

# JAVA OOP: GREEN-SCREEN PROCESSING\*

R.G. (Dick) Baldwin

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

## Abstract

Learn to write a program to do green-screen processing.

## 1 Table of Contents

- Preface (p. 1)
  - Viewing tip (p. 1)
    - \* Figures (p. 2)
    - \* Listings (p. 2)
- Preview (p. 2)
- General background information (p. 8)
- Discussion and sample code (p. 8)
- Run the program (p. 14)
- Summary (p. 14)
- What's next? (p. 14)
- Online video links (p. 15)
- Miscellaneous (p. 15)
- Complete program listing (p. 15)

## 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 <sup>1</sup>.

### 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: Nov 14, 2012 11:40 am -0600

<sup>†</sup><http://creativecommons.org/licenses/by/3.0/>

<sup>1</sup><http://cnx.org/content/m44148/latest/>

### 2.1.1 Figures

- Figure 1 (p. 3) . Input image file Prob03a.bmp.
- Figure 2 (p. 4) . Input image file Prob03b.bmp.
- Figure 3 (p. 5) . Input image file Prob03c.bmp.
- Figure 4 (p. 6) . Input image file Prob03d.jpg.
- Figure 5 (p. 7) . Output picture.
- Figure 6 (p. 8) . Required output text.
- Figure 7 (p. 11) . Front view of the skater after cropping.

### 2.1.2 Listings

- Listing 1 (p. 8) . The driver class named Prob03.
- Listing 2 (p. 9) . Beginning of the class named Prob03Runner.
- Listing 3 (p. 9) . Beginning of the run method.
- Listing 4 (p. 11) . Remainder of the run method.
- Listing 5 (p. 13) . The greenScreenDraw method.
- Listing 6 (p. 15) . Complete program listing.

## 3 Preview

In this lesson, you will learn how to write a program to do *green-screen* processing to superimpose a source image onto a destination image while making the green background of the source image appear to be transparent.

### Program specifications

Write a program named **Prob03** that uses the class definition shown in Listing 1 (p. 8) and Ericson's media library along with the image files in the following list to produce the five graphic output images shown in Figure 1 (p. 3) through Figure 5 (p. 7) .

- Prob03a.bmp
- Prob03b.bmp
- Prob03c.bmp
- Prob03d.jpg

Input image file Prob03a.bmp.

