Java OOP: Background Color, Text Color, Mouse Clicks, etc.*

R.G. (Dick) Baldwin

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

Abstract

Learn how to modify the SimplePicture class to control the color of the text that is placed on the image in a Picture object. Place a turtle object in a world and cause it to draw a square spiral.

1 Table of Contents

- Preface (p. 1)
  - Viewing tip (p. 1)
    - Figures (p. 2)
    - Listings (p. 2)
- Preview (p. 2)
- Discussion and sample code (p. 6)
- Run the program (p. 11)
- Summary (p. 11)
- What's next? (p. 11)
- Miscellaneous (p. 11)
- Complete program listings (p. 12)

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: Aug 20, 2012 8:59 am -0500

†http://creativecommons.org/licenses/by/3.0/

[1] http://cnx.org/content/m44148/latest/

2.1.1 Figures

- Figure 1 (p. 3) . Output image at startup.
- Figure 2 (p. 4) . Output image after ten mouse clicks.
- Figure 3 (p. 5) . Output image after eleven mouse clicks.

2.1.2 Listings

- Listing 1 (p. 6) . The new setMessageColor method.
- Listing 2 (p. 7) . The modified addMessage method.
- Listing 3 (p. 7) . Beginning of the Prob04Runner class.
- Listing 4 (p. 8) . Beginning of the anonymous listener class.
- Listing 5 (p. 9) . Add a turtle to the world.
- Listing 6 (p. 9) . Not the first click.
- Listing 7 (p. 9) . Process odd or even clicks.
- Listing 8 (p. 10) . Cause the program to terminate properly.
- Listing 9 (p. 12) . Source code for the program named Prob04.
- Listing 10 (p. 13) . Modified World class.
- Listing 11 (p. 19) . Modified SimplePicture class.

3 Preview

The program that I will explain in this module requires you to modify both the **World** class and the **SimplePicture** class from Ericson's media library.

Just as you did in an earlier module, you will modify the **World** class to make it possible to get access to the **JFrame** object that is encapsulated in a **World** object.

You will modify the **SimplePicture** class to make it possible to control the color of the text that is placed on the image in a **Picture** object.

Program specifications

Write a program named **Prob04** that uses the class definition for the **Prob04** class shown in Listing 9 (p. 12) along with Ericson's media library to produce the graphic output images shown in Figure 1 (p. 3) , Figure 2 (p. 4) , and Figure 3 (p. 5) .

Output image at startup

Figure 1 (p. 3) shows the output image when you first start the program.

Output image at startup.

Figure 1: Output image at startup.

Output image after ten mouse clicks

This program adds a **JButton** object to the SOUTH location of the **World** object as shown in Figure 1 (p. 3) . Figure 2 (p. 4) shows the output image after you click button ten times.

Output image after ten mouse clicks.

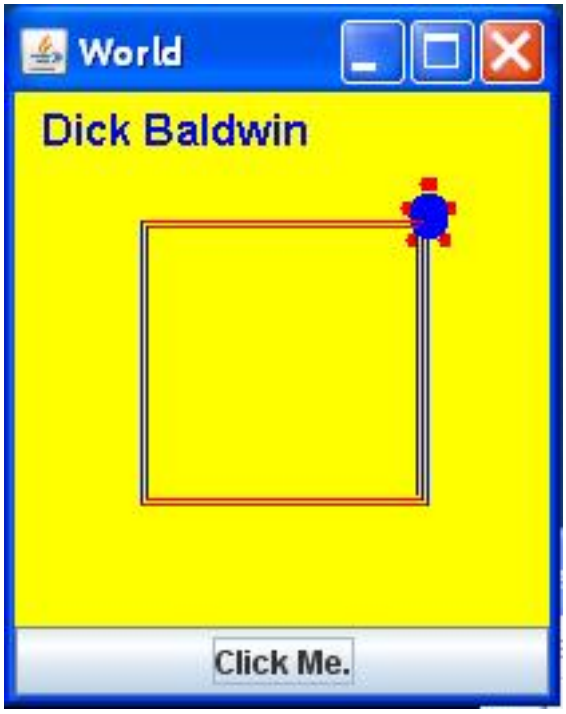

Figure 2: Output image after ten mouse clicks.

Output image after eleven mouse clicks

Figure 3 (p. 5) shows the output image after you click button eleven times.

Output image after eleven mouse clicks.

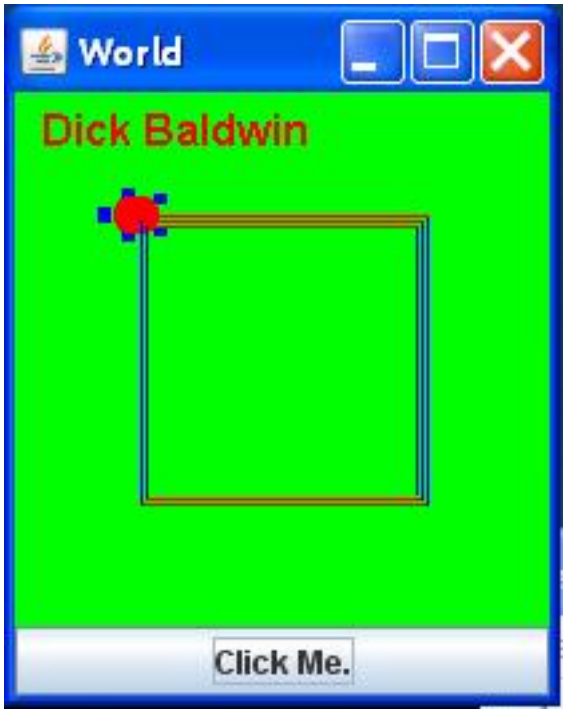

Figure 3: Output image after eleven mouse clicks.

