Java OOP: Image Processing Algorithms, Image Inversion, and PictureExplorer Objects*

R.G. (Dick) Baldwin

This work is produced by The Connexions Project and licensed under the
Creative Commons Attribution License 3.0†

Abstract

Learn how to invert images and how to display images in Ericson's PictureExplorer object.

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] .

---

*Version 1.3: Nov 14, 2012 9:31 am +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

2.1.2 Listings

3 Preview

The program that I will explain in this module is designed to be used as a test of the student's understanding of programming using Java and Ericson's media library.

The student is provided an image file named **Prob02.jpg** along with a pair of **PictureExplorer** windows containing the raw image and a modified version of the image. *(See Figure 1 (p. 3) and Figure 2 (p. 4) .)*

Deduce the algorithm

The first part of the test is to determine if the student can examine the raw image shown in the **PictureExplorer** window in *Figure 1 (p. 3)* and deduce the algorithm required to produce the output shown in the **PictureExplorer** window in *Figure 2 (p. 4)* .

Implement the algorithm
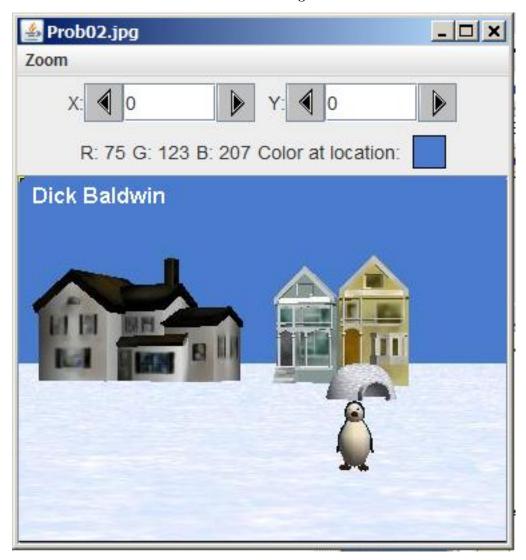
The second part of the test is to determine if the student can implement the algorithm once it is established and also satisfy some requirements for text output on the command line screen. Among other things, this requires that the student be able to:

- Create a **Picture** object from an image file.
- Write an accessor method to return a reference to the **Picture** object.
- Modify the pixels in the picture according to the algorithm.
- Display the raw picture and the modified picture in **PictureExplorer** objects by calling the **explore** method on the **Picture** object before and after it is modified.

Program output

The raw image is displayed in the **PictureExplorer** window shown in Figure 1 (p. 3) .

The raw image.

Figure 1: The raw image.

The modified image is shown in the **PictureExplorer** window in Figure 2 (p. 4) .
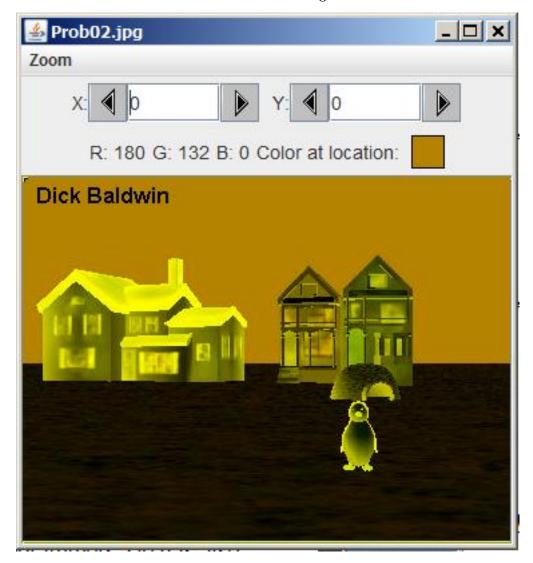
The modified image.

Figure 2: The modified image.

The required output on the command-line screen is shown by the last two lines of text in Figure 3 (p. 5) . The other text in Figure 3 (p. 5) is produced by the system during the compilation and execution process.

Output text on the command line screen.

```
    java version "1.6.0_14"
Java(TM) SE Runtime Environment (build 1.6.0_14-b08)
Java HotSpot(TM) Client VM (build 14.0-b16, mixed mode,
 sharing)
javac 1.6.0_14

Dick Baldwin
Picture, filename Prob02.jpg height 274 width 365
```

Figure 3: Output text on the command line screen.

The algorithm

The algorithm required to transform the image from Figure 1 (p. 3) to Figure 2 (p. 4) is:

- Set the blue color value for every pixel to zero.
- Invert the red and green color values for every pixel.

A color value is inverted by subtracting the value from 255.

Obvious that the blue color value is reduced to zero

It should be obvious to the student when comparing the two images in the **PictureExplorer** objects that the blue pixel value has been set to zero for every pixel in the modified image.

Color inversion is not quite so obvious

Deducing that the red and green colors in the output pixels are the inverse of the red and green colors in the input image isn't as straightforward. However, color inversion is one of the examples provided in the Ericson textbook, so that should serve as a clue to the student. I have also published several online tutorials that involve color inversion.

The implementation of the algorithm will be explained below.

