Java OOP: Controlling Opacity with a Slider[*]

Richard Baldwin

Abstract

Learn how to use a slider to continuously change the opacity of an image and to draw that image onto a background image.

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: Sep 6, 2012 1:36 pm -0500
[†]http://creativecommons.org/licenses/by/3.0/
[1]http://cnx.org/content/m44148/latest/

2.1.1 Figures

2.1.2 Listings

3 Preview

The primary objective of this module is to illustrate how to use a slider to continuously change the opacity of an image and to draw that image onto a background image.

Two approaches

This module builds on an earlier module involving transparency. In that module, you learned that there are at least two ways to incorporate alpha transparency into Ericson's media library, The easiest way, which is not necessarily the best way, is to make a relatively simple modification to a constructor in Ericson's **SimplePicture** class. That is the approach used in this module.

The second approach

The second approach is more complicated, but does not require the modification of the classes in Ericson's library. That is probably a better approach due simply to the fact that modifications to Ericson's library are not required. However, that approach is not shown in this module.

Outside research

As with the earlier module, the program that I will explain in this module may require a significant amount of outside research on the part of the student in order to learn about:

- Alpha transparency
- A buffered image of type **TYPE_INT_ARGB**
- The ability to use Ericson's **getBasicPixel** and **setBasicPixel** methods,
- The use of the bitwise AND, OR, and left-shift operators.
- The use of the **drawImage** method of the **Graphics** class.

Modification to the SimplePicture class

The student must modify the **SimplePicture** class to cause the buffered image used to store the image to be **TYPE_INT_ARGB** instead of **TYPE_INT_RGB** , which is its normal type.

Generally speaking, this program:

- Instantiates a new visual object that extends the **JFrame** class and contains a **JSlider** object.
- Instantiates **Picture** objects from two image files *(beach and butterfly)* along with some blank **Picture** objects of the same size.

- Defines a method named **setOpacity** that can be called to set the opacity of every pixel in a picture to a specified value.
- Defines a method named **drawPictureOnPicture** that can be called to draw one picture onto another picture.
- Registers a **ChangeEvent** handler on the slider to:
    - Extract a percent-opacity value from the slider based on the position of the thumb.
    - Apply that opacity value to the butterfly image.
    - Draw the modified butterfly image on the beach image and display it.

Brief program specifications

Write a program named **Prob07** that uses the class definition for the class named **Prob07** in Listing 10 (p. 13) along with Ericson's media library and the image files named Prob07a.jpg [2] and Prob07b.jpg [3] to produce the two output images shown in Figure 1 (p. 4) .

[2]http://cnx.org/content/m44912/latest/Prob07a.jpg
[3]http://cnx.org/content/m44912/latest/Prob07b.jpg

Screen output at startup.

