Using Flood-Fill in Java Programs

Published: March 19, 2009, **Incomplete**
By Richard G. Baldwin

Java Programming Notes # 365

Note: The text for this lesson is incomplete. However, I am publishing the source code for the program in case you find it useful.

Resources

- Creative Commons Attribution 3.0 United States License
- Media Computation book in Java - numerous downloads available
- Introduction to Computing and Programming with Java: A Multimedia Approach
- DrJava download site
- DrJava, the JavaPLT group at Rice University
- DrJava Open Source License
- The Essence of OOP using Java, The this and super Keywords
- Threads of Control
- Painting in AWT and Swing
- Wikipedia Turtle Graphics
- IsA or HasA
- Vector Cad-Cam XI Lathe Tutorial
- Classification of 3D to 2D projections
- Color model from Wikipedia
- Light and color: an introduction by Norman Koren
- Color Principles - Hue, Saturation, and Value
- 200 Implementing the Model-View-Controller Paradigm using Observer and Observable
- 300 Java 2D Graphics, Nested Top-Level Classes and Interfaces
- 302 Java 2D Graphics, The Point2D Class
- 304 Java 2D Graphics, The Graphics2D Class
- 306 Java 2D Graphics, Simple Affine Transforms
- 308 Java 2D Graphics, The Shape Interface, Part 1
- 310 Java 2D Graphics, The Shape Interface, Part 2
- 312 Java 2D Graphics, Solid Color Fill
- 314 Java 2D Graphics, Gradient Color Fill
- 316 Java 2D Graphics, Texture Fill
- 318 Java 2D Graphics, The Stroke Interface
- 320 Java 2D Graphics, The Composite Interface and Transparency
- 322 Java 2D Graphics, The Composite Interface, GradientPaint, and Transparency
- 324 Java 2D Graphics, The Color Constructors and Transparency

- 400 Processing Image Pixels using Java, Getting Started
  402 Processing Image Pixels using Java, Creating a Spotlight
  404 Processing Image Pixels Using Java: Controlling Contrast and Brightness
  406 Processing Image Pixels, Color Intensity, Color Filtering, and Color Inversion
  408 Processing Image Pixels, Performing Convolution on Images
  410 Processing Image Pixels, Understanding Image Convolution in Java
  412 Processing Image Pixels, Applying Image Convolution in Java, Part 1
  414 Processing Image Pixels, Applying Image Convolution in Java, Part 2
  416 Processing Image Pixels, An Improved Image-Processing Framework in Java
  418 Processing Image Pixels, Creating Visible Watermarks in Java
  450 A Framework for Experimenting with Java 2D Image-Processing Filters
  452 Using the Java 2D LookupOp Filter Class to Process Images
  454 Using the Java 2D AffineTransformOp Filter Class to Process Images
  456 Using the Java 2D LookupOp Filter Class to Scramble and Unscramble Images
  458 Using the Java 2D BandCombineOp Filter Class to Process Images
  460 Using the Java 2D ConvolveOp Filter Class to Process Images
  462 Using the Java 2D ColorConvertOp and RescaleOp Filter Classes to Process Images
- 506 JavaBeans, Introspection
- 2100 Understanding Properties in Java and C#
- 2300 Generics in J2SE, Getting Started
- 340 Multimedia Programming with Java, Getting Started
- 342 Getting Started with the Turtle Class: Multimedia Programming with Java
- 344 Continuing with the SimpleTurtle Class: Multimedia Programming with Java
- 346 Wrapping Up the SimpleTurtle Class: Multimedia Programming with Java
- 348 The Pen and PathSegment Classes: Multimedia Programming with Java
- 349 A Pixel Editor Program in Java: Multimedia Programming with Java
- 350 3D Displays, Color Distance, and Edge Detection
- 351 A Slider-Controlled Softening Program for Digital Photos
- 352 Adding Animated Movement to Your Java Application
- 353 A Slider-Controlled Sharpening Program for Digital Photos
- 354 The DigitalPicture Interface
- 355 The HSB Color Model
- 356 The show Method and the PictureFrame Class
- 357 An HSB Color-Editing Program for Digital Photos
- 358 Applying Affine Transforms to Picture Objects
- 359 Creating a lasso for editing digital photos in Java
- 360 Wrapping Up the SimplePicture Class
- 361 A Temperature and Tint Editing Program for Digital Photos
- 362 Getting Started with the PictureExplorer Class
- 363 Redeye Correction in Digital Photographs
- 364 Building the Information Panel for the PictureExplorer GUI

Complete program listing

A complete listings of the program is shown in Listing 1 below.

