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.

- INPUT IMAGE:

Features:

*1.**Convert to Gray Scale (Option 1):***

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

2. Vertical Invert (Option 2):

- **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.*

3.Horizontal Invert (Option 3):

- **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.

4.Left Rotation (Option 4):

- **Usage:** This feature rotates the input image 90 degrees to the left (counterclockwise). It effectively turns the image on its side.

5.Right Rotation (Option 5):

- **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.

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.

Code Documentation:

1. `Convert To Grayscale`

```
public static BufferedImage convertToGray(BufferedImage input) {
    int height = input.getHeight();
    int width = input.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, input.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 :

2. **`Right Rotated Image`**

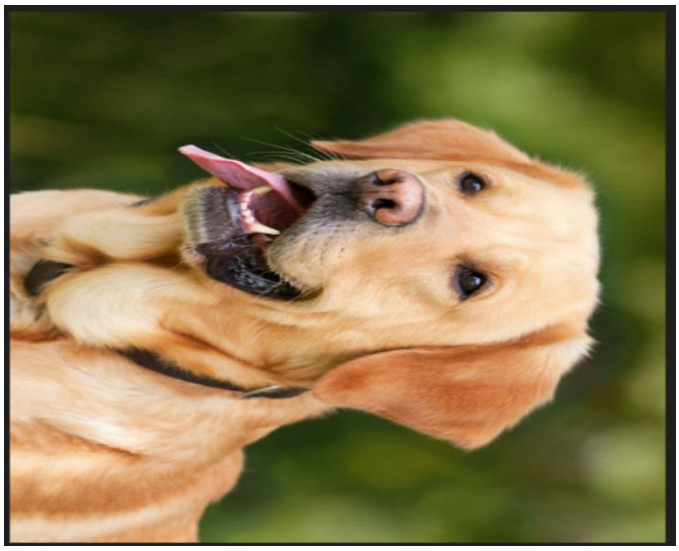
```java
public static BufferedImage rightRotate(BufferedImage input) {
    int height = input.getHeight();
    int width = input.getWidth();
    BufferedImage OutputImage = new BufferedImage(height, width, BufferedImage.TYPE_3BYTE_BGR);
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            OutputImage.setRGB(i, j, input.getRGB(j, i));
        }
    }
    int cols = OutputImage.getWidth() - 1;
    for (int i = 0; i < OutputImage.getHeight(); i++) {
        for (int j = 0; j < (OutputImage.getWidth() / 2); j++) {
            Color pixel = new Color(OutputImage.getRGB(j, i));
            OutputImage.setRGB(j, i, OutputImage.getRGB(cols - j, i));
            OutputImage.setRGB(cols - j, i, pixel.getRGB());
        }
    }
    return OutputImage;
}
```

- **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.



3. **`Left Rotated Image`**

```java
public static BufferedImage leftRotate(BufferedImage input) {
    int height = input.getHeight();
    int width = input.getWidth();
    BufferedImage OutputImage = new BufferedImage(height, width, BufferedImage.TYPE_3BYTE_BGR);
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            OutputImage.setRGB(i, j, input.getRGB(j, i));
        }
    }
    int rows = OutputImage.getHeight() - 1;
    for (int j = 0; j < OutputImage.getWidth(); j++) {
        for (int i = 0; i < OutputImage.getHeight() / 2; i++) {
            Color pixel = new Color(OutputImage.getRGB(j, i));
            OutputImage.setRGB(j, i, OutputImage.getRGB(j, rows - i));
            OutputImage.setRGB(j, rows - i, pixel.getRGB());
        }
    }
    return OutputImage;
}
```

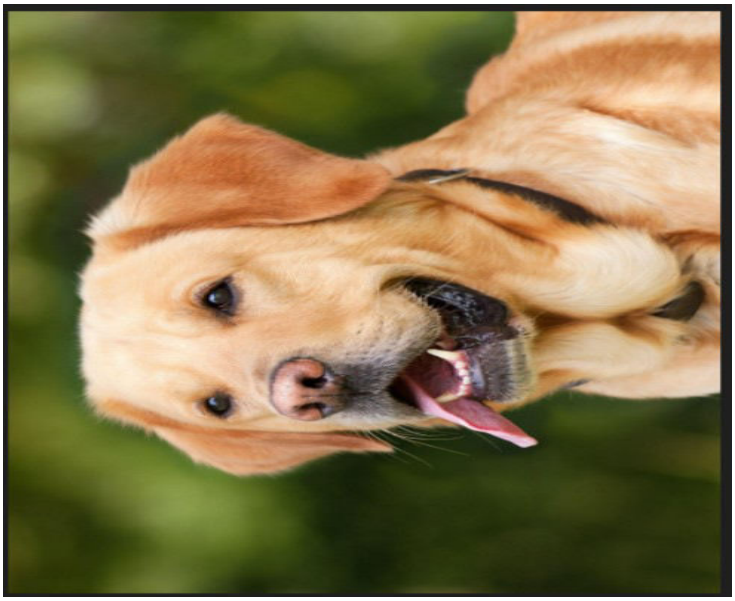
- **Description:** This function performs a left 90-degree rotation of the input image.

- **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 :



4. **`Horizontally Invert`**

