Java OOP: Using Alpha Transparency with Ericson's Media Library[*]

R.G. (Dick) Baldwin

Abstract

Learn how to use alpha transparency with Ericson's media library.

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 1.1: Sep 5, 2012 11:28 am -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 incorporate alpha transparency into the use of Ericson's media library.

Two approaches

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

This program 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 and OR operators, and
- The use of the **drawImage** method of the **Graphics** class.

The getBasicPixel and setBasicPixel methods

The program uses the **getBasicPixel** and **setBasicPixel** methods from Ericson's library along with bitwise operations to set the alpha value for all the pixels in a cropped and flipped image of a butterfly to a hexadecimal value of 5F.

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.

Crop, flip, and set alpha values

Then the student must write a method that will crop and flip an image of a butterfly and set the value of every alpha byte to a hexadecimal value of 5F.

Draw a partially transparent image of a butterfly

Finally, the student must use the standard **drawImage** method of the **Graphics** class to draw the image of the butterfly onto an image of a beach with transparency.

Brief program specifications

Write a program named **Prob06** that uses the class definition for the class named **Prob06** in Listing 7 (p. 12) along with Ericson's media library and the image files named **Prob06a.jpg** [2] and **Prob06b.jpg** [3] to produce the three graphic output images shown in Figure 1 (p. 3) , Figure 2 (p. 4) , and Figure 3 (p. 5) .

Image from file named Prob06a.

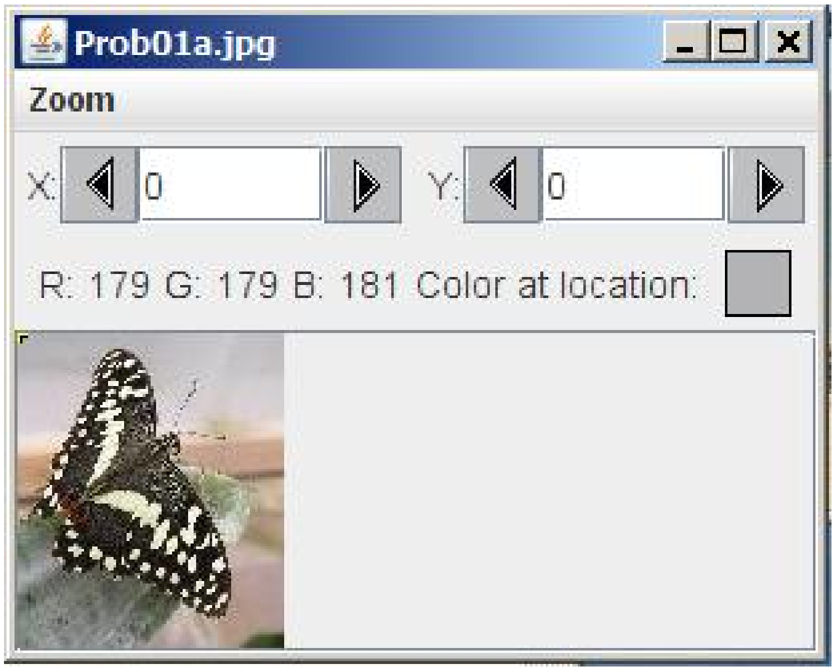

Figure 1: Image from file named Prob06a.

---

[2]http://cnx.org/content/m44911/latest/Prob06a.jpg
[3]http://cnx.org/content/m44911/latest/Prob06b.jpg

Image from file named Prob06b.

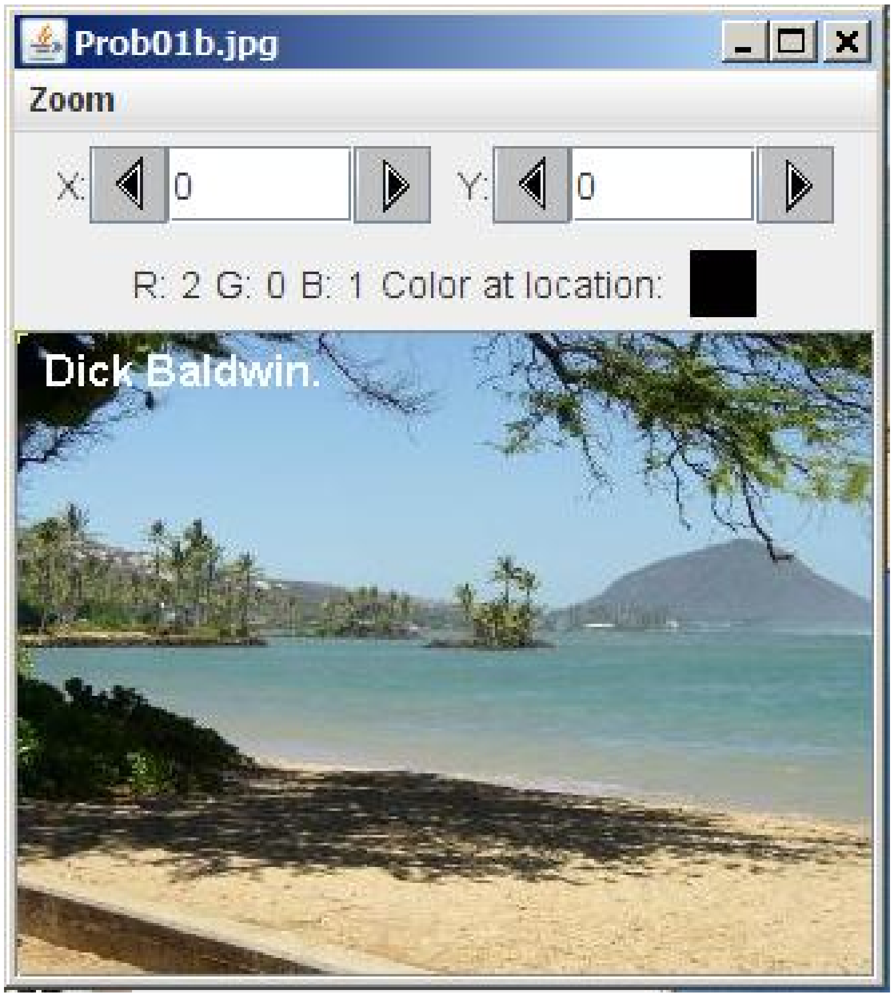

Figure 2: Image from file named Prob06b.

Processed output image.

