Java OOP: Controlling Image Rotation with a Slider and Affine Transforms*

R.G. (Dick) Baldwin

This work is produced by OpenStax-CNX and licensed under the
Creative Commons Attribution License 3.0†

Abstract

Learn to use a JSlider object along with Affine Transforms to control the rotation of an image.

1 Table of Contents

- Preface (p. 1)
  - Viewing tip (p. 2)
    - Figures (p. 2)
    - Listings (p. 2)
- Preview (p. 2)
- Discussion and sample code (p. 8)
- Run the program (p. 13)
- Summary (p. 13)
- What's next? (p. 13)
- Miscellaneous (p. 13)
- Complete program listing (p. 14)

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] .

*Version 1.3: Nov 10, 2012 4:44 pm +0000
†http://creativecommons.org/licenses/by/3.0/
[1]http://cnx.org/content/m44148/latest/

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.

2.1.1 Figures

- Figure 1 (p. 3) . Program output at startup.
- Figure 2 (p. 5) . Program output for slider at zero.
- Figure 3 (p. 7) . Program output for slider at 240.

2.1.2 Listings

- Listing 1 (p. 8) . Beginning of the class named Prob10Runner.
- Listing 2 (p. 9) . Beginning of the constructor for the Prob10Runner class.
- Listing 3 (p. 9) . Construct the GUI for the slider.
- Listing 4 (p. 10) . Rotate and display the butterfly.
- Listing 5 (p. 10) . Register a ChangeListener on the slider.
- Listing 6 (p. 11) . Beginning of the rotatePicture method.
- Listing 7 (p. 12) . Set up the translation transform.
- Listing 8 (p. 12) . Concatenate the transforms.
- Listing 9 (p. 12) . Transform and draw the butterfly image.
- Listing 10 (p. 14) . Complete program listing.

3 Preview

In this lecture, I will explain a program that uses **Affine Transforms** to rotate an image by a specified angle around a specified anchor point.

Then the program translates the image so as to center it in a **JFrame** object.

Specification of the rotation angle

A **JSlider** is used to specify the rotation angle.

The range of the slider is from 0 to 360 degrees.

The position of the thumb on the slider specifies a counter-clockwise rotation angle in degrees.

Program output at startup

The thumb on the slider is at 45 degrees when the program starts running.

This causes the initial rotation angle of the butterfly image to be 45 degrees counter-clockwise as shown in Figure 1 (p. 3) .

Program output at startup.

