```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 Editor4
 {

  public static void main (String[] args)
  {
    String NameOfFile = args[0];
    int NumOfTimes=Integer.parseInt(args[1]);
    // Write your code here
    Instush.morph(Instush.read(NameOfFile),
Instush.greyscaled(Instush.read(NameOfFile)), NumOfTimes);
  }
}


```

```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 NameOfFile1 = args[0];
    String NameOfFile2 = args[1];
    int NumOfTimes=Integer.parseInt(args[2]);
    // Write your code here
    Instush.morph(Instush.read(NameOfFile1), Instush.read(NameOfFile2), NumOfTimes);

  }
}

```

```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)
  {
    String NameOfFile = args[0];
    int wid=Integer.parseInt(args[1]);
    int heig=Integer.parseInt(args[2]);

    Instush.show(Instush.scaled(Instush.read(NameOfFile),wid,heig));


    // Write your code here
  }
}

```

```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)
  {
    String NameOfFile = args[0];
    String Operation = args[1];

    if (Operation.equals("fv"))
      Instush.show(Instush.flippedVertically(Instush.read(NameOfFile)));

    if (Operation.equals("fh"))
    Instush.show(Instush.flippedHorizontally(Instush.read(NameOfFile)));

    if (Operation.equals("gs"))
    Instush.show(Instush.greyscaled(Instush.read(NameOfFile)));


    // Write your code here
  }
}

```

```java
import java.awt.Color;

import javax.swing.text.Highlighter.Highlight;

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

  public static void main(String[] args)
  {
      // Can be used for testing, as needed.
    ///Instush.print(read("tinypic.ppm"));
    ///Instush.print(flippedHorizontally(read("tinypic.ppm")));
    ///Instush.print(flippedVertically(read("tinypic.ppm")));
    ///Instush.print(greyscaled(read("tinypic.ppm")));
    ///Instush.show(scaled(read("ironman.ppm"),600,300));
    ///Color c1 = new Color (100,40,100);
    ///Color c2 = new Color (200,20,40);
    ///System.out.println(blend(c1, c2, 0.25);
    ///Instush.print(blend(read("tinypic.ppm"), read("thor.ppm"), 0.25));

  }

  /**
   * 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<image.length; i++)
    {
      for (int j=0; j<image[0].length; j++)
      {
        int numone = StdIn.readInt();
        int numtwo = StdIn.readInt();
        int numthree = StdIn.readInt();
        image[i][j]= new Color(numone, numtwo, numthree);
      }

    }
    // Reads the RGB values from the file, into the image.
    // 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;
  }

  /**
```

```java
   * 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)
  {
      // Write your code here
    for (int i=0; i<image.length; i++)
    {
      for (int j=0; j<image[0].length; j++)
      {
        System.out.print("(");
        System.out.printf("%4s", image[i][j].getRed() +",");    // Prints the color's red component
        System.out.printf("%4s", image[i][j].getGreen()+",");  // Prints the color's green component
        System.out.printf("%3s", image[i][j].getBlue());   // Prints the color's blue component
        System.out.print(") ");
        System.out.print(" ");

      }
      System.out.println("");
    }
  }

  /**
   * 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 [][] newarr = new Color [image.length][image[0].length];
    for (int i=0; i<image.length; i++)
    {
      for (int j=0; j<image[0].length; j++)
      {
        newarr[i][j]=image[i][image[0].length-j-1];
      }
    }
    return newarr;
  }

  /**
   * 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)
  {
    // Replace the following statement with your code
    Color [][] newarr = new Color [image.length][image[0].length];
    for (int i=0; i<image.length; i++)
    {
      for (int j=0; j<image[0].length; j++)
      {
        newarr[i][j]=image[image.length-i-1][j];
      }
```

```
117     }
118     return newarr;
119   }
120 
121   /**
122    * Returns the average of the RGB values of all the pixels in a given image.
123    * @param image - the image
124    * @return the average of all the RGB values of the image
125    */
126   public static double average(Color[][] image) {
127     // Replace the following statement with your code
128     return 0.0;
129   }
130 
131   /**
132    * Returns the luminance value of a given pixel. Luminance is a weighted average
133    * of the RGB values of the pixel, given by 0.299 * r + 0.587 * g + 0.114 * b.
134    * Used as a shade of grey, as part of the greyscaling process.
135    * @param pixel - the pixel
136    * @return the greyscale value of the pixel, as a Color object
137    *         (r = g = b = the greyscale value)
138    */
139   public static Color luminance(Color pixel)
140   {
141     // Replace the following statement with your code
142     int value= ((int)((pixel.getRed()*0.299)+(pixel.getGreen()*0.587)+
(pixel.getBlue()*0.114)));
143     return new Color(value, value, value);
144   }
145 
146   /**
147    * Returns an image which is the greyscaled version of the given image.
148    * @param image - the image
149    * @return rhe greyscaled version of the image
150    */
151   public static Color[][] greyscaled(Color[][] image)
152    {
153     // Replace the following statement with your code
154     // I am saving the original array before I change it, so it won't have a side
effect.
155     Color [][] newarr = new Color [image.length][image[0].length];
156 
157     for (int k=0; k<image.length; k++)
158     {
159       for (int l=0; l<image[0].length; l++)
160       {
161         newarr[k][l]=image[k][l];
162       }
163     }
164     for (int i=0; i<newarr.length; i++)
165     {
166       for (int j=0; j<newarr[0].length; j++)
167       {
168         newarr[i][j]=luminance(image[i][j]);
169       }
170     }
171 
172     return newarr;
173   }
174 
```

```
175   /**
176    * Returns an umage which is the scaled version of the given image.
177    * The image is scaled (resized) to be of the given width and height.
178    * @param image - the image
179    * @param width - the width of the scaled image
180    * @param height - the height of the scaled image
181    * @return - the scaled image
182    */
183   public static Color[][] scaled(Color[][] image, int width, int height)
184   {
185     // Replace the following statement with your code
186     Color [][] newsize = new Color [height][width];
187     double newh= ((double)(image.length)/height);
188     double neww= ((double)(image[0].length)/width);
189     for (int i=0; i<height; i++)
190     {
191       for (int j=0; j< width; j++)
192       {
193         newsize[i][j]=image[(int)(i*newh)][(int)(j*neww)];
194       }
195 
196     }
197     return newsize;
198   }
199 
200   /**
201    * Returns a blended color which is the linear combination of two colors.
202    * Each r, g, b, value v is calculated using v = (1 - alpha) * v1 + alpha * v2.
203    *
204    * @param pixel1 - the first color
205    * @param pixel2 - the second color
206    * @param alpha - the linear combination parameter
207    * @return the blended color
208    */
209   public static Color blend(Color c1, Color c2, double alpha)
210    {
211      double newred = (double)(c1.getRed()*alpha)+ (double)((1-alpha)*(c2.getRed()));
212      double newgreen = (double)(c1.getGreen()*alpha)+ (double)((1-alpha)*
(c2.getGreen()));
213      double newblue = (double)(c1.getBlue()*alpha)+ (double)((1-alpha)*
(c2.getBlue()));
214 
215     Color c = new Color ((int)newred,(int)newgreen,(int)newblue);
216     return c;
217     // Replace the following statement with your code
218   }
219 
220   /**
221    * Returns an image which is the blending of the two given images.
222    * The blending is the linear combination of (1 - alpha) parts the
223    * first image and (alpha) parts the second image.
224    * The two images must have the same dimensions.
225    * @param image1 - the first image
226    * @param image2 - the second image
227    * @param alpha - the linear combination parameter
228    * @return - the blended image
229    */
230   public static Color[][] blend(Color[][] image1, Color[][] image2, double alpha)
231   {
232     Color [][] blended = new Color [image1.length][image1[0].length];
```

```
    for (int i=0; i<image1.length; i++)
    {
      for (int j=0; j<image1[0].length; j++)
      {
        blended[i][j]=blend(image1[i][j],image2[i][j], alpha);
      }
    }
    // Replace the following statement with your code
    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)
  {
    Color [][] backup = new Color [target.length][target[0].length];
    backup = scaled(target, source[0].length,source.length);

    for(int i=0; i<=n; i++)
    {
      show(blend(source,backup,(double)(n-i)/n));
    }
    // Write your code here
  }

     /**
   * 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();
  }
}
```