Java OOP: Implementing a space-wise linear color-modification algorithm.*

Richard Baldwin

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

Abstract

The cnxml version of this module is incomplete. You will find a copy of the original version at http://www.austincc.edu/baldwin/ITSE2321LectureNotesAndSlides/LectureNotes/Lecture03/Lecture03.htm

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. 5)
- Run the program (p. 8)
- Summary (p. 9)
- What's next? (p. 9)
- Online video links (p. 9)
- Miscellaneous (p. 9)
- Complete program listing (p. 10)

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 9:34 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

- Figure 1 (p. 3) . The raw image.
- Figure 2 (p. 4) . The modified image.
- Figure 3 (p. 5) . Text output on the command-line screen.

2.1.2 Listings

- Listing 1 (p. 5) . The driver class named Prob03.
- Listing 2 (p. 5) . Beginning of the class named Prob03Runner.
- Listing 3 (p. 6) . The beginning of the run method.
- Listing 4 (p. 7) . Beginning of the for loop.
- Listing 5 (p. 7) . Compute the column number and scale factors.
- Listing 6 (p. 8) . Apply the scale factors.
- Listing 7 (p. 8) . Display the modified image.
- Listing 8 (p. 10) . Complete program listing.

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 **Prob03.jpg** along with a written specification of a space-wise linear image modification algorithm.

Implement the algorithm

The primary purpose of the test is to determine if the student can implement the algorithm 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 specified 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.

The algorithm

Scale the blue and green color components by a scale factor that is less than or equal to 1.0. The green scale factor:

- Is equal to 1.0 on the left side of the image
- Is equal to 0.0 on the right side of the image
- Decreases linearly with distance going from left to right across the image.

The blue scale factor

- Is 0.0 on the left side of the image
- Is 1.0 on the right side of the image
- Increases linearly with distance going from left to right across the image.

Do not scale the red color component.

The program output

The program produces the images shown in Figure 1 (p. 3) and Figure 2 (p. 4) and produces the output text shown in Figure 3 (p. 5) on the command line screen.

The raw image.

