HW6 – neta tarshish

```
 //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 = scaled(tinypic,3,5);

                System.out.println;()

                print(imageOut);


                Color one = new Color;(100,40,100)

                Color two = new Color;(200,20,40)


                print(blend(one, two, 0.25));


                //Color[][] newPic = new Color [tinypic.length][tinypic[0].length];
        //for(int i = 0;i<tinypic.length;i++)}
                //for(int j = 0;j<tinypic[0].length;j++)}
                        //newPic [i][j] = luminance(tinypic[i][j]);
        //        {
        {//
        //print(newPic);


         ////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];
        for(int i = 0;i<numRows;i++)}
                for(int j = 0;j<numCols;j++)}
                        image[i][j] = new Color (in.readInt(),in.readInt(),in.readInt());
                {
        {
         //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.
                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++)}
                                System.out.print;(")")
                                System.out.printf("%3s,", image[i][j].getRed())    ;
                                System.out.printf("%3s,", image[i][j].getGreen()) ;
                                System.out.printf("%3s",  image[i][j].getBlue())  ;
                                System.out.print;("  (")
                        {
                        System.out.println;()
                {
        {
```

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

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

 //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 newColor = (int)(0.299 * red + 0.587 * green + 0.114 *
blue);
                Color newPixal = new Color(newColor, newColor, newColor);

        return newPixal;
    {

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

    **/
     * 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)}
        Color [][] newPic = new Color [height][width];
        int zeroH = image.length;
        int zeroW = image[0].length;
        for(int i = 0;i<newPic.length;i++)}
```

```
                    for(int j = 0;j<newPic[0].length;j++)}
                              int originalI = i * zeroH / height;
int originalJ = j * zeroW / width;
newPic[i][j] = image[originalI][originalJ];
                    {
          {
          return newPic;
{

**/
 * 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 newRed = (int)((alpha * c1.getRed()) + ((1 - alpha) * c2.getRed()));
          int newGreen = (int)((alpha * c1.getGreen()) + ((1 - alpha) * c2.getGreen()));
          int newBlue = (int)((alpha * c1.getBlue()) + ((1 - alpha) * c2.getBlue()));
          Color newColor = new Color (newRed, newGreen, newBlue);
          return newColor;
{

**/
 * 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)}
          Color [][] newImage = new Color [image1.length][image1[0].length];
```

```
    for(int i = 0;i<image1.length;i++)}
        for(int j = 0;j<image1[0].length;j++)}
            newImage[i][j] = blend (image1[i][j], image2[i][j], alpha);
        {
    {
    return newImage;
{

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