Listing 1. Source code for the program named FloodFill01.

```
/*File FloodFill01 Copyright 2009 R.G.Baldwin

01/24/09 This program is working very well. All the comments need to be
updated. Also, the code needs to be reviewed for clarity, descriptive
names, etc.

This program makes it possible for the user to modify the colors in a
picture using a flood fill algorithm.

The user adjusts sliders to specify the following and then clicks a Flood
Fill button:

Matching tolerances for hue, saturation, and brightness.
Values for hue, saturation, and brightness to specify a new color.

When the user clicks the Flood Fill button, the program
uses a flood fill algorithm to find all of the pixels that
are connected to the selected pixel, and all the pixels that are connected
to those pixels for which the color
matches the color of the selected pixel to withing the
specified tolerances. Then the program changes the color of all of those
pixels to the new color.

The user can repeat this process, specifying new pixels, tolerances, and
colors until the colors of all of the pixels in the picture have been
changed.

The program maintains two copies of the picture: an active picture and a
committed picture. Clicking the Commit button copies the active picture
into the committed
picture. The program also provides an Undo button. Clicking the Undo button
undoes all of the operations subsequent back to the most recent commit
operation. Once a picture
is committed, it cannot be undone.

When the user moves the knob on the New Hue, New Saturation, or New
Brightness sliders, the background color of all three sliders changes to
reflect the new color.
The text color on those three sliders is the color on the opposite side of
the color wheel from the hue value with full saturation and full
brightness. The purpose
of changing the text color is to cause the text to always be visible
regardless of the color of the background.

This program capitalizes on the availability of the
PictureExplorer class released under a Creative Commons
Attribution 3.0 United States License by Barb Ericson
at Georgia Institute of Technology.

Ericson's PictureExplorer class was
modified to make certain internals of PictureExplorer
```

```
objects accessible to objects instantiated from other
classes. Then a Java GUI was written that makes it
possible to open an image file in a PictureExplorer
object and process the pixels in the image as described earlier.

Specify an input image file using only the file name and
extension if the file is in the current directory.
Specify the full path to the file if it is not in the
current directory.

Both jpg and bmp file types are supported as input files.

The program starts with a GUI in the upper-left corner of
the screen. At that point in time, all of the user input controls are
disabled except for a text field for entry of the file name.
When the user enters the name of the input
image file, the image is opened in a PictureExplorer
object in the upper-left corner of the screen and the GUI
is moved to a location immediately below the
PictureExplorer object. All of the user input controls are enabled.

From that point on, the user can process the image as described earlier
using the buttons and sliders in the GUI. A Write
button is provided to allow the user to save intermediate
versions of the edited image. Note, however, that each
time the Write button is clicked, the previously written
output file is overwritten. The user should manually save
the intermediate versions if they will be needed later.

The final edited version of the image is automatically
written to the disk when the user clicks the Quit button
or clicks the X in the upper-right corner of the GUI.

Tested using Windows Vista Premium Home Edition,
Java v1.6, and the version of Ericson's multimedia library
contained in bookClasses10-1-07.zip.
**********************************************************/


import java.awt.event.MouseEvent;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowListener;
import java.awt.event.WindowEvent;
import java.awt.Color;
import java.awt.Point;
import java.awt.FlowLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.JLabel;
import javax.swing.JColorChooser;
import javax.swing.JSlider;
import javax.swing.WindowConstants;
```

```
import javax.swing.event.DocumentListener;
import javax.swing.event.DocumentEvent;
import javax.swing.event.ChangeListener;
import javax.swing.event.ChangeEvent;
import javax.swing.border.LineBorder;
import java.awt.Color;
import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.geom.Rectangle2D;
import java.awt.geom.Rectangle2D.Double;
import java.io.File;

import java.util.LinkedList;
import java.util.TreeSet;
import java.util.Comparator;




public class FloodFill01 extends JFrame{

  public static void main(String[] