Runigram.java

```java
// This class uses the Color class, which is part of a package called awt,
// which is part of Java's standard class library.
import java.awt.Color;

/** A library of image processing functions. */
public class Runigram {

    public static void main(String[] args) {

        //// Hide / change / add to the testing code below, as needed.

        // Tests the reading and printing of an image:

        //Color[][] tinypic = read("tinypic.ppm");
        //print(tinypic);

        // Creates an image which will be the result of various
        // image processing operations:

        //Color[][] imageOut;

        // Tests the horizontal flipping of an image:

        //imageOut = flippedHorizontally(tinypic);
        //System.out.println();
        //print(imageOut);

        //// Write here whatever code you need in order to test your work.
        //// You can reuse / overide the contents of the imageOut array.
    }

    /** Returns a 2D array of Color values, representing the image data
     * stored in the given PPM file. */
    public static Color[][] read(String fileName) {
        In in = new In(fileName);
        // Reads the file header, ignoring the first and the third lines.
        in.readString();
        int numCols = in.readInt();
        int numRows = in.readInt();
        in.readInt();
        // Creates the image array
        Color[][] image = new Color[numRows][numCols];
        // Reads the RGB values from the file, into the image array.
        // For each pixel (i,j), reads 3 values from the file,
        // creates from the 3 colors a new Color object, and
        // makes pixel (i,j) refer to that object.
        for (int i = 0; i < numRows; i++){
            for (int j = 0; j < numCols; j++){
                int red = in.readInt();
```

```java
                int green = in.readInt();
                int blue = in.readInt();
                image[i][j] = new Color(red, green, blue);
            }
        }
        return image;
    }

    // Prints the RGB values of a given color.
    private static void print(Color c) {
        System.out.print("(");
        System.out.printf("%3s,", c.getRed());   // Prints the red component
        System.out.printf("%3s,", c.getGreen()); // Prints the green component
        System.out.printf("%3s",  c.getBlue());  // Prints the blue component
        System.out.print(")  ");
    }

    // Prints the pixels of the given image.
    // Each pixel is printed as a triplet of (r,g,b) values.
    // This function is used for debugging purposes.
    // For example, to check that some image processing function works correctly,
    // we can apply the function and then use this function to print the resulting
image.
    private static void print(Color[][] image) {
        for (int i = 0; i < image.length; i++){
            for (int j = 0; j < image[0].length; j++){
                print(image[i][j]);
            }
        }
    }

    /**
     * Returns an image which is the horizontally flipped version of the given
image.
     */
    public static Color[][] flippedHorizontally(Color[][] image) {
        int numRows = image.length;
        int numCols = image[0].length;
        Color[][] flipped = new Color[numRows][numCols];
        for (int i = 0; i < numRows; i++){
            for (int j = 0; j < numCols; j++){
                flipped[i][numCols - 1 - j] = image[i][j];
            }
        }
        return flipped;
    }

    /**
     * Returns an image which is the vertically flipped version of the given image.
     */
    public static Color[][] flippedVertically(Color[][] image){
```

```
        int numRows = image.length;
        int numCols = image[0].length;
        Color[][] flipped = new Color[numRows][numCols];
        for (int i = 0; i < numRows; i++){
            for (int j = 0; j < numCols; j++){
                flipped[numRows - 1 - i][j] = image[i][j];
            }
        }
        return flipped;
    }

    // Computes the luminance of the RGB values of the given pixel, using the
formula
    // lum = 0.299 * r + 0.587 * g + 0.114 * b, and returns a Color object
consisting
    // the three values r = lum, g = lum, b = lum.
    public static Color luminance(Color pixel) {
        int red = pixel.getRed();
        int green = pixel.getGreen();
        int blue = pixel.getBlue();
        int lum = (int) (0.299 * red + 0.587 * green + 0.114 * blue);
        return new Color(lum, lum, lum);
    }

    /**
     * Returns an image which is the grayscaled version of the given image.
