# JAVA3024: MIRRORING IMAGES\*

# R.G. (Dick) Baldwin

This work is produced by OpenStax-CNX and licensed under the Creative Commons Attribution License  $3.0^{\dagger}$ 

#### Abstract

Learn how to mirror images both horizontally and vertically.

# **1** Table of Contents

- Preface (p. 1)
  - · Viewing tip (p. 1)
    - \* Figures (p. 2)
    - \* Listings (p. 2)
- Preview (p. 2)
- Discussion and sample code (p. 7)
- Run the program (p. 11)
- Summary (p. 12)
- What's next? (p. 12)
- Online video link (p. 12)
- Miscellaneous (p. 12)
- Complete program listing (p. 13)

# 2 Preface

This module is one of a series of modules designed to teach you about Object-Oriented Programming (OOP) using Java.

The program described in this module requires the use of the Guzdial-Ericson multimedia class library. You will find download, installation, and usage instructions for the library at Java OOP: The Guzdial-Ericson Multimedia Class Library  $^1$ .

# 2.1 Viewing tip

I recommend that you open another copy of this document in a separate browser window and use the following links to easily find and view the figures and listings while you are reading about them.

<sup>\*</sup>Version 1.4: Dec 12, 2012 6:43 am -0600

<sup>&</sup>lt;sup>†</sup>http://creativecommons.org/licenses/by/3.0/

 $<sup>^{1}</sup> http://cnx.org/content/m44148/latest/$ 

# 2.1.1 Figures

- Figure 1 (p. 3). Input file named Prob02a.jpg.
- Figure 2 (p. 4) . First output image.
- Figure 3 (p. 5) . Second output image.
- Figure 4 (p. 6). Third output image.
- Figure 5 (p. 7) . Required output text.
- Figure 6 (p. 10). Picture output from the mirrorUpperQuads method.

#### 2.1.2 Listings

- Listing 1 (p. 7). The driver class named Prob02.
- Listing 2 (p. 8). Beginning of the class named Prob02Runner.
- Listing 3 (p. 8). The run method.
- Listing 4 (p. 8). The method named mirrorUpperQuads.
- Listing 5 (p. 11). The method named mirrorHoriz.
- Listing 6 (p. 13) . Complete program listing.

# **3** Preview

In this module, you will learn how to mirror images, both horizontally and vertically.

**Program specifications** 

Write a program named **Prob02** that uses the class definition shown in Listing 1 (p. 7) and Ericson's media library along with the image file named **Prob02a.jpg** (shown in Figure 1 (p. 3)) to produce the three graphic output images shown in Figure 2 (p. 4), Figure 3 (p. 5), and Figure 4 (p. 6).

 $\mathbf{2}$ 



Input file named Prob02a.jpg.

Figure 1: Input file named Prob02a.jpg.



Figure 2: First output image.

# Second output image.



Figure 3: Second output image.

# Third output image.



Figure 4: Third output image.

#### New classes

You may define new classes as necessary to cause your program to behave as required, but you may not modify the class definition for the class named **Prob02** shown in Listing 1 (p. 7).

#### Rotate and mirror

The image from the file named **Prob02a.jpg** is rotated by 35 degrees. It is not scaled. Then the top-left quadrant of the picture containing the rotated image is mirrored into the top-right quadrant. Following this, the top half of the picture is mirrored into the bottom half.

#### **Required output text**

In addition to the three output images mentioned above, your program must display your name and the other line of text shown in Figure 5 (p. 7) on the command-line screen

#### Required output text.

Display your name here. Picture, filename None height 404 width 425

Figure 5: Required output text.

#### 4 Discussion and sample code

#### Will discuss in fragments

I will discuss and explain this program in fragments. A complete listing of the program is provided in Listing 6 (p. 13) near the end of the module.

The driver class named Prob02

The driver class containing the **main** method is shown in Listing 1 (p. 7).