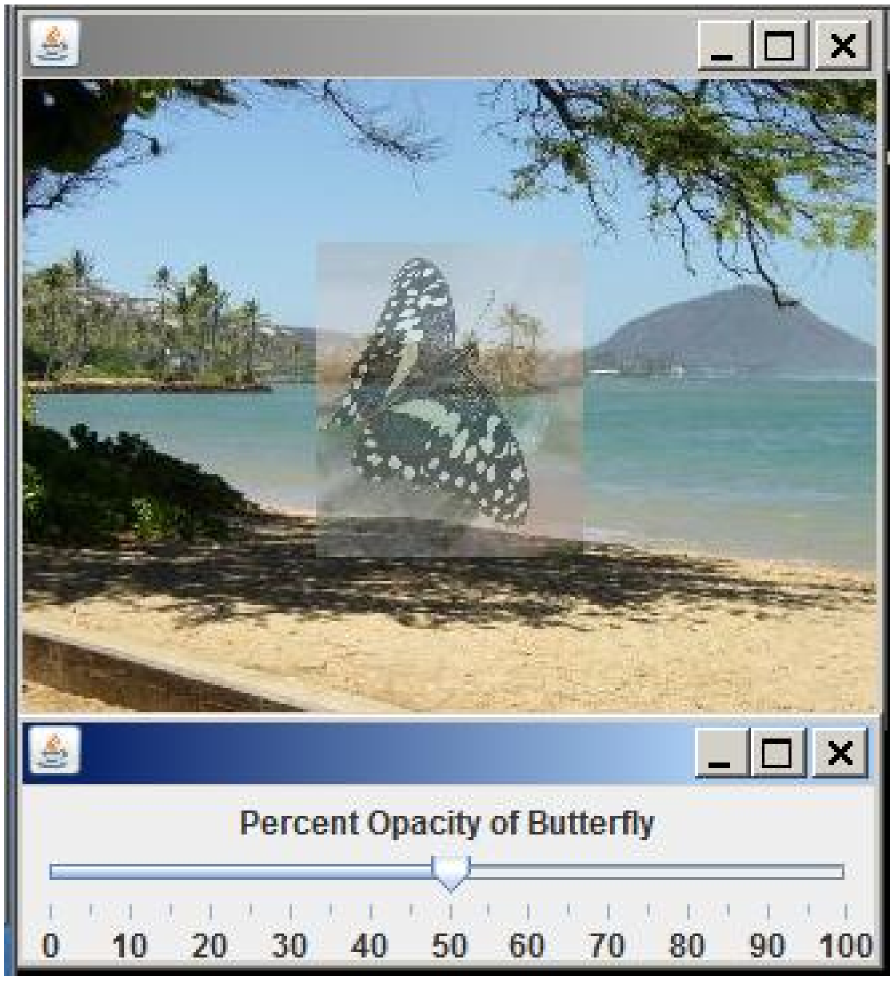

Figure 1: Screen output at startup.

Two output images

Note that Figure 1 (p. 4) actually consists of two output images, one positioned below the other.

Move the thumb to the left

When you move the thumb on the slider to the left, the butterfly becomes less opaque *(more transparent)* as shown in Figure 2 (p. 5) with total transparency at the extreme left end of the slider.

Twenty-percent opacity.

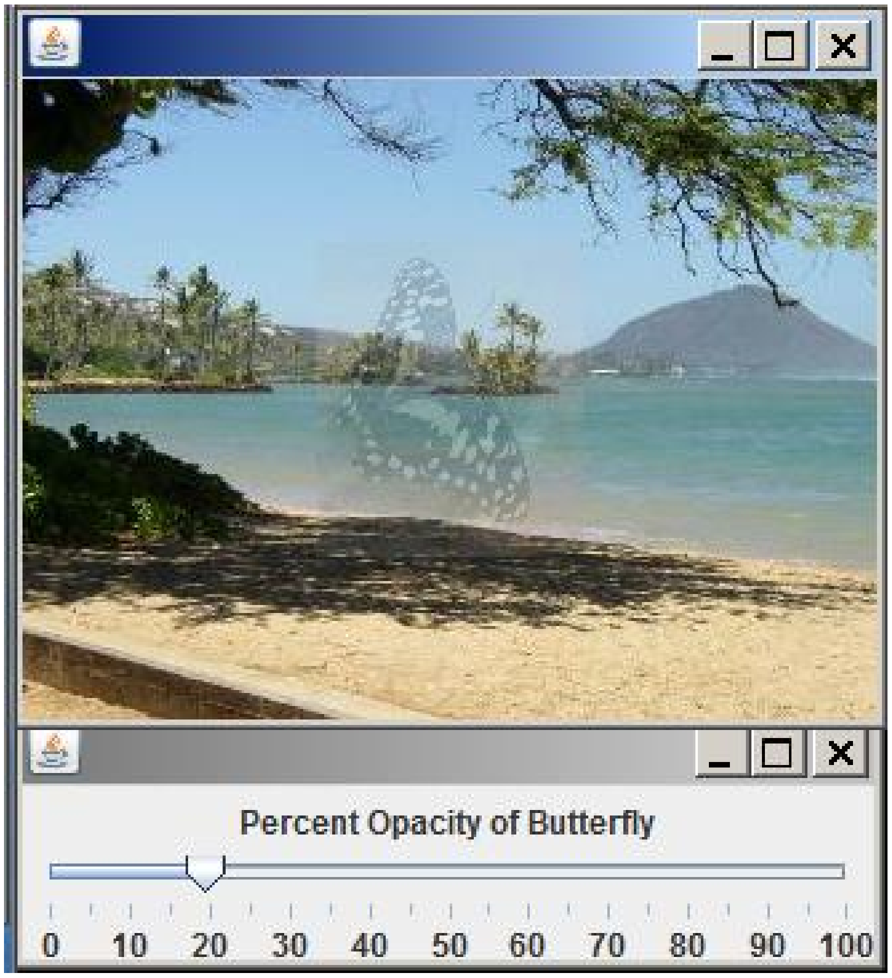

Figure 2: Twenty-percent opacity.

Move the thumb to the right

When you move the thumb on the slider to the right, the butterfly becomes more opaque *(less transparent)* as shown in Figure 3 (p. 6) with total opacity at the extreme right end of the slider.

Eighty-percent opacity.

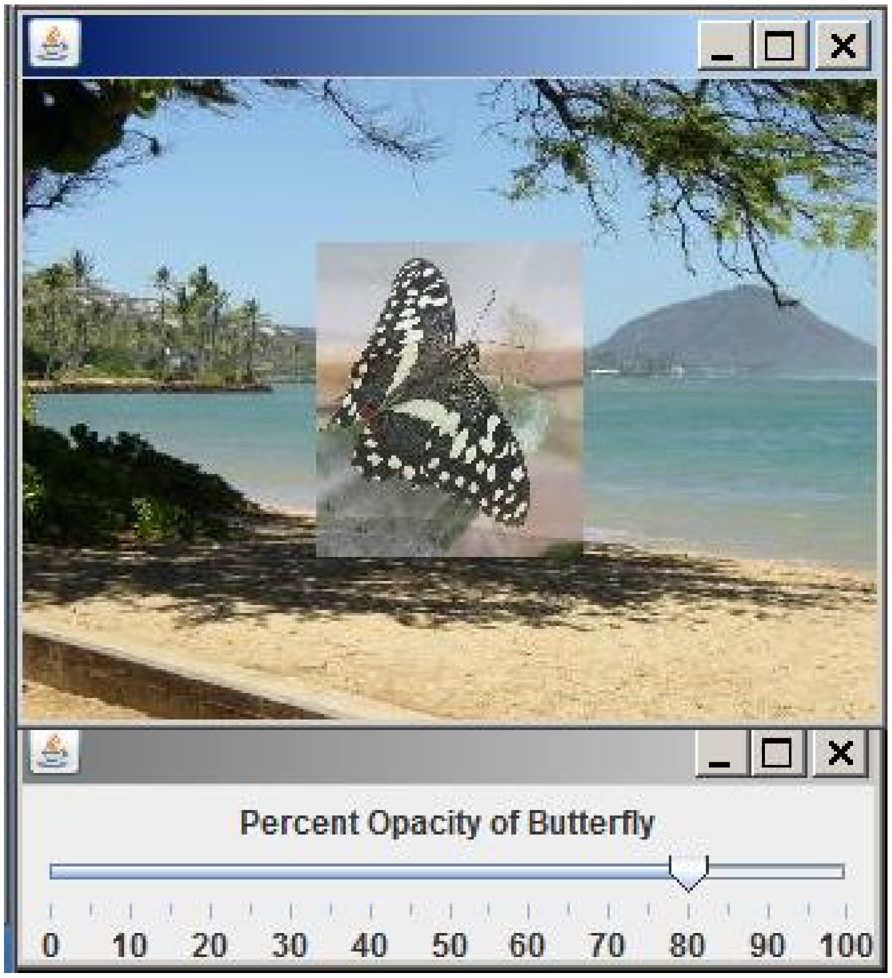

Figure 3: Eighty-percent opacity.

Define 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 **Prob07** given in Listing 10 (p. 13) .

4 General background information

The image in a **SimplePicture** object is stored in an object of the **BufferedImage** class, which is a class in the standard Sun Java library.

Image data formats

An examination of the documentation for the **BufferedImage** class shows that the red, green, blue, and alpha values for each pixel can be formatted in about fourteen different ways in an object of the **BufferedImage** class.

No alpha data

Some of those formats, including the way that information is stored in a **SimplePicture** object, don't include an alpha value.

Modification of the SimplePicture class

One way to modify the **SimplePicture** class to force it to accommodate alpha transparency data is to modify one of the constructors for the **SimplePicture** class as shown in Listing 1 (p. 7) . Note the change indicated by comments in Listing 1 . *(There are probably other ways that you can modify the class to achieve the same result as well.)*

Listing 1: Modification of the SimplePicture class.

```
    /**
  * A constructor that takes the width and height desired
  * for a picture and creates a buffered image of that
  * size.  This constructor doesn't show the picture.
  */
 public  SimplePicture(int width, int height){
//Disable the following statement
//   bufferedImage = new BufferedImage(
//            width, height, BufferedImage.
```

Future Picture objects will accommodate alpha transparency

Having made this modification, future objects instantiated from the **SimplePicture** class using this constructor will accommodate alpha transparency. *(The* ***SimplePicture*** *class is the superclass of the* ***Picture*** *class.)*

No complete listing of SimplePicture provided

Because of the simplicity of this modification, a complete listing of the modified **SimplePicture** class will not be provided in this module.

5 Discussion and sample code

5.1 The class named Prob07

You can view the driver class named **Prob07** at the beginning of the source code in Listing 10 (p. 13) . You are already familiar with the code in the **main** method of that class from earlier modules so I won't spend any time explaining it.

Briefly, the **main** method instantiates a new object of the class named **Prob07Runner** and calls the **run** method on that object. When the **run** method returns, the GUI shown in Figure 1 (p. 4) has been displayed on the screen.

At that point, the program simply goes into an idle state and waits for the user to take some action that causes an event to be fired. When an event is fired, it is handled and the program goes idle again waiting for another event.

(Because there are images on the screen, the program does not actually terminate until the user forces it to terminate.)

5.2 The class named Prob07Runner

Will explain in fragments

I will explain this program in fragments. A complete listing of the program is provided in Listing 10 (p. 13) near the end of the module.

Beginning of the class named Prob07Runner

The class named **Prob07Runner** begins in Listing 2 (p. 8) .

Listing 2: Beginning of the class named Prob07Runner.

```
  class Prob07Runner extends JFrame{

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

private Picture background = new Picture("Prob07b.jpg");
private Picture butterfly = new Picture("Prob07a.jpg");


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

private Picture display =
        new Picture(backgroundWidth,backgroundHeight);
private Picture tempPicture =
          new Picture(butterflyWidth,butterflyHeight);

private Image image = null;
private Graphics graphics = null;
```

Class extends JFrame

Note that this class extends **JFrame** . An object of this class forms the lower part of the image shown in Figure 1 (p. 4) that contains the slider.

The code in Listing 1 (p. 7) is straightforward and shouldn't require an explanation.

When Listing 2 finishes executing...

When the code in Listing 2 (p. 8) has finished executing, four new **Picture** objects have been instantiated and referred to by the following reference variables:

- **background** - The beach scene shown in the background in Figure 1 (p. 4) .
- **butterfly** - Contains an opaque image of the butterfly shown in Figure 1 (p. 4) .
- **display** - Empty picture the same size as the beach scene.
- **tempPicture** - Empty picture the same size as the butterfly.

In addition, a pair of working variables named **image** and **graphics** of the types **Image** and **Graphics** have been declared.

Finally, when the code in Listing 2 (p. 8) has finished executing, two new **JPanel** objects and one new **JSlider** object have been instantiated and referred to by the variables named **mainPanel** , **titlePanel** , and **slider** .

Beginning of the constructor

The beginning of the constructor is shown in Listing 3 (p. 8) .