     */
    public static Color[][] grayScaled(Color[][] image) {
        int numRows = image.length;
        int numCols = image[0].length;
        Color[][] gray = new Color[numRows][numCols];
        for (int i = 0; i < numRows; i++) {
            for (int j = 0; j < numCols; j++) {
                gray[i][j] = luminance(image[i][j]);
            }
        }
        return gray;
    }

    /**
     * Returns an image which is the scaled version of the given image.
     * The image is scaled (resized) to have the given width and height.
     */
    public static Color[][] scaled(Color[][] image, int width, int height) {
        int numRows = image.length;
        int numCols = image[0].length;
        Color[][] scaled = new Color[width][height];
        for (int i = 0; i < width; i++) {
            for (int j = 0; j < height; j++) {
                int x = (int)(i * (numRows /(double) width));
                int y = (int)(j * (numCols / (double) height));
```

```
                scaled[i][j] = image[x][y];
            }
        }
        return scaled;
    }

    /**
     * Computes and returns a blended color which is a linear combination of the
two given
     * colors. Each r, g, b, value v in the returned color is calculated using the
formula
     * v = alpha * v1 + (1 - alpha) * v2, where v1 and v2 are the corresponding r,
g, b
     * values in the two input color.
     */
    public static Color blend(Color c1, Color c2, double alpha) {
        int red = (int) (alpha * c1.getRed() + (1 - alpha) * c2.getRed());
        int green = (int) (alpha * c1.getGreen() + (1 - alpha) * c2.getGreen());
        int blue = (int) (alpha * c1.getBlue() + (1 - alpha) * c2.getBlue());
        return new Color(red, green, blue);
    }

    /**
     * Cosntructs and returns an image which is the blending of the two given
images.
     * The blended image is the linear combination of (alpha) part of the first
image
     * and (1 - alpha) part the second image.
     * The two images must have the same dimensions.
     */
    public static Color[][] blend(Color[][] image1, Color[][] image2, double alpha)
{
        int numRows = image1.length;
        int numCols = image1[0].length;
        Color[][] blended = new Color[numRows][numCols];
        for (int i = 0; i < numRows; i++) {
            for (int j = 0; j < numCols; j++) {
                blended[i][j] = blend(image1[i][j], image2[i][j], alpha);
            }
        }
        return blended;
    }

    /**
     * Morphs the source image into the target image, gradually, in n steps.
     * Animates the morphing process by displaying the morphed image in each step.
     * Before starting the process, scales the target image to the dimensions
     * of the source image.
     */
    public static void morph(Color[][] source, Color[][] target, int n) {
        target = scaled(target, source.length, source[0].length);
```

```
        setCanvas(source);
        for (int i = 0; i <= n; i++) {
            double alpha = (double)(n - i) / n;
            Color[][] morphed = blend(source, target, alpha);
            display(morphed);
            StdDraw.pause(500);
        }
    }

    /** Creates a canvas for the given image. */
    public static void setCanvas(Color[][] image) {
        StdDraw.setTitle("Runigram 2023");
        int height = image.length;
        int width = image[0].length;
        StdDraw.setCanvasSize(height, width);
        StdDraw.setXscale(0, width);
        StdDraw.setYscale(0, height);
        // Enables drawing graphics in memory and showing it on the screen only
when
        // the StdDraw.show function is called.
        StdDraw.enableDoubleBuffering();
    }

    /** Displays the given image on the current canvas. */
    public static void display(Color[][] image) {
        int height = image.length;
        int width = image[0].length;
        for (int i = 0; i < height; i++) {
            for (int j = 0; j < width; j++) {
                // Sets the pen color to the pixel color
                StdDraw.setPenColor( image[i][j].getRed(),
                                     image[i][j].getGreen(),
                                     image[i][j].getBlue() );
                // Draws the pixel as a filled square of size 1
                StdDraw.filledSquare(j + 0.5, height - i - 0.5, 0.5);
            }
        }
        StdDraw.show();
    }
}
```

Editor 1 :

```
import java.awt.Color;

/**
 * Demonstrates three basic image processing operations that are featured by
Runigram.java.