Operational description

The program initially displays an empty white world with a button at the bottom as shown in Figure 1 (p. 3) .

When the user clicks the button:

- The world's background color changes to green.
- A turtle appears near the bottom right corner of the World.
- The student's name appears near the top left corner of the world in red.
- The turtle has a blue body and a red shell.

Click the button again

When you click the button the second time:

- The background color changes to yellow.
- The student's name changes to blue.
- The turtle changes to a red body with a blue shell.
- The turtle turns 90 degrees left and moves forward 100 pixels plus the value of a click counter.
- The turtle leaves a blue trail.

Click the button another time

On the next click:

- The colors revert to the color scheme with the yellow background.
- The turtle turns 90 degrees left and moves forward 100 pixels plus the value of the click counter leaving a red trail.

Click the button repeatedly

This cycle repeats on each click with the turtle's trail drawing a square spiral of increasing size with red lines on the top and bottom of the spiral and blue lines on the right and left of the spiral.

Output text

In addition to the output images described above, your program must produce some output text on the command- line screen

4 Discussion and sample code

The driver class

The driver class named **Prob04** is shown at the beginning of Listing 9 (p. 12) near the end of the module. The **main** method simply instantiates a new object of the class named **Prob04Runner** , which I will explain later. The event driven behavior of the program is controlled by a listener object that is registered on the button in the constructor of the **Prob04Runner** class.

Modification of the World class

This program adds a method named **getFrame** to the **World** class. The method returns a reference to the **JFrame** object that is used to display the world.

The program uses that reference to add a **JButton** object to the SOUTH location of the **World** . I explained a modification very similar to this in an earlier module, so I won't repeat that explanation here.

A complete listing of the modified **World** class is provided in Listing 10 (p. 13) near the end of the module.

Modification of the SimplePicture class

This program modifies the **addMessage** method of the **SimplePicture** class to cause it to use a color variable named **messageColor** to set the color of the text. The modification also declares and initializes the private instance variable named **messageColor** .

The **SimplePicture** class was also modified to include a **setMessageColor** method that can be used to set the color value stored in the variable named **messageColor** .

The new setMessageColor method

The new method named **setMessageColor** that was added to the **SimplePicture** class is shown in Listing 1 (p. 6) .

Listing 1: The new setMessageColor method.

```
    public void setMessageColor(Color color){
  System.out.println("Dick Baldwin");
  messageColor = color;
}//end setMessageColor
```

Save the color value

The **setMessageColor** method saves the incoming color parameter in a private instance variable named **messageColor** that was added to the class.

The default value of the variable is **Color.WHITE** , thereby preserving the default behavior of the **addMessage** method.

You can view the new variable named **messageColor** in Listing 11 (p. 19) near the end of the module.

Display the student's name

The new **setMessageColor** method also causes my name to be displayed each time the method is called. This is of no operational value, but is useful during the testing stage of the modified class. This is part of the code that produces text output on the command line screen.

The modified addMessage method

The modified version of the **addMessage** method is shown in Listing 2 (p. 7) .

Listing 2: The modified addMessage method.

```
  public void addMessage(
                  String message, int xPos, int yPos){
 // get a graphics context to use to draw on the
 // buffered image
 Graphics2D graphics2d = bufferedImage.createGraphics();

 // set the color to white
 //graphics2d.setPaint(Color.white);

 //modified by Baldwin on 12/23/08
 graphics2d.setPaint(messageColor);

 // set the font to Helvetica bold style and size 16
 graphics2d.setFont(new Font("Helvetica",Font.BOLD,16));

 // draw the message
 graphics2d.drawString(message,xPos,yPos);

}//end addMessage
```

The original statement shown in Listing 2 (p. 7) was disabled and replaced by the statement shown following the modification comment. This causes the text to be displayed on the image using the color stored in the new private instance variable named **messageColor** .

The Prob04Runner class

I will explain this program in fragments. A complete listing is shown in Listing 9 (p. 12) near the end of the module.

The driver class

The driver class for this program is named **Prob04** . As I mentioned earlier, you can view the class definition in its entirety near the beginning of Listing 9 (p. 12) .

Beginning of the Prob04Runner class

The **Prob04Runner** class begins in Listing 3 (p. 7) .

Listing 3: Beginning of the Prob04Runner class.

```
  class Prob04Runner{
 Turtle turtle = null;
 Picture picture = null;
 int counter = 0;
 World world = new World(200,200);
 JButton button = new JButton("Click Me.");

 public Prob04Runner(){
   System.out.println("Dick Baldwin");
   System.out.println(world.getPicture());

   //Get a reference to the JFrame object that is used
```

```
// to display the World.
JFrame frame = world.getFrame();

//Add the JButton object to the
// SOUTH location in the JFrame object.
frame.getContentPane().add(button,BorderLayout.SOUTH);

frame.pack();
```

Very familiar code

You should already be familiar with all of the code in Listing 3 (p. 7) . When the code in Listing 3 (p. 7) has finished executing, the image shown in Figure 1 (p. 3) should have appeared on the screen.

Waiting for an event

At this point, the program is essentially idle waiting for the user to either click the button at the bottom of Figure 1 (p. 3) , or click the red X-button in the upper-right corner of Figure 1 (p. 3) . *(More on the red X-button later.)*

Beginning of the anonymous listener class

This program uses an anonymous inner class to register an action listener on the button. The anonymous class begins in Listing 4 (p. 8) .