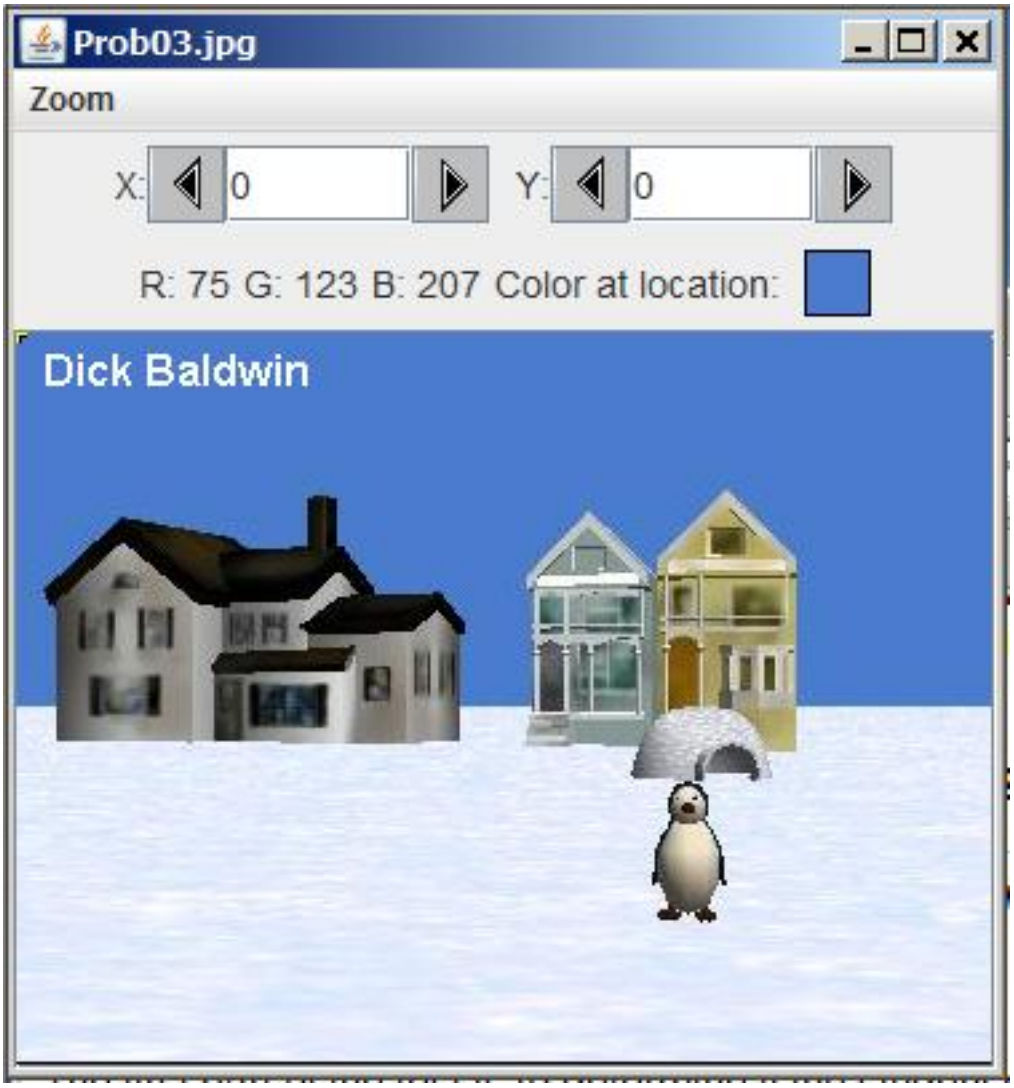

Figure 1: The raw image.

The modified image.

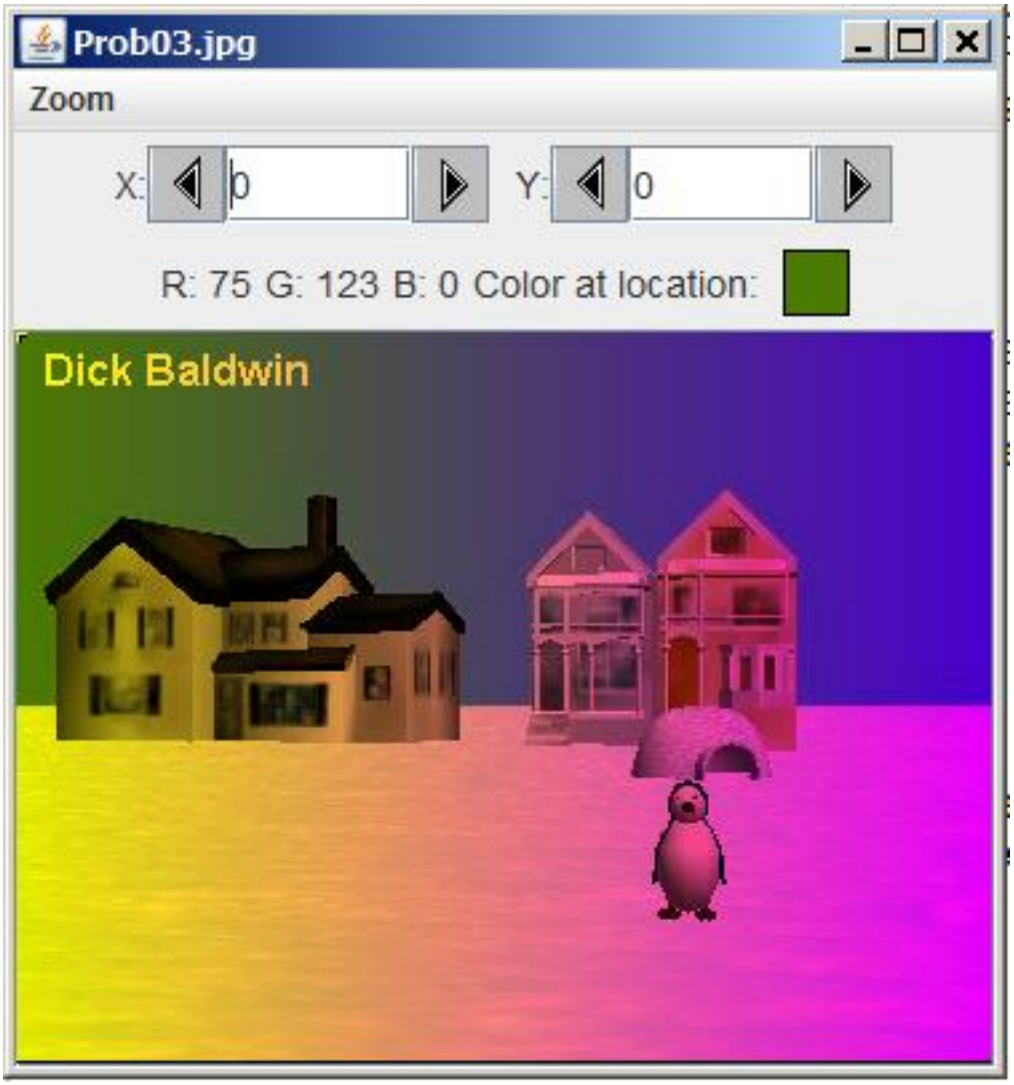

Figure 2: The modified image.