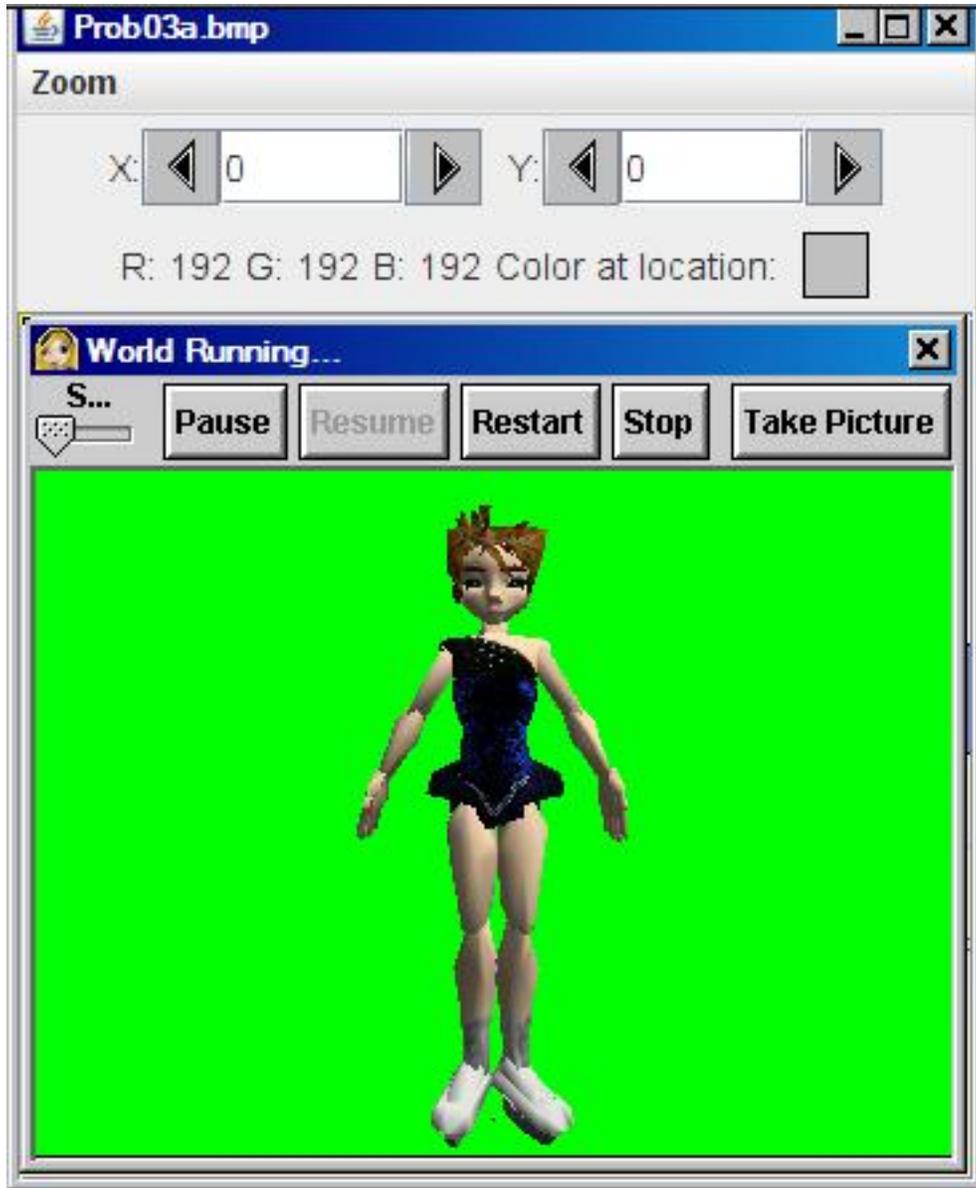


Figure 1: Input image file Prob03a.bmp.

Input image file Prob03b.bmp.

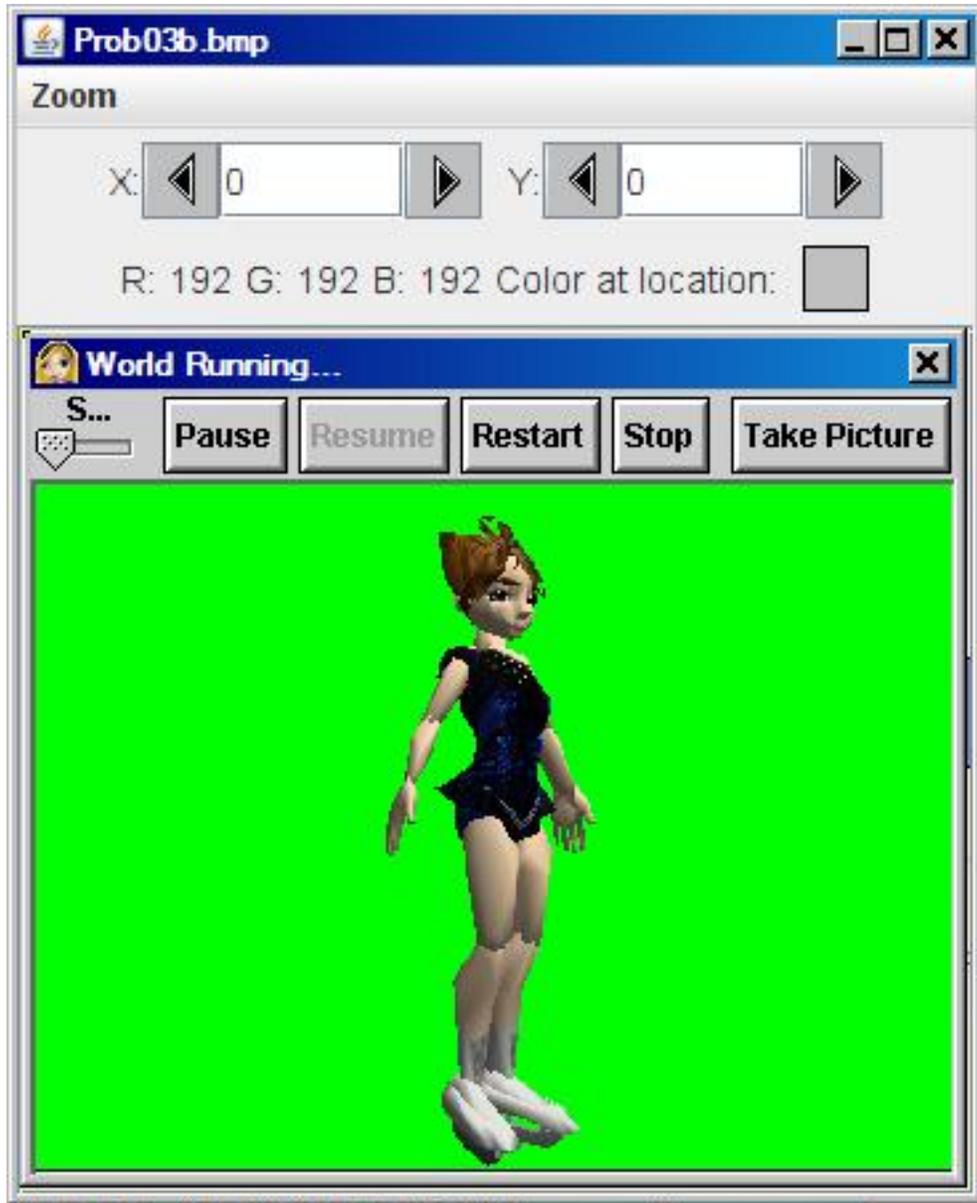


Figure 2: Input image file Prob03b.bmp.

Input image file Prob03c.bmp.