Listing 4: Beginning of the anonymous listener class.

```
  button.addActionListener(new ActionListener()
{//Begin the anonymous class definition

  public void actionPerformed(ActionEvent e){
    picture = world.getPicture();

    //Set picture background to green.
    picture.setAllPixelsToAColor(Color.GREEN);
    picture.setMessageColor(Color.RED);


    //Display the student's name on the picture.
    picture.addMessage("Dick Baldwin",10,20);
```

The event handler method named actionPerformed

Once the listener object is instantiated from the anonymous class and registered on the button, the method named **actionPerformed** , which begins in Listing 4 (p. 8) , will be executed each time the button is clicked.

Get a reference to the background picture

The **actionPerformed** method begins by getting and saving a reference to the **Picture** object that provides the background image for the **World** object. By default, all of the pixels in this image are white, as shown in Figure 1 (p. 3) .

Set the background color to green

Then the method calls Ericson's standard method named **setAllPixelsToAColor** method to set the color of all of the background pixels to green.

Display student's name in red

Following that, the method calls the new **setMessageColor** method to set the text color to red, and calls the modified **addMessage** method to display my name in red near the upper-left corner of the image. Figure 3 (p. 5) shows an example of a green background and red text.

Add a turtle to the world

Listing 5 (p. 9) tests to determine if the variable named **turtle** that was declared in Listing 3 (p. 7) still contains null. If so, that means that this is the first time that the button has been clicked and the **Turtle** object has not yet been added to the world.

Listing 5: Add a turtle to the world.

```
  if(turtle == null){
turtle = new Turtle(150,150,world);
turtle.setHeading(90);
turtle.setShellColor(Color.RED);
turtle.setBodyColor(Color.BLUE);
```

Add a Turtle object and set its colors

Listing 5 (p. 9) instantiates a new **Turtle** object and adds it to the world near the lower-right corner facing due east *(90 degrees)* .

The shell color is set to red and the body color *(feet and head)* is set to blue. An example of the turtle with this color scheme is shown in Figure 3 (p. 5) .

Not the first click

If the conditional clause in Listing 5 (p. 9) returns false, that means that this is not the first time the button has been clicked and the else clause, which begins in Listing 6 (p. 9) will be executed.

Listing 6: Not the first click.

```
  }else{
turtle.turnLeft();
turtle.forward(100+counter);
```

Rotate and move

The **else** clause begins by causing the turtle to rotate to the left by a default angle of 90 degrees. Then the turtle moves forward by a distance equal to 100 plus the value of a counter that is incremented by one each time the button is clicked.

For example, on the second click of the button, the turtle moves toward the north drawing a blue line along the way. The default width of the line is one pixel and the default color of the line is the same as the shell color.

Process odd or even clicks

The behavior of the **actionPerformed** method at this point depends on whether the incremented value of the **counter** variable is even or odd. The code to accomplish this is shown in Listing 7 (p. 9) .

Listing 7: Process odd or even clicks.

```
  if(counter++ %2 != 0){
picture.setAllPixelsToAColor(Color.GREEN);
picture.setMessageColor(Color.RED);

picture.addMessage(
           "Dick Baldwin",10,20);
turtle.setShellColor(Color.RED);
turtle.setBodyColor(Color.BLUE);
}else{
picture.setAllPixelsToAColor(Color.YELLOW);
```

```
              picture.setMessageColor(Color.BLUE);


              picture.addMessage(
                         "Dick Baldwin",10,20);
              turtle.setShellColor(Color.BLUE);
              turtle.setBodyColor(Color.RED);
            }//end else

            picture.addMessage(
                         "Dick Baldwin",10,20);
          }//end else

        }//end actionPerformed
      }//end class definition
    );//end addActionListener

  }//end constructor
  //-----------------------------------------------------//

}//end class Prob04Runner
```

If the counter value is odd...

If the value of the counter (*before it is incremented – note the post-increment operator)* is odd, the color scheme for the background, the message, and the turtle is set to that shown in Figure 3 (p. 5) with the green background.

If the counter value is even...

Otherwise, if the counter value is even, the color scheme is set to that shown in Figure 2 (p. 4) with the yellow background.

The end of several sections of code

Listing 7 (p. 9) also completes the **else** clause that began in Listing 6 (p. 9) .

In addition, Listing 7 (p. 9) signals the end of the **actionPerformed** method, the end of the anonymous class definition, the end of the constructor, and the end of the class named **Prob04Runner** .

Waiting for an event

As mentioned earlier, once the constructor finishes execution, the program becomes idle waiting for the user to either click the button at the bottom of Figure 1 (p. 3) , or click the red X-button in the upper-right corner of Figure 1 (p. 3) .

Does not terminate as expected

Normally a user would expect the program to terminate and return control to the operating system when the user clicks the red X-button in the upper-right corner of the last remaining window. However, this program does not do that. Instead, clicking this button simply hides the window and control is not returned to the operating system.

A programming oversight

This was a programming oversight on my part, which can be corrected by adding the second statement in Listing 8 (p. 10) to the definition of the **Prob04Runner** class immediately following the first statement shown in Listing 8 (p. 10) .