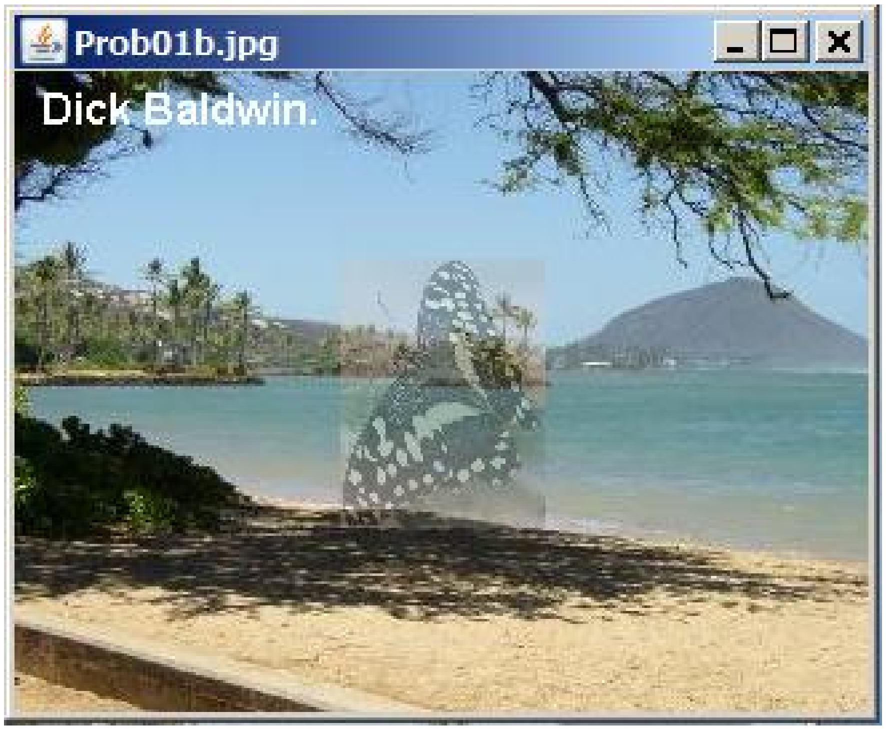

Figure 3: Processed output image.

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 **Prob06** given in Listing 7 (p. 12) .

A partially transparent image of a butterfly

Just in case you haven't noticed it, the final image of the beach contains a partially transparent image of a butterfly superimposed and centered on the beach image.

Modification to the SimplePicture class

In order to write this program, you will need to modify the class from Ericson's media library named **SimplePicture** .

Your modifications must make it possible for you to display a partially transparent image on top of another image with the background image showing through.

Transparency

The degree of transparency can range from being completely transparent at one extreme to being totally opaque at the other extreme. In this case, the butterfly image shown in Figure 3 (p. 5) is about 37-percent opaque *(or 63-percent transparent)* .

Outside research

You will probably need to do some outside research in order to write this program. For example, you will need to learn about the following topics and probably some other topics as well:

- Alpha transparency

- BufferedImage objects of TYPE_INT_ARGB
- The representation of a pixel as type int
- Bit manipulation of pixels
- The drawImage method of the Graphics class

Required text output

In addition to the output images described above, your program must produce the text output shown in Figure 4 (p. 6) on the command- line screen.

Required text output.

```
    Dick Baldwin.
Dick Baldwin
Picture, filename Prob06a.jpg height 118 width 100
Picture, filename Prob06b.jpg height 240 width 320
Picture, filename None height 101 width 77
```

Figure 4: Required text output.

You must substitute your name for my name wherever my name appears both in the images and on the command-line screen.

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. 6) . Note that **BufferedImage.TYPE_INT_RGB** was replaced by **BufferedImage.TYPE_INT_ARGB** 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.TYPE_INT_RGB);

   //Modify constructor to support alpha transparency.
   System.out.println("Dick Baldwin");
   bufferedImage = new BufferedImage(
             width, height, BufferedImage.TYPE_INT_ARGB);

   title = "None";
   fileName = "None";
   extension = "jpg";
   setAllPixelsToAColor(Color.white);
 }//end constructor
```

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.)*

Display the student's name

Note that the constructor in Listing 1 (p. 6) is also modified to cause it to display the student's name, which is a requirement of the program.

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 Prob06

You can view the driver class named **Prob06** at the beginning of the source code in Listing 7 (p. 12) . 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 **Prob06Runner** and calls the **run** method on that object. When the **run** method returns, the code in the **main** method displays some information about the three images and terminates.

(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 Prob06Runner

Will explain in fragments

I will explain this program in fragments. A complete listing of the program is provided in Listing 7 (p. 12) near the end of the module

The class named **Prob06Runner** begins in Listing 2 (p. 8) , which shows the constructor for the class.

Listing 2: Beginning of the class named Prob06Runner.

```
  class Prob06Runner{

public Prob06Runner(){//constructor
  System.out.println("Dick Baldwin.");
}//end constructor
```

The constructor simply displays the student's name to satisfy one of the requirements of the program.

The run method

The run method, which is called from the **main** method in Listing 7 (p. 12) , is shown in its entirety in Listing 3 (p. 8) .

Listing 3: The run method.