Listing 3: Beginning of the constructor.

```
  public Prob07Runner(){//constructor
//Do some initial setup.
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
    slider.setMajorTickSpacing(10);
    slider.setMinorTickSpacing(5);
    slider.setPaintTicks(true);
    slider.setPaintLabels(true);

    mainPanel.setLayout(new BorderLayout());
    titlePanel.add(new JLabel(
                     "Percent Opacity of Butterfly"));
    mainPanel.add(titlePanel,BorderLayout.NORTH);
    mainPanel.add(slider,BorderLayout.CENTER);

    getContentPane().add(mainPanel);

    setSize(backgroundWidth + 7,97);
    setLocation(0,backgroundHeight + 25);
    setVisible(true);
```

Although it may be necessary for you to go to Sun's Java documentation to learn about the detailed behavior of some of the methods that are called in Listing 3 (p. 8) , the code in Listing 3 (p. 8) is straightforward and should not require further explanation.

Display the initial background image

Listing 4 (p. 9) displays the initial background image.

Instantiating and destroying a lot of new **Picture** objects as the user moves the slider to change the opacity would be very inefficient. To avoid this inefficiency, this program gets images from existing **Picture** objects and draws them on existing **Picture** objects without modifying the originals.

Listing 4: Display the initial image.

```
        graphics = display.getGraphics();
    graphics.drawImage(background.getImage(),0,0,null);
```

Display the butterfly at 50-percent opacity

Listing 5 (p. 9) calls the **setOpacity** and **drawPictureOnPicture** methods to set the opacity of the butterfly and draw it onto the display with 50-percent opacity. The image of the butterfly is centered on the background.

Listing 5: Display the butterfly at 50-percent opacity.

```
    butterfly = setOpacity(butterfly,50);
    drawPictureOnPicture(
                 butterfly,
                 display,
                 backgroundWidth/2 - butterflyWidth/2,
                 backgroundHeight/2 - butterflyHeight/2);
    display.show();
```

Put the constructor on hold

At this point, I will put the discussion of the constructor on hold and explain the **setOpacity** and **drawPictureOnPicture** methods.

The setOpacity method

The **setOpacity** method is shown in its entirety in Listing 6 (p. 10) .

Listing 6: The setOpacity method.

```
private Picture setOpacity(
                Picture pic,double percentOpacity){

  int opacity = (int)(255*percentOpacity/100);
  int opacityMask = opacity << 24;

  for(int col = 0;col < butterflyWidth;col++){
    for(int row = 0;row < butterflyHeight;row++){
      //Get the pixel in basic int format.
      int basicPixel = pic.getBasicPixel(col,row);

      //Set the alpha value for the pixel.
      basicPixel = basicPixel & 0x00FFFFFF;
      basicPixel = basicPixel | opacityMask;

      //Set the modified pixel into tempPicture.
      tempPicture.setBasicPixel(col,row,basicPixel);
    }//end inner loop
  }//end outer loop

  return tempPicture;

}//end setOpacity
```

This method copies an incoming picture into an existing temporary picture, setting the alpha value for every pixel to a specified value in the process. Then it returns the modified picture object's reference where it is saved in the reference variable named **butterfly** in Listing 5 (p. 9) ,

A bitwise left-shift operation

The only thing in Listing 6 (p. 10) that is new to this module is the use of a bitwise left-shift operation.

A 24-bit left shift

Listing 6 (p. 10) converts the incoming **percentOpacity** value to an integer value ranging from 0 to 255. This value resides in the least significant eight bits of an **int** variable named **opacity** .

Then Listing 6 (p. 10) applies the bitwise left-shift operator *(two left angle brackets)* to shift those eight bits into the eight most significant bits and stores the result in another **int** variable named **opacityMask** .

Apply the opacityMask to the pixels

A pair of nested **for** loops is used to set the alpha value of every pixel to the value of **opacityMask** using an overall bit-masking methodology that I explained in an earlier module.

The drawPictureOnPicture method

After the alpha value for every pixel in the butterfly image has been set to the specified opacity, Listing 5 (p. 9) calls the method named **drawPictureOnPicture** to draw the modified butterfly image on the beach scene as shown in Figure 1 (p. 4) .

The **drawPictureOnPicture** method is shown in its entirety in Listing 7 (p. 11) .

