IMAGE EDITOR

IMPORT STATEMENTS

import statements allow us to use the specified classes in the ImageEditor class.

- **java.awt.image.BufferedImage - Allows us to load images into BufferedImage objects**
- **java.io.File - Used for reading/writing files**
- **java.util.Scanner - For reading input from the user**
- **javax.imageio.ImageIO - Contains methods for loading/saving images**
- **java.io.IOException - Checked exception thrown by image reading methods**
-

PSVM

```
public static void main(String[] args) throws Exception {

        Scanner sc = new Scanner(System.in);

        System.out.print("Enter name of the image: ");
        String imagePath = sc.nextLine();

        File imgFile = new File(imagePath);
        try {
            BufferedImage image = ImageIO.read(imgFile);
            System.out.println("Enter 1 for grayscale");
            System.out.println("Enter 2 to rotate image CLOCKWISE");
            System.out.println("Enter 3 to rotate image ANTI-CLOCKWISE");
            System.out.println("Enter 4 to INVERT IMAGE HORIZANTALLY");
            System.out.println("Enter 5 to INVERT IMAGE VERTICALLY");
            System.out.println("Enter 6 to CHANGE BRIGHTNESS OF THE IMAGE");
            System.out.println("Enter 7 to TO BLUR THE IMAGE");

            int choice = sc.nextInt();

            switch (choice) {
                case 1:
                    image = grayscale(image);
                    break;

                case 2:
                    image = rotateClockwise(image);
```

```
                    break;

                case 3:
                    image = rotateAntiClockwise(image);
                    break;
                case 4:
                    image = invertHorizontal(image);
                    break;
                case 5:
                    image = invertVertical(image);
                    break;
                case 6:
                    System.out.println("Amount by which brightness should be
increased");
                    int amount = sc.nextInt();
                    image = changeBrightness(image, amount);
                    break;
                case 7:
                    image = blur(image);
                    break;
                default:
                    System.out.println("Invalid choice");
                    break;
            }
            ImageIO.write(image, "jpg", new File("output.jpg"));
        }
        catch(IOException e){
            throw new RuntimeException(e);
        }
    }

}
```

This code allows the user to load an image file, apply various image processing operations like grayscale, rotation, brightness change etc, and save the result to an output file.

It uses a Scanner to read the image file path and choice from the user. The File and ImageIO classes load the image into a BufferedImage.

Based on the user's choice, it calls appropriate methods like grayscale(), rotateClockwise() etc to process the image and save it using ImageIO.write().

This code allows the user to load an image file, apply various image processing operations like grayscale, rotation, brightness change etc, and save the result to an output file.

It uses a Scanner to read the image file path and choice from the user. The File and ImageIO classes load the image into a BufferedImage.

Based on the user's choice, it calls appropriate methods like grayscale(), rotateClockwise() etc to process the image and save it using ImageIO.write(g

Grayscale

```
public static BufferedImage grayscale(BufferedImage img) {
   int height = img.getHeight();
   int width = img.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, img.getRGB(j, i));
       }
   }
   return outputImage;
}
```

This method takes a BufferedImage and converts it to grayscale.

It first gets the width and height of the original image.

Then it creates a new BufferedImage of the same size and sets it to store grayscale values (TYPE_BYTE_GRAY).

It loops through every pixel in the original image. For each pixel, it gets the RGB color components using getRGB().

It calculates a grayscale value by averaging the R, G, and B components.

It sets this new grayscale value as the RGB components in the output image using setRGB().

Finally, it returns the new grayscale image.

So it basically converts each pixel from color to a grayscale shade, creating a grayscale version of the original image.

ROTATE ANTICLOCKWISE

```
public static BufferedImage rotateAntiClockwise(BufferedImage img) {
   int width = img.getWidth();
   int height = img.getHeight();

   BufferedImage rotated = new BufferedImage(height, width, img.getType());

   for(int i = 0; i < height; i++) {
       for(int j = 0; j < width; j++) {
           rotated.setRGB(i, width - j - 1, img.getRGB(j, i));
       }
   }

   return rotated;
}
```

- Create a new empty BufferedImage of rotated dimensions
- Loop through each pixel of original image
- To find rotated location, swap x and y coordinate and invert y
- Set the pixel color in rotated image

This rotates the image by directly mapping each pixel from original to rotated image without using Graphics2D. The key steps are swapping x and y coordinates, and inverting y to get the 90 degree anti-clockwise rotation.

BLUR

```
public static BufferedImage blur(BufferedImage img) {
   int radius = 5;
   // Basic blur implementation
   for(int i=0; i<img.getWidth(); i++) {
```

```
        for(int j=0; j<img.getHeight(); j++) {
            int rgb = 0;
            int blurPixelCount = 0;

            for(int k=-radius; k<=radius; k++) {
                for(int l=-radius; l<=radius; l++) {
                    if(i+k >= 0 && i+k < img.getWidth() && j+l >= 0 && j+l <
img.getHeight()) {
                        rgb += img.getRGB(i+k, j+l);
                        blurPixelCount++;
                    }
                }
            }
            rgb = rgb/blurPixelCount;
            img.setRGB(i, j, rgb);
        }
    }
    return img;
}
```

- Create a uniform kernel with radius 5, where each value is 1/(size*size)
- Use the kernel to create a ConvolveOp filter
- Apply the ConvolveOp filter to the source image and return the result

This allows the blur() method to take a BufferedImage as input, apply the blur filter, and return the blurred image, without needing to handle loading/saving the images explicitly.