 * The program recieves two command-line arguments: a string representing the name
of the PPM file
 * of a source image, and one of the strings "fh", "fv", or "gs". The program
creates and displays
 * a new image which is either the horizontally flipped version of the source image
("fh"),
 * or the vertically flipped version of the source image ("fv"), or the grayscaled
version of the
 * source image ("gs"). For example, to create a grayscale version of thor.ppm,
use:
 * java Editor1 thor.ppm gs
 */
public class Editor1 {

    public static void main (String[] args){
        String fileName = args[0];
        String action = args[1];
        // Reads the input image and creates an empty output image
        Color[][] imageIn = Runigram.read(fileName);
        Color[][] imageOut = null;
        // Applies the specified image processing function
        if (action.equals("fh")) {
            imageOut = Runigram.flippedHorizontally(imageIn);
        } else if (action.equals("fv")) {
            imageOut = Runigram.flippedVertically(imageIn);
        } else if (action.equals("gs")) {
            imageOut = Runigram.grayScaled(imageIn);
        }
        // Creates a canvas in which both images will be displayed, one after the
other.
        // Next, displays the input image, and pauses for a few seconds.
        // Finally, displays the output image.
        // (Notice that both images have the same dimensions).
        Runigram.setCanvas(imageIn);
        Runigram.display(imageIn);
        StdDraw.pause(3000);
        Runigram.display(imageOut);
    }
}
```

Editor 2 :

```
import java.awt.Color;

/**
 * Demonstrates the scaling (resizing) operation featured by Runigram.java.
 * The program recieves three command-line arguments: a string representing the
name
 * of the PPM file of a source image, and two integers that specify the width and
the
 * height of the scaled, output image. For example, to scale/resize ironman.ppm to
a width
 * of 100 pixels and a height of 900 pixels, use: java Editor2 ironman.ppm 100 900
 */
public class Editor2 {

    public static void main (String[] args){
        String fileName = args[0];
        int width = Integer.parseInt(args[1]);
        int height = Integer.parseInt(args[2]);
        Color[][] imageIn = Runigram.read(fileName);
        Color[][] scaledImage = Runigram.scaled(imageIn, width, height);
        // Displays the source image
        Runigram.setCanvas(imageIn);
        Runigram.display(imageIn);
        StdDraw.pause(3000);
        // Displays the scaled image.
        Runigram.setCanvas(scaledImage);
        Runigram.display(scaledImage);
    }
}
```

Editor 3 :

```java
import java.awt.Color;

/**
 * Demonstrates the morphing operation featured by Runigram.java.
 * The program recieves three command-line arguments: a string representing the
name
 * of the PPM file of a source image, a string representing the name of the PPM
file
 * of a target image, and the number of morphing steps (an int).
 * For example, to morph the cake into ironman in 50 steps, use:
 * java Editor3 cake.ppm ironman.ppm 50
 * Note: There is no need to scale the target image to the size of the source
 * image, since Runigram.morph performs this action.
 */
public class Editor3 {

    public static void main (String[] args) {
        String source = args[0];
        String target = args[1];
        int n = Integer.parseInt(args[2]);
        Color[][] sourceImage = Runigram.read(source);
        Color[][] targetImage = Runigram.read(target);
        Runigram.setCanvas(sourceImage);
        Runigram.morph(sourceImage, targetImage, n);
    }
}
```

Editor 4 :

```java
import java.awt.Color;

public class Editor4 {

    public static void main(String[] args) {
        String source = args[0];
        int n = Integer.parseInt(args[1]);
        Color[][] sourceImage = Runigram.read(source);
        Color[][] targetImage = Runigram.grayScaled(sourceImage);
        Runigram.setCanvas(sourceImage);
        Runigram.morph(sourceImage, targetImage, n);
    }
}
```