Image Editor Documentation

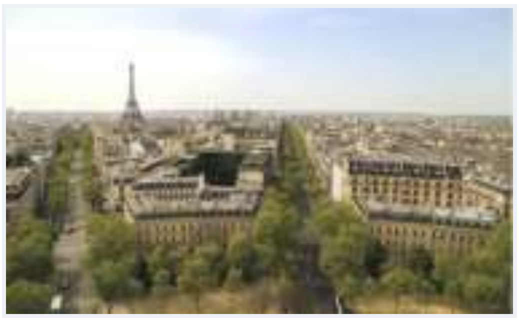
The Image Editor developed is a command line image editor made with Java. You can use its cool basic features to edit your images relentlessly. The features of this editor is described below. Please be free to comment on the changes needed to made in the editor.

Test Image

Features of the editor -:

Change brightness of your image → (BufferedImage Brightentheimg())

It is a feature which is used to increase the brightness of your image. This function takes an input image and a percentage increase value, and it iterates through each pixel in the image to adjust the color intensity. It ensures that the color components remain within the valid range (0-255) after the adjustment and returns a new BufferedImage with the modified colors. The test output is shown below.


The image is brightened by 20%

Convert your image to Grey-scale →(BufferedImage greyscale())

It is a simple and basic feature which is used to convert your image to grey-scale image. This function takes an input BufferedImage, `inpimg`, and converts it into a grayscale image. It creates a new BufferedImage, `outimg`, of type `BufferedImage.TYPE_BYTE_GRAY` with the same dimensions as the input image. Then, it iterates through each pixel of the input image and sets the corresponding pixel value in the output image to convert it to grayscale. Finally, it returns the grayscale BufferedImage.The test output is shown below.

Grey-Scale Image

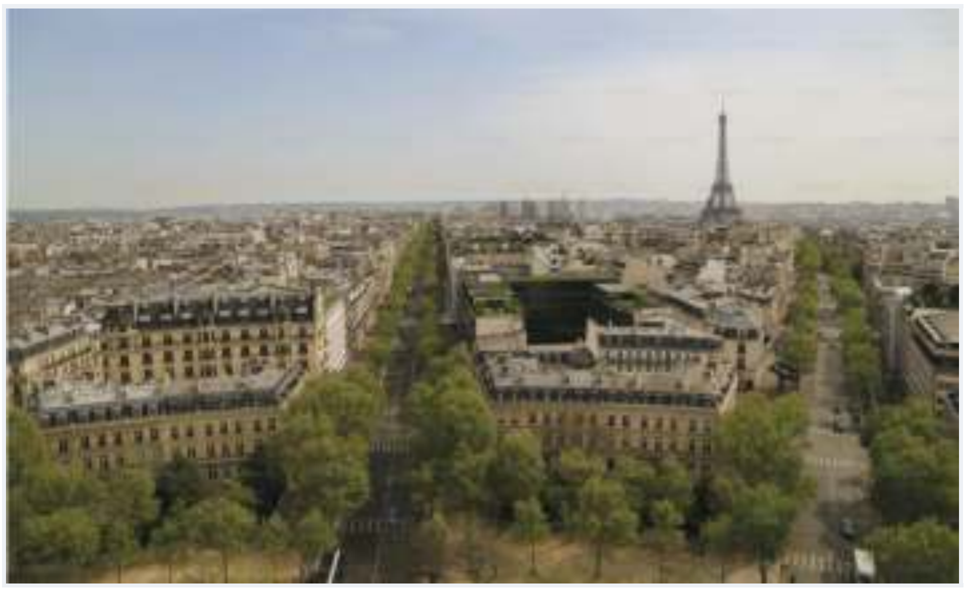
Blur your Image → (BufferedImage Blur())

It is a feature which is used to blur your image. This function takes an input image (`inpimg`) and an amount of blur (`amtOfBlur`) as parameters. It iterates through the input image in blocks of size specified by `amtOfBlur`, calculates the average color values for each block, and creates a blurred output image (`outimg`) by replacing each block with the calculated average color. Finally, it returns the blurred image as a `BufferedImage`.The test output is shown below.