```
    public Picture[] run(){
  //Insert executable code here
  Picture picA = new Picture("Prob06a.jpg");
  picA.explore();
  Picture picB = new Picture("Prob06b.jpg");
  picB.addMessage("Dick Baldwin.",10,20);
  picB.explore();

  Picture picC = cropAndFlip(picA,4,5,80,105);

  copyPictureWithCrop(picC,picB,122,70);

  picB.show();

  Picture[] output = {picA,picB,picC};
  return output;
}//end run
```

New material

The only thing in Listing 3 (p. 8) that is new to this module is the pair of calls to the following methods. I will explain these methods in the paragraphs that follow:

- **cropAndFlip**
- **copyPictureWithCrop**

Beginning of the cropAndFlip method

The **cropAndFlip** method begins in Listing 4 (p. 8) . This method receives an incoming reference to a **Picture** object. It crops the picture to a set of specified coordinate values and flips it around a vertical line at its center.

Listing 4: Beginning of the cropAndFlip method.

```
    private Picture cropAndFlip(
              Picture pic,int x1,int y1,int x2,int y2){
  Picture output = new Picture(x2-x1+1,y2-y1+1);
```

```
int width = output.getWidth();
Pixel pixel = null;
Color color = null;
for(int col = x1;col < (x2+1);col++){
  for(int row = y1;row < (y2+1);row++){
    color = pic.getPixel(col,row).getColor();
    pixel = output.getPixel(width-col+x1-1,row-y1);
    pixel.setColor(color);
  }//end inner loop
}//end outer loop
```

Receives a reference to the butterfly image

Note from Listing 3 (p. 8) that the **cropAndFlip** method receives a reference to the **Picture** object of the butterfly that is displayed in Figure 1 (p. 3) .

Also note that the butterfly in Figure 1 (p. 3) is facing toward the right while the butterfly in the output image in Figure 3 (p. 5) has been cropped to a smaller size and is facing toward the left.

Crop and flip is not new

The capability to crop and flip an image is not new to this module. However, the **cropAndFlip** method also makes the image partially transparent as shown in Figure 3 (p. 5) . That capability is new to this module. I will explain how that is done shortly.

A call to the modified SimplePicture constructor

Although there is nothing new in the code in Listing 4 (p. 8) , it is important to note that the first statement in Listing 4 (p. 8) causes the **SimplePicture** constructor that was modified in Listing 1 (p. 6) to be called.

As a result, the **Picture** object referred to by the reference variable named **output** in Listing 4 (p. 8) will accommodate alpha transparency data.

Make the pixels partially transparent

The code in Listing 5 (p. 9) uses a pair of nested **for** loops to iterate through all of the pixels in the picture referred to by **output** and modify each pixel.

The four statements in the body of the inner loop in Listing 5 (p. 9) cause the current pixel to become partially transparent.

Listing 5: Make the pixels partially transparent.

```
    width = output.getWidth();
int height = output.getHeight();
pixel = null;
color = null;
for(int col = 0;col < width;col++){
  for(int row = 0;row < height;row++){

    int basicPixel = output.getBasicPixel(col,row);

    basicPixel = basicPixel & 0x00FFFFFF;
    basicPixel = basicPixel | 0x5F000000;

    output.setBasicPixel(col,row,basicPixel);