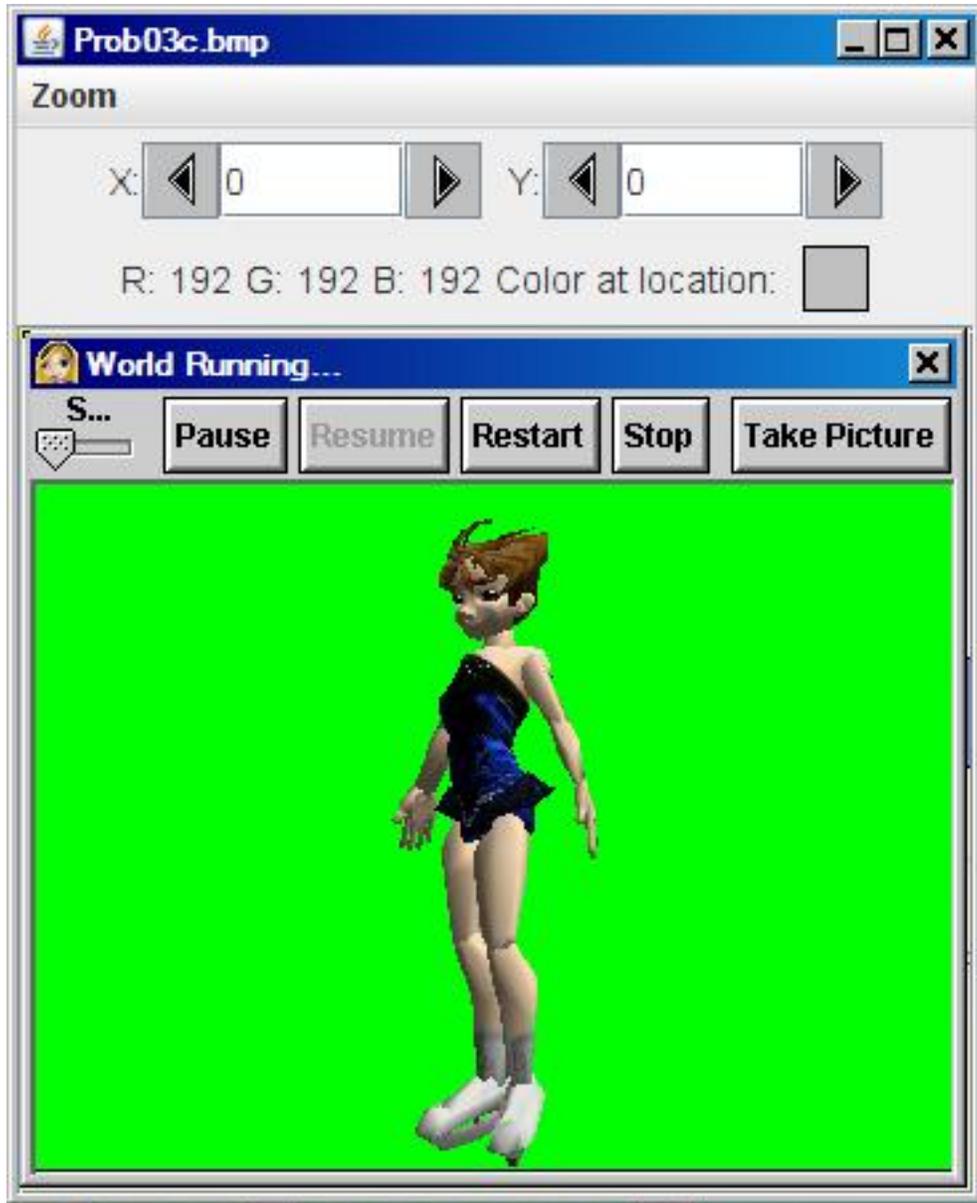


Figure 3: Input image file Prob03c.bmp.

Input image file Prob03d.jpg.