Test output image

To print the mirror image → (BufferedImage mirrorimg())

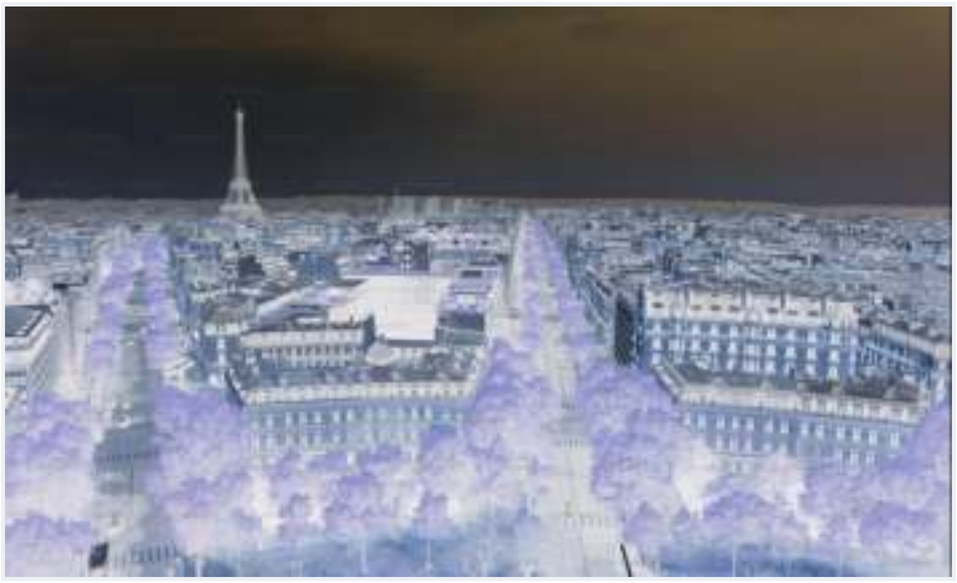
It is a feature which is used to print the mirror(image) your image.This function takes an input image (`inpimg`) and creates a new BufferedImage (`outimg`) with the same dimensions and type. It then iterates through each row and half of the columns in the input image, mirroring the pixels from the right side to the left side and vice versa to create the mirrored image. Finally, it returns the horizontally mirrored image.The test output is shown below.



Test output image

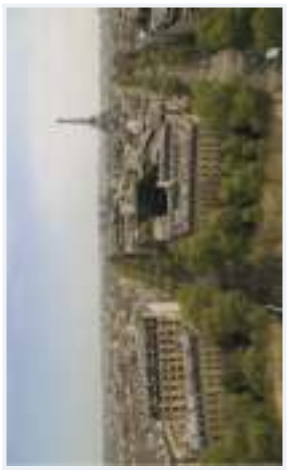
To print the inverted color image → (BufferedImage colorinversion())

It is a feature which is used to print the inverted color of your image. This function takes an input image, iterates through each pixel, and inverts its color by subtracting each RGB component from 255 (the maximum value for each component). It then creates a new image with the inverted colors and returns it. The function uses nested loops to traverse each pixel and several variables to store and manipulate pixel colors.The test output is shown below.


Test output image

To rotate your image 90deg Anti-clockwise → (BufferedImage AntiClockwise())

It is a feature which is used to rotate your image in anti-clockwise direction.This function takes an image as input, creates a new image with swapped dimensions, and then iterates through the pixels of the input image, copying them to the appropriate location in the output image to achieve a 90-degree anticlockwise rotation.The test output is shown below.


Test Output Image

To rotate your image 90deg Clockwise → (BufferedImage Clockwise())

It is a feature which is used to rotate your image in clockwise direction.This function takes an input `BufferedImage` called `inpimg` as an argument.It extracts the height and width of the input image.It creates a new `BufferedImage` called `outimg` with swapped height and width to store the rotated image.The first loop (nested within the second loop) iterates through each pixel in the input image, and for each pixel, it does the following:

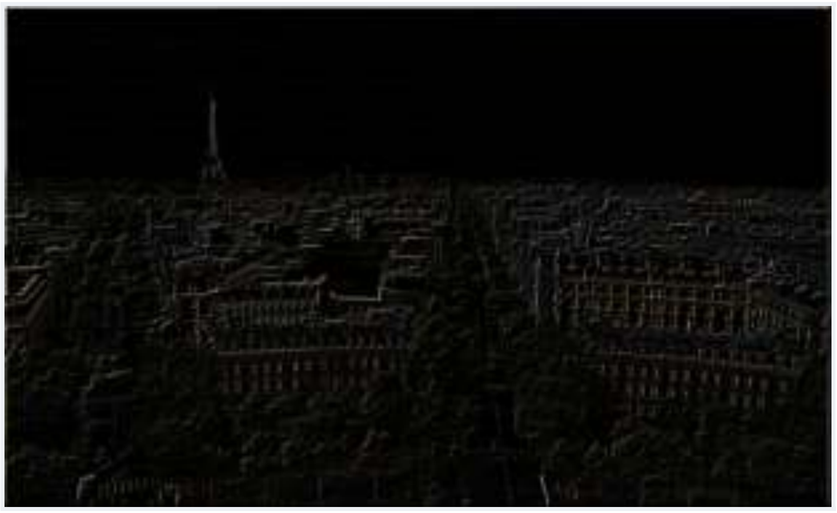
- Retrieves the color of the pixel from the input image.
- Sets the color of the corresponding pixel in the output image with swapped x and y coordinates, effectively transposing the image.

After the transposition, it calls the `mirrorimg` function (not provided in the code snippet) to flip the rotated image horizontally.Finally, it returns the rotated and mirrored image as `outimg`.The test output is shown below.

Test Output Image

To print the edge detected image → (BufferedImage edgeDetectionimg())

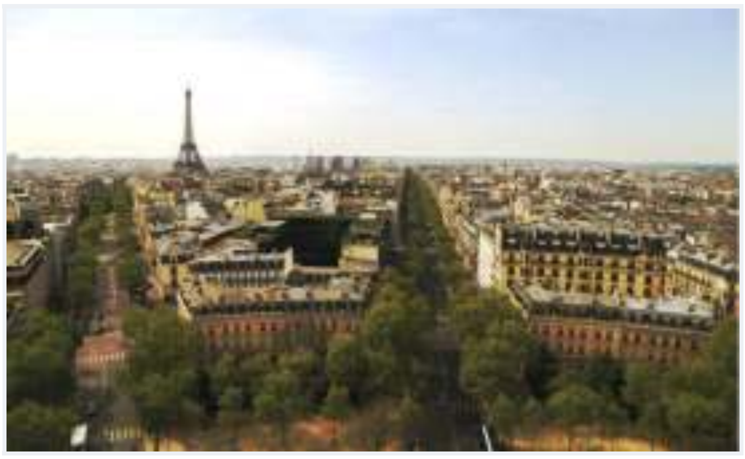
This function takes an input image and creates an output image with edge detection applied. It copies the input image to the output image and then calculates the differences in pixel values in a 5x5 neighborhood to detect edges. Finally, it sets the modified pixel values in the output image and returns the result. The test output is shown below.



Test Output Image

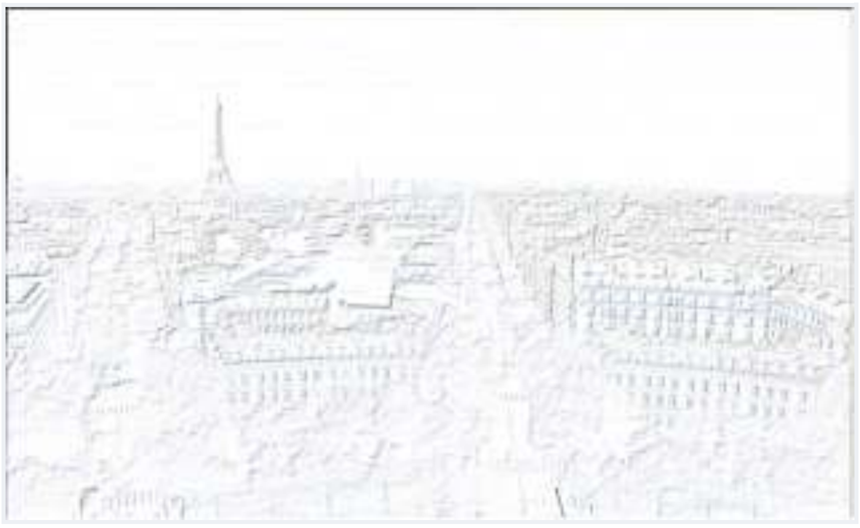
To print the contrast of a image → (BufferedImage contrastimg())

This function iterates through each pixel in the input image, adjusts its color components to increase or decrease contrast, and stores the result in the output image. It performs this adjustment separately for the red, green, and blue channels while ensuring the adjusted values stay within the valid color component range. The test output is shown below.


