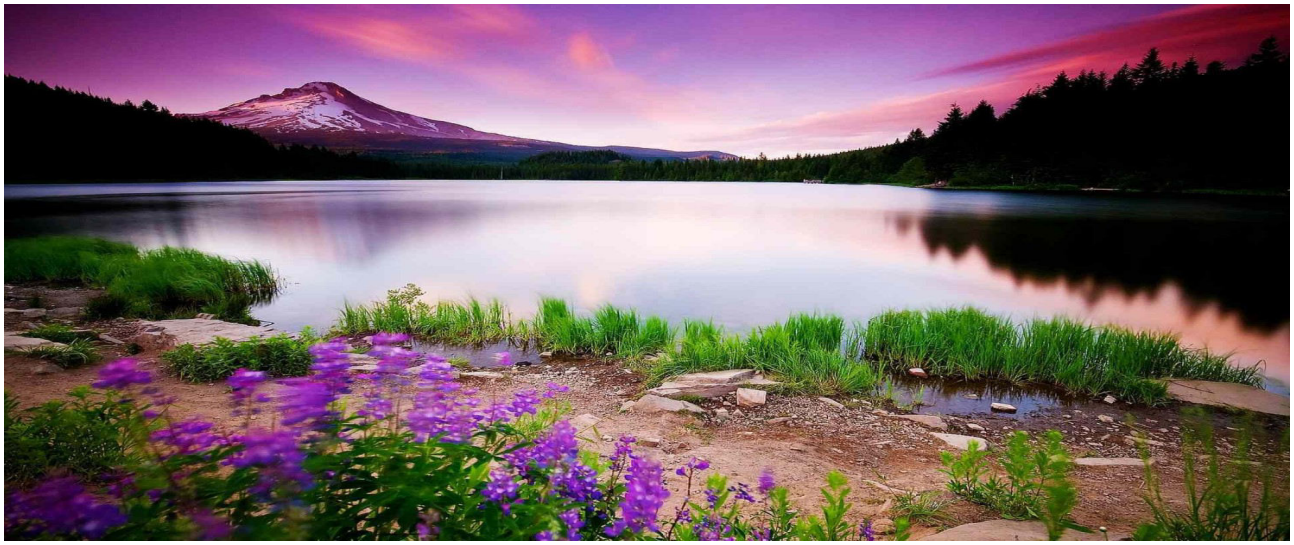
8.Blur Image

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

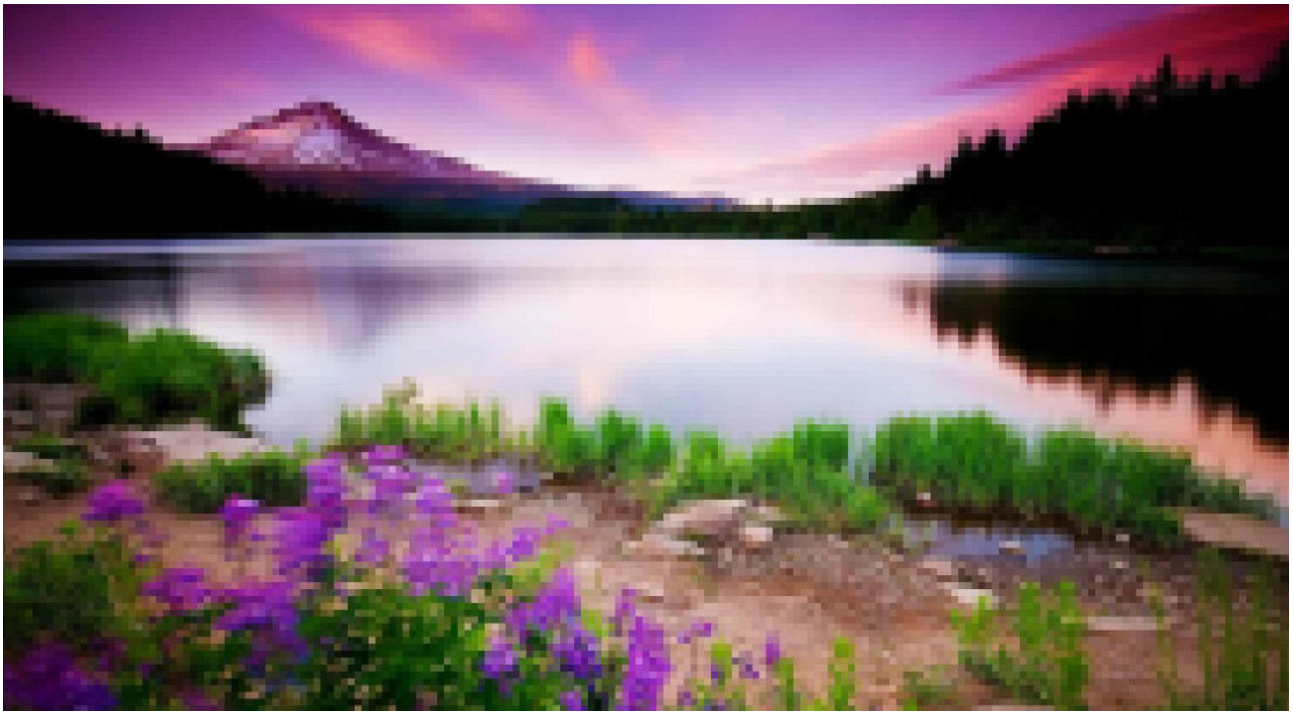
how code works:

- *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.

Sample input:



Sample output:



Packages Imported or Used:-

java.io.File: For working with files and file paths.

java.io.IOException: For handling exceptions related to input/output operations.

java.awt.image.BufferedImage: For representing and manipulating images.

javax.imageio.ImageIO: For reading and writing images.

java.awt.*: AWT (Abstract Window Toolkit) classes for graphical user interface (GUI) elements.

java.util.*: Java utility classes, potentially for future use.

java.lang.*: The core Java language classes, typically imported by default.

Known Issues:

Blurring large images may result in performance issues.