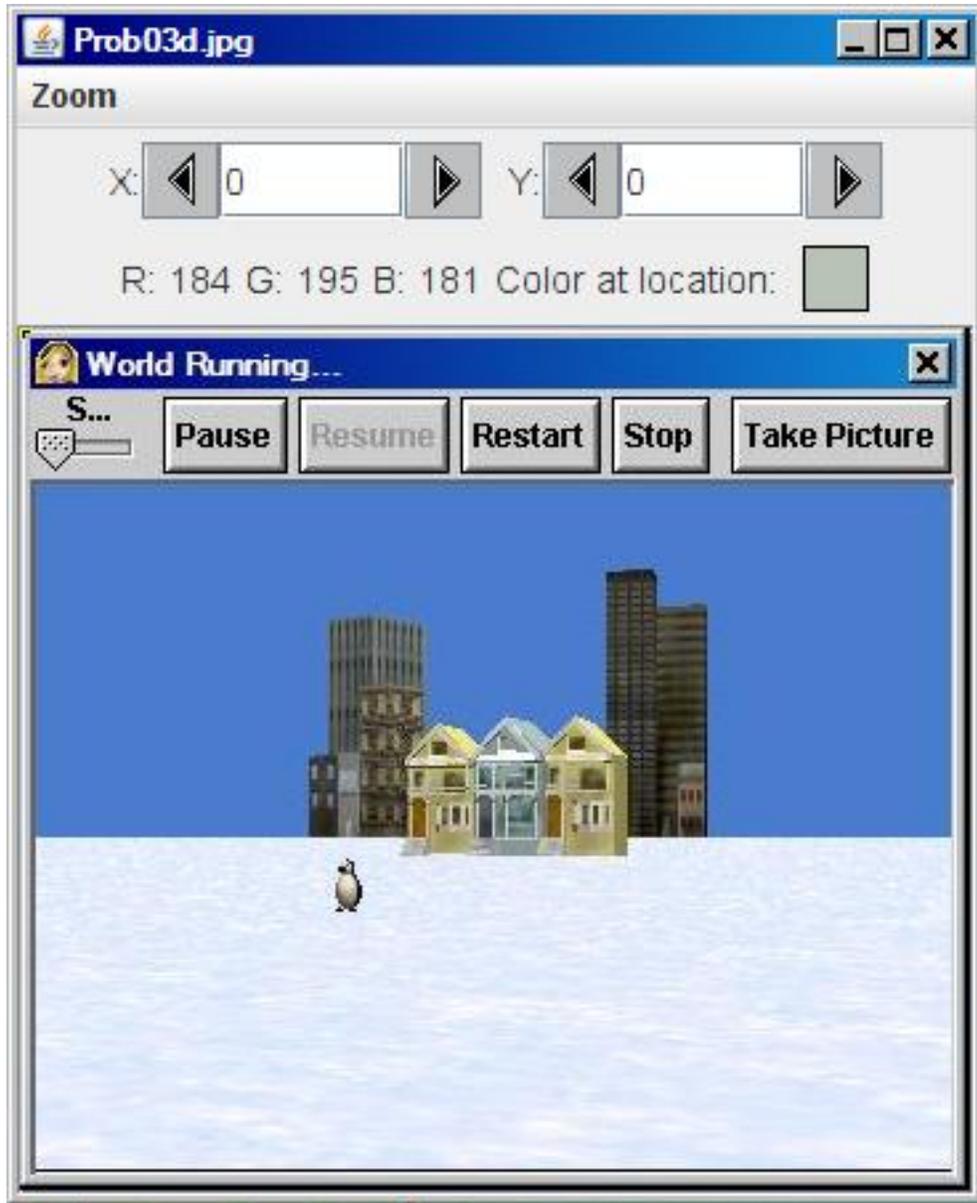
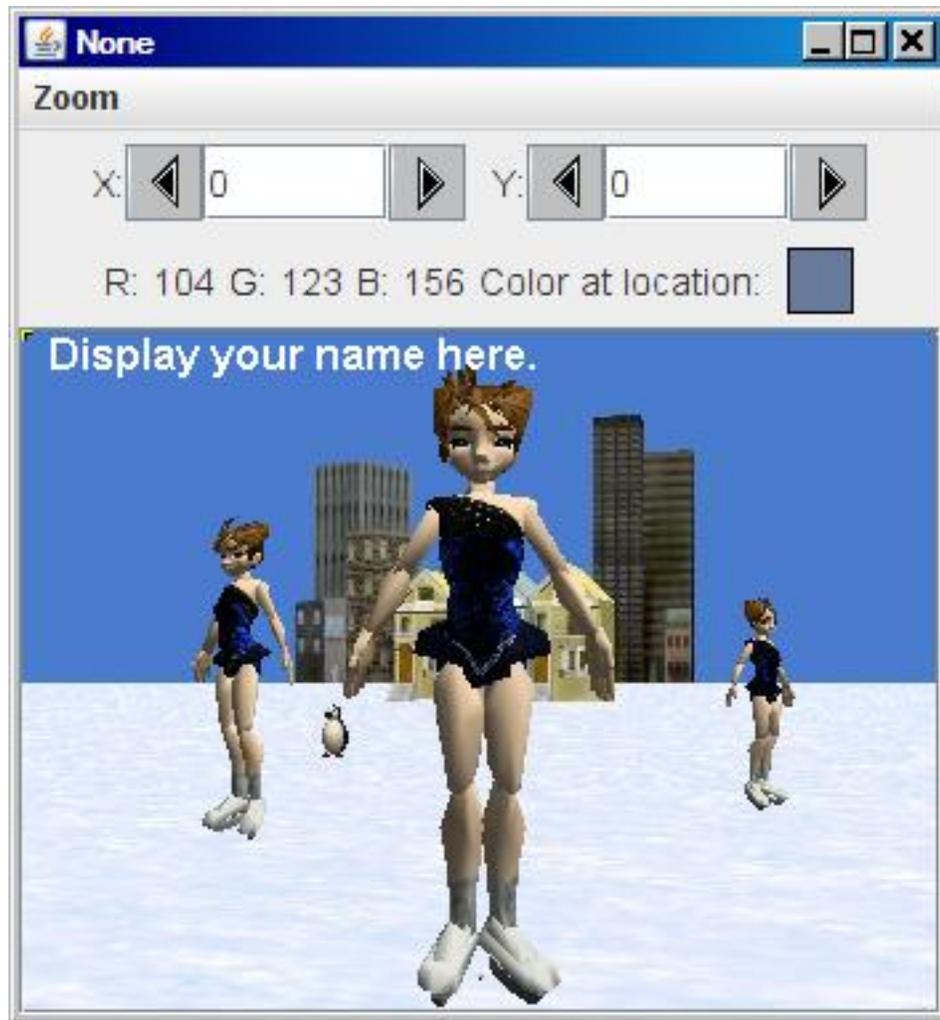