Listing 8: Cause the program to terminate properly.

```
    JFrame frame = world.getFrame();
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

To understand why this is necessary to cause the program to terminate, I recommend that you visit the standard Sun javadocs and examine the description of the method named **setDefaultCloseOperation** in the **JFrame** class.

5 Run the program

I encourage you to copy the code from Listing 9 (p. 12) , Listing 10 (p. 13) , and Listing 11 (p. 19) . 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.

6 Summary

Just as you did in an earlier module, you modified the **World** class to make it possible to get access to the **JFrame** object that is encapsulated in a **World** object.

You learned how to modify the **SimplePicture** class to make it possible to control the color of the text that is placed on the image in a **Picture** object.

Then you placed a turtle object in a world and performed a series of maneuvers causing the turtle to draw a square spiral.

7 What's next?

In the next module, you will learn how to create and service a graphical user interface containing panels, labels, text fields, and buttons.

8 Miscellaneous

This section contains a variety of miscellaneous information.

NOTE: **Housekeeping material**

- Module name: Java OOP: Background Color, Text Color, Mouse Clicks, etc.
- File: Java3108.htm
- Revised: 08/20/12

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.

9 Complete program listings

Complete listings of the programs discussed in this module are shown below.

Listing 9: Source code for the program named Prob04.

```
    /*File Prob04 Copyright 2008 R.G.Baldwin
*******************************************************/
import java.awt.BorderLayout;
import javax.swing.JFrame;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.Color;
import java.awt.Toolkit;

public class Prob04{
  //DO NOT MODIFY THE CODE IN THIS CLASS DEFINITION.
  public static void main(String[] args){
    new Prob04Runner();
  }//end main method
}//end class Prob04
//=====================================================//

class Prob04Runner{
  Turtle turtle = null;
  Picture picture = null;
  int counter = 0;
  World world = new World(200,200);
  JButton button = new JButton("Click Me.");

  public Prob04Runner(){
    System.out.println("Dick Baldwin");
    System.out.println(world.getPicture());

    //Get a reference to the JFrame object that is used
    // to display the World.
    JFrame frame = world.getFrame();

    //Add the JButton object to the
    // SOUTH location in the JFrame object.
    frame.getContentPane().add(button,BorderLayout.SOUTH);

    frame.pack();

    //Use an anonymous class to register an action
    // listener on the button. Note that the student is
    // not required to use an anonymous class.
    button.addActionListener(new ActionListener()
      {//Begin the class definition
        public void actionPerformed(ActionEvent e){
```

```
          picture = world.getPicture();

          //Set picture background to green.
          picture.setAllPixelsToAColor(Color.GREEN);
          picture.setMessageColor(Color.RED);

          //Display the student's name on the picture.
          picture.addMessage(
                         "Dick Baldwin",10,20);
          //Add a turtle to the world. This causes the
          // world to be repainted.
          if(turtle == null){
            turtle = new Turtle(150,150,world);
            turtle.setHeading(90);
            turtle.setShellColor(Color.RED);
            turtle.setBodyColor(Color.BLUE);
          }else{
            turtle.turnLeft();
            turtle.forward(100+counter);
            if(counter++ %2 != 0){
              picture.setAllPixelsToAColor(Color.GREEN);
              picture.setMessageColor(Color.RED);

              picture.addMessage(
                         "Dick Baldwin",10,20);
              turtle.setShellColor(Color.RED);
              turtle.setBodyColor(Color.BLUE);
            }else{
              picture.setAllPixelsToAColor(Color.YELLOW);
              picture.setMessageColor(Color.BLUE);


              picture.addMessage(
                         "Dick Baldwin",10,20);
              turtle.setShellColor(Color.BLUE);
              turtle.setBodyColor(Color.RED);
            }//end else
            picture.addMessage(
                         "Dick Baldwin",10,20);
          }//end else
        }//end actionPerformed
      }//end class definition
    );//end addActionListener

  }//end constructor
  //-----------------------------------------------------//

}//end class Prob04Runner
```

Listing 10: Modified World class.

```
    import javax.swing.*;
import java.util.List;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Observer;
import java.awt.*;

/*12/23/08 Modified the World class. Added a method named
 *getFrame that returns a reference to the JFrame object
 *in which the turtles are displayed.
 */

/**
 * Class to represent a 2d world that can hold turtles and
 * display them
 *
 * Copyright Georgia Institute of Technology 2004
 * @author Barb Ericson ericson@cc.gatech.edu
 */
public class World extends JComponent implements ModelDisplay
{
  ///////////////// fields ///////////////////////

  /** should automatically repaint when model changed */
  private boolean autoRepaint = true;

  /** the background color for the world */
  private Color background = Color.white;

  /** the width of the world */
  private int width = 640;

  /** the height of the world */
  private int height = 480;

  /** the list of turtles in the world */
  private List<Turtle> turtleList = new ArrayList<Turtle>();

  /** the JFrame to show this world in */
  private JFrame frame = new JFrame("World");

  /** background picture */
  private Picture picture = null;

  ///////////////// the constructors ///////////////

  /**
   * Constructor that takes no arguments
   */
  public World()
  {
```

```
    // set up the world and make it visible
    initWorld(true);
  }

  /**
   * Constructor that takes a boolean to
   * say if this world should be visible
   * or not
   * @param visibleFlag if true will be visible
   * else if false will not be visible
   */
  public World(boolean visibleFlag)
  {
    initWorld(visibleFlag);
  }

  /**
   * Constructor that takes a width and height for this
   * world
   * @param w the width for the world
   * @param h the height for the world
   */
  public World(int w, int h)
  {
    width = w;
    height = h;

    // set up the world and make it visible
    initWorld(true);
  }

  ///////////////// methods ///////////////////////////
  /**
   *Method to return a reference to the JFrame.
   */
  public JFrame getFrame(){
    System.out.println("Dick Baldwin");
    return frame;
  }//end getFrame

  /**
   * Method to initialize the world
   * @param visibleFlag the flag to make the world
   * visible or not
   */
  private void initWorld(boolean visibleFlag)
  {
    // set the preferred size
    this.setPreferredSize(new Dimension(width,height));