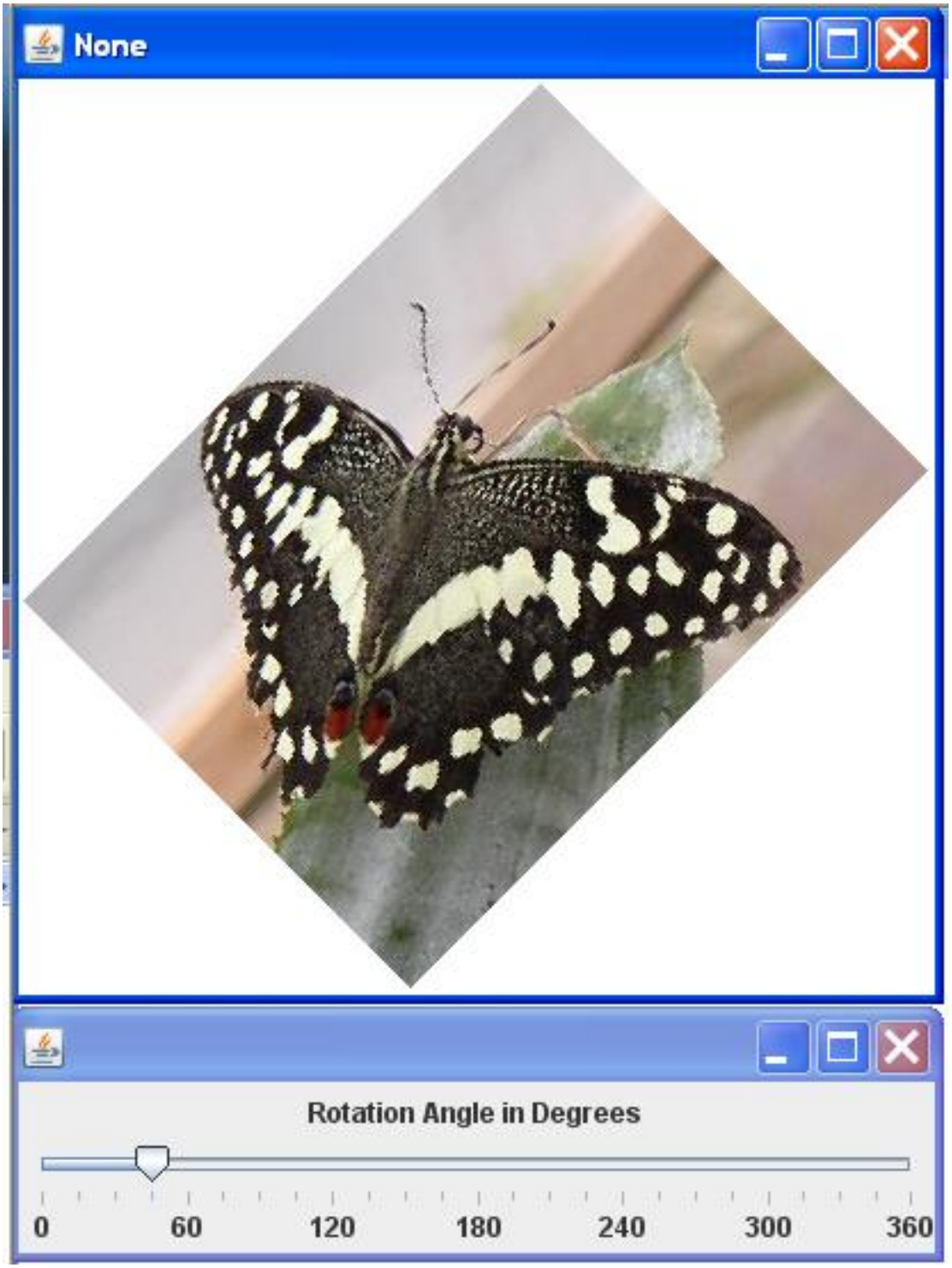

Figure 1: Program output at startup.

Corners barely graze inner-edges of the frame's borders

Note that for the initial rotation angle shown in Figure 1 (p. 3) , the corners of the image almost touch the inner-edges of the borders on the frame.

If you run the program and rotate the image, you will see that the size of the frame is such that the corners of the image barely graze the inner-edges for those cases where the diagonals of the image are horizontal and vertical.

Program output for slider at zero degrees

Figure 2 (p. 5) shows the program output when the thumb on the slider has been moved to zero degrees at the far-left end of the slider.

Program output for slider at zero.

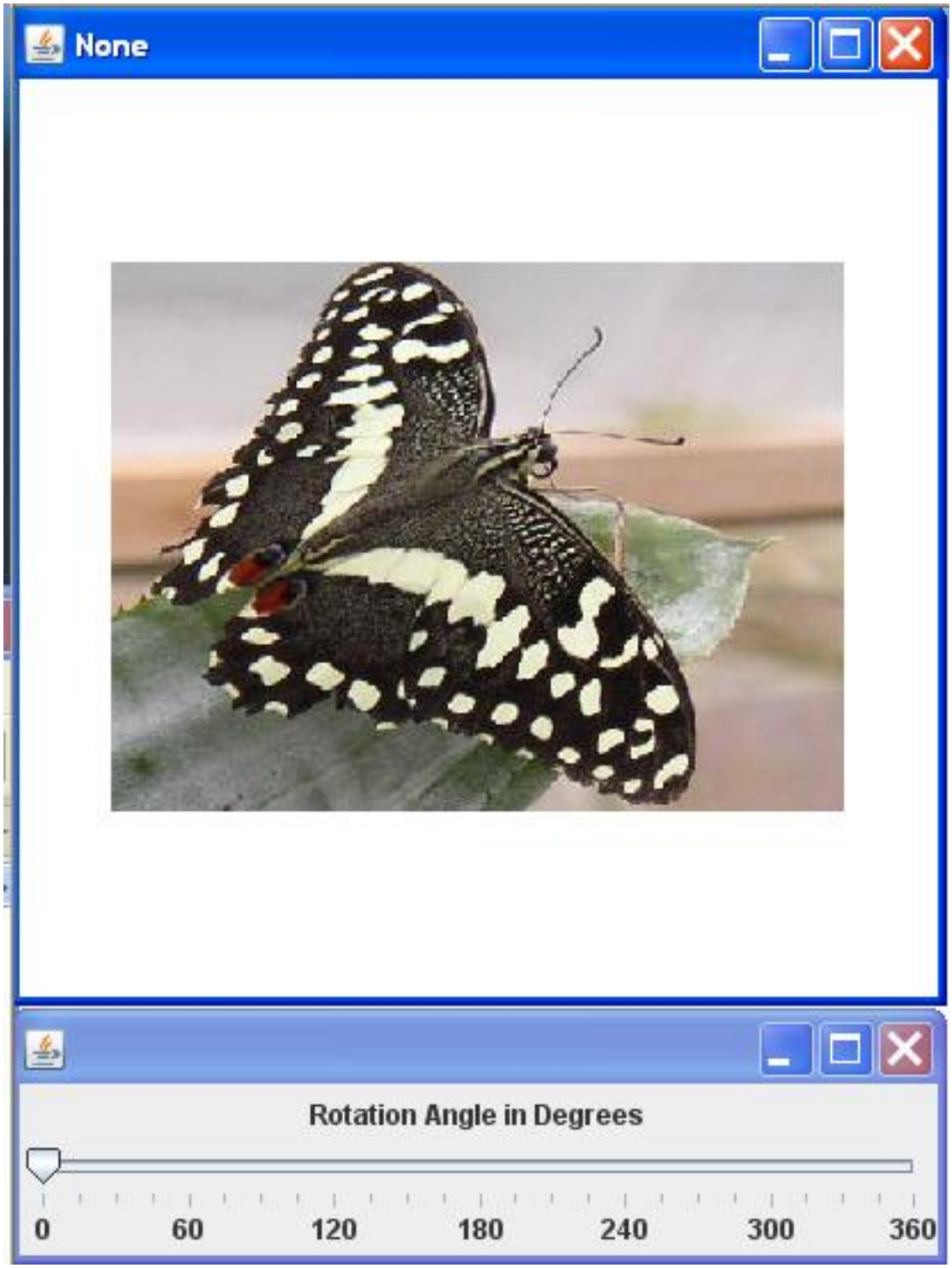

Figure 2: Program output for slider at zero.

Program output for slider at 240 degrees

Figure 3 (p. 7) shows the result of positioning the thumb on the slider at 240 degrees.

Program output for slider at 240.