INVERT VERTICALLY

```
public static BufferedImage invertVertical(BufferedImage img) {
    int width = img.getWidth();
    int height = img.getHeight();
    int[][] pixels = new int[width][height];
    for(int y = 0; y < height; y++) {
        for(int x = 0; x < width; x++) {
            pixels[x][y] = img.getRGB(x, y);
        }
    }
```

```
   BufferedImage inverted = new BufferedImage(width, height,
BufferedImage.TYPE_INT_RGB);
   for(int y = 0; y < height; y++) {
       for(int x = 0; x < width; x++) {
           inverted.setRGB(x, height - y - 1, pixels[x][y]);
       }
   }
   return inverted;
}
```

it loops through the original image pixel-by-pixel, reads the RGB values, and writes them to the inverted image at flipped y-coordinates.

INVERT HORIZONTALLY

```
public static BufferedImage invertHorizontal(BufferedImage img) {
   int width = img.getWidth();
   int height = img.getHeight();
   BufferedImage invertedImage = new BufferedImage(width, height,
img.getType());
   for(int i=0; i<height; i++) {
       for(int j=0; j<width; j++) {
           invertedImage.setRGB(width-j-1, i, img.getRGB(j, i));
       }
   }
   return invertedImage;
}
```

This method takes a BufferedImage as input and returns a new BufferedImage with the pixels inverted horizontally.

It first gets the width and height of the input image using `getWidth()` and `getHeight()`.

It creates a new BufferedImage of the same size and type using the width, height and `getType()`.

It then loops through each pixel of the image using the nested for loops:

- The outer loop `i` goes through each row
- The inner loop `j` goes through each column in that row

For each pixel at (j, i), it gets the original RGB value using `img.getRGB(j, i)`

It then sets the RGB value on the inverted image at (width - j - 1, i) - this flips the column position horizontally.

Finally, it returns the new inverted image. So it inverts the image by flipping the column positions of the pixels while keeping the rows intact.

BRIGHTNESS

```
public static BufferedImage changeBrightness(BufferedImage img,int amount) {
   int width = img.getWidth();
   int height = img.getHeight();

   BufferedImage brightened = new BufferedImage(width, height,
BufferedImage.TYPE_INT_RGB);

   for(int y = 0; y < height; y++) {
       for(int x = 0; x < width; x++) {
           // Getting pixel brightness (0-255)
           int brightness = img.getRGB(x, y);
           // Increasing brightness
           brightness = Math.min(255, brightness + amount);
           // Setting new pixel brightness
           brightened.setRGB(x, y, brightness);
       }
   }
   return brightened;
}
```

It takes a BufferedImage and an integer amount as input.

It creates a new BufferedImage of the same width and height to hold the brightened image.

It loops through every pixel x,y coordinate in the original image.

For each pixel, it gets the RGB brightness value (0-255).

It increases the brightness by adding the amount parameter. It caps the value at 255 to avoid overflow.

The new brighter pixel value is set in the output image at the same x,y coordinate.

This continues for every pixel until the entire image is brightened.

Finally, it returns the new brightened BufferedImage.

So in summary, it manually loops through all pixels to brighten the image by increasing each pixel's RGB brightness value.

BLUR IMAGE

```
public static BufferedImage blur(BufferedImage img) {
   int radius = 5;
   // Basic blur implementation
   for(int i=0; i<img.getWidth(); i++) {
       for(int j=0; j<img.getHeight(); j++) {
           int rgb = 0;
           int blurPixelCount = 0;

           for(int k=-radius; k<=radius; k++) {
               for(int l=-radius; l<=radius; l++) {
                   if(i+k >= 0 && i+k < img.getWidth() && j+l >= 0 && j+l <
img.getHeight()) {
                       rgb += img.getRGB(i+k, j+l);
                       blurPixelCount++;
                   }
               }
           }
           rgb = rgb/blurPixelCount;
           img.setRGB(i, j, rgb);
       }
   }
   return img;
}
```

The blur() method takes a BufferedImage as input and returns a blurred version of that image.

It starts by defining an integer radius, which controls how much blurring is applied.

It then has two nested for loops going through every pixel in the image, with i and j as the x and y coordinates.

For each pixel, it calculates a blurred RGB value by looking at other pixels around it within the blur radius.

It initializes rgb and blurPixelCount variables to store the summed RGB value and count of sampled pixels.

It then has two more nested for loops with k and l to iterate over the blur area around the current i, j pixel.

It checks that the sampled pixels are within the image bounds, and if so, adds their RGB value to rgb and increments the blurPixelCount.

After sampling the surrounding pixels, it averages the rgb sum by dividing it by the blurPixelCount.

Finally, it sets the RGB value of the current pixel to the calculated blurred rgb value.

So in summary, it implements a basic box blur by sampling surrounding pixels within a radius, averaging their color, and setting each pixel to this averaged blurred value.

ROTATE CLOCKWISE

```
public static BufferedImage rotateClockwise(BufferedImage img) {
   int width = img.getWidth();
   int height = img.getHeight();

   BufferedImage rotated = new BufferedImage(height, width, img.getType());

   for(int i = 0; i < height; i++) {
       for(int j = 0; j < width; j++) {
           rotated.setRGB(height - i - 1, j, img.getRGB(j, i));
       }
   }
```

```
    return rotated;

}
```

This Java code rotates a BufferedImage 90 degrees clockwise.

It first gets the width and height of the original image.

Then it creates a new BufferedImage of the swapped width and height.

It iterates through each pixel of the original image, and sets the corresponding pixel on the rotated image by swapping the x and y coordinates.After looping through each pixel, it returns the rotated image. So it performs an in-place 90 degree rotation by reading the pixels from the original image and writing them to the new rotated image.