Figure 4: Input image file Prob03d.jpg.

---

**Output picture.**

**Figure 5:** Output picture.

---

**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 **Prob03** given in Listing 1 (p. 8) .

**Required output text**

In addition to the output images mentioned above, your program must display your name and one other line of text on the command-line screen as shown in Figure 6 (p. 8) .

---

### Required output text.

```
Display your name here.  
Picture, filename None height 256 width 344
```

**Figure 6:** Required output text.

---

## 4 General background information

This program receives three views of an ice skater in **bmp** files with a pure green background along with a **jpg** file containing a snow scene.

All four files show the Alice <sup>2</sup> runtime panel with the words **World Running...** and sever associated buttons. (See *Figure 1 (p. 3)* .)

### Program behavior

The program performs the following actions:

- Crops the snow scene to remove the Alice runtime panel.
- Crops the three views of the skater to remove the Alice runtime panel along with excess blank green background.
- Scales two of the views of the skater to smaller sizes.
- Does green-screen processing to place the three views of the skater at different locations in the snow scene.
- Uses position along with size to create an optical illusion of a 3D scene of three ice skaters and a penguin standing at different locations on a frozen lake (see *Figure 5 (p. 7)* ) .

### Programming skills required

In order to write this program, the student must be able to, as a minimum, write a green-screen processing method.

## 5 Discussion and sample code

### Will discuss in fragments

I will discuss this program in fragments. A complete listing of the program is provided in Listing 6 (p. 15) near the end of the lesson.

### The driver class named Prob03

The driver class containing the **main** method is shown in Listing 1 (p. 8) .

### Listing 1: The driver class named Prob03.

---

<sup>2</sup><http://www.alice.org/>

```

import java.awt.Color;

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

```

The **main** method in Listing 1 (p. 8) instantiates a new object of the class named **Prob03Runner** and calls the method named **run** that belongs to that object.

When the **run** method returns, the program terminates.

#### **Beginning of the class named Prob03Runner**

The beginning of the class named **Prob03Runner**, and its constructor, is shown in Listing 2 (p. 9).

#### **Listing 2: Beginning of the class named Prob03Runner.**

```

class Prob03Runner{

public Prob03Runner(){ //constructor
    System.out.println("Display your name here.");
} //end constructor

```

The constructor simply displays the student's name on the command line screen producing the first line of text shown in Figure 6 (p. 8).

#### **Beginning of the run method**

The beginning of the **run** method that is called in Listing 1 (p. 8) is shown in Listing 3 (p. 9).

#### **Listing 3: Beginning of the run method.**

```

public void run(){

//A view facing the front of the skater.
Picture front = new Picture("Prob03a.bmp");
front.explore();
front = crop(front,123,59,110,256);

//A view showing the right side of the skater.
Picture right = new Picture("Prob03b.bmp");
right.explore();
right = crop(right,123,59,110,256);

//A view showing the left side of the skater.
Picture left = new Picture("Prob03c.bmp");
left.explore();
left = crop(left,123,59,110,256);

//This will be the background for the new picture.
Picture snowScene = new Picture("Prob03d.jpg");
snowScene.explore();
snowScene = crop(snowScene,6,59,344,256);

```

The code in Listing 3 (p. 9) instantiates, displays, and crops the four input images.

All four images must be cropped to remove the Alice runtime window. In addition, the three skater images are also cropped to remove excess blank green background material.

#### **Image formats: bmp versus jpg**

Note that the three views of the skater are extracted from **bmp** image files instead of **jpg** image files. This is necessary in order to preserve the pure green background color. Storing the images as **jpg** files would corrupt the background color in the low order bits making it more difficult to achieve the green-screen processing required by this program.

#### **The method named crop**

Listing 3 calls the **crop** method four times in succession. once for each of the four **Picture** objects instantiated from the image files.

I explained image cropping in an earlier module. The **crop** method used in this program is very similar to the methods that I explained in the earlier module, so I won't explain it again in this module. You can view the **crop** method in detail in Listing 6 (p. 15) near the end of the module.

#### **Five incoming parameters**

The **crop** method requires five incoming parameters. The first parameter is a reference to the **Picture** object that is to be cropped. The remaining four integer parameters specify the rectangular area that is to be preserved after the picture is cropped.