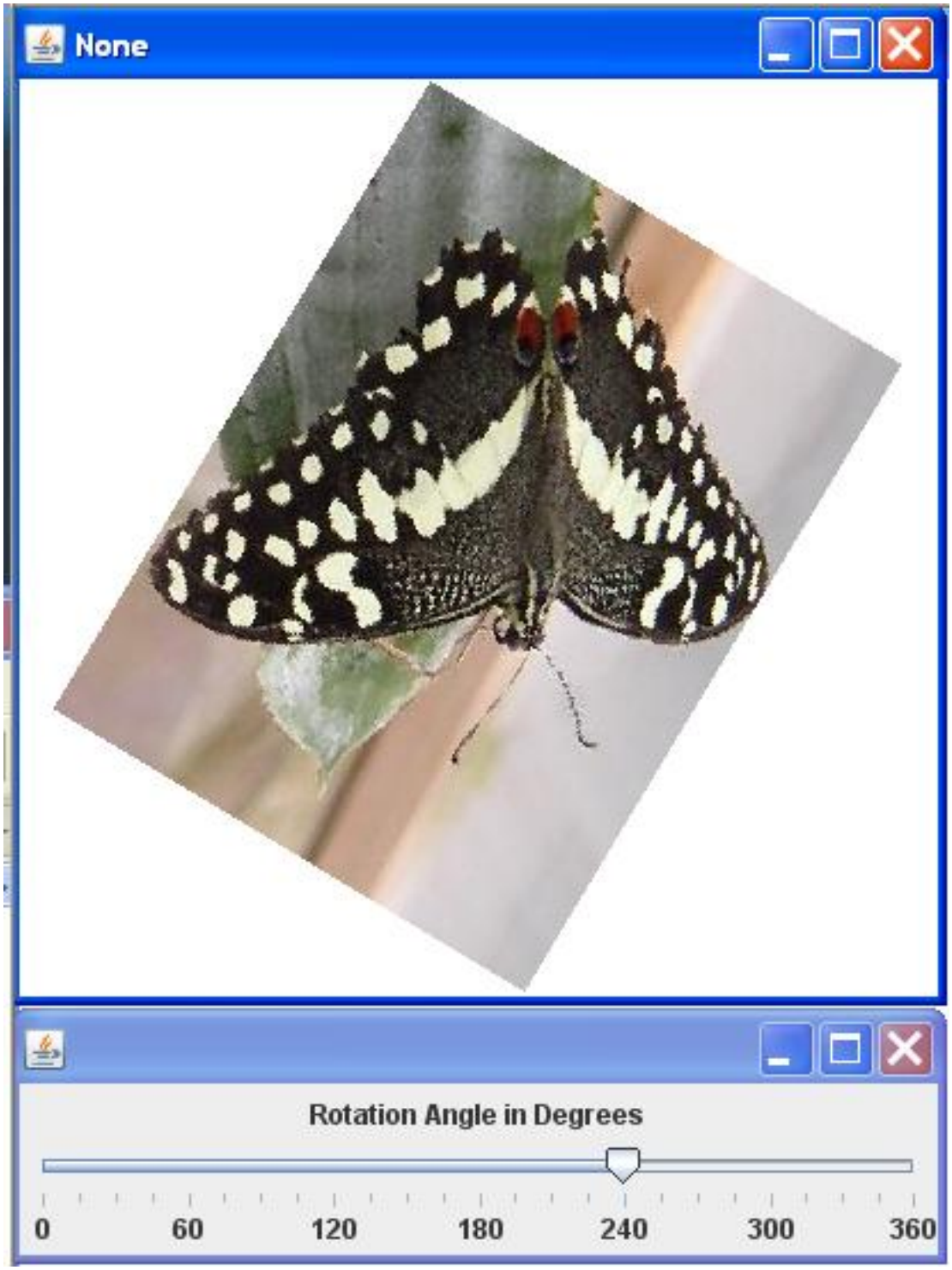

Figure 3: Program output for slider at 240.

Brief program specifications

Write a program named **Prob10** that uses the **Prob10** class definition shown in Listing 10 (p. 14) and Ericson's media library along with the image file named Prob10.jpg [2] to produce the graphic output images shown in Figure 1 (p. 3) through Figure 3 (p. 7) .

The butterfly image rotates smoothly around its center as you move the slider back and forth.

The program must terminate and return control to the operating system when you click the large X in the upper-right corner of the GUI containing the slider.