Listing 1: The driver class named Prob02.

```
public class Prob02{
  public static void main(String[] args){
    new Prob02Runner().run();
  }//end main method
}//end class Prob02
```

# Instantiate a new object and call its run method

The code in the **main** method instantiates a new object of the class named **Prob02Runner** and calls the **run** method on that object.

When the **run** method returns, the **main** method terminates causing the program to terminate. Beginning of the class named Prob02Runner

The beginning of the class named  $\mathbf{Prob02Runner}$ , including the constructor, is shown in Listing 2 (p. 8).

Listing 2: Beginning of the class named Prob02Runner.

```
class Prob02Runner{
public Prob02Runner(){
   System.out.println("Display your name here.");
}//end constructor
```

The constructor displays the student's name, producing the first line of output text shown in Figure 5 (p. 7).

#### The run method

The **run** method that is called in Listing 1 (p. 7) is shown in its entirety in Listing 3 (p. 8).

Listing 3: The run method.

```
public void run(){
Picture pix = new Picture("Prob02a.jpg");
//Add your name and display the output picture.
pix.addMessage("Display your name here.",10,20);
//Display the input picture.
pix.explore();
pix = rotatePicture(pix,35);
pix.explore();
pix = mirrorUpperQuads(pix);
pix = mirrorHoriz(pix);
pix.explore();
System.out.println(pix);
}//end run
```

#### Very familiar code

Except for the calls to the methods named **mirrorUpperQuads** and **mirrorHoriz** in Listing 3 (p. 8), you should already be familiar with all of the code in Listing 3.

# The rotatePicture method

For example, the call to the method named **rotatePicture** is essentially the same as code that I explained in an earlier module. Therefore, I won't explain that method again in this module. You will find the code for the method named **rotatePicture** in Listing 6 (p. 13) near the end of the module.

#### Operate on the picture with the rotated image

The original picture is replaced by a picture containing the rotated image shown in Figure 3 (p. 5). From this point forward, all operations are performed on the **Picture** object containing the rotated image.

#### The method named mirrorUpperQuads

The method named mirrorUpperQuads that is called in the run method in Listing 3 (p. 8) is shown in Listing 4 (p. 8).

#### Behavior of the method named mirrorUpperQuads

This method mirrors the upper-left quadrant of a picture into the upper-right quadrant as shown in Figure 6 (p. 10).

#### Listing 4: The method named mirrorUpperQuads.

return pix;
}//end mirrorUpperQuads

#### Declare four working variables

Listing 4 (p. 8) begins by declaring and initializing four working variables. The purpose of these variables should be obvious on the basis of their names and their initialization expressions.

#### Copy the pixel colors

Then Listing 4 (p. 8) uses a double nested **for** loop to copy the colors from the pixels in the upper-left quadrant into the pixels in the upper-right quadrant. This is done in such a way as to form a mirror image about the center point as shown in Figure 6 (p. 10).

#### The outer loop

The outer loop iterates on the rows of pixels in the top half of the image. Only the top half of the image is processed in this method because the top half will be mirrored into the bottom half later on.

# The inner loop

The inner loop iterates on the columns in the left half of the image, copying pixel colors from the left half into the pixels in the right half.

#### **Destruction of pixel colors**

The colors in the pixels in the upper-right quadrant are overwritten by this method.

In effect, this method and the one following it destroys all of the pixel colors originally in the right half of the picture of the rotated image and all of the pixel colors originally in the bottom half of the picture.

The final picture shown in Listing 4 (p. 8) contains only pixel from the upper-left quadrant of the picture with the rotated image.

#### Return a modified Picture object

Finally, the code in Listing 4 (p. 8) returns the modified **Picture** object to the **run** method in Listing 3 (p. 8).

At this point, the picture with the rotated image is replaced by the version of the picture returned by the **mirrorUpperQuads** method.

# Picture output from the mirrorUpperQuads method

If you were to display the picture at that point, you would see the image shown in Figure 6 (p. 10).



Picture output from the mirrorUpperQuads method.

Figure 6: Picture output from the mirrorUpperQuads method.

#### The upper-left quadrant has been mirrored

As you can see from Figure 6 (p. 10), at this point in the process, the upper-left quadrant has been mirrored into the upper-right quadrant, but the bottom half of the picture is undisturbed. It's time to do something about that.

#### Call the mirrorHoriz method

The next statement in the **run** method in Listing 3 (p. 8) is a call to the **mirrorHoriz** method passing the picture shown in Figure 6 (p. 10) as a parameter.

# The method named mirrorHoriz

The method named **mirrorHoriz** is shown in Listing 5 (p. 11). This method mirrors the top half of a picture into the bottom half of the same picture. It will be used to mirror the top half of the picture in Figure 6 (p. 10) into the bottom half.

## Listing 5: The method named mirrorHoriz.

```
private Picture mirrorHoriz(Picture pix){
```

```
Pixel topPixel = null;
Pixel bottomPixel = null;
int midpoint = pix.getHeight()/2;
int height = pix.getHeight();
for(int col = 0;col < pix.getWidth();col++){</pre>
```

```
for(int row = 0;row < midpoint;row++){
   topPixel = pix.getPixel(col,row);
   bottomPixel =
        pix.getPixel(col,height-1-row);
   bottomPixel.setColor(topPixel.getColor());
}//end inner loop
}//end outer loop</pre>
```

return pix;
}//end mirrorHoriz method

}//end class Prob02Runner

#### Very similar to the mirrorUpperQuads method

This method is very similar to the previous method named mirrorUpperQuads . Four working variables and a nested for loop

As before, Listing 5 (p. 11) declares and initializes four working variables. These variables are used in a nested **for** loop to copy pixel colors from the top half of the picture into the pixels in the bottom half.

#### The outer and inner loops

In this case, the outer loop iterates on all of the columns going from left to right.

The inner loop iterates on rows, from the top row to the vertical midpoint, copying the colors from the pixels from the top half into the pixels in the bottom half.

#### The end of the class

Listing 5 (p. 11) also signals the end of the class named **Prob02Runner** and the end of the program.

# 5 Run the program

I encourage you to copy the code from Listing 6 (p. 13). Compile the code and execute it. Experiment with the code, making changes, and observing the results of your changes. Make certain that you can explain why your changes behave as they do.

Click Prob02a.jpg<sup>2</sup> to download the required input image file for this program.

# 6 Summary

You learned how to mirror images both horizontally and vertically.

# 7 What's next?

In the next module, you will learn to use a variety of Java2D classes including GradientPaint.

# 8 Online video link

Select the following link to view an online video lecture on the material in this module.

• ITSE 2321 Lecture 12 <sup>3</sup>

# 9 Miscellaneous

This section contains a variety of miscellaneous information.

#### NOTE: Housekeeping material

- Module name: Java OOP: Mirroring Images
- $\bullet$  File: Java3024.htm
- Published: August 4, 2012
- Revised: November 14, 2012

NOTE: **Disclaimers:** Financial : Although the Connexions site makes it possible for you to download a PDF file for this module at no charge, and also makes it possible for you to purchase a pre-printed version of the PDF file, you should be aware that some of the HTML elements in this module may not translate well into PDF.

I also want you to know that, I receive no financial compensation from the Connexions website even if you purchase the PDF version of the module.

In the past, unknown individuals have copied my modules from cnx.org, converted them to Kindle books, and placed them for sale on Amazon.com showing me as the author. I neither receive compensation for those sales nor do I know who does receive compensation. If you purchase such a book, please be aware that it is a copy of a module that is freely available on cnx.org and that it was made and published without my prior knowledge.

**Affiliation** : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

 $<sup>^{2}</sup>$  http://cnx.org/content/m44228/1.4/Prob02a.jpg  $^{3}$  http://vimeo.com/channels/itse2321/21217708

# 10 Complete program listing

A complete listing of the program discussed in this module is shown in Listing 6 (p. 13) below.