    // create the background picture
```

```
    picture = new Picture(width,height);

    // add this panel to the frame
    frame.getContentPane().add(this);

    // pack the frame
    frame.pack();

    // show this world
    frame.setVisible(visibleFlag);
  }

  /**
   * Method to get the graphics context for drawing on
   * @return the graphics context of the background picture
   */
  public Graphics getGraphics() { return picture.getGraphics(); }

  /**
   * Method to clear the background picture
   */
  public void clearBackground() { picture = new Picture(width,height); }

  /**
   * Method to get the background picture
   * @return the background picture
   */
  public Picture getPicture() { return picture; }

  /**
   * Method to set the background picture
   * @param pict the background picture to use
   */
  public void setPicture(Picture pict) { picture = pict; }

  /**
   * Method to paint this component
   * @param g the graphics context
   */
  public synchronized void paintComponent(Graphics g)
  {
    Turtle turtle = null;

    // draw the background image
    g.drawImage(picture.getImage(),0,0,null);

    // loop drawing each turtle on the background image
    Iterator iterator = turtleList.iterator();
    while (iterator.hasNext())
    {
      turtle = (Turtle) iterator.next();
```

```
      turtle.paintComponent(g);
    }
  }

  /**
   * Metod to get the last turtle in this world
   * @return the last turtle added to this world
   */
  public Turtle getLastTurtle()
  {
    return (Turtle) turtleList.get(turtleList.size() - 1);
  }


  /**
   * Method to add a model to this model displayer
   * @param model the model object to add
   */
  public void addModel(Object model)
  {
    turtleList.add((Turtle) model);
    if (autoRepaint)
       repaint();
  }

  /**
   * Method to check if this world contains the passed
   * turtle
   * @return true if there else false
   */
  public boolean containsTurtle(Turtle turtle)
  {
    return (turtleList.contains(turtle));
  }

  /**
   * Method to remove the passed object from the world
   * @param model the model object to remove
   */
  public void remove(Object model)
  {
    turtleList.remove(model);
  }

  /**
   * Method to get the width in pixels
   * @return the width in pixels
   */
  public int getWidth() { return width; }

  /**
```

```
   * Method to get the height in pixels
   * @return the height in pixels
   */
  public int getHeight() { return height; }

  /**
   * Method that allows the model to notify the display
   */
  public void modelChanged()
  {
    if (autoRepaint)
       repaint();
  }

  /**
   * Method to set the automatically repaint flag
   * @param value if true will auto repaint
   */
  public void setAutoRepaint(boolean value) { autoRepaint = value; }

  /**
   * Method to hide the frame
   */
//  public void hide()
//  {
//    frame.setVisible(false);
//  }

  /**
   * Method to show the frame
   */
//  public void show()
//  {
//    frame.setVisible(true);
//  }

  /**
   * Method to set the visibility of the world
   * @param value a boolean value to say if should show or hide
   */
  public void setVisible(boolean value)
  {
    frame.setVisible(value);
  }

  /**
   * Method to get the list of turtles in the world
   * @return a list of turtles in the world
   */
  public List getTurtleList()
  { return turtleList;}
```

```
  /**
   * Method to get an iterator on the list of turtles
   * @return an iterator for the list of turtles
   */
  public Iterator getTurtleIterator()
  { return turtleList.iterator();}

  /**
   * Method that returns information about this world
   * in the form of a string
   * @return a string of information about this world
   */
  public String toString()
  {
    return "A " + getWidth() + " by " + getHeight() +
      " world with " + turtleList.size() + " turtles in it.";
  }

} // end of World class
```

Listing 11: Modified SimplePicture class.

```
    import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import javax.swing.ImageIcon;
import java.awt.*;
import java.io.*;
import java.awt.geom.*;

/*
12/23/08 Modified the addMessage method to cause it to
use a color variable to set the color of the message.
Also provided a setMessageColor method to set the color
and a variable named messageColor to contain the color.
/*


/**
 * A class that represents a simple picture.  A simple picture may have
 * an associated file name and a title.  A simple picture has pixels,
 * width, and height.  A simple picture uses a BufferedImage to
 * hold the pixels.  You can show a simple picture in a
 * PictureFrame (a JFrame).
 *
 * Copyright Georgia Institute of Technology 2004
 * @author Barb Ericson ericson@cc.gatech.edu
 */
public class SimplePicture implements DigitalPicture
{
```

```
  ////////////////////// Fields ////////////////////////
  /**
   * the color of the message
   */
   private Color messageColor = Color.WHITE;


  /**
   * the file name associated with the simple picture
   */
  private String fileName;

  /**
   * the title of the simple picture
   */
  private String title;

  /**
   * buffered image to hold pixels for the simple picture
   */
  private BufferedImage bufferedImage;

  /**
   * frame used to display the simple picture
   */
  private PictureFrame pictureFrame;

  /**
   * extension for this file (jpg or bmp)
   */
  private String extension;


 ////////////////////// Constructors ////////////////////////

 /**
  * A Constructor that takes no arguments.  All fields will be null.
  * A no-argument constructor must be given in order for a class to
  * be able to be subclassed.  By default all subclasses will implicitly
  * call this in their parent's no argument constructor unless a
  * different call to super() is explicitly made as the first line
  * of code in a constructor.
  */
 public SimplePicture()
 {this(200,100);}

 /**
  * A Constructor that takes a file name and uses the file to create
  * a picture
  * @param fileName the file name to use in creating the picture
```

```
 */
public SimplePicture(String fileName)
{

  // load the picture into the buffered image
  load(fileName);

}

/**
 * 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.
 * @param width the desired width
 * @param height the desired height
 */
public  SimplePicture(int width, int height)
{
  bufferedImage = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);
  title = "None";
  fileName = "None";
  extension = "jpg";
  setAllPixelsToAColor(Color.white);
}

/**
 * A constructor that takes the width and height desired for a picture and
 * creates a buffered image of that size.  It also takes the
 * color to use for the background of the picture.
 * @param width the desired width
 * @param height the desired height
 * @param theColor the background color for the picture
 */
public  SimplePicture(int width, int height, Color theColor)
{
  this(width,height);
  setAllPixelsToAColor(theColor);
}

/**
 * A Constructor that takes a picture to copy information from
 * @param copyPicture the picture to copy from
 */
public SimplePicture(SimplePicture copyPicture)
{
  if (copyPicture.fileName != null)
  {
     this.fileName = new String(copyPicture.fileName);
     this.extension = copyPicture.extension;
  }
  if (copyPicture.title != null)
```

```
      this.title = new String(copyPicture.title);
   if (copyPicture.bufferedImage != null)
   {
     this.bufferedImage = new BufferedImage(copyPicture.getWidth(),
                                            copyPicture.getHeight(), BufferedImage.TYPE_INT_RGB);
     this.copyPicture(copyPicture);
   }
 }

 /**
  * A constructor that takes a buffered image
  * @param image the buffered image
  */
 public SimplePicture(BufferedImage image)
 {
   this.bufferedImage = image;
   title = "None";
   fileName = "None";
   extension = "jpg";
 }

 ////////////////////////// Methods ////////////////////////////////
 /**
  * Method to set the color used for a message.
  */
  public void setMessageColor(Color color){
    System.out.println("Dick Baldwin");
    messageColor = color;
  }//end setMessageColor


 /**
  * Method to get the extension for this picture
  * @return the extendsion (jpg or bmp)
  */
 public String getExtension() { return extension; }


 /**
  * Method that will copy all of the passed source picture into
  * the current picture object
  * @param sourcePicture  the picture object to copy
  */
 public void copyPicture(SimplePicture sourcePicture)
 {
   Pixel sourcePixel = null;
   Pixel targetPixel = null;

   // loop through the columns
   for (int sourceX = 0, targetX = 0;
        sourceX < sourcePicture.getWidth() &&
```

```
        targetX < this.getWidth();
        sourceX++, targetX++)
  {
    // loop through the rows
    for (int sourceY = 0, targetY = 0;
         sourceY < sourcePicture.getHeight() &&
         targetY < this.getHeight();
         sourceY++, targetY++)
    {
      sourcePixel = sourcePicture.getPixel(sourceX,sourceY);
      targetPixel = this.getPixel(targetX,targetY);
      targetPixel.setColor(sourcePixel.getColor());
    }
  }

}

/**
 * Method to set the color in the picture to the passed color
 * @param color the color to set to
 */
public void setAllPixelsToAColor(Color color)
{
  // loop through all x
  for (int x = 0; x < this.getWidth(); x++)
  {
    // loop through all y
    for (int y = 0; y < this.getHeight(); y++)
    {
      getPixel(x,y).setColor(color);
    }
  }
}

/**
 * Method to get the buffered image
 * @return the buffered image
 */
public BufferedImage getBufferedImage()
{
   return bufferedImage;
}

/**
 * Method to get a graphics object for this picture to use to draw on
 * @return a graphics object to use for drawing
 */
public Graphics getGraphics()
{
  return bufferedImage.getGraphics();
}
```

```
/**
 * Method to get a Graphics2D object for this picture which can
 * be used to do 2D drawing on the picture
 */
public Graphics2D createGraphics()
{
  return bufferedImage.createGraphics();
}

/**
 * Method to get the file name associated with the picture
 * @return  the file name associated with the picture
 */
public String getFileName() { return fileName; }

/**
 * Method to set the file name
 * @param name the full pathname of the file
 */
public void setFileName(String name)
{
  fileName = name;
}

/**
 * Method to get the title of the picture
 * @return the title of the picture
 */
public String getTitle()
{ return title; }

/**
 * Method to set the title for the picture
 * @param title the title to use for the picture
 */
public void setTitle(String title)
{
  this.title = title;
  if (pictureFrame != null)
      pictureFrame.setTitle(title);
}

/**
 * Method to get the width of the picture in pixels
 * @return the width of the picture in pixels
 */
public int getWidth() { return bufferedImage.getWidth(); }

/**
 * Method to get the height of the picture in pixels
```

```
 * @return  the height of the picture in pixels
 */
public int getHeight() { return bufferedImage.getHeight(); }

/**
 * Method to get the picture frame for the picture
 * @return the picture frame associated with this picture
 * (it may be null)
 */
public PictureFrame getPictureFrame() { return pictureFrame; }

/**
 * Method to set the picture frame for this picture
 * @param pictureFrame the picture frame to use
 */
public void setPictureFrame(PictureFrame pictureFrame)
{
  // set this picture objects' picture frame to the passed one
  this.pictureFrame = pictureFrame;
}

/**
 * Method to get an image from the picture
 * @return  the buffered image since it is an image
 */
public Image getImage()
{
  return bufferedImage;
}

/**
 * Method to return the pixel value as an int for the given x and y location
 * @param x the x coordinate of the pixel
 * @param y the y coordinate of the pixel
 * @return the pixel value as an integer (alpha, red, green, blue)
 */
public int getBasicPixel(int x, int y)
{
   return bufferedImage.getRGB(x,y);
}