Listing 7: The drawPictureOnPicture method.

```
    private void drawPictureOnPicture(
                   Picture source,Picture dest,int xOff,
                                         int yOff){

    Graphics destGraphics = dest.getGraphics();
    Image sourceImage = source.getImage();
    destGraphics.drawImage(sourceImage,
                           xOff,
                           yOff,
                           null);
}//end drawPictureOnPicture method
```

This method draws the source picture onto the destination picture with an offset on both axes. There is nothing in Listing 7 (p. 11) that I haven't explained in an earlier module.

Return to the explanation of the constructor

You are already familiar with the use of anonymous inner classes to create and register listener objects on Java source objects. The slider is a source object.

Listing 8 (p. 11) begins the registration of an anonymous **ChangeEvent** listener on the slider.

Listing 8: Begin the registration of an event handler on the slider.

```
      slider.addChangeListener(
    new ChangeListener(){
      public void stateChanged(ChangeEvent e){
        //Draw a new copy of the background on the
        // display.
        graphics = display.getGraphics();
        graphics.drawImage(
                        background.getImage(),0,0,null);
```

Restore the background image

Each time the slider fires a **ChangeEvent** , this event handler draws a new background image on the display. This erases what was previously drawn there, restoring a pristine image of the beach scene.

Draw a partially opaque butterfly image on the background

Then it uses the current value of the slider to set the opacity of the butterfly image and draws it centered on the display on top of the background image.

A series of events

The slider fires a series of **ChangeEvents** as the user moves the thumb on the slider. Listing 8 (p. 11) begins the definition of the event handler method named **stateChanged** , which is registered on the slider. This method is called each time the slider fires a **ChangeEvent** .

Listing 8 (p. 11) draws a new copy of the beach background image on the **Picture** object referred to by the reference variable named **background** . This image replaces the image that was previously drawn there.

Draw the butterfly and repaint

Listing 9 (p. 12) calls the **setOpacity** and **drawPictureOnPicture** methods to:

- Set the opacity of the butterfly to the value currently represented by the position of the thumb on the slider. This is the value returned by the slider's **getValue** method.
- Draw the butterfly image on the background image.

Listing 9: Draw the butterfly and repaint.

```
          //Set the opacity of butterfly and copy it onto
      // the display. Then repaint the display.
      butterfly =
              setOpacity(butterfly,slider.getValue());
      drawPictureOnPicture(
              butterfly,
              display,
              backgroundWidth/2 - butterflyWidth/2,
              backgroundHeight/2 - butterflyHeight/2);

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

Repaint the image

Then Listing 9 (p. 12) calls the **repaint** method to cause the modified image to be rendered onto the computer screen.

The end of the program

Listing 9 (p. 12) also signals the end of the constructor, the end of the class named **Prob07Runner** , and the end of the program.

6 Run the program

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

7 Summary

In this module, you learned how to use a slider to continuously change the opacity of an image and draw that image onto a background image.

8 What's next?

In the next module, you will learn how to use a slider to continuously change the threshold detection level of an edge detector and to draw the edge-detected image on the screen.

9 Miscellaneous

This section contains a variety of miscellaneous information.

NOTE: **Housekeeping material**

- Module name: Java OOP: Controlling Opacity with a Slider
- File: Java3114.htm
- Published: 05/13/12
- Revised: 09/06/12

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.

10 Complete program listing

A complete listing of the program discussed in this module is shown in Listing 10 (p. 13) below.

Listing 10: Complete program listing.

```
    /*File Prob07 Copyright 2008 R.G.Baldwin
*******************************************************/
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 Prob07{
  //DO NOT MODIFY THE CODE IN THIS CLASS DEFINITION.
  public static void main(String[] args){
    new Prob07Runner();
  }//end main method
}//end class Prob07
//=====================================================//

class Prob07Runner extends JFrame{
  private JPanel mainPanel = new JPanel();
  private JPanel titlePanel = new JPanel();
  private JSlider slider = new JSlider();

  private Picture background = new Picture("Prob07b.jpg");
  private Picture butterfly = new Picture("Prob07a.jpg");


