Documentation for Command line-based Image Editor

- Souvik Kumar

Table of Contents

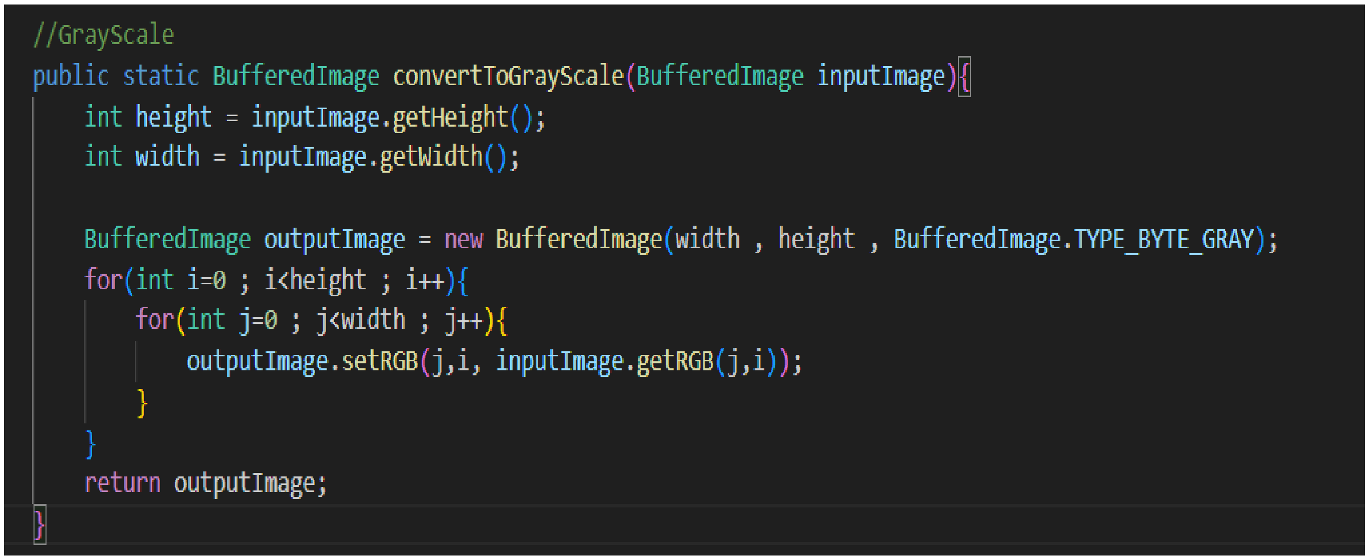
Overview

The Image Editor is a Java-based command-line application that allows users to perform various image manipulation operations on digital images. This comprehensive tool offers a range of functionalities, including converting images to grayscale, rotating images, adjusting brightness, inverting colors, and applying a blur effect. Users can choose from these operations to enhance and modify their images as needed.

Functionalities

- Convert to Grayscale
- Rotate 90 degrees Clockwise
- Rotate 90 degrees Anticlockwise
- Adjust Brightness
- Invert the Image
- Blur the Image

★ **Convert to Grayscale**



```
//GrayScale
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;
}
```

-->This function takes a BufferedImage as input and converts it into a grayscale bufferedImage by using the type 'BufferedImage.TYPE_BYTE_GRAY'.

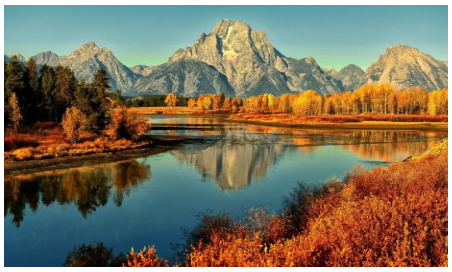
Creates a new BufferedImage with the same dimensions as the input image.

It iterates through each pixel of the input image and retrieves the RGB color value of each pixel.

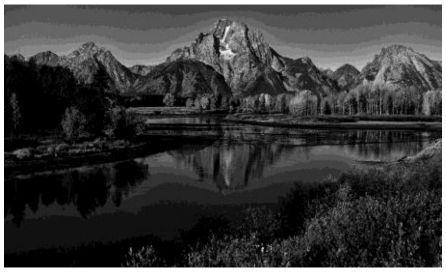
It then sets the corresponding pixel value in the grayscale image using the same RGB value, which gets automatically converted to grayscale.

