```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.
       //// Replace the following statement with your code.
```

```
        for ( int row = 0 ; row < numRows; row++ ){
            for (int col = 0 ; col < numCols; col++ ){
                int red = in.readInt();
                int green = in.readInt();
                int blue  = in.readInt();
                image[row][col] = 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) {
        int rows = image.length;
        int columns = image[0].length;
        for (int i = 0 ; i < rows ; i++ ){
            for (int j = 0 ; j < columns ; j++ ){
                print(image[i][j]);
            }
            System.out.println();
        }
    }

    /**
     * Returns an image which is the horizontally flipped version of the given
image.
     */
    public static Color[][] flippedHorizontally(Color[][] image) {
        int rows = image.length;
        int columns = image[0].length;
```

```
        Color[][] imageFlipHorizo = new Color[rows][columns] ;

        for (int i = 0 ; i < rows ; i++ ){
            for (int j = columns - 1 ; j >= 0 ; j-- ){
                imageFlipHorizo[i][j] = image [i][columns - 1- j];
            }
        }

        return imageFlipHorizo;
    }

    /**
     * Returns an image which is the vertically flipped version of the given image.
     */
    public static Color[][] flippedVertically(Color[][] image){
        int rows = image.length;
        int columns = image[0].length;
        Color[][] imageFlipVartically  = new Color[rows][columns] ;

        for (int i = 0 ; i < rows ; i++ ){
            for (int j = 0 ; j < columns ; j++ ){

                imageFlipVartically[i][j] = image[rows - 1 - i ][j];
            }
        }
        return imageFlipVartically;
    }

    // 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();
        // new gary values

        int lum = (int) (0.299 * red + 0.587 * green + blue * 0.114);
        // new gray color

        Color grayColor = new Color (lum, lum, lum );

        return grayColor;
```

```
    }

    /**
     * Returns an image which is the grayscaled version of the given image.
     */
    public static Color[][] grayScaled(Color[][] image) {

        int rows = image.length;
        int columns = image[0].length;
        Color[][] imageGrayScaled = new Color[rows][columns] ;

        for (int i = 0 ; i < rows ; i++ ){
            for (int j = 0 ; j < columns ; j++ ){

                imageGrayScaled[i][j]= luminance(image[i][j]) ;
            }
        }

        return imageGrayScaled;
    }

    /**
     * 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 h0 = image.length;
        int w0 = image[0].length;

        Color[][] scaledImage = new Color[height][width];

        for (int i = 0; i < height; i++) {
            for (int j = 0; j < width; j++) {
                int sourceI = (int) (i * ((double) h0 / height));
                int sourceJ = (int) (j * ((double) w0 / width));
                scaledImage[i][j] = image[sourceI][sourceJ];
            }
        }

        return scaledImage;
    }

    /**
     * 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());

       Color blendedColor = new Color (red , green , blue );

       return blendedColor;
   }

   /**
    * 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 imageWidth = image1.length  ;
       int imageHeight =  image1[0].length ;
       Color[][] blendedImage = new Color[imageWidth][imageHeight] ;

       for( int i = 0 ; i < imageWidth ; i++ ){
           for (int j = 0 ; j < imageHeight ; j++ ){

               Color c1 = image1[i][j];
               Color c2 = image2[i][j];

               blendedImage[i][j] = blend(c1, c2, alpha);
           }
       }

       return blendedImage;
   }
   /**
```

```
    * 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) {
       Color[][] scaledTarget = scaled(target, source[0].length, source.length);
       for (int i = 0; i <= n; i++) {
           double alpha = (n - i) / (double) n;
           Color[][] sourceImage = blend(source, scaledTarget, alpha);
           display(sourceImage);
           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();
   }
}
```