```java
import java.awt.Color;

/**
 * Demonstrates three Instush.java services: flipping an image horizontally,
 * flipping an image vertically, and greyscaling an image.
 *
 * The program recieves two command-line arguments: the name of the PPM file
 * that represents the source image (a string), and one of the strings "fh",
 * "fv", or "gs" (a string). 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 greyscaled
 * version of the source image ("gs"). For example: java Editor1 thor.ppm gs
 */
public class Editor1 {

    public static void main(String[] args) {
        Color[][] image = Instush.read(args[0]);
        if (args[1].equals("fv")) {
            Color[][] flippedImage = Instush.flippedVertically(image);
            Instush.show(flippedImage);
        } else if (args[1].equals("fh")) {
            Color[][] flippedImage = Instush.flippedHorizontally(image);
            Instush.show(flippedImage);
        } else if (args[1].equals("gs")) {
            Color[][] greyImage = Instush.greyscaled(image);
            Instush.show(greyImage);
        } else {
            System.out.println("Invalid function name");
        }
    }
}
```

```java
import java.awt.Color;

/**
 * Demonstrates the scaling function of Instush.java. The program recieves two
 * command-line arguments: the name of the PPM file (a string) representing the
 * image that should be scaled, and two integers that specify the width and the
 * height of the scaled image. For example: java Editor2 ironman.ppm 100 900
 */
public class Editor2 {

    public static void main(String[] args) {
        Color[][] image = Instush.read(args[0]);
        int width = Integer.parseInt(args[1]);
        int height = Integer.parseInt(args[2]);
        Instush.show(Instush.scaled(image, width, height));
    }
}
```

```java
import java.awt.Color;

/**
 * Demonstrates the morphing service of Instush.java. The program recieves three
 * command-line arguments: the name of a PPM file that represents the source
 * image (a string), the name of a PPM file that represents the target image (a
 * string), and the number of morphing steps (an int). For example: java Editor3
 * cake.ppm ironman.ppm 300 If the two images don't have the same dimensions,
 * the program scales the target image to the dimensions of the source image.
 */
public class Editor3 {

    public static void main(String[] args) {
        String sourceFile = args[0];
        String targetFile = args[1];
        Color[][] source = Instush.read(sourceFile);
        Color[][] target = Instush.read(targetFile);
        if (source.length != target.length || source[0].length != target[0].length) {
            target = Instush.scaled(target, source[0].length, source.length);
        }
        Instush.morph(source, target, Integer.parseInt(args[2]));
    }
}
```

```java
import java.awt.Color;

/**
 * The program recieves two command-line arguments: the name of a PPM file that
 * represents the source image (a string), and the number of morphing steps (an
 * int). For example: java Editor4 ironman.ppm 300
 */
public class Editor4 {

    public static void main(String[] args) {
        String sourceFile = args[0];
        Color[][] source = Instush.read(sourceFile);
        Color[][] target = Instush.greyscaled(source);
        Instush.morph(source, target, Integer.parseInt(args[1]));
    }
}
```

```java
import java.awt.Color;
import java.util.concurrent.TimeUnit;
import javax.swing.plaf.ColorUIResource;

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

    public static void main(String[] args) {
        Color[][] image = read(args[0]);
        Color[][] target = read(args[1]);
        // // Test read
        // print(image);
        // show(image);
        // // Test flip horizontal
        // print(flippedHorizontally(image));
        // show(flippedHorizontally(image));
        // // Test flip vertical
        // print(flippedVertically(image));
        // show(flippedVertically(image));
        // // Test flip gray scale
        // print(greyscaled(image));
        // show(greyscaled(image));
        // // Test flip scale
        // print(scaled(image, Integer.parseInt(args[1]), Integer.parseInt(args[2])));
        // show(scaled(image, Integer.parseInt(args[1]), Integer.parseInt(args[2])));
        // // Test flip morph
        // print(morph(image, target,50);
        // show(morph(image, target,50);

    }

    /**
     * Returns an image created from a given PPM file. SIDE EFFECT: Sets standard
     * input to the given file.
     *
     * @return the image, as a 2D array of Color values
     */
    public static Color[][] read(String filename) {
        StdIn.setInput(filename);
        // Reads the PPM file header (ignoring some items)
        StdIn.readString();
        int numRows = StdIn.readInt();
        int numCols = StdIn.readInt();
        StdIn.readInt();
        // Creates the image
        Color[][] image = new Color[numCols][numRows];
        for (int i = 0; i < numCols; i++) {
            for (int j = 0; j < numRows; j++) {
                Color pixelColor = new Color(StdIn.readInt(), StdIn.readInt(),
StdIn.readInt());
                image[i][j] = pixelColor;
            }
        }
        return image;
    }

    /**
     * Prints the pixels of a given image. Each pixel is printed as a triplet of
     * (r,g,b) values. For debugging purposes.
