Java OOP: Clipping Images[*]

R.G. (Dick) Baldwin

Abstract

Learn how to use shapes to clip images during the drawing process.

1 Table of Contents

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.

*Version 1.3: Nov 14, 2012 12:18 pm -0600
†http://creativecommons.org/licenses/by/3.0/
1http://cnx.org/content/m44148/latest/

2.1.1 Figures

2.1.2 Listings

3 Preview

In this module, you will learn how to use shapes to clip images during the drawing process.

Program specifications

Write a program named **Prob04** that uses the class definition shown in Listing 1 (p. 6) and Ericson's media library along with the image file named **Prob04a.jpg** *(see Figure 1 (p. 3))* to produce the graphic output images shown in Figure 2 (p. 4) and Figure 3 (p. 5) . Don't forget to display your name in the output image as shown.

Input file named Prob04a.jpg.

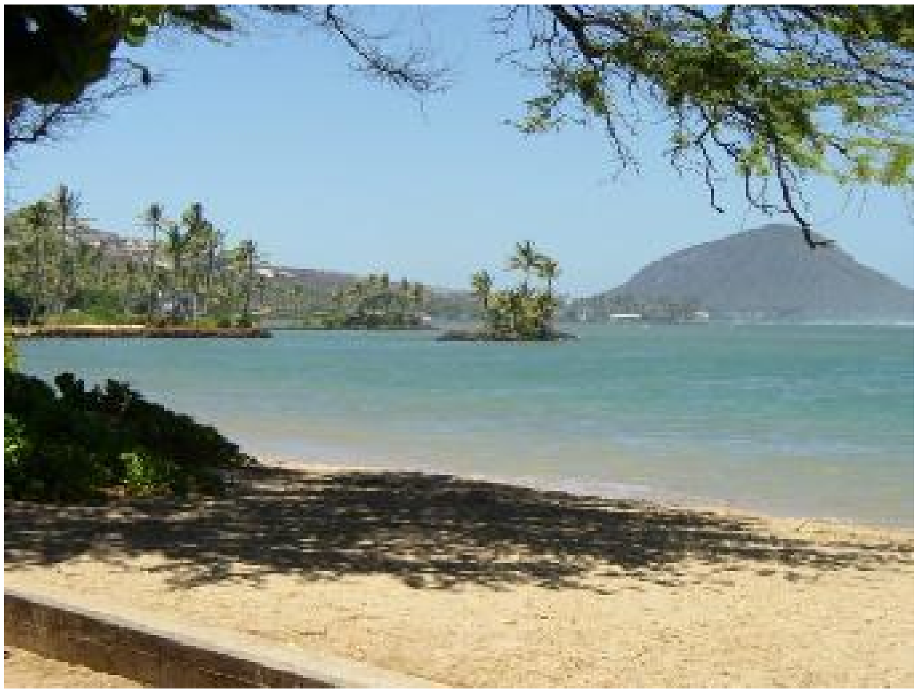
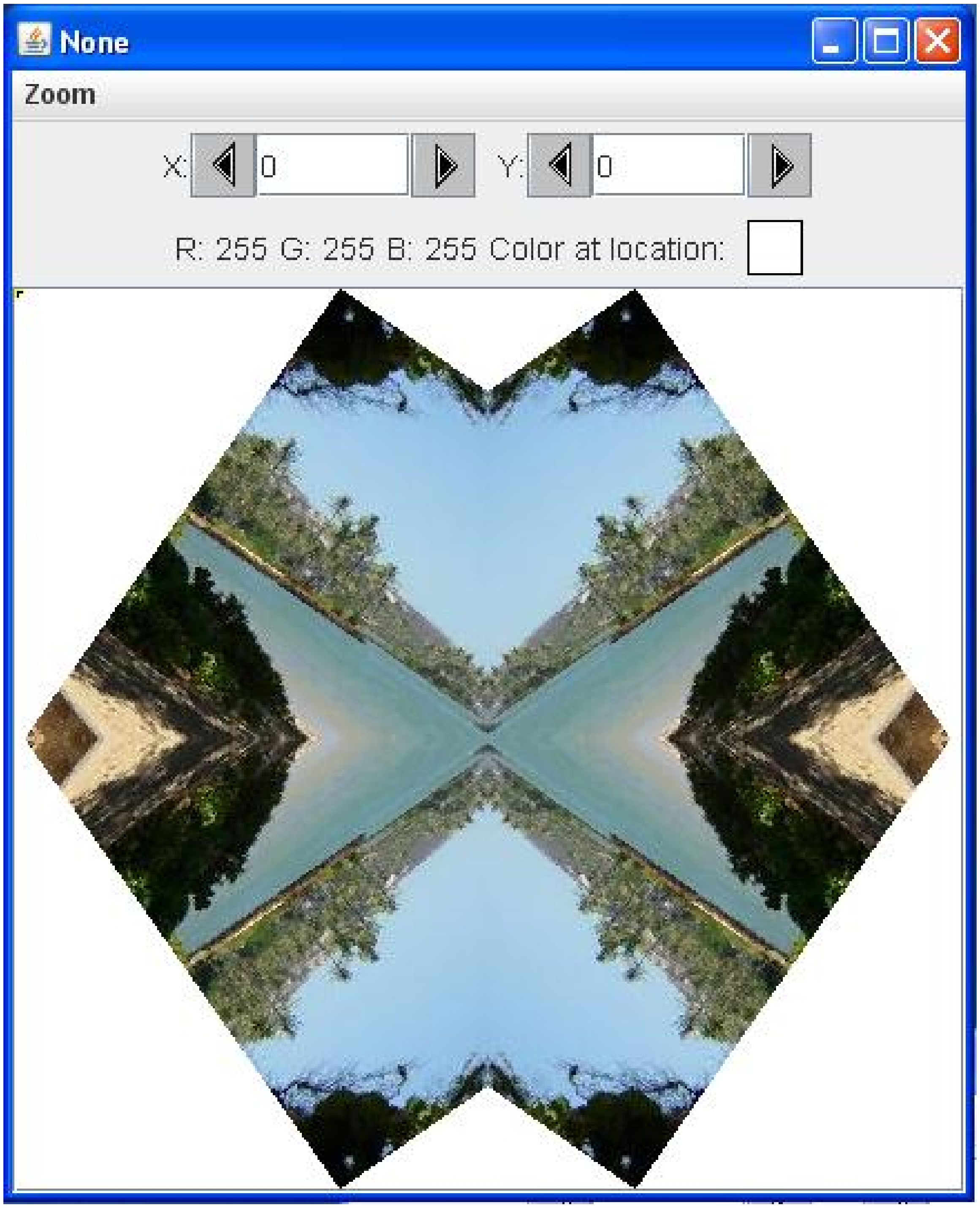