  }//end inner loop
}//end outer loop
```

```
    return output;
  }//end crop and flip
```

The getBasicPixel method

According to Ericson's documentation, the **getBasicPixel** method will *"return the pixel value as an int for the given x and y location."* In other words, a call to the **getBasicPixel** method will return an **int** value containing the red, green, blue, and alpha values for the pixel at the specified location.

A bitwise AND operation

Listing 5 (p. 9) uses a bitwise ***AND*** operation *(note the single ampersand)* to force the eight most significant bits *(the alpha byte)* in the **int** representation of the current pixel to zero while preserving the bit values stored in the least significant 24 bits.

A bitwise OR operation

Then Listing 5 (p. 9) uses a bitwise ***OR*** operation (|) to store the hexadecimal value 5F in the eight most significant bits *(the alpha byte)* without changing the values stored in the 24 least significant bits.

The alpha byte

The value of the alpha byte can range from 0 to 255. When rendered using a mechanism that supports alpha transparency, an alpha value of zero causes the pixel to be totally transparent.

Similarly, an alpha value of 255 causes the pixel to be totally opaque.

Values between zero and 255 cause the pixel to be rendered as partially opaque or partially transparent, whichever terminology you prefer.

Thirty-seven percent opaque

If I did the arithmetic correctly, a hexadecimal value of 5F represents a decimal value of 95. Therefore, this value will cause the pixel to be about 37-percent opaque *(or 63-percent transparent)* .

The setBasicPixel method

As the name implies, the **setBasicPixel** method can be used to *"set the value of a pixel in the picture from an int."*

Therefore, the last statement in the body of the inner loop in Listing 5 (p. 9) replaces the value of the current pixel with the modified value containing a value of 95 in the alpha byte.

The end of the cropAndFlip method

When the pair of nested **for** loops in Listing 5 (p. 9) terminates, the **cropAndFlip** method returns control to the **run** method in Listing 3 (p. 8) , returning a copy of the reference from the variable named **output** *(see Listing 4 (p. 8))* in the process.

Save the Picture object's reference

The returned reference is stored in the reference variable named **picC** in Listing 3 (p. 8) .

At this point, **picC** contains a reference to a butterfly image that has been cropped, flipped, and formatted into a buffered image that contains alpha transparency information.

Call the copyPictureWithCrop method

Listing 3 (p. 8) immediately calls the **copyPictureWithCrop** method passing copies of the references stored in **picC** and **picB** along with a pair of integer coordinate values.

The copyPictureWithCrop method

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

Listing 6: The copyPictureWithCrop method.

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

  Graphics destGraphics = dest.getGraphics();
  Image sourceImage = source.getImage();
  destGraphics.drawImage(sourceImage,
                              xOff,
                              yOff,
                              null);
  }//end copyPictureWithCrop method
}//end class Prob06Runner
```

The purpose of the **copyPictureWithCrop** method is to copy a source picture onto a destination picture with an offset on each axis.

An exercise for the student

I won't attempt to explain the code in Listing 6 (p. 10) in this module. Instead, I will simply suggest that you go to Google and search for the following or similar keywords:

baldwin java drawImage

You will find many tutorials that I have written that deal with topics in this area.

Modify the destination pixel colors

I will tell you that the use of the **drawImage** method in Listing 6 (p. 10) modifies the destination picture in such a way that the color of each pixel in the resulting image is a combination of the colors in the original destination image and the corresponding pixel in the source image.

An illusion of transparency

If a source pixel is totally transparent, it has no effect on the color of the destination pixel.

If the source pixel is totally opaque, the color of the destination pixel is changed to the color of the source pixel.

For alpha values between these two extremes, the final color of the destination pixel produces the illusion of a partially transparent image in front of the original destination image.

Termination of the copyPictureWithCrop method

When the **copyPictureWithCrop** method terminates in Listing 6 (p. 10) , control returns to the run method in Listing 3 (p. 8) .

Listing 3 (p. 8) calls the **show** method to display the image in the now-modified **Picture** object referred to by **picB** , as shown in Figure 3 (p. 5) .

Return a reference to an array object

Then the **run** method encapsulates references to each of the three images in an array object and returns control to the **main** method in Listing 7 (p. 12) , returning a copy of the array object's reference in the process.

The **main** method in Listing 7 (p. 12) displays information about each of the three **Picture** objects, producing the output shown in Figure 4 (p. 6) . Then the main method terminates.

Images don't go away immediately

Because there are images belonging to the program still on the screen, the program doesn't return control to the operating system. It will simply wait until it is forced to terminate by the user before returning control to the operating system.

Clicking the X-buttons in the upper-right corners of the images will simply hide the frames and won't terminate the program. Some extra work is required to deal with this issue.