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

  private Picture display =
            new Picture(backgroundWidth,backgroundHeight);
  private Picture tempPicture =
              new Picture(butterflyWidth,butterflyHeight);

  private Image image = null;
  private Graphics graphics = null;

  public Prob07Runner(){//constructor
    //Do some initial setup.
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    slider.setMajorTickSpacing(10);
    slider.setMinorTickSpacing(5);
    slider.setPaintTicks(true);
    slider.setPaintLabels(true);

    mainPanel.setLayout(new BorderLayout());
    titlePanel.add(new JLabel(
                         "Percent Opacity of Butterfly"));
    mainPanel.add(titlePanel,BorderLayout.NORTH);
    mainPanel.add(slider,BorderLayout.CENTER);

    getContentPane().add(mainPanel);

    setSize(backgroundWidth + 7,97);
    setLocation(0,backgroundHeight + 25);
    setVisible(true);

    //Draw and display the initial image with 50-percent
    // opacity. In order to avoid instantiating and
    // destroying a lot of Picture objects, the
    // procedure is to simply get images from existing
    // picture objects and draw them on other existing
    // picture objects.
    graphics = display.getGraphics();
    graphics.drawImage(background.getImage(),0,0,null);
    //Set the opacity of butterfly and draw it onto the
    // display. In this case, the opacity is set to
    // 50-percent. The image of the butterfly is centered
    // on the background.
    butterfly = setOpacity(butterfly,50);
    drawPictureOnPicture(
                  butterfly,
                  display,
                  backgroundWidth/2 - butterflyWidth/2,
                  backgroundHeight/2 - butterflyHeight/2);
    display.show();
    //--------------------------------------------------//
    //Register an anonymous listener object on the slider.
    //Each time the slider fires a ChangeEvent, this event
    // handler draws a new background image on the
    // display. This erases what was previously drawn
    // there. Then it uses the current value of the slider
    // to set the opacity of the butterfly image and
    // draws it on the display on top of the background
    // image. It is centered on the background image.
    slider.addChangeListener(
      new ChangeListener(){
        public void stateChanged(ChangeEvent e){
          //Draw a new copy of the background on the
          // display.
          graphics = display.getGraphics();
          graphics.drawImage(
                        background.getImage(),0,0,null);

          //Set the opacity of butterfly and copy it onto
          // the display. Then repaint the display.
          butterfly =
                  setOpacity(butterfly,slider.getValue());
          drawPictureOnPicture(
                  butterfly,
                  display,
                  backgroundWidth/2 - butterflyWidth/2,
                  backgroundHeight/2 - butterflyHeight/2);
          display.repaint();
        }//end stateChanged
      }//end new ChangeListener
    );//end addChangeListener
    //--------------------------------------------------//
  }//end constructor
  //----------------------------------------------------//


  //This method copies an incoming picture into an
  // existing temporary picture, setting the alpha value
  // for every pixel to a specified value. Then it returns
  // the modified temporary picture object.
  private Picture setOpacity(
                      Picture pic,double percentOpacity){

    int opacity = (int)(255*percentOpacity/100);
    int opacityMask = opacity << 24;

    for(int col = 0;col < butterflyWidth;col++){
      for(int row = 0;row < butterflyHeight;row++){
        //Get the pixel in basic int format.
        int basicPixel = pic.getBasicPixel(col,row);

        //Set the alpha value for the pixel.
        basicPixel = basicPixel & 0x00FFFFFF;
        basicPixel = basicPixel | opacityMask;

        //Set the modified pixel into tempPicture.
        tempPicture.setBasicPixel(col,row,basicPixel);
      }//end inner loop
    }//end outer loop

    return tempPicture;

  }//end setOpacity
  //----------------------------------------------------//

  //Draws the source picture onto the destination
  // picture with an offset on both axes.
  private void drawPictureOnPicture(
                     Picture source,Picture dest,int xOff,
                                               int yOff){

      Graphics destGraphics = dest.getGraphics();
      Image sourceImage = source.getImage();
      destGraphics.drawImage(sourceImage,
                             xOff,
                             yOff,
                             null);
  }//end drawPictureOnPicture method
}//end class Prob07Runner
```

-end-