Figure 1: Input file named Prob04a.jpg.

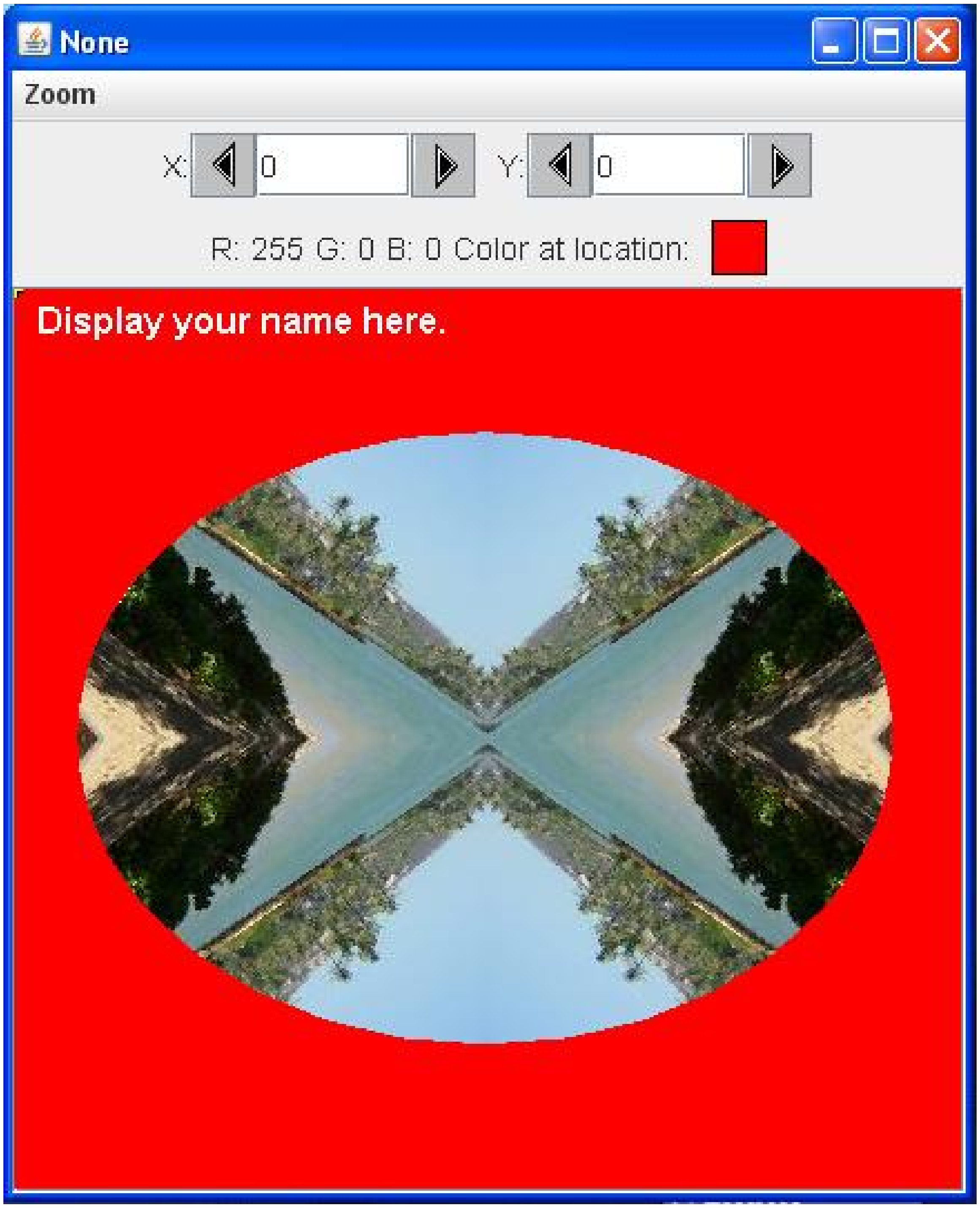
First output image.



R: 255 G: 255 B: 255 Color at location:

Figure 2: First output image.

Second output image.



R: 255 G: 0 B: 0 Color at location: 🟥

Display your name here.

Figure 3: Second 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 **Prob04** shown in Listing 1 (p. 6) .

Rotate, mirror, and clip

The program rotates a **Picture** object by 35 degrees with no scaling. Then it does a four-way mirror on the rotated picture. Finally, it clips the image to an elliptical format as shown in Figure 3 (p. 5) .

Required output text

In addition to the two output images shown above, your program must display your name and the other line of text shown in Figure 4 (p. 6) .

Required text output.

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

Figure 4: Required text output.

---

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 5 (p. 10) near the end of the module.

The driver class named Prob04

The driver class containing the **main** method is shown in Listing 1 (p. 6) .

Listing 1: The driver class named Prob04.

```
    public class Prob04{
  public static void main(String[] args){
    new Prob04Runner().run();
  }//end main method
}//end class Prob04
```

If you have been studying the earlier modules in this collection, no explanation of Listing 1 (p. 6) should be required.

Beginning of the class named Prob04Runner

The class named **Prob04Runner** begins in Listing 2 (p. 6) .