Text output 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 Prob03.jpg height 274 width 365
```

Figure 3: Text output on the command-line screen.

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

4 Discussion and sample code

Will explain in fragments

I will explain this program in fragments. A complete listing is provided in Listing 8 (p. 10) near the end of the module.

I will begin with the driver class named **Prob03** , which is shown in its entirety in Listing 1 (p. 5) .

Listing 1: The driver class named Prob03.

```
    public class Prob03{
  public static void main(String[] args){
    Prob03Runner obj = new Prob03Runner();
    obj.run();
    System.out.println(obj.getPicture());
  }//end main
}//end class Prob03
```

There is nothing in Listing 1 (p. 5) that I haven't explained in earlier modules. Therefore, no explanation of the code in Listing 1 (p. 5) should be required.

Beginning of the class named Prob03Runner

The class definition for the class named **Prob03Runner** begins in Listing 2 (p. 5) .

Listing 2: Beginning of the class named Prob03Runner.

```
  class Prob03Runner{
//Instantiate the Picture object.
private Picture pic = new Picture("Prob03.jpg");

public Prob03Runner(){//constructor
  System.out.println("Dick Baldwin");
}//end constructor
//-----------------------------------------------------//

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

Once again, there is nothing in Listing 2 (p. 5) that I haven't explained before. I included it here simply for the sake of continuity.

The beginning of the run method

The **run** method begins in Listing 3 (p. 6) . The **run** method is where most of the interesting action takes place.

Listing 3: The beginning of the run method.

```
  public void run(){
pic.addMessage("Dick Baldwin",10,20);
//Display a PictureExplorer object.
pic.explore();

//Get an array of Pixel objects.
Pixel[] pixels = pic.getPixels();

//Declare working variables
Pixel pixel = null;
int green = 0;
int blue = 0;
int width = pic.getWidth();
double greenScale = 0;
double blueScale = 0;
```

Much of what you see in Listing 3 (p. 6) has been explained in earlier modules. However, Listing 3 (p. 6) does deserve a few comments.

Display the raw image

The call to the **explore** method produces the output shown in Figure 1 (p. 3) .

Get an array of Pixel data

The call to the **getPixels** method in Listing 3 (p. 6) returns a reference to a one-dimensional array object. The elements in the array are references to **Pixel** objects, where each **Pixel** object represents a single pixel in the image. I will explain the organization of the pixel data later [2] .

Get the width of the image

The call to the **getWidth** method in Listing 3 (p. 6) returns an **int** value that specifies the width of the image in pixels. This value will be used later to compute the column to which each pixel belongs.

Local variables

Listing 3 (p. 6) declares six local variables. The purpose of these variables should become clear during the explanation of the code that implements the algorithm.

[2] http://cnx.org/content/m44204/1.3/Java3006old.htm#Organization_of_the_pixel_data

Implementation of the algorithm

The algorithm [3] is implemented by the code in a conventional **for** loop, which begins in Listing 4 (p. 7) .

Listing 4: Beginning of the for loop.

```
  for(int cnt = 0;cnt < pixels.length;cnt++){

pixel = pixels[cnt];

green = pixel.getGreen();

blue = pixel.getBlue();
```

The loop iterates through the array of **Pixel** data, modifying the colors in one pixel during each iteration.

The length property of the array object

Every array object in Java contains a **length** property that contains the number of elements in the array. The value of this property is used in the conditional clause in the **for** loop in Listing 4 (p. 7) to establish when the end of the array has been reached in order to terminate the loop.

Get reference to the next Pixel object

The first statement inside the **for** loop in Listing 4 (p. 7) gets a reference to a **Pixel** object from the next array element. That reference is stored in the local variable of type **Pixel** named **pixel** that was declared in Listing 3 (p. 6) .

Get the red and green color values for the current pixel

Having gotten a reference to the **Pixel** object, the next statement calls the **getGreen** method on that reference to get and save the value of the green color component in the current pixel.

Similarly, the statement following that one gets and saves the value of the blue color component in the current pixel.

Both values are returned as type **int** , and can range in value from 0 up to and including 255.

Objective is to scale the green and blue color values

Recall that the objective is to scale the green and blue color values on a column by column basis, going from left to right across the image shown in Figure 1 (p. 3) in order to produce the output image shown in Figure 2 (p. 4) .

Organization of the pixel data

The pixel data is stored in the array on a row by row basis. In other words, the first **width** elements contain references to pixels in the first row of pixels going from left to right across the screen. The next **width** elements contain references to pixels in the second row of pixels, etc.

Compute the column number and scale factors

Listing 5 (p. 7) uses the modulus operator to compute the column number for each **Pixel** object.