Then it returns the resulting grayscale image as output.

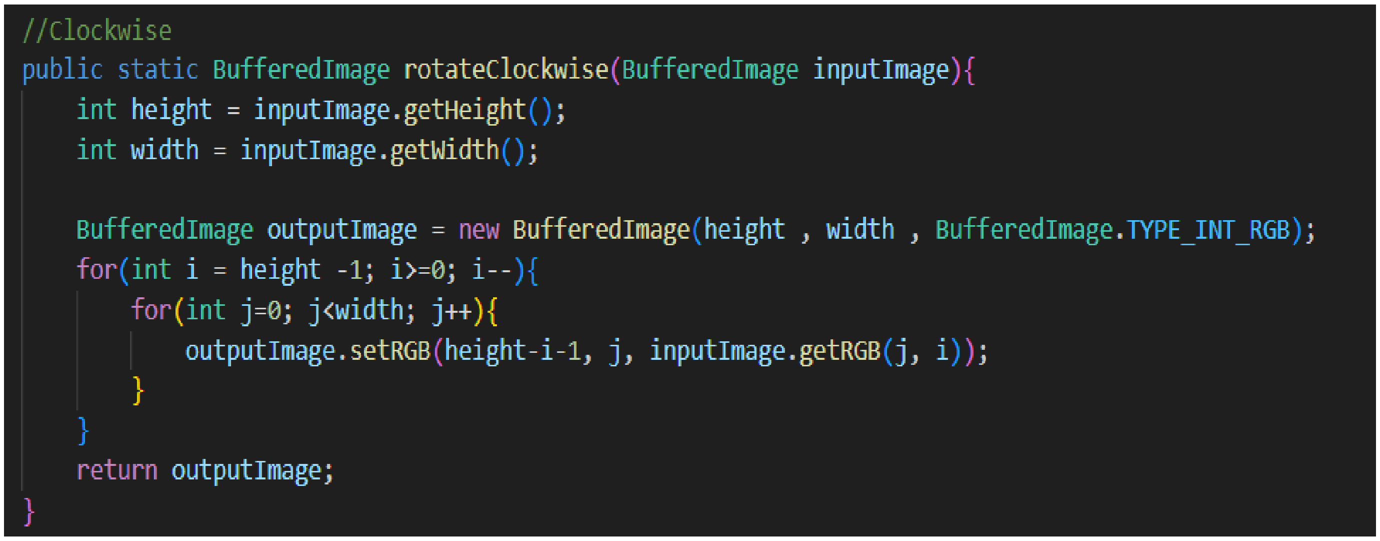
Original Image



Grayscale Image



★ Rotate 90 degrees Clockwise



```java
//Clockwise
public 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 = height -1; i>=0; i--){
        for(int j=0; j<width; j++){
            outputImage.setRGB(height-i-1, j, inputImage.getRGB(j, i));
        }
    }
    return outputImage;
}
```

--> This function takes an input image and rotates it 90 degrees clockwise

This code takes an input image and creates an empty output image.

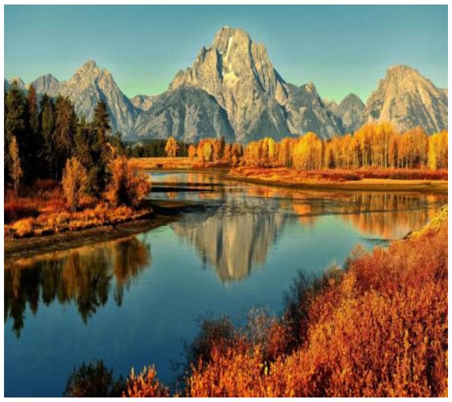
Then it finds the dimensions of the input image.

Using nested loops, it iterates through all the pixels of the input image.

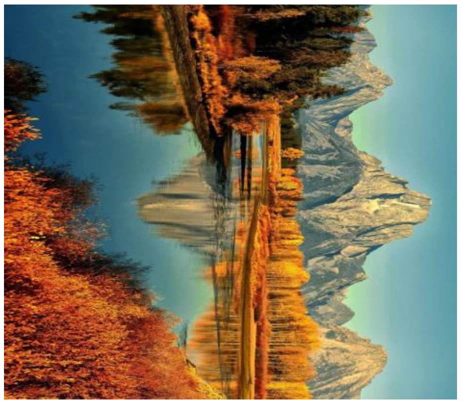
For each pixel it calculates its new position in the rotated image (90 degrees clockwise) and assigns the corresponding color.