Listing 2: Beginning of the class named Prob04Runner.

```
    class Prob04Runner{

  public Prob04Runner(){
    System.out.println("Display your name here.");
  }//end constructor
  //--------------------------------------------------//

  public void run(){
    Picture pix = new Picture("Prob04a.jpg");

    //Rotate and mirror the picture.
    pix = rotatePicture(pix,35);
    pix = mirrorUpperQuads(pix);
    pix = mirrorHoriz(pix);

    pix.explore();
```

Nothing new here

There is nothing new in Listing 2 (p. 6) .

After instantiating a new **Picture** object from the given image file, Listing 2 (p. 6) calls three methods to rotate, mirror, and display the picture, producing the graphic output shown in Figure 2 (p. 4) .

All of the code to accomplish this is essentially the same as code that I have explained in earlier modules.

Clip the picture and display your name

Then Listing 3 (p. 7) calls the **clipToEllipse** method to clip the picture to an ellipse on a red background as shown in Figure 3 (p. 5) . The **clipToEllipse** method is new to this module, so I will explain it shortly.

Listing 3: Clip the picture and display your name.

```
        pix = clipToEllipse(pix);

    //Add your name and display the output picture.
    pix.addMessage("Display your name here.",10,20);
    pix.explore();

    System.out.println(pix);
  }//end run
```

The remaining code in Listing 3 (p. 7) is a repeat of code that I have explained in earlier modules, so I won't have anything further to say about it.

The method named clipToEllipse

The method named **clipToEllipse** is shown in its entirety in Listing 4 (p. 7) .

Listing 4: The method named clipToEllipse.

```
      private Picture clipToEllipse(Picture pix){
    Picture result =
              new Picture(pix.getWidth(),pix.getHeight());
    result.setAllPixelsToAColor(Color.RED);

    //Get the graphics2D object
    Graphics2D g2 = (Graphics2D)(result.getGraphics());

    //Create an ellipse for clipping
    Ellipse2D.Double ellipse =
      new Ellipse2D.Double(28,64,366,275);

    //Use the ellipse for clipping
    g2.setClip(ellipse);

    //Draw the image
    g2.drawImage(pix.getImage(),0,0,pix.getWidth(),
                                pix.getHeight(),
                                null);

    return result;
  }//end clipToEllipse
```

Behavior of the clipToEllipse method

The **clipToEllipse** method receives an incoming parameter that is a reference to an object of the **Picture** class. Basically, here is what the method does:

- Instantiate a **Picture** object with an all white background that is the same size as the incoming **Picture** object.
- Call Ericson's **setAllPixelsToAColor** method to convert the white background into a red background.
- Call Ericson's **getGraphics** method to get the **Graphics** object encapsulated in the red **Picture** object.
- Cast the **Graphics** object's reference to type **Graphics2D** .
- Construct a new **Ellipse2D.Double** object with the position, width, and height specified by the constructor parameters.
- Call Sun's **setClip** method to set the clipping area on the red **Picture** object to match the position and shape of the ellipse.
- Call Ericson's **getImage** method to get the **Image** object encapsulated in the incoming **Picture** object.
- Call Sun's **drawImage** method to draw that portion of the incoming picture that fits inside the ellipse on the red **Picture** object.

The new code

The only code in Listing 4 (p. 7) that is new to this module is the call to the **setClip** method.

The **setClip** method is defined in the **Graphics** class and inherited into the **Graphics2D** class.

*(Among other things, that means that it wasn't necessary for me to cast the **Graphics** object to type **Graphics2D** in Listing 4 (p. 7) .)*

The setClip method

There are a couple of overloaded versions of the **setClip** method. The one used in Listing 4 (p. 7) requires an incoming parameter of the interface type **Shape** .

The Shape interface

Briefly, Sun tells us that the **Shape** interface *"provides definitions for objects that represent some form of geometric shape."*

There are several dozen classes that implement the **Shape** interface, one of which is the class named **Ellipse2D.Double** . Therefore, the object of that type that is instantiated in Listing 4 (p. 7) satisfies the type requirement for being passed to the **setClip** method.