4 Discussion and sample code

Will explain in fragments

I will explain this program in fragments. A complete listing is provided in Listing 6 (p. 11) near the end of the module.

I will begin with the driver class named **Prob02** , which is shown in its entirety in Listing 1 (p. 5) .

Listing 1: The driver class.

```
   public class Prob02{//the driver class
 public static void main(String[] args){
   Prob02Runner obj = new Prob02Runner();
```

```
    obj.run();

    System.out.println(obj.getPicture());
  }//end main
}//end class Prob02
```

You should already be familiar with everything in Listing 1 (p. 5) . The most important aspect of Listing 1 (p. 5) for purposes of this discussion is the call to the **run** method belonging to the object instantiated from the **Prob02Runner** class. I will explain the **run** method shortly.

Beginning of the class named Prob02Runner

The class definition for the class named **Prob02Runner** begins in Listing 2 (p. 6) .

Listing 2: Beginning of the class named Prob02Runner.

```
  class Prob02Runner{
private Picture pic = new Picture("Prob02.jpg");

public Prob02Runner(){//constructor
   System.out.println("Dick Baldwin");
}//end constructor
//-----------------------------------------------------//

//Accessor method
public Picture getPicture(){return pic;}
```

Again, you should be familiar with everything in Listing 2 (p. 6) . I will simply highlight the instantiation of a new **Picture** object using an image file as input and the saving of a reference to that object in the private instance variable named **pic** .

The beginning of the run method

The **run** method begins in Listing 3 (p. 6) . This is where the action is, so to speak.

Listing 3: The beginning of the run method.

```
public void run(){

pic.addMessage("Dick Baldwin",10,20);

pic.explore();
```

You are already familiar with the call to the **addMessage** method to add my name as text to the image encapsulated in the **Picture** object. (See Figure 1 (p. 3) .)

The explore method

The call to the **explore** method is new to this module.

The **explore** method is defined in the **SimplePicture** class, which is the superclass of the **Picture** class. The method is inherited into the **Picture** class.

javadocs description of the explore method

The javadocs description of this method is shown in Figure 4 (p. ??) .

> Method to open a picture explorer on a copy of this simple picture.

Figure 4: javadocs description of the explore method.

Result of calling the explore method

The result of calling the **explore** method in Listing 3 (p. 6) is to create and display the **PictureExplorer** object shown in Figure 1 (p. 3) .

Very important capability

The availability of the **explore** method and the **PictureExplorer** class is very important in at least two respects:

- The **explore** method makes it easy to display copies of an image at various stages during the processing of the image. Once the **PictureExplorer** object is created and displayed, it won't be effected by subsequent changes to the image.
- The availability of a **PictureExplorer** object makes it easy to manually analyze the colors of the individual pixels in an image encapsulated in that object.

Implementing the algorithm

The code in Listing 4 (p. 7) implements the algorithm required to modify the original image to make it look like the image shown in Figure 2 (p. 4) .

Listing 4: Implementing the algorithm.

```
    Pixel[] pixelArray = pic.getPixels();

for(Pixel pixel:pixelArray ){
  pixel.setRed(255 - pixel.getRed());
  pixel.setGreen(255 - pixel.getGreen());
  pixel.setBlue(0);
}//end for loop
```

In particular, the code in Listing 4 (p. 7) sets the blue color components to 0 and inverts the red and green color components for every pixel in the picture.

One of several approaches

There are several ways to do this, and this is only one of those ways. This approach makes use of a method named **getPixels** that is defined in the **SimplePicture** class and inherited into the **Picture** class.

Very useful when...

This approach is particularly useful when you want to perform the same action on every pixel in an image. The advantage is that you don't have to worry about horizontal and vertical coordinates with this approach. Access to all of the pixels is provided in a one-dimensional array.

javadocs description of the getPixels method

The javadocs description of this method is shown in Figure 5 (p. **??**) .

Returns: a one-dimensional array of **Pixel** objects starting with y=0 to y=height-1 and x=0 to x=width-1.

Returns: a one-dimensional array of **Pixel** objects starting with y=0 to y=height-1 and x=0 to x=width-1.

Method to get a one-dimensional array of **Pixels** for this simple picture. Returns: a one-dimensional array of **Pixel** ob

---

Figure 5: javadocs description of the getPixels method.

What is a Pixel object?

An object of Ericson's **Pixel** class encapsulates an individual pixel from an image. Figure 6 (p. **??**) shows the javadocs description of the **Pixel** class.

A pixel has an x and y location in a picture.

A pixel knows how to get and set the red, green, blue, and alpha values in the picture.

A pixel also knows how to get and set the color using a Color object.

A pixel has an x and y location in a picture.

A pixel knows how to get and set the red, green, blue, and alpha values in the picture.

A pixel also knows how to get and set the color using a Color object.

Class that references a pixel in a picture. A pixel has an x and y location in a picture. A pixel knows how to get and set t

---

Figure 6: javadocs description of the Pixel class.

Many methods available

The **Pixel** class defines a large number of methods. Once you have a reference to a **Pixel** object, you can manipulate the underlying pixel encapsulated in that object in a variety of ways.