The code returns the rotated image as a new ‘BufferedImage’, effectively rotating the input image 90 degrees clockwise.

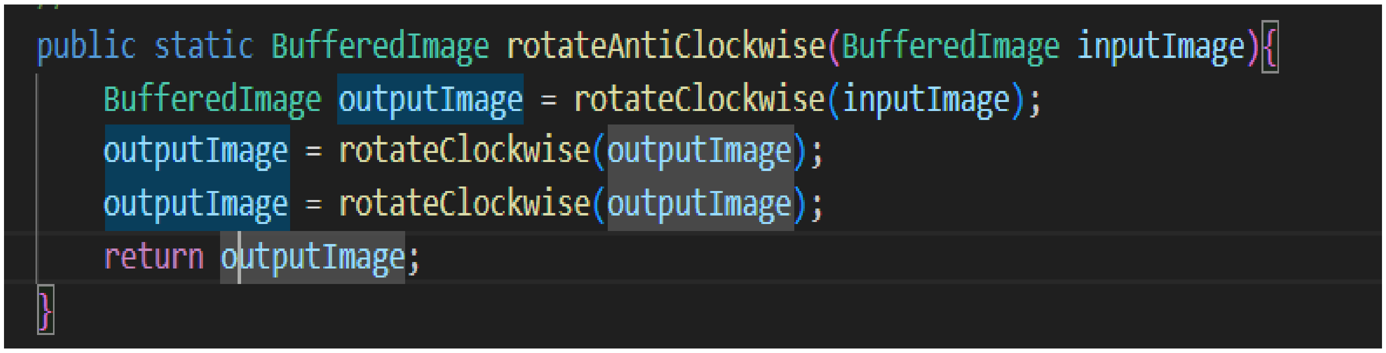
Original Image



Rotated 90 degrees clockwise



★ Rotate 90 degrees Anti-clockwise



```
public static BufferedImage rotateAntiClockwise(BufferedImage inputImage){
    BufferedImage outputImage = rotateClockwise(inputImage);
    outputImage = rotateClockwise(outputImage);
    outputImage = rotateClockwise(outputImage);
    return outputImage;
}
```

--> This function takes an input image and rotates it 90 degrees anti-clockwise

This code takes an input image and creates an empty output image.

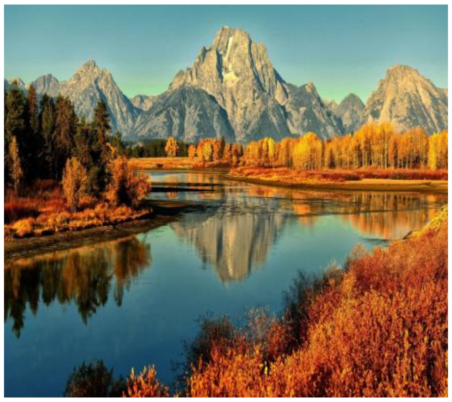
Then it finds the dimensions of the input image.

Using nested loops, it iterates through all the pixels of the input image.

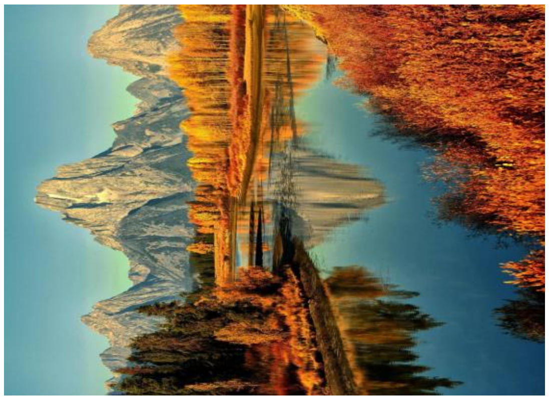
For each pixel it calculates its new position in the rotated image (90 degrees clockwise) and assigns the corresponding color.

The code returns the rotated image as a new 'BufferedImage', effectively rotating the input image 90 degrees clockwise three times consecutively.