```java
public static BufferedImage horizantalInvert(BufferedImage input) {
    int height = input.getHeight();
    int width = input.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(width - 1 - j, i, input.getRGB(j, i));
        }
    }
    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 :

5. **`Vertically Invert`**

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

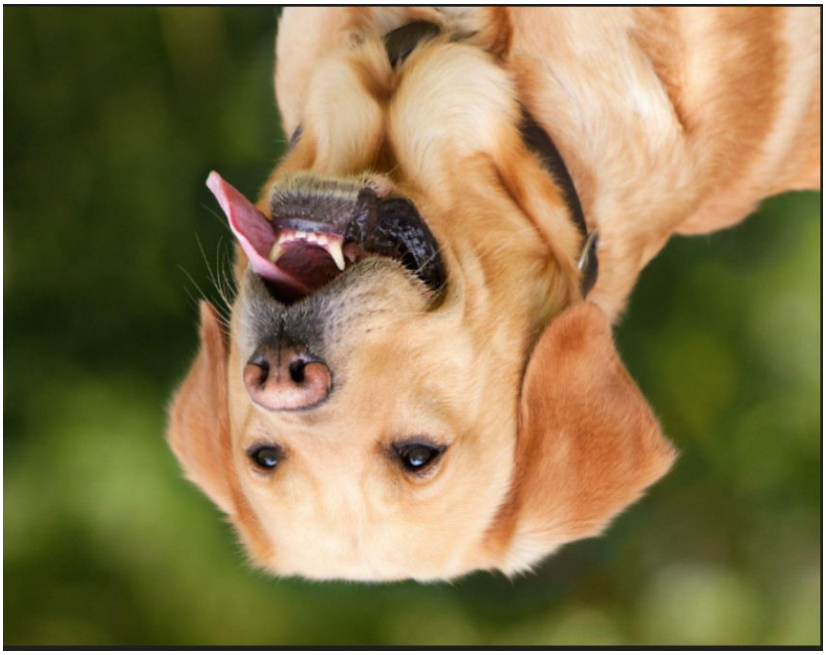
How code works:

- 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.

```
public static BufferedImage verticalInvert(BufferedImage input) {
    int height = input.getHeight();
    int width = input.getWidth();
    BufferedImage OutputImage = new BufferedImage(width, height,BufferedImage.TYPE_3BYTE_BGR);
    for (int j = 0; j < width; j++) {
        for (int i = 0; i < height / 2; i++) {
            Color temp = new Color(input.getRGB(j, i));
            OutputImage.setRGB(j, i, input.getRGB(j, height - 1 - i));
            OutputImage.setRGB(j, height - 1 - i, temp.getRGB());
        }
    }
    return OutputImage;
}
```

- 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 :



6. **`Image brightness`**

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

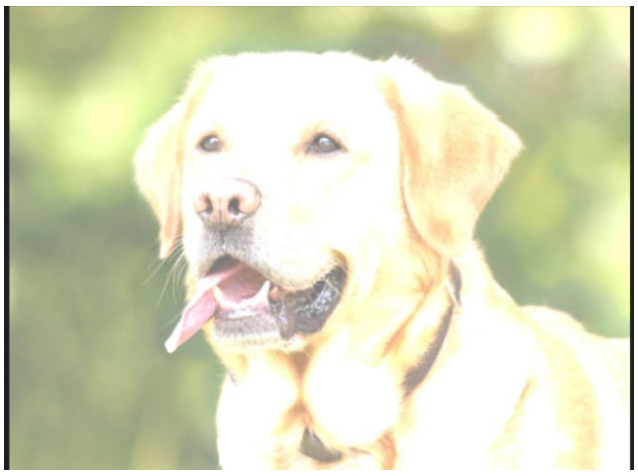
How code works:

- 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.`

```java
public static BufferedImage changeBrightness(BufferedImage input, int Brightness) {
    int height = input.getHeight();
    int width = input.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(input.getRGB(j, i));
            int red = pixel.getRed();
            int green = pixel.getGreen();
            int blue = pixel.getBlue();
            red += Brightness;
            green += Brightness;
            blue += Brightness;
            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);
            OutputImage.setRGB(j, i, newpixel.getRGB());
        }
    }
    return OutputImage;
}
```

OUTPUT:



7. Blurred Image:

Description: This function adjusts the blurness of the input image based on specified percentage of increase or decrease.

`How this code works:`

→ It takes two parameters: the input image and the number of pixels to use for the blur.

→ The function first creates a new output image with the same dimensions as the input image.

→ Then, it iterates over the input image, averaging the RGB values of each pixel's neighbors and storing the average value in the corresponding pixel in the output image.

→ The function returns the blurred output 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;
}
```

OUTPUT:

Packages Used:

1. java.awt : Contains classes for creating and managing windows frames and graphical elements.

2. java.awt.image : Provides classes for creating and modifying images.

3. javax.imageIo : Enables reading and writing of images in various formats.

4. java.io : Provides classes for input and output operations.

5. java.util : Provides scanner for user inputs.