General background information

The overall structure of this program is very similar to programs that I explained in earlier lectures titled

- Controlling an Edge Detector with a Slider [3] , and
- Controlling an Image-Scaling Program with a Slider [4] .

The most significant difference...

Is the code that is executed when the slider fires a **ChangeEvent** . There are a few other differences as well.

New and different code

I will explain the code that is new and different in this program.

I will refer you back to the earlier lectures for an explanation of the remainder of the code.

You can view the entire program in Listing 10 (p. 14) .

4 Discussion and sample code

Beginning of the class named Prob10Runner

The code in the **main** method in Listing 10 (p. 14) instantiates a new object of the class named **Prob10Runner** .

Listing 1 (p. 8) shows the beginning of the class named **Prob10Runner** .

Listing 1 (p. 8) declares several instance variables and initializes some of them.

I will refer back to some of these variables in later paragraphs.

Listing 1: Beginning of the class named Prob10Runner.

```
 class Prob10Runner extends JFrame{

private JPanel mainPanel = new JPanel();
private JPanel titlePanel = new JPanel();

//Instantiate a new slider setting the limits and the
// initial position of the thumb.
private JSlider slider = new JSlider(0,360,45);

private Picture butterfly = new Picture("Prob10.jpg");
private Picture background = null;

private int butterflyWidth = butterfly.getWidth();
private int butterflyHeight = butterfly.getHeight();

private Picture display = null;
```

[2] http://cnx.org/content/m44915/latest/Prob10.jpg
[3] http://cnx.org/content/m44913/latest
[4] http://cnx.org/content/m44914/latest

```
private int displayWidth = 0;
private int displayHeight = 0;

private Image image = null;
private Graphics graphics = null;
private AffineTransform rotateTransform = null;
private AffineTransform translateTransform = null;
private Graphics2D g2 = null;
```

Beginning of the constructor for the Prob10Runner class

Listing 2 (p. 9) shows the beginning of the constructor for the class named **Prob10Runner** .

Listing 2: Beginning of the constructor for the Prob10Runner class.

```
  public Prob10Runner(){//constructor

setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

//Compute the minimum dimensions allowed for the
// display window that will contain the butterfly
// image rotated at any angle. This turns out to
// be a square with the length of each side equal
// to the diagonal length of the butterfly picture.
//The length of each side was increased by one
// pixel to guard against loss of precision when
// converting from double to int.
int diagonal = 1 + (int)(Math.sqrt(
                    butterflyWidth*butterflyWidth +
                    butterflyHeight*butterflyHeight));

//Instantiate the picture in which the rotated
// butterfly image will be displayed.
display = new Picture(diagonal,diagonal);
displayWidth = displayHeight = diagonal;

//This picture provides a white background the same
// size as the display picture.
background = new Picture(diagonal,diagonal);
```

Not much that is new here

The only thing that might be considered new in Listing 2 (p. 8) is the code that computes the minimum dimensions for a display window that will contain the butterfly image rotated by any angle.

The rationale for this computation is explained in the comments.

Construct the GUI for the slider

The code in Listing 3 (p. 9) constructs the GUI that contains the slider shown in Figure 1 (p. 3) .

Although the code in Listing 3 (p. 9) is a little tedious, there are no new concepts associated with that code.

Listing 3: Construct the GUI for the slider.

```
    //Construct the GUI for the slider.
slider.setMajorTickSpacing(60);
slider.setMinorTickSpacing(15);
slider.setPaintTicks(true);
slider.setPaintLabels(true);

mainPanel.setLayout(new BorderLayout());
titlePanel.add(new JLabel(
                        "Rotation Angle in Degrees"));
mainPanel.add(titlePanel,BorderLayout.NORTH);
mainPanel.add(slider,BorderLayout.CENTER);

getContentPane().add(mainPanel);
pack();//Adjust the size of the slider GUI.

//Compute an improved size and location for the
// GUI containing the slider.
setSize(displayWidth + 2 * getInsets().left,
        mainPanel.getHeight() + slider.getHeight());
setLocation(0,displayHeight + getInsets().top
                            + getInsets().bottom + 1);
setVisible(true);
```

Rotate and display the butterfly

The code in Listing 4 (p. 10) calls the method named **rotatePicture** , passing the initial value of the thumb on the slider as a parameter to cause the initial display of the butterfly to be properly rotated.