6 Run the program

I encourage you to copy the code from Listing 7 (p. 12) . 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 about:

- Alpha transparency

- A buffered image of type TYPE_INT_ARGB
- The ability to use the **getBasicPixel** and **setBasicPixel** methods,
- The use of the bitwise AND and OR operators,
- The use of the **drawImage** method of the **Graphics** class.

You modified 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.

You wrote a method that cropped and flipped an image of a butterfly.

You used the **getBasicPixel** and **setBasicPixel** methods from Ericson's library along with bitwise operations to set the alpha value for all the pixels in the cropped and flipped image of the butterfly to a hexadecimal value of 5F.

Finally, you used the standard **drawImage** method of the **Graphics** class to draw the image of the butterfly onto an image of a beach with transparency.

8 What's next?

In the next module, you will learn how to use a slider to continuously change the opacity of an image and to draw that modified image onto a background image.

9 Miscellaneous

This section contains a variety of miscellaneous information.

NOTE: **Housekeeping material**

- Module name: Java OOP: Using Alpha Transparency with Ericson's Media Library
- File: Java3112.htm
- Published: 05/13/12
- Revised: 09/05/12

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 7 (p. 12) below.

Listing 7: Complete program listing.

```
    /*File Prob06 Copyright 2008 R.G.Baldwin
Revised 12/31/08
*********************************************************/
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Image;

public class Prob06{
  //DO NOT MODIFY THE CODE IN THIS CLASS DEFINITION.
  public static void main(String[] args){
    Picture[] pictures = new Prob06Runner().run();
    System.out.println(pictures[0]);
    System.out.println(pictures[1]);
    System.out.println(pictures[2]);
  }//end main method
}//end class Prob06
//======================================================//

class Prob06Runner{

  public Prob06Runner(){//constructor
    System.out.println("Dick Baldwin.");
  }//end constructor
  //----------------------------------------------------//
  public Picture[] run(){
    //Insert executable code here
    Picture picA = new Picture("Prob06a.jpg");
    picA.explore();
    Picture picB = new Picture("Prob06b.jpg");
    picB.addMessage("Dick Baldwin.",10,20);
    picB.explore();

    Picture picC = cropAndFlip(picA,4,5,80,105);
    copyPictureWithCrop(picC,picB,122,70);

    picB.show();

    Picture[] output = {picA,picB,picC};
    return output;
  }//end run
  //----------------------------------------------------//

  //Crops a picture to the specified coordinate values and
  // flips it around a vertical line at its center.
  //Also makes it partially transparent
  private Picture cropAndFlip(
               Picture pic,int x1,int y1,int x2,int y2){
    Picture output = new Picture(x2-x1+1,y2-y1+1);

    int width = output.getWidth();
    Pixel pixel = null;
    Color color = null;
    for(int col = x1;col < (x2+1);col++){
      for(int row = y1;row < (y2+1);row++){
        color = pic.getPixel(col,row).getColor();
        pixel = output.getPixel(width-col+x1-1,row-y1);
        pixel.setColor(color);
      }//end inner loop
    }//end outer loop

    width = output.getWidth();
    int height = output.getHeight();
    pixel = null;
    color = null;
    for(int col = 0;col < width;col++){
      for(int row = 0;row < height;row++){

        int basicPixel = output.getBasicPixel(col,row);
        basicPixel = basicPixel & 0x00FFFFFF;
        basicPixel = basicPixel | 0x5F000000;
        output.setBasicPixel(col,row,basicPixel);
      }//end inner loop
    }//end outer loop


    return output;
  }//end crop and flip
  //----------------------------------------------------//

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

      Graphics destGraphics = dest.getGraphics();
      Image sourceImage = source.getImage();
      destGraphics.drawImage(sourceImage,
                             xOff,
                             yOff,
                             null);
  }//end copyPictureWithCrop method
}//end class Prob06Runner
```

-end-