Listing 5: Compute the column number and scale factors.

```
    //Compute the column number and use it to compute
// the scale factor.
int col = cnt%width;

greenScale = (double)(width - col)/width;
blueScale = (double)(col)/width;
```

[3] http://cnx.org/content/m44204/1.3/Java3006old.htm#The_algorithm

An exercise for the student

Knowing the column number in which the pixel is located, the next step is to compute the green and blue scale factors necessary to satisfy the algorithm.

I will leave it as an exercise for the student to think about how the expressions contained in the last two statements in Listing 5 (p. 7) cause the two scale factors to vary linearly from left to right across the image in accordance with the requirements of the algorithm. *(Think about the equation of a straight line from your high school math classes.)*

Apply the scale factors

The **Pixel** class contains methods named **setRed** , **setGreen** , and **setBlue** that can be called to set the color values for the pixel represented by a **Pixel** object.

Listing 6 (p. 8) computes new values for the red and green components based on the existing color values for the pixel and the scale factors computed in Listing 5 (p. 7) .

Listing 6: Apply the scale factors.

```
      pixel.setGreen((int)(green * greenScale));
  pixel.setBlue((int)(blue * blueScale));
 }//end for loop
```

Then Listing 6 (p. 8) calls the **setGreen** and **setBlue** methods on the **Pixel** object to set the green and blue color values to the newly computed values.

The end of the for loop

Listing 6 (p. 8) also signals the end of the **for** loop that began in Listing 4 (p. 7) .

Display the modified image

Finally, Listing 7 (p. 8) calls the **explore** method again to display the image shown in Figure 2 (p. 4) .

Listing 7: Display the modified image.

```
      pic.explore();

  }//end run method
}//end class Prob03Runner
```

The end of the program

Listing 7 (p. 8) also signals the end of the **run** method and the end of the **Prob03Runner** class.

Return control to main

The **run** method terminates and returns control to the **main** method shown 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 8 (p. 10) , 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 required input image file here [4] .

[4] http://cnx.org/content/m44204/1.3/Prob03.jpg

5 Summary

In this module, I showed you how to implement an algorithm that causes the green and blue color values in an image to change in a linear fashion going from left to right across the image.

6 What's next?

You will learn more about abstract methods, abstract classes, and overridden methods in the next lesson. Very importantly, you will learn more about overriding the toString method.

7 Online video links

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

- ITSE 2321 Lecture 03 [5]
  - Part01 [6]
  - Part02 [7]
  - Part03 [8]

8 Miscellaneous

This section contains a variety of miscellaneous information.

NOTE: **Housekeeping material**

- Module name: Java OOP: Implementing a space-wise linear color-modification algorithm
- File: Java3006.htm
- Published: July 30, 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.

[5] http://www.youtube.com/playlist?list=PL43014B9ED4419642
[6] http://www.youtube.com/watch?v=pdqKvAuzkhg
[7] http://www.youtube.com/watch?v=aXoSD7oauLQ
[8] http://www.youtube.com/watch?v=1iRiXhTPmMU

9 Complete program listing

A complete listing of the program discussed in this module is shown in Listing 8 (p. 10) below.

Listing 8: Complete program listing.

```
    /*File Prob03 Copyright 2008 R.G.Baldwin
*******************************************************/

public class Prob03{
  public static void main(String[] args){
    Prob03Runner obj = new Prob03Runner();
    obj.run();
    System.out.println(obj.getPicture());
  }//end main
}//end class Prob03
//=====================================================//

class Prob03Runner{
  //Instantiate the Picture object.
  private Picture pic = new Picture("Prob03.jpg");

  public Prob03Runner(){//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(){
    pic.addMessage("Dick Baldwin",10,20);
    //Display a PictureExplorer object.
    pic.explore();

    //Get an array of Pixel objects.
    Pixel[] pixels = pic.getPixels();

    //Declare working variables
    Pixel pixel = null;
    int green = 0;
    int blue = 0;
    int width = pic.getWidth();
    double greenScale = 0;
    double blueScale = 0;

    //Scale the blue and green color components according
    // to the algorithm given above.
    //Do not scale the red component.
```

```
    for(int cnt = 0;cnt < pixels.length;cnt++){
      //Get blue and green values for current pixel.
      pixel = pixels[cnt];
      green = pixel.getGreen();
      blue = pixel.getBlue();

      //Compute the column number and use it to compute
      // the scale factor.
      int col = cnt%width;
      greenScale = (double)(width - col)/width;
      blueScale = (double)(col)/width;

      //Apply the scale factor.
      pixel.setGreen((int)(green * greenScale));
      pixel.setBlue((int)(blue * blueScale));
    }//end for loop

    //Display a PictureExplorer object.
    pic.explore();

  }//end run method
}//end class Prob03Runner
```

-end-