Then Listing 4 (p. 10) calls the **show** method on the display to cause the rotated butterfly image to be displayed as shown in Figure 1 (p. 3) .

Listing 4: Rotate and display the butterfly.

```
    rotatePicture(slider.getValue());

display.show();
```

Register a ChangeListener on the slider

The code in Listing 5 (p. 10) registers an anonymous **ChangeListener** object on the slider.

Listing 5: Register a ChangeListener on the slider.

```
    slider.addChangeListener(
  new ChangeListener(){
    public void stateChanged(ChangeEvent e){
      //Restore the background image of the display
      // to all white.
      graphics = display.getGraphics();
      graphics.drawImage(
                      background.getImage(),0,0,null);

      //Rotate the butterfly image, draw it on the
      // display, and repaint the display on the
```

```
          // screen..
          rotatePicture(slider.getValue());
          display.repaint();
        }//end stateChanged
      }//end new ChangeListener
    );//end addChangeListener
    //--------------------------------------------------//
  }//end constructor
```

Each time the slider fires a **ChangeEvent** , the **stateChanged** method in Listing 5 (p. 10) is executed.

The **stateChanged** method begins by restoring the background image of thedisplay.

Then the **stateChanged** method calls the **rotatePicture** method to draw a rotated version of thebutterfly on top of the background image using theslider value, *(which ranges from 0 to +360)* , asthe rotation angle in degrees.

The image of thebutterfly *(see Figure 1* (p. 3)) is always centered in the displaypicture.

Finally, the **stateChanged** method causes the display to be repainted to force the rotated image to appear on the screen.

End of the constructor

Listing 5 (p. 10) signals the end of the constructor for the class named **Prob10Runner** .

Beginning of the rotatePicture method

The **rotatePicture** method begins in Listing 6 (p. 11) .

Listing 6: Beginning of the rotatePicture method.

```
  private void rotatePicture(double angle){

 //Set up the rotation transform
 rotateTransform = new AffineTransform();
 //Negate the angle for counter-clockwise rotation.
 rotateTransform.rotate(-Math.toRadians(angle),
                        butterflyWidth/2,
                        butterflyHeight/2);
```

The **rotatePicture** method accepts a rotation angle in degrees as an incoming parameter.

The **rotatePicture** method

- rotates a butterfly image by that angle around its center,
- translates the rotated image, and
- draws the rotated image in the center of a display picture.

Set up the rotation transform

Listing 6 (p. 11) begins by instantiating a new object of the **AffineTransform** class.

Then it calls the **rotate** method on the transform object to configure that object for use in rotating the image of the butterfly.

Overloaded versions of the *rotate* method

There are several overloaded versions of the **rotate** method.

The version called in Listing 6 (p. 11) requires three parameters:

- The rotation angle in radians
- The X coordinate of the anchor point *(point around which the rotation will take place)* .
- The Y coordinate of the anchor point.

A positive rotation value indicates a clockwise rotation.

Convert from degrees to radians

Listing 6 (p. 11) converts the incoming angle in degrees to radians

The code in Listing 6 (p. 11) applies a minus sign to convert the angle from a positive value to a negative value for ***counter-clockwise*** rotation, and passes that value as the rotation parameter.

Specify the rotation *anchor point*

Listing 6 (p. 11) also specifies the center of the butterfly image as the anchor point.

Set up the translation transform

Listing 7 (p. 12) sets up the translation transform that will be used to translate the rotated image to the center of the new **Picture** object.