#### **The rectangular area to be preserved**

The first two integers specify the column and row coordinates for the upper-left corner of the rectangular area that is to be preserved. The last two integers specify the width and the height of the rectangle that is to be preserved.

Note that in all four cases, the height of the rectangular area that is to be preserved is 256 pixels. This will be important later on with respect to scaling two of the images.

#### **Returns a reference to a cropped picture**

The **crop** method returns a reference to a **Picture** object that is a cropped version of the picture whose reference is passed to the method. In each case, the code in Listing 3 (p. 9) replaces the reference to the original **Picture** object with the reference to the cropped **Picture** object.

#### **Front view of the skater after cropping**

If you were to display the **Picture** object referred to by the variable **front** after cropping, you would see the image shown in Figure 7 (p. 11) .

Front view of the skater after cropping.

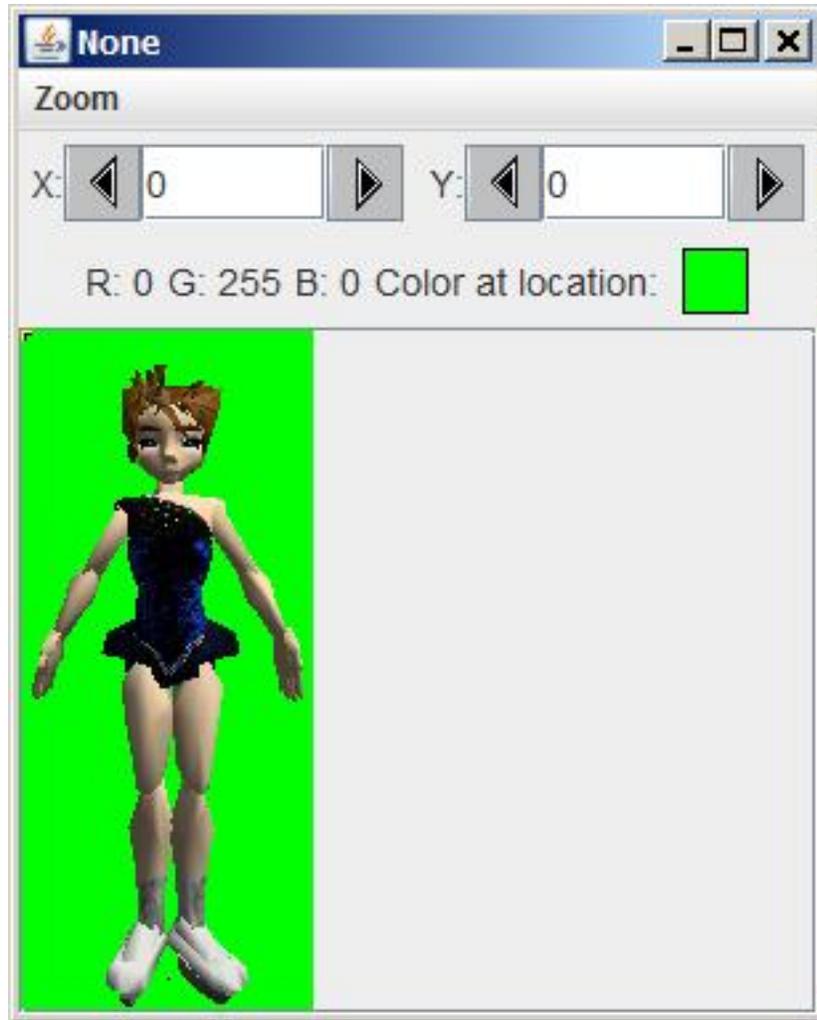


Figure 7: Front view of the skater after cropping.

### Transparent pixels

This is the image that appears as the center ice skater in Figure 5 (p. 7) after green-screen processing. Note that all of the green pixels in Figure 7 (p. 11) appear to be transparent in Figure 5 (p. 7) .

### Remainder of the run method

Continuing with the `run` method, Listing 4 (p. 11) calls the method named `greenScreenDraw` three times in succession to draw the three skater images at specific locations on the snow scene with green-screen processing.

### Listing 4: Remainder of the run method.

```

        //Draw the front view of the skater on the snowScene
// at full size.
greenScreenDraw(front,snowScene,117,0);

//Draw the left side view of the skater on the
// snowScene at half size.
left = left.getPictureWithHeight(256/2);
greenScreenDraw(left,snowScene,55,64);

//Draw the right side view of the skater on the
// snowScene at one-third size.
right = right.getPictureWithHeight(256/3);
greenScreenDraw(right,snowScene,260,96);

//Display students name on the final output and
// display it.
snowScene.addMessage("Display your name here.",10,15);
snowScene.explore();
System.out.println(snowScene);
} //end run method

```

### Two skater images are scaled

In two cases in Listing 4 (p. 11) , the method named **getPictureWithHeight** is called before calling the **greenScreenDraw** method. The **getPictureWithHeight** method is used to scale two of the images to a smaller size as shown in Figure 5 (p. 7) .

#### The **getPictureWithHeight** method

The **getPictureWithHeight** method is defined in Ericson's **SimplePicture** class and inherited into Ericson's **Picture** class.