Listing 6: Complete program listing.

```
/*File Prob02 Copyright 2008 R.G.Baldwin
import java.awt.Graphics2D;
import java.awt.geom.AffineTransform;
import java.awt.geom.Rectangle2D;
import java.awt.Graphics;
public class Prob02{
 //DO NOT MODIFY THE CODE IN THIS CLASS DEFINITION.
 public static void main(String[] args){
   new Prob02Runner().run();
 }//end main method
}//end class Prob02
class Prob02Runner{
 public Prob02Runner(){
   System.out.println("Display your name here.");
 }//end constructor
 //-----//
 public void run(){
   Picture pix = new Picture("Prob02a.jpg");
   //Add your name and display the output picture.
   pix.addMessage("Display your name here.",10,20);
   //Display the input picture.
   pix.explore();
   pix = rotatePicture(pix,35);
   pix.explore();
   pix = mirrorUpperQuads(pix);
   pix = mirrorHoriz(pix);
   pix.explore();
   System.out.println(pix);
 }//end run
 //-----//
 private Picture rotatePicture(Picture pix,
                          double angle){
   //Set up the rotation transform
   AffineTransform rotateTransform =
                              new AffineTransform();
   rotateTransform.rotate(Math.toRadians(angle),
                      pix.getWidth()/2,
```

```
pix.getHeight()/2);
```

```
//Get the required dimensions of a rectangle that will
 // contain the rotated image.
 Rectangle2D rectangle2D =
        pix.getTransformEnclosingRect(rotateTransform);
 int resultWidth = (int)(rectangle2D.getWidth());
 int resultHeight = (int)(rectangle2D.getHeight());
 //Set up the translation transform that will translate
 // the rotated image to the center of the new Picture
 // object.
 AffineTransform translateTransform =
                                 new AffineTransform();
 translateTransform.translate(
                   (resultWidth - pix.getWidth())/2,
                   (resultHeight - pix.getHeight())/2);
 //Concatenate the two transforms so that the image
 // will first be rotated around its center and then
 // translated to the center of the new Picture object.
 translateTransform.concatenate(rotateTransform);
 //Create a new Picture object to contain the results
 // of the transformation.
 Picture result = new Picture(
                           resultWidth,resultHeight);
 //Get the graphics context of the new Picture object,
 // apply the transform to the incoming picture and
 // draw the transformed picture on the new Picture
 // object.
 Graphics2D g2 = (Graphics2D)result.getGraphics();
 g2.drawImage(pix.getImage(),translateTransform,null);
 return result;
}//end rotatePicture
//-----//
//This method mirrors the upper-left quadrant of a
// picture into the upper-right quadrant.
private Picture mirrorUpperQuads(Picture pix){
 Pixel leftPixel = null;
 Pixel rightPixel = null;
 int midpoint = pix.getWidth()/2;
 int width = pix.getWidth();
 for(int row = 0;row < pix.getHeight()/2;row++){</pre>
   for(int col = 0;col < midpoint;col++){</pre>
     leftPixel = pix.getPixel(col,row);
     rightPixel =
                pix.getPixel(width-1-col,row);
     rightPixel.setColor(leftPixel.getColor());
```

```
}//end inner loop
 }//end outer loop
 return pix;
}//end mirrorUpperQuads
//-----//
//This method mirrors the top half of a picture into
// the bottom half.
private Picture mirrorHoriz(Picture pix){
 Pixel topPixel = null;
 Pixel bottomPixel = null;
 int midpoint = pix.getHeight()/2;
 int height = pix.getHeight();
 for(int col = 0;col < pix.getWidth();col++){</pre>
   for(int row = 0;row < midpoint;row++){</pre>
     topPixel = pix.getPixel(col,row);
     bottomPixel =
               pix.getPixel(col,height-1-row);
     bottomPixel.setColor(topPixel.getColor());
   }//end inner loop
 }//end outer loop
 return pix;
}//end mirrorHoriz
//-----//
```

}//end class Prob02Runner

-end-