Listing 7: Set up the translation transform.

```
    translateTransform = new AffineTransform();
 translateTransform.translate(
                  (displayWidth - butterflyWidth)/2,
                  (displayHeight - butterflyHeight)/2);
```

Listing 7 (p. 12) instantiates a new **AffineTransform** object.

Then it calls the **translate** method to configure that object for use in translating *(moving)* the rotated image of the butterfly.

The translation transform object will be concatenated with the rotation object from Listing 6 (p. 11) to produce an **AffineTransform** object that can ***rotate*** and ***translate*** , in that order.

Concatenate the transforms

Listing 8 (p. 12) concatenates the **AffineTransform** object referred to by **translateTransform** with the transform object referred to by **rotateTransform** .

Listing 8: Concatenate the transforms.

```
    translateTransform.concatenate(rotateTransform);
```

The resulting composite transform

The concatenation in Listing 8 (p. 12) results in an **Affine Transform** object referred to by **translateTransform** that will first rotate the image around its center and then translate the rotated image to the center of the new **Picture** object.

The order of operations is very important

When rotating and translating images, it is important that the two operations be performed in the correct order.

Otherwise, the results might not be what you want.

Transform and draw the butterfly

Listing 9 (p. 12) applies the composite transform to the butterfly image and draws the transformed image on the output picture as shown in Figure 1 (p. 3) , Figure 2 (p. 5) , and Figure 3 (p. 7) .

Listing 9: Transform and draw the butterfly image.

```
    Graphics2D g2 = (Graphics2D)display.getGraphics();
 g2.drawImage(butterfly.getImage(),
              translateTransform,
              null);

}//end rotatePicture
```

```
}//end class Prob10Runner
```

The end of the program

Listing 9 (p. 12) also signals the end of the **rotatePicture** method, the end of the **Prob10Runner** class, and the end of the program.

5 Run the program

I encourage you to copy the code from Listing 10 (p. 14) . 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.

6 Summary

In this lecture, you learned how to use a **JSlider** object along with **Affine Transforms** to control the rotation of an image.

7 What's next?

In the next module, you will learn how to open an image file *(specified by a string in a text field)* in a **PictureExplorer** object.

8 Miscellaneous

This section contains a variety of miscellaneous information.

NOTE: **Housekeeping material**

- Module name: Java OOP: Controlling Image Rotation with a Slider and Affine Transforms
- File: Java3120.htm
- Published: 09/07/12

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.

9 Complete program listing

A complete listing of the program discussed in this lecture is shown in Listing 10 (p. 14) below.

