Runigram.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("thor.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 = grayScaled(scaled(tinypic, 200, 200))
;()System.out.println
;print(imageOut) //
;setCanvas(imageOut)
;display(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]
} for ( int y = 0; y < numRows; y++)
} for ( int x = 0; x < numCols; x++)
;()int red = in.readInt
;()int green = in.readInt
;()int blue = in.readInt
;image [y][x] = new Color (red, green, blue)
{
{

;return image
{

.Prints the RGB values of a given color //
} private static void printPixel(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 y = 0; y < image.length; y++)
} for ( int x = 0; x < image[0].length; x++)
;printPixel(image[y][x])
{
;()System.out.println
{
{


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

```
;return flipImage
{


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


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 lum = (int) (0.299 * pixel.getRed() + 0.587 * pixel.getGreen()
;+ 0.114 * pixel.getBlue())
;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 width = image[0].length
;int height = image.length
;Color [][] grayImage = new Color[height][width]
} for ( int y = 0; y < height; y++)
} for ( int x = 0; x < width; x++)
;grayImage [y][x] =luminance(image [y][x])
{
{
;return grayImage

        {

**/
.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 widthO = image[0].length
;int heightO = image.length
;Color [][] scaleImage = new Color[height][width]
} for ( int y = 0; y < height; y++)
} for ( int x = 0; x < width; x++)
;int yO = y * heightO/height
;int xO = x * widthO/width
;scaleImage [y][x] = image [yO][xO]
{
{
```

```
;return scaleImage


{


**/

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)

;()double r1 = c1.getRed

;()double g1 = c1.getGreen

;()double b1 = c1.getBlue

;()double r2 = c2.getRed

;()double g2 = c2.getGreen

;()double b2 = c2.getBlue

;int red = (int) (alpha * r1 + (1 - alpha) * r2)

;int green = (int) (alpha * g1 + (1 - alpha) * g2)

;int blue = (int) (alpha * b1 + (1 - alpha) * b2)

;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)
Color[][] blendedImage = new
;Color[image1.length][image1[0].length]
} for (int i = 0; i < blendedImage.length; i++)
} for (int j = 0; j < blendedImage[0].length; j++)
blendedImage[i][j] = blend(image1[i][j], image2[i][j],
;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)
if (source.length != target.length || source[0].length !=
} target[0].length)
;target = scaled(target, source[0].length, source.length)
{
Color[][] morphImage = new
;Color[source.length][source[0].length]
} for (int i = 0; i <= n; i++)
;double alpha = (double) (n - i) / n
```

```
;morphImage = blend(source, target, alpha)
;display(morphImage)
;(500)StdDraw.pause

{
{

/* .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
{
{
```