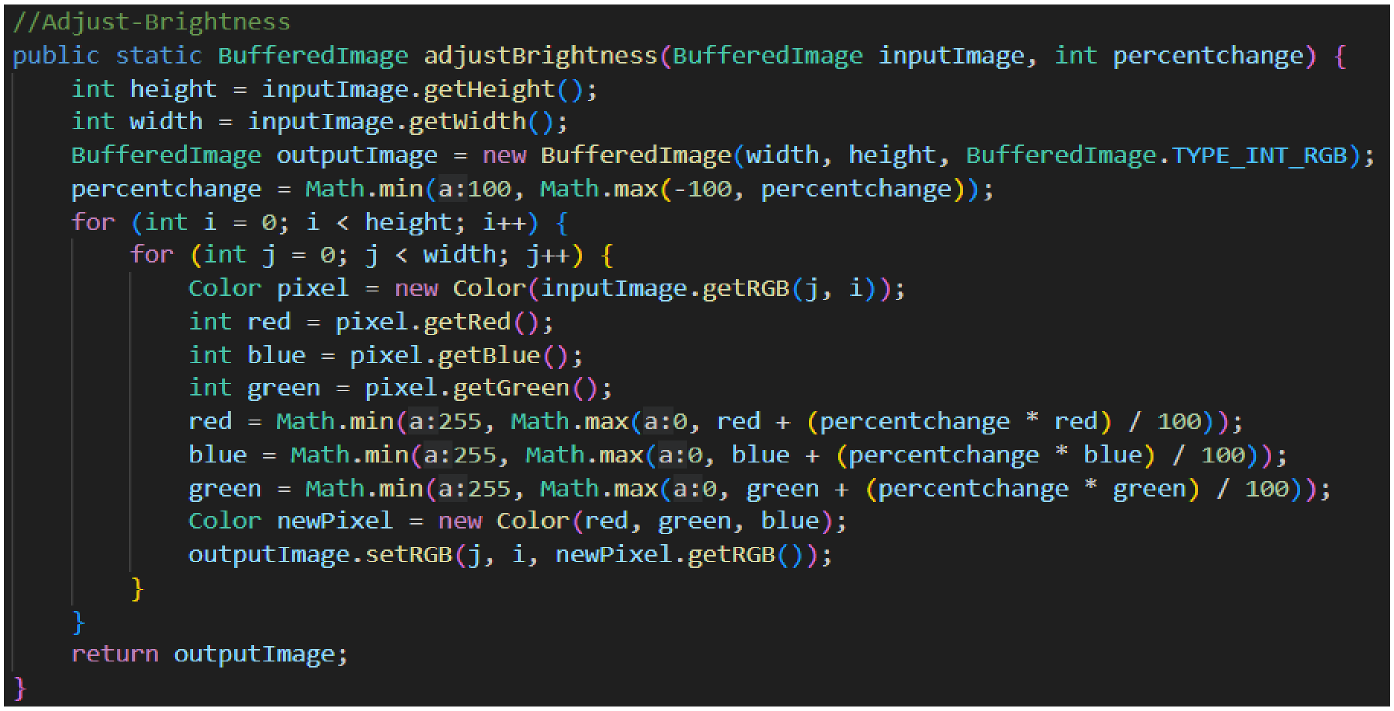
Original Image



Rotated Anti-Clockwise



★ Adjust Brightness



```
//Adjust-Brightness
public static BufferedImage adjustBrightness(BufferedImage inputImage, int percentchange) {
    int height = inputImage.getHeight();
    int width = inputImage.getWidth();
    BufferedImage outputImage = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);
    percentchange = Math.min(a:100, Math.max(-100, percentchange));
    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 = Math.min(a:255, Math.max(a:0, red + (percentchange * red) / 100));
            blue = Math.min(a:255, Math.max(a:0, blue + (percentchange * blue) / 100));
            green = Math.min(a:255, Math.max(a:0, green + (percentchange * green) / 100));
            Color newPixel = new Color(red, green, blue);
            outputImage.setRGB(j, i, newPixel.getRGB());
        }
    }
    return outputImage;
}
```

--> This function is used to change the brightness of the input image by a specified percentage.

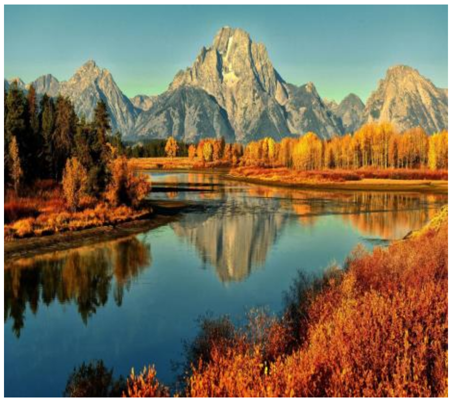
The code takes an input image and creates an empty image to store the adjusted image.

