```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);
        // luminance test
        //Color originalColor = new Color(128, 0, 0);
        //System.out.println(luminance(originalColor));

        //// 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) {
```

```java
    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++) {
            int r = in.readInt();
            int g = in.readInt();
            int b = in.readInt();
            image[i][j] = new Color(r, g, b);
        }
    }
    return image;
}

// Prints the RGB values of a given color.
private static void printColor(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[i].length; j++) {
            printColor(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 numRows = image.length;
    int numCols = image[0].length;

    Color[][] newImage = new Color[numRows][numCols];

    for (int i = 0; i < numRows; i++) {
        for (int j = 0; j < numCols; j++) {
            newImage[i][j] = image[i][numCols - 1 - j];
        }
    }

    return newImage;
}

/**
 * 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[][] newImage = new Color[numRows][numCols];

    for (int i = 0; i < numRows; i++) {
        for (int j = 0; j < numCols; j++) {
            newImage[i][j] = image[numRows - 1 - i][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 r = pixel.getRed();
    int g = pixel.getGreen();
    int b = pixel.getBlue();
    // now we compute the luminance of the pixel using the given formula
    int lum = (int) (0.299 * r + 0.587 * g + 0.114 * b);
    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[][] newImage = new Color[numRows][numCols];

    for (int i = 0; i < numRows; i++) {
        for (int j = 0; j < numCols; j++) {
            // use luminance function to get a new grey color
            Color greyPixel = luminance(image[i][j]);
            // we assign the grey color pixels to the new image
            newImage[i][j] = greyPixel;

        }
    }
    return newImage;
}

/**
 * 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 newWidth, int newHeight) {
    int originalHeight = image.length;
    int originalWidth = image[0].length;

    // create new color image array with the new desired size
    Color[][] newImage = new Color[newHeight][newWidth];
    // calculate the 2 scaling factors for width and hight
    double scaleWidth = (double) originalWidth / newWidth;
    double scaleHeight = (double) originalHeight / newHeight;

    for (int i = 0; i < newHeight; i++) {
        for (int j = 0; j < newWidth; j++) {
            // calculate the i and j indexes of the pixel in the original image
            // correspond to the new i j indexes of the new pixel in the scaled i
            int iOriginal = (int) (i * scaleHeight);
            int jOriginal = (int) (j * scaleWidth);

            // now we assign the original pixel to its position in the new scaled image
            newImage[i][j] = image[iOriginal][jOriginal];
        }
    }
    return newImage;
}

/**
 * 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
```

```java
 * 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 r1 = c1.getRed();
    int g1 = c1.getGreen();
    int b1 = c1.getBlue();

    int r2 = c2.getRed();
    int g2 = c2.getGreen();
    int b2 = c2.getBlue();

    // linear combination of both colors to get the new color
    int rBlend = (int) ((alpha * r1) + ((1 - alpha) * r2));
    int gBlend = (int) ((alpha * g1) + ((1 - alpha) * g2));
    int bBlend = (int) ((alpha * b1) + ((1 - alpha) * b2));
    return new Color(rBlend, gBlend, bBlend);
}

/**
 * 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) {
    // since both images have the same dimensions we'll use dimensions of image1
    int numRows = image1.length;
    int numCols = image1[0].length;

    Color [][]blendedImage = new Color[numRows][numCols];

    for (int i = 0; i < numRows; i++) {
        for (int j = 0; j < numCols; j++) {
            // we create a new blended color per pixel by calling the blended function
            Color newBlendedColor = blend(image1[i][j], image2[i][j], alpha);

            blendedImage[i][j] = newBlendedColor;

        }
    }

    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) {
    int sourceRows = source.length;
    int sourceCols = source[0].length;
    Color[][] scaledTarget = target;

    // if the dimensions are different then we scale the image
    if (target.length != sourceRows || target[0].length != sourceCols) {
        scaledTarget = scaled(target, sourceCols, sourceRows);
    }

    for (int i = 0; i <= n; i++) {
        double alpha = (double) (n - i) / n;
        Color[][] morphedStep = blend(source, scaledTarget, alpha); //

        Runigram.display(morphedStep);
        StdDraw.pause(500);
    }
}

/** Creates a canvas for the given image. */
```

```java
    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();
    }
}
```