Listing 10: Complete program listing.

```
    /*File Prob10 Copyright 2008 R.G.Baldwin
Revised 09/14/10

*********************************************************/
import java.awt.geom.AffineTransform;
import java.awt.Graphics2D;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.Image;
import java.awt.BorderLayout;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JSlider;
import javax.swing.JLabel;
import javax.swing.event.ChangeListener;
import javax.swing.event.ChangeEvent;

public class Prob10{
  public static void main(String[] args){
    new Prob10Runner();
  }//end main method
}//end class Prob10
/*********************************************************/

class Prob10Runner extends JFrame{

  private JPanel mainPanel = new JPanel();
  private JPanel titlePanel = new JPanel();

  //Instantiate a new slider setting the limits and the
  // initial position of the thumb.
  private JSlider slider = new JSlider(0,360,45);

  private Picture butterfly = new Picture("Prob10.jpg");
  private Picture background = null;

  private int butterflyWidth = butterfly.getWidth();
  private int butterflyHeight = butterfly.getHeight();

  private Picture display = null;
  private int displayWidth = 0;
  private int displayHeight = 0;
```

```
  private Image image = null;
  private Graphics graphics = null;
  private AffineTransform rotateTransform = null;
  private AffineTransform translateTransform = null;
  private Graphics2D g2 = null;

  public Prob10Runner(){//constructor

    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    //Compute the minimum dimensions allowed for the
    // display window that will contain the butterfly
    // image rotated at any angle. This turns out to
    // be a square with the length of each side equal
    // to the diagonal length of the butterfly picture.
    //The length of each side was increased by one
    // pixel to guard against loss of precision when
    // converting from double to int.
    int diagonal = 1 + (int)(Math.sqrt(
                        butterflyWidth*butterflyWidth +
                        butterflyHeight*butterflyHeight));

    //Instantiate the picture in which the rotated
    // butterfly image will be displayed.
    display = new Picture(diagonal,diagonal);
    displayWidth = displayHeight = diagonal;

    //This picture provides a white background the same
    // size as the display picture.
    background = new Picture(diagonal,diagonal);

    //Construct the GUI for the slider.
    slider.setMajorTickSpacing(60);
    slider.setMinorTickSpacing(15);
    slider.setPaintTicks(true);
    slider.setPaintLabels(true);

    mainPanel.setLayout(new BorderLayout());
    titlePanel.add(new JLabel(
                           "Rotation Angle in Degrees"));
    mainPanel.add(titlePanel,BorderLayout.NORTH);
    mainPanel.add(slider,BorderLayout.CENTER);

    getContentPane().add(mainPanel);
    pack();//Adjust the size of the slider GUI.

    //Compute an improved size and location for the
    // GUI containing the slider.
    setSize(displayWidth + 2 * getInsets().left,
            mainPanel.getHeight() + slider.getHeight());
    setLocation(0,displayHeight + getInsets().top
```

```
                              + getInsets().bottom + 1);
    setVisible(true);

    //Place the butterfly image in the display picture
    // with a rotation angle specified by the initial
    // position of the thumb on the slider.

    rotatePicture(slider.getValue());

    display.show();
    //---------------------------------------------------//
    //Register an anonymous listener object on the slider.
    //Each time the slider fires a ChangeEvent, this event
    // handler restores the background image of the
    // display. Then it draws a rotated version of the
    // butterfly on top of the background image using the
    // slider value, which ranges from 0 to +360, as
    // the rotation angle in degrees. The image of the
    // butterfly is always centered in the display
    // picture.
    slider.addChangeListener(
      new ChangeListener(){
        public void stateChanged(ChangeEvent e){
          //Restore the background image of the display
          // to all white.
          graphics = display.getGraphics();
          graphics.drawImage(
                          background.getImage(),0,0,null);
          //Rotate the butterfly image, draw it on the
          // display, and repaint the display on the
          // screen..
          rotatePicture(slider.getValue());
          display.repaint();
        }//end stateChanged
      }//end new ChangeListener
    );//end addChangeListener
    //---------------------------------------------------//
  }//end constructor
  //---------------------------------------------------//

  //This method accepts a rotation angle in degrees. It
  // rotates a butterfly image by that angle around its
  // center, translates, and draws the rotated image in
  // the center of a display picture.
  private void rotatePicture(double angle){

    //Set up the rotation transform
    rotateTransform = new AffineTransform();
    //Negate the angle for counter-clockwise rotation.
    rotateTransform.rotate(-Math.toRadians(angle),
                           butterflyWidth/2,
```

```
                          butterflyHeight/2);

    //Set up the translation transform that will translate
    // the rotated image to the center of the new Picture
    // object.
    translateTransform = new AffineTransform();
    translateTransform.translate(
                    (displayWidth - butterflyWidth)/2,
                    (displayHeight - butterflyHeight)/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);

    //Get the graphics context of the display picture,
    // apply the transform to the butterfly image, and
    // draw the transformed picture on the display
    // picture.
    Graphics2D g2 = (Graphics2D)display.getGraphics();
    g2.drawImage(butterfly.getImage(),
                 translateTransform,
                 null);

  }//end rotatePicture

}//end class Prob10Runner
```

-end-