```

```
     *
     * @param image - the image to be printed
     */
    public static void print(Color[][] image) {
        for (int i = 0; i < image.length; i++) {
            for (int j = 0; j < image[0].length; j++) {
                int r = image[i][j].getRed();
                int g = image[i][j].getGreen();
                int b = image[i][j].getBlue();
                System.out.print("(");
                System.out.printf("%4s", r + ",");
                System.out.printf("%4s", g + ",");
                System.out.printf("%3s", b);
                System.out.print(") ");
            }
            System.out.printf("%1s", "\n");
        }
        System.out.printf("%1s", "\n");
    }

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

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

    /**
     * Returns the average of the RGB values of all the pixels in a given image.
     *
     * @param image - the image
     * @return the average of all the RGB values of the image
     */
    public static double average(Color[][] image) {
        // Replace the following statement with your code
```

```
122         return 0.0;
123     }
124 
125     /**
126      * Returns the luminance value of a given pixel. Luminance is a weighted average
127      * of the RGB values of the pixel, given by 0.299 * r + 0.587 * g + 0.114 * b.
128      * Used as a shade of grey, as part of the greyscaling process.
129      *
130      * @param pixel - the pixel
131      * @return the greyscale value of the pixel, as a Color object (r = g = b = the
132      *         greyscale value)
133      */
134     public static Color luminance(Color pixel) {
135         int lum = (int) (0.299 * pixel.getRed() + 0.587 * pixel.getGreen() + 0.114 *
pixel.getBlue());
136         Color greyPixel = new Color(lum, lum, lum);
137         return greyPixel;
138     }
139 
140     /**
141      * Returns an image which is the greyscaled version of the given image.
142      *
143      * @param image - the image
144      * @return rhe greyscaled version of the image
145      */
146     public static Color[][] greyscaled(Color[][] image) {
147         Color[][] greyScaled = new Color[image.length][image[0].length];
148         for (int i = 0; i < image.length; i++) {
149             for (int j = 0; j < image[0].length; j++) {
150                 greyScaled[i][j] = luminance(image[i][j]);
151             }
152         }
153         return greyScaled;
154     }
155 
156     /**
157      * Returns an umage which is the scaled version of the given image. The image is
158      * scaled (resized) to be of the given width and height.
159      *
160      * @param image  - the image
161      * @param width  - the width of the scaled image
162      * @param height - the height of the scaled image
163      * @return - the scaled image
164      */
165     public static Color[][] scaled(Color[][] image, int width, int height) {
166         Color[][] scaledImage = new Color[height][width];
167         for (int i = 0; i < height; i++) {
168             for (int j = 0; j < width; j++) {
169                 scaledImage[i][j] = image[(int) (i * ((double) image.length / height))]
[(int) (j
170                         * ((double) image[0].length / width))];
171             }
172         }
173         return scaledImage;
174     }
175 
176     /**
177      * Returns a blended color which is the linear combination of two colors. Each
178      * r, g, b, value v is calculated using v = (1 - alpha) * v1 + alpha * v2.
179      *
180      * @param pixel1 - the first color
```

```
     * @param pixel2 - the second color
     * @param alpha  - the linear combination parameter
     * @return the blended color
     */
    public static Color blend(Color c1, Color c2, double alpha) {
        int red = (int) ((c1.getRed() * alpha) + (c2.getRed() * (1 - alpha)));
        int green = (int) ((c1.getGreen() * alpha) + (c2.getGreen() * (1 - alpha)));
        int blue = (int) ((c1.getBlue() * alpha) + (c2.getBlue() * (1 - alpha)));
        Color newColor = new Color(red, green, blue);
        return newColor;
    }

    /**
     * Returns an image which is the blending of the two given images. The blending
     * is the linear combination of (1 - alpha) parts the first image and (alpha)
     * parts the second image. The two images must have the same dimensions.
     *
     * @param image1 - the first image
     * @param image2 - the second image
     * @param alpha  - the linear combination parameter
     * @return - the blended image
     */
    public static Color[][] blend(Color[][] image1, Color[][] image2, double alpha) {
        Color[][] blended = new Color[image1.length][image1[0].length];
        for (int i = 0; i < image1.length; i++) {
            for (int j = 0; j < image1[i].length; 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.
     * The target image is an image which is scaled to be a version of the target
     * image, scaled to have the width and height of the source image.
     *
     * @param source - source image
     * @param target - target image
     * @param n      - number of morphing steps
     */
    public static void morph(Color[][] source, Color[][] target, int n) {
        for (double i = n; i >= 0; i--) {
            Color[][] blended = blend(source, target, i / n);
            show(blended);
        }
    }

    /**
     * Renders (displays) an image on the screen, using StdDraw.
     *
     * @param image - the image to show
     */
    public static void show(Color[][] image) {
        StdDraw.setCanvasSize(image[0].length, image.length);
        int width = image[0].length;
        int height = image.length;
        StdDraw.setXscale(0, width);
        StdDraw.setYscale(0, height);
        StdDraw.show(25);
```

```
        for (int i = 0; i < height; i++) {
            for (int j = 0; j < width; j++) {
                // Sets the pen color to the color of the pixel
                StdDraw.setPenColor(image[i][j].getRed(), image[i][j].getGreen(), image[i]
[j].getBlue());
                // Draws the pixel as a tiny filled square of size 1
                StdDraw.filledSquare(j + 0.5, height - i - 0.5, 0.5);
            }
        }
        StdDraw.show();
    }
}
```