Test Output Image

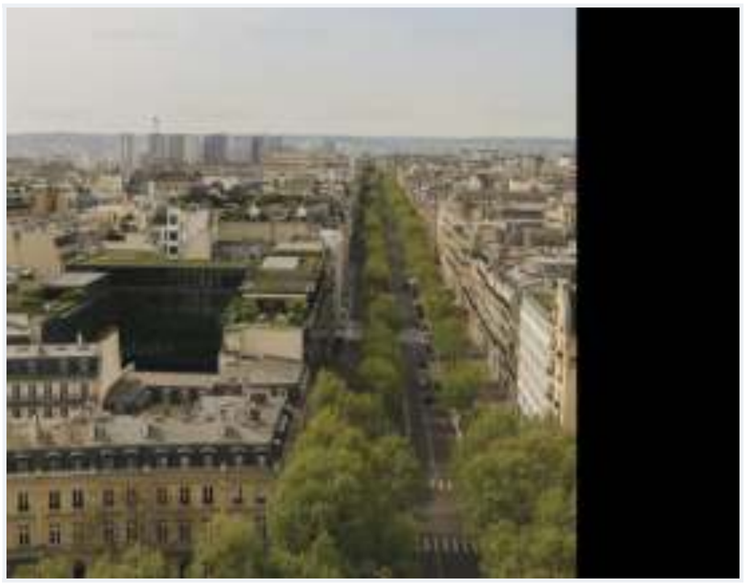
To print your image as sketch → (BufferedImage pencilSketchimg())

The function `pencilSketchimg` primarily acts as a driver for calling these two sub-functions (`edgeDetectionimg` and `colorinversion`) to create a pencil sketch effect. The specifics of how the loops within these sub-functions work would depend on the implementation details of those functions, which are not provided in the code snippet. . The test output is shown below.



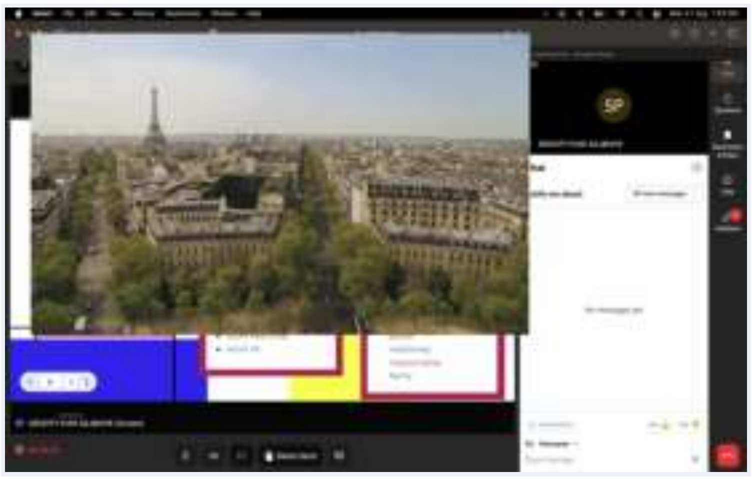
To Crop your image → (BufferedImage crop())

The provided function `crop` takes an input `BufferedImage` and an integer `choice` as arguments and returns a modified `BufferedImage` based on the choice made. The function performs cropping in three different shapes: a circle, a square, or a rectangle, depending on the value of the `choice` parameter. Finally, the function returns the `outimg` containing the cropped image according to the user's choice. The test output is shown below.


Test Output Image

To insert a image over the other background image → (BufferedImage addimg())

This function takes an input image `inpimg`, prompts the user for the path and position to insert another image, checks if the insertion position is valid, and then adds the second image to the first image at the specified position. If any errors occur during the process, it handles them and returns the resulting image. The test output is shown below.


Test Output Image

These features are called in the main function using a switch case reading the choice from the user and executing the program.