The method requires a single integer input parameter, which specifies the height in pixels of a scaled output version of the picture object on which the method is called.

According to the documentation, the method can be used to create a new picture with the specified height. The aspect ratio of the width and height will stay the same.

#### Original height is 256 pixels

Referring back to the parameters that were passed to the **crop** method in Listing 3 (p. 9) , you can see the height of all three cropped images is 256 pixels.

#### Scale by 1/2 and 1/3

Referring to the two calls to the **getPictureWithHeight** method in Listing 4 (p. 11) , you can see that one of the cropped images was replaced with an image having a height of  $256/2$  or 128 pixels. The other cropped image was replaced with an image having a height of  $256/3$  or 85 pixels. You can see the effect of this scaling in Figure 5 (p. 7) .

The height of one of the images was not changed, which you can also see in Figure 5 (p. 7) .

#### Put the run method on hold

I will put the explanation of the **run** method on temporary hold at this point and explain the method named **greenScreenDraw** , which is shown in Listing 5 (p. 13) .

#### Behavior of the **greenScreenDraw** method

The **greenScreenDraw** method requires four incoming parameters:

- A reference to a source image.
- A reference to a destination image
- The horizontal coordinate on the destination image where the upper-left corner of the source image will be drawn.

- The vertical coordinate on the destination image where the upper-left corner of the source image will be drawn.

### Pure green pixels are required for transparency

The method draws the source image onto the destination image at the specification location. However, pixels having a pure green color are not drawn on the destination image. The effect is to make it appear that those portions of the source image with pure green pixels become totally transparent allowing the pixels belonging to the destination image to show through.

#### jpg image files are not satisfactory for this program

**Picture** objects created from jpg image files typically won't have a pure green background even if they had a pure green background before being compressed into the jpg format file. However, the bmp file format does not corrupt the pixel colors. Therefore, bmp images work well for this type of processing.

#### The greenScreenDraw method

The **greenScreenDraw** method is shown in its entirety in Listing 5 (p. 13) .

#### Listing 5: The greenScreenDraw method.

```
private void greenScreenDraw(
    Picture source,
    Picture dest,
    //Place the upper-left corner
    // of the source image at the
    // following location in the
    // destination image.
    int destX,
    int destY){
    int width = source.getWidth();
    int height = source.getHeight();
    Pixel pixel = null;
    Color color = null;

    for(int row = 0;row < height;row++){
        for(int col = 0;col < width;col++){
            color = source.getPixel(col,row).getColor();
            if(!(color.equals(Color.GREEN))){
                pixel = dest.getPixel(destX + col,destY + row);
                pixel.setColor(color);
            }//end if
        }//end inner loop
    }//end outer loop

} //end greenScreenDraw
```

#### Very similar to earlier methods

This method is very similar to other methods that I have explained in earlier modules that use nested **for** loops to draw one image onto another image.

#### The one new thing...

The only thing that is really new in Listing 5 (p. 13) is the **if** statement that tests the color of source image pixels for a value of exactly **Color.GREEN** . If the color of the source pixel does not match that color exactly, it is drawn on the destination image replacing the pixel color previously at that location on the destination image.

If the source pixel color does exactly match that color, it is not drawn on the destination image thereby leaving the color of the corresponding destination pixel unchanged.

Listing 5 (p. 13) signals the end of the `greenScreenDraw` method.

### The weather forecast on television

This is roughly how the TV stations superimpose a human weather forecaster onto a giant animated weather map. The forecaster is photographed with a video camera standing in front of a green or blue screen. At the same time, an animated video of the weather map is also created.

Then each video frame of the forecaster is superimposed onto a video frame of the weather map. The green or blue pixels in the forecaster frame are not copied onto the weather map frame. This allows the weather map pixels to show with the exception of those that are replaced by the pixels that comprise the human forecaster. (*The forecaster must be careful to avoid wearing clothing that matches the color of the green or blue screen.*)

### Returning to the run method

When the third call to the `greenScreenDraw` method returns in Listing 4 (p. 11), the `run` method:

- Adds the student's name to the snow scene picture.
- Displays the snow scene picture (see Figure 5 (p. 7) ).
- Displays information about the snow scene picture on the command line screen.

### The end of the program

Then the `run` method in Listing 4 (p. 11) returns control to the `main` method in Listing 1 (p. 8) causing the program to terminate.

## 6 Run the program

I encourage you to copy the code from Listing 6 (p. 15) . 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.

Click the following links to download the required input images:

1. Prob03a.bmp <sup>3</sup>
2. Prob03b.bmp <sup>4</sup>
3. Prob03c.bmp <sup>5</sup>
4. Prob03d.jpg <sup>6</sup>

## 7 Summary

In this module, you learned how to write a program to do *green-screen* processing to superimpose a source image onto a destination image while making the green background of the source image appear to be transparent.

## 8 What's next?

You will learn how to darken, brighten, and tint the colors in a `Picture` object in the next module.