Behavior of the setClip method

With regard to the behavior of the **setClip** method, Sun tells us that the method

"Sets the current clipping area to an arbitrary clip shape."

What is the significance of the clipping area?

The closest answer that I can find for that question is the following statement in Sun's description of the **Graphics** class:

"All rendering operations modify only pixels which lie within the area bounded by the current clip, which is specified by a ***Shape*** *in user space and is controlled by the program using the* ***Graphics*** *object."*

In other words...

The *clipping area* is analogous to the *current clip* . In this case, the position and shape of the current clip is the position and shape of the ellipse.

When the image is later drawn on the red **Picture** object, only those pixels within the ellipse are modified to show the image. The remaining pixels retain their original color, which was set to red early in Listing 4 (p. 7) .

End of discussion

That concludes my explanation of this program. You will find the methods that I didn't discuss in Listing 5 (p. 10) near the end of the module.

5 Run the program

I encourage you to copy the code from Listing 5 (p. 10) . 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 Prob04a.jpg [2] to download the required input image file.

6 Summary

In this module, you learned how to use shapes to clip images during the drawing process.

7 What's next?

In the next module, you will learn how to merge pictures.

8 Online video link

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

- ITSE 2321 Lecture 14 [3]

9 Miscellaneous

This section contains a variety of miscellaneous information.

NOTE: **Housekeeping material**

- Module name: Java OOP: Clipping Images
- File: Java3028.htm
- Published: August 6, 2012
- Revised: November 14, 2012

[2]http://cnx.org/content/m44246/1.3/Prob04a.jpg
[3]http://vimeo.com/channels/itse2321/21221510

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.

10 Complete program listing

A complete listing of the program discussed in this module is provided in Listing 5 (p. 10) below.

Listing 5: Complete program listing.

```
    /*File Prob04 Copyright 2008 R.G.Baldwin
*********************************************************/
import java.awt.Graphics2D;
import java.awt.geom.AffineTransform;
import java.awt.geom.Rectangle2D;
import java.awt.Graphics;
import java.awt.geom.Ellipse2D;
import java.awt.Color;

public class Prob04{
  public static void main(String[] args){
    new Prob04Runner().run();
  }//end main method
}//end class Prob04
//=======================================================//

class Prob04Runner{
  public Prob04Runner(){
    System.out.println("Display your name here.");
  }//end constructor
  //-----------------------------------------------------//
  public void run(){
    Picture pix = new Picture("Prob04a.jpg");

    //Rotate and mirror the picture.
    pix = rotatePicture(pix,35);
    pix = mirrorUpperQuads(pix);
    pix = mirrorHoriz(pix);
    pix.explore();

    //Clip the picture to an ellipse on a red background.
    pix = clipToEllipse(pix);

    //Add your name and display the output picture.
    pix.addMessage("Display your name here.",10,20);
    pix.explore();

    System.out.println(pix);
  }//end run
  //----------------------------------------------------//

  private Picture clipToEllipse(Picture pix){
    Picture result =
             new Picture(pix.getWidth(),pix.getHeight());
    result.setAllPixelsToAColor(Color.RED);

    //Get the graphics2D object
    Graphics2D g2 = (Graphics2D)(result.getGraphics());

    //Create an ellipse for clipping
    Ellipse2D.Double ellipse =
      new Ellipse2D.Double(28,64,366,275);

    //Use the ellipse for clipping
    g2.setClip(ellipse);

    //Draw the image
    g2.drawImage(pix.getImage(),0,0,pix.getWidth(),
                              pix.getHeight(),
                              null);

    return result;
  }//end clipToEllipse
  //----------------------------------------------------//

  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++){
      for(int col = 0;col < midpoint;col++){
        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++){
      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

    return pix;
  }//end mirrorHoriz
  //----------------------------------------------------//

}//end class Prob04Runner
```

-end-