It finds the dimensions of the input image.

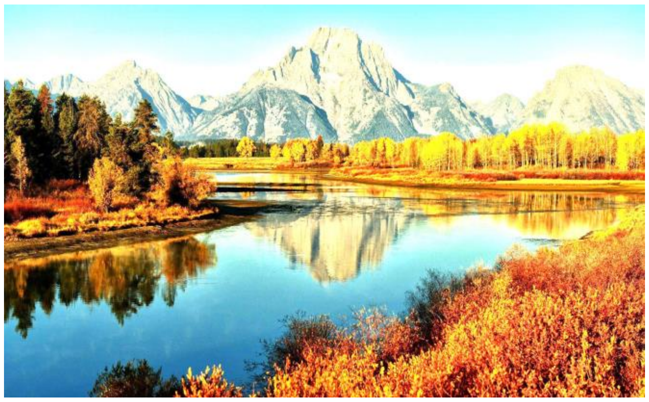
Using nested loops, it iterates over all pixels of the input image.

The code returns the adjusted image as a new ‘BufferedImage’, effectively changing the image’s brightness according to the given percentage.

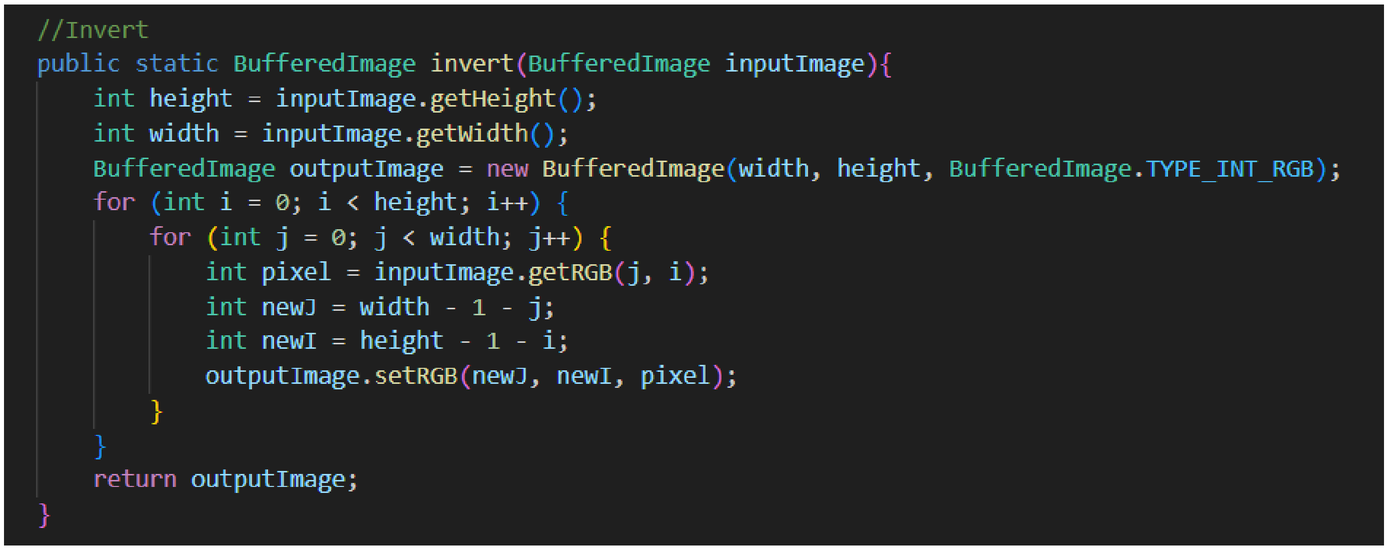
Original Image



Brightness Adjusted



★ Invert the Image



```java
//Invert
public static BufferedImage invert(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++) {
            int pixel = inputImage.getRGB(j, i);
            int newJ = width - 1 - j;
            int newI = height - 1 - i;
            outputImage.setRGB(newJ, newI, pixel);
        }
    }
    return outputImage;
}
```

--> Inverts a buffered Image by flipping it both horizontally and vertically.

Gets the dimensions of the image.