<sup>3</sup><http://cnx.org/content/m44210/1.3/Prob03a.bmp>

<sup>4</sup><http://cnx.org/content/m44210/1.3/Prob03b.bmp>

<sup>5</sup><http://cnx.org/content/m44210/1.3/Prob03c.bmp>

<sup>6</sup><http://cnx.org/content/m44210/1.3/Prob03d.jpg>

## 9 Online video links

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

- ITSE 2321 Lecture 08 <sup>7</sup>
  - Part01 <sup>8</sup>
  - Part02 <sup>9</sup>
  - Part03 <sup>10</sup>

## 10 Miscellaneous

This section contains a variety of miscellaneous information.

NOTE: **Housekeeping material**

- Module name: Java OOP: Green-Screen Processing
- File: Java3016.htm
- Published: August 1, 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.

## 11 Complete program listing

A complete listing of the program discussed in this module is shown in Listing 6 (p. 15) below.

**Listing 6: Complete program listing.**

```
/*File Prob03 Copyright 2008 R.G.Baldwin
Revised 12/17/08
*****/
import java.awt.Color;
public class Prob03{
```

<sup>7</sup><http://www.youtube.com/playlist?list=PLF12CDA1C72ACC167>

<sup>8</sup><http://www.youtube.com/watch?v=EW6ZEDGJi2w>

<sup>9</sup>[http://www.youtube.com/watch?v=JqK\\_42UnXoI](http://www.youtube.com/watch?v=JqK_42UnXoI)

<sup>10</sup><http://www.youtube.com/watch?v=4bM6qElbxpc>

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

class Prob03Runner{

    public Prob03Runner(){ //constructor
        System.out.println("Display your name here.");
    } //end constructor
    //-----//

    public void run(){
        //Instantiate, display and crop the four input
        // images. They must be cropped to remove the Alice
        // runtime window material. The three skater images
        // are also cropped to remove excess blank green
        // background.

        //Note that the three views of the skater are bmp
        // images instead of jpg images in order to preserve
        // the pure green background color. Storing the
        // images as jpg files would corrupt the background
        // color in the low order bits.
        //A view facing the front of the skater.
        Picture front = new Picture("Prob03a.bmp");
        front.explore();
        front = crop(front,123,59,110,256);

        //A view showing the right side of the skater.
        Picture right = new Picture("Prob03b.bmp");
        right.explore();
        right = crop(right,123,59,110,256);

        //A view showing the left side of the skater.
        Picture left = new Picture("Prob03c.bmp");
        left.explore();
        left = crop(left,123,59,110,256);

        //This will be the background for the new picture.
        Picture snowScene = new Picture("Prob03d.jpg");
        snowScene.explore();
        snowScene = crop(snowScene,6,59,344,256);

        //Draw the front view of the skater on the snowScene
        // at full size.
        greenScreenDraw(front,snowScene,117,0);

        //Draw the left side view of the skater on the
```

```

// snowScene at half size.
left = left.getPictureWithHeight(256/2);
greenScreenDraw(left,snowScene,55,64);

//Draw the right side view of the skater on the
// snowScene at one-third size.
right = right.getPictureWithHeight(256/3);
greenScreenDraw(right,snowScene,260,96);

//Display students name on the final output and
// display it.
snowScene.addMessage("Display your name here.",10,15);
snowScene.explore();
System.out.println(snowScene);
} //end run method
//-----//

//Assumes a source image with a pure green background.
// Copies all non-green pixels from the source image to
// the destination image at the location explained
// below. Note that jpg images typically won't have
// a pure green background even if they had a pure
// green background before being compressed into the
// jpg format.  bmp images work well for this.
private void greenScreenDraw(
    Picture source,
    Picture dest,
    //Place the upper-left corner
    // of the source image at the
    // following location in the
    // destination image.
    int destX,
    int destY){
    int width = source.getWidth();
    int height = source.getHeight();
    Pixel pixel = null;
    Color color = null;

    for(int row = 0;row < height;row++){
        for(int col = 0;col < width;col++){
            color = source.getPixel(col,row).getColor();
            if(!(color.equals(Color.GREEN))){
                pixel = dest.getPixel(destX + col,destY + row);
                pixel.setColor(color);
            } //end if
        } //end inner loop
    } //end outer loop
} //end greenScreenDraw

//-----//

```

```
//Crops a Picture object to the given width and height
// with the upper-left corner located at startCol and
// startRow.
private Picture crop(Picture pic,int startCol,
                    int startRow,
                    int width,
                    int height){
    Picture output = new Picture(width,height);

    int colOut = 0;
    int rowOut = 0;
    int col = 0;
    int row = 0;
    Pixel pixel = null;
    Color color = null;
    for(col = startCol;col < startCol+width;col++){
        for(row = startRow;row < startRow+height;row++){
            color = pic.getPixel(col,row).getColor();
            pixel = output.getPixel(colOut,rowOut);
            pixel.setColor(color);
            rowOut++;
        }//end inner loop
        rowOut = 0;
        colOut++;
    }//end outer loop
    return output;
} //end crop

} //end class Prob03Runner

-end-
```