Get the pixels in the image

Recall that a reference to the **Picture** object that encapsulates our image is stored in the variable named **pic** . *(See Listing 2 (p. 6) .)*

Listing 4 (p. 7) begins by calling the **getPixels** method on that reference.

All of the pixels in the image are returned in a one-dimensional array.

A reference to the array is stored in a local reference variable of type **Pixel[]** named **pixelArray** .

A for-each loop

A special kind of **for** loop *(often called a for-each loop)* is used to access and process each pixel in the array. *(A conventional **for** loop could also be used here.)*

During each iteration of the loop...

The three statements inside the loop modify the color values of a single pixel.

The first two statements invert the red and green color values by subtracting the values from 255.

The third statement in the loop sets the blue color value to zero.

When the loop terminates...
Every pixel in the image will have been modified as described above.
Not a reversible process
Because the blue color values were set to zero, the image has now been modified in an irreversible manner.
A reversible process
However, if the blue color values had also been inverted, the process would be reversible.
All that would be necessary to recover the original image would be to invert all of the pixels again.
An important process
Color inversion is a very important process in many areas of computing that involve images. The process
is:

- Computationally cheap
- Very fast
- Usually visually obvious
- Totally reversible

Often used to highlight selected images
For example, many software program invert all of the colors in an image when it is selected for some
purpose, such as copying to the clipboard. Then the colors are restored to their original values when the
image is deselected.

Next to redeye correction, color inversion is probably the most commonly used color modification algo-
rithm in use in modern image processing.
Display again and terminate
The variable named **pic** still contains a reference to the original **Picture** object. However, the image
that is encapsulated in that object has been significantly modified.

Listing 5 (p. 9) calls the **explore** method again, creating and displaying another **PictureExplorer**
object that encapsulates a copy of the **Picture** object with the modified image.

Listing 5: Display again and terminate.

```
        pic.explore();

  }//end run method
}//end class Prob02Runner
```

The result is shown in Figure 2 (p. 4) .
The end of the program
Listing 5 (p. 9) also signals the end of the **run** method and the end of the **Prob02Runner** class.
Return control to main
The **run** method terminates and returns control to the **main** method in Listing 1 (p. 5) .
The code in the **main** method calls a *getter* method to get a reference to the **Picture** object.
The reference is passed to the **println** method, which displays the information about the **Picture**
object in the last line of Figure 3 (p. 5) .
The program terminates
Then the **main** method terminates, at which time the program terminates and returns control to the
operating system.

4.1 Run the program

I encourage you to copy the code from Listing 6 (p. 11) , compile it 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.

You can download a copy of the raw image file here [2] .

5 Summary

In this module, you learned how to invert images and how to display images in **PictureExplorer** objects.

6 What's next?

Y ou will learn how to implement a space-wise linear color-modification algorithm in the next module.

7 Online video links

Select the following links to view online video lectures on the material in this module.

- ITSE 2321 Lecture 02 [3]
    - Part01 [4]
    - Part02 [5]
    - Part03 [6]
    - Part04 [7]

8 Miscellaneous

This section contains a variety of miscellaneous information.

NOTE: **Housekeeping material**

- Module name: Java OOP: Image Processing Algorithms, Image Inversion, and PictureExplorer Objects
- File: Java3004.htm
- Published: July 29, 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.

---

[2]http://cnx.org/content/m44203/1.3/Prob02.jpg
[3]http://www.youtube.com/playlist?list=PL713DB9A1FF4B92DF
[4]http://www.youtube.com/watch?v=SVq_IN4TsTs
[5]http://www.youtube.com/watch?v=vcVLr8Z1mo4
[6]http://www.youtube.com/watch?v=M1Nns7vYTiM
[7]http://www.youtube.com/watch?v=Qw-yzEGuFJU

9 Complete program listing

A complete listing of the program discussed in this module is shown in Listing 6 (p. 11) below.

Listing 6: Complete program listing.

```
    /*File Prob02 Copyright 2008 R.G.Baldwin
***********************************************************/

public class Prob02{//the driver class
  public static void main(String[] args){
    Prob02Runner obj = new Prob02Runner();
    obj.run();
    System.out.println(obj.getPicture());
  }//end main
}//end class Prob02
//=========================================================//

class Prob02Runner{
  private Picture pic = new Picture("Prob02.jpg");

  public Prob02Runner(){//constructor
    System.out.println("Dick Baldwin");
  }//end constructor
  //-------------------------------------------------------//

  //Accessor method
  public Picture getPicture(){return pic;}
  //-------------------------------------------------------//

  //This method is where the action is.
  public void run(){
    //Display the raw picture.
    pic.addMessage("Dick Baldwin",10,20);
    pic.explore();

    //Set the blue color components to 0 and invert the
    // red and green color components for every pixel in
    // the picture.
    Pixel[] pixelArray = pic.getPixels();
    for(Pixel pixel:pixelArray ){
      pixel.setRed(255 - pixel.getRed());
      pixel.setGreen(255 - pixel.getGreen());
      pixel.setBlue(0);
    }//end for loop

    //Display the modified picture
    pic.explore();
  }//end run method
}//end class Prob02Runner

 -end-
```