Create a new BufferedImage to store the inverted image.

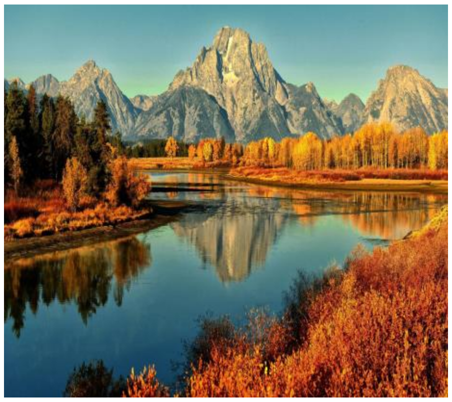
Iterate through each pixel in the input image and invert its position in the output image.

Calculate the new position for the pixel in the inverted image

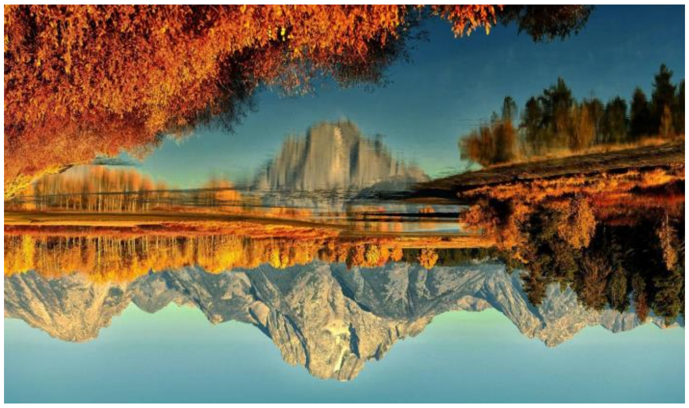
Set the pixel in the inverted image

Return the inverted image

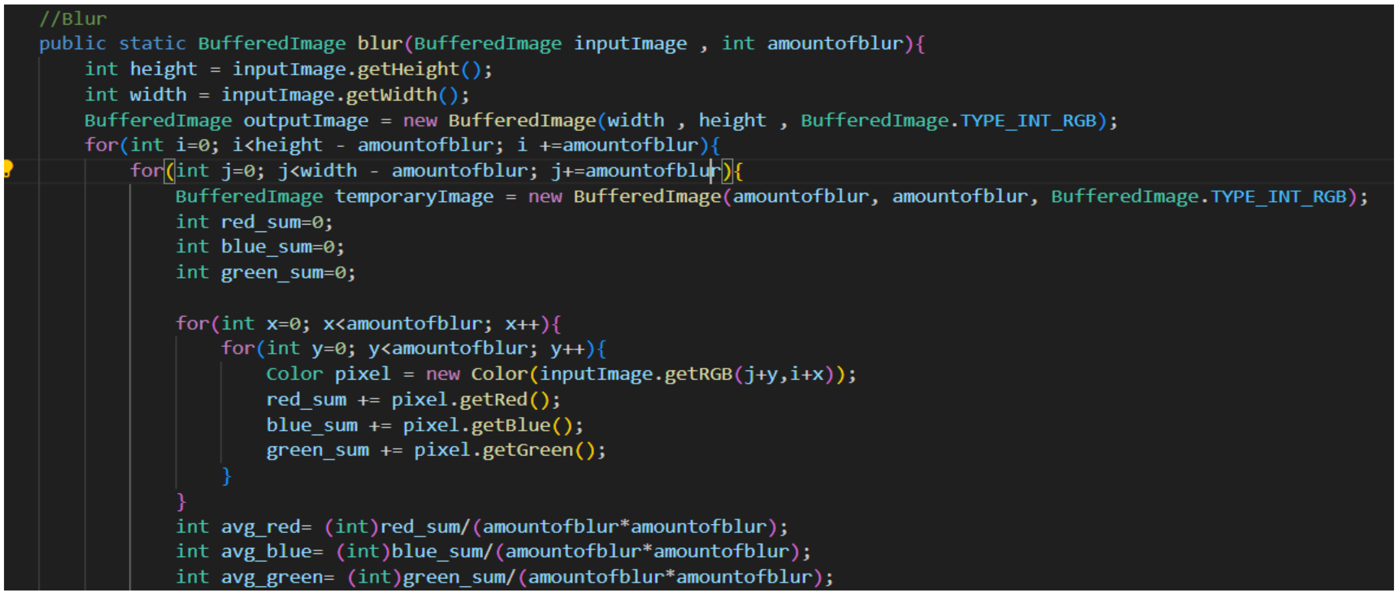
Original Image



Inverted Image

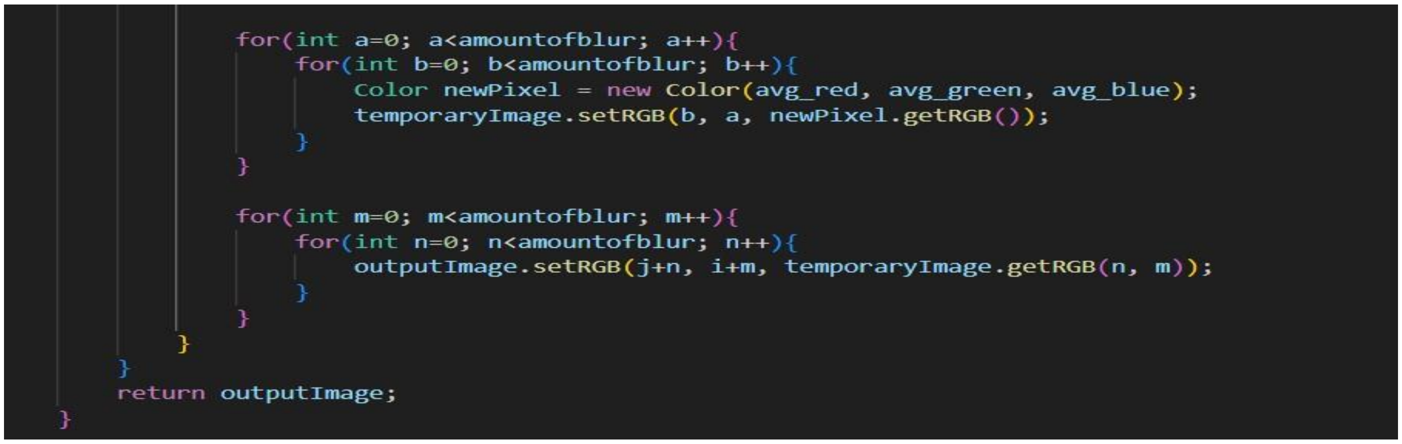


★ Blur the Image



```
//Blur
public static BufferedImage blur(BufferedImage inputImage , int amountofblur){
    int height = inputImage.getHeight();
    int width = inputImage.getWidth();
    BufferedImage outputImage = new BufferedImage(width , height , BufferedImage.TYPE_INT_RGB);
    for(int i=0; i<height - amountofblur; i +=amountofblur){
        for(int j=0; j<width - amountofblur; j+=amountofblur){
            BufferedImage temporaryImage = new BufferedImage(amountofblur, amountofblur, BufferedImage.TYPE_INT_RGB);
            int red_sum=0;
            int blue_sum=0;
            int green_sum=0;

            for(int x=0; x<amountofblur; x++){
                for(int y=0; y<amountofblur; y++){
                    Color pixel = new Color(inputImage.getRGB(j+y,i+x));
                    red_sum += pixel.getRed();
                    blue_sum += pixel.getBlue();
                    green_sum += pixel.getGreen();
                }
            }
            int avg_red= (int)red_sum/(amountofblur*amountofblur);
            int avg_blue= (int)blue_sum/(amountofblur*amountofblur);
            int avg_green= (int)green_sum/(amountofblur*amountofblur);
```



```
            for(int a=0; a<amountofblur; a++){
                for(int b=0; b<amountofblur; b++){
                    Color newPixel = new Color(avg_red, avg_green, avg_blue);
                    temporaryImage.setRGB(b, a, newPixel.getRGB());
                }
            }

            for(int m=0; m<amountofblur; m++){
                for(int n=0; n<amountofblur; n++){
                    outputImage.setRGB(j+n, i+m, temporaryImage.getRGB(n, m));
                }
            }
        }
    }
    return outputImage;
}
```

--> Applies a blur effect to a BufferedImage.

Get the height and width of the input image.

Create a new BufferedImage to store the blurred image.

Iterate through the input image, applying blur in chunks.

Create a temporary BufferedImage to store the blurred chunk.

Copy the blurred chunk back to the output Image.

Return the blurred image.

Original Image

Blurred Image