/**
 * Method to set the value of a pixel in the picture from an int
 * @param x the x coordinate of the pixel
 * @param y the y coordinate of the pixel
 * @param rgb the new rgb value of the pixel (alpha, red, green, blue)
 */
public void setBasicPixel(int x, int y, int rgb)
{
  bufferedImage.setRGB(x,y,rgb);
}
```

```
/**
 * Method to get a pixel object for the given x and y location
 * @param x  the x location of the pixel in the picture
 * @param y  the y location of the pixel in the picture
 * @return a Pixel object for this location
 */
public Pixel getPixel(int x, int y)
{
  // create the pixel object for this picture and the given x and y location
  Pixel pixel = new Pixel(this,x,y);
  return pixel;
}

/**
 * Method to get a one-dimensional array of Pixels for this simple picture
 * @return a one-dimensional array of Pixel objects starting with y=0
 * to y=height-1 and x=0 to x=width-1.
 */
public Pixel[] getPixels()
{
  int width = getWidth();
  int height = getHeight();
  Pixel[] pixelArray = new Pixel[width * height];

  // loop through height rows from top to bottom
  for (int row = 0; row < height; row++)
    for (int col = 0; col < width; col++)
      pixelArray[row * width + col] = new Pixel(this,col,row);

  return pixelArray;
}



/**
 * Method to load the buffered image with the passed image
 * @param image  the image to use
 */
public void load(Image image)
{
  // get a graphics context to use to draw on the buffered image
  Graphics2D graphics2d = bufferedImage.createGraphics();

  // draw the image on the buffered image starting at 0,0
  graphics2d.drawImage(image,0,0,null);

  // show the new image
  show();
}
```

```
/**
 * Method to show the picture in a picture frame
 */
public void show()
{
   // if there is a current picture frame then use it
  if (pictureFrame != null)
    pictureFrame.updateImageAndShowIt();

  // else create a new picture frame with this picture
  else
    pictureFrame = new PictureFrame(this);
}

/**
 * Method to hide the picture
 */
public void hide()
{
  if (pictureFrame != null)
    pictureFrame.setVisible(false);
}

/**
 * Method to make this picture visible or not
 * @param flag true if you want it visible else false
 */
public void setVisible(boolean flag)
{
  if (flag)
    this.show();
  else
    this.hide();
}

/**
 * Method to open a picture explorer on a copy of this simple picture
 */
public void explore()
{
  // create a copy of the current picture and explore it
  new PictureExplorer(new SimplePicture(this));
}

/**
 * Method to force the picture to redraw itself.  This is very
 * useful after you have changed the pixels in a picture.
 */
public void repaint()
{
  // if there is a picture frame tell it to repaint
```

```
  if (pictureFrame != null)
    pictureFrame.repaint();

  // else create a new picture frame
  else
    pictureFrame = new PictureFrame(this);
}

/**
 * Method to load the picture from the passed file name
 * @param fileName the file name to use to load the picture from
 */
public void loadOrFail(String fileName) throws IOException
{
   // set the current picture's file name
  this.fileName = fileName;

  // set the extension
  int posDot = fileName.indexOf('.');
  if (posDot >= 0)
    this.extension = fileName.substring(posDot + 1);

  // if the current title is null use the file name
  if (title == null)
    title = fileName;

  File file = new File(this.fileName);

  if (!file.canRead())
  {
    // try adding the media path
    file = new File(FileChooser.getMediaPath(this.fileName));
    if (!file.canRead())
    {
      throw new IOException(this.fileName +
                            " could not be opened. Check that you specified the path");
    }
  }

  bufferedImage = ImageIO.read(file);
}


/**
 * Method to write the contents of the picture to a file with
 * the passed name without throwing errors
 * @param fileName the name of the file to write the picture to
 * @return true if success else false
 */
public boolean load(String fileName)
{
```

```
    try {
        this.loadOrFail(fileName);
        return true;

    } catch (Exception ex) {
        System.out.println("There was an error trying to open " + fileName);
        bufferedImage = new BufferedImage(600,200,
                                          BufferedImage.TYPE_INT_RGB);
        addMessage("Couldn't load " + fileName,5,100);
        return false;
    }

 }


 /**
  * Method to load the picture from the passed file name
  * this just calls load(fileName) and is for name compatibility
  * @param fileName the file name to use to load the picture from
  * @return true if success else false
  */
 public boolean loadImage(String fileName)
 {
     return load(fileName);
}

 /**
  * Method to draw a message as a string on the buffered image
  * @param message the message to draw on the buffered image
  * @param xPos  the leftmost point of the string in x
  * @param yPos  the bottom of the string in y
  */
 public void addMessage(String message, int xPos, int yPos)
 {
   // get a graphics context to use to draw on the buffered image
   Graphics2D graphics2d = bufferedImage.createGraphics();

   // set the color to white
//   graphics2d.setPaint(Color.white);
   //modified by baldwin on 12/23/08
   graphics2d.setPaint(messageColor);

   // set the font to Helvetica bold style and size 16
   graphics2d.setFont(new Font("Helvetica",Font.BOLD,16));

   // draw the message
   graphics2d.drawString(message,xPos,yPos);

 }

 /**
```

```
 * Method to draw a string at the given location on the picture
 * @param text the text to draw
 * @param xPos the left x for the text
 * @param yPos the top y for the text
 */
public void drawString(String text, int xPos, int yPos)
{
  addMessage(text,xPos,yPos);
}

/**
  * Method to create a new picture by scaling the current
  * picture by the given x and y factors
  * @param xFactor the amount to scale in x
  * @param yFactor the amount to scale in y
  * @return the resulting picture
  */
 public Picture scale(double xFactor, double yFactor)
 {
   // set up the scale tranform
   AffineTransform scaleTransform = new AffineTransform();
   scaleTransform.scale(xFactor,yFactor);

   // create a new picture object that is the right size
   Picture result = new Picture((int) (getWidth() * xFactor),
                                (int) (getHeight() * yFactor));

   // get the graphics 2d object to draw on the result
   Graphics graphics = result.getGraphics();
   Graphics2D g2 = (Graphics2D) graphics;

   // draw the current image onto the result image scaled
   g2.drawImage(this.getImage(),scaleTransform,null);

   return result;
 }

 /**
  * Method to create a new picture of the passed width.
  * The aspect ratio of the width and height will stay
  * the same.
  * @param width the desired width
  * @return the resulting picture
  */
 public Picture getPictureWithWidth(int width)
 {
   // set up the scale tranform
   double xFactor = (double) width / this.getWidth();
   Picture result = scale(xFactor,xFactor);
   return result;
 }
```

```
 /**
  * Method to create a new picture of the passed height.
  * The aspect ratio of the width and height will stay
  * the same.
  * @param height the desired height
  * @return the resulting picture
  */
 public Picture getPictureWithHeight(int height)
 {
   // set up the scale tranform
   double yFactor = (double) height / this.getHeight();
   Picture result = scale(yFactor,yFactor);
   return result;
 }

/**
 * Method to load a picture from a file name and show it in a picture frame
 * @param fileName the file name to load the picture from
 * @return true if success else false
 */
public boolean loadPictureAndShowIt(String fileName)
{
  boolean result = true;  // the default is that it worked

  // try to load the picture into the buffered image from the file name
  result = load(fileName);

  // show the picture in a picture frame
  show();

  return result;
}

/**
 * Method to write the contents of the picture to a file with
 * the passed name
 * @param fileName the name of the file to write the picture to
 */
public void writeOrFail(String fileName) throws IOException
{
  String extension = this.extension; // the default is current

  // create the file object
  File file = new File(fileName);
  File fileLoc = file.getParentFile();

  // canWrite is true only when the file exists already! (alexr)
  if (!fileLoc.canWrite()) {
      // System.err.println("can't write the file but trying anyway? ...");
       throw new IOException(fileName +
```

```
        " could not be opened. Check to see if you can write to the directory.");
  }

  // get the extension
  int posDot = fileName.indexOf('.');
  if (posDot >= 0)
      extension = fileName.substring(posDot + 1);

  // write the contents of the buffered image to the file as jpeg
  ImageIO.write(bufferedImage, extension, file);

}

/**
 * Method to write the contents of the picture to a file with
 * the passed name without throwing errors
 * @param fileName the name of the file to write the picture to
 * @return true if success else false
 */
public boolean write(String fileName)
{
    try {
        this.writeOrFail(fileName);
        return true;
    } catch (Exception ex) {
        System.out.println("There was an error trying to write " + fileName);
        return false;
    }

}

/**
 * Method to set the media path by setting the directory to use
 * @param directory the directory to use for the media path
 */
public static void setMediaPath(String directory) {
  FileChooser.setMediaPath(directory);
}

/**
 * Method to get the directory for the media
 * @param fileName the base file name to use
 * @return the full path name by appending
 * the file name to the media directory
 */
public static String getMediaPath(String fileName) {
  return FileChooser.getMediaPath(fileName);
}

 /**
  * Method to get the coordinates of the enclosing rectangle after this
```

```
   * transformation is applied to the current picture
   * @return the enclosing rectangle
   */
  public Rectangle2D getTransformEnclosingRect(AffineTransform trans)
  {
    int width = getWidth();
    int height = getHeight();
    double maxX = width - 1;
    double maxY = height - 1;
    double minX, minY;
    Point2D.Double p1 = new Point2D.Double(0,0);
    Point2D.Double p2 = new Point2D.Double(maxX,0);
    Point2D.Double p3 = new Point2D.Double(maxX,maxY);
    Point2D.Double p4 = new Point2D.Double(0,maxY);
    Point2D.Double result = new Point2D.Double(0,0);
    Rectangle2D.Double rect = null;

    // get the new points and min x and y and max x and y
    trans.deltaTransform(p1,result);
    minX = result.getX();
    maxX = result.getX();
    minY = result.getY();
    maxY = result.getY();
    trans.deltaTransform(p2,result);
    minX = Math.min(minX,result.getX());
    maxX = Math.max(maxX,result.getX());
    minY = Math.min(minY,result.getY());
    maxY = Math.max(maxY,result.getY());
    trans.deltaTransform(p3,result);
    minX = Math.min(minX,result.getX());
    maxX = Math.max(maxX,result.getX());
    minY = Math.min(minY,result.getY());
    maxY = Math.max(maxY,result.getY());
    trans.deltaTransform(p4,result);
    minX = Math.min(minX,result.getX());
    maxX = Math.max(maxX,result.getX());
    minY = Math.min(minY,result.getY());
    maxY = Math.max(maxY,result.getY());

    // create the bounding rectangle to return
    rect = new Rectangle2D.Double(minX,minY,maxX - minX + 1, maxY - minY + 1);
    return rect;
  }

 /**
  * Method to return a string with information about this picture
  * @return a string with information about the picture
  */
 public String toString()
 {
   String output = "Simple Picture, filename " + fileName +
```

```
     " height " + getHeight() + " width " + getWidth();
   return output;
 }

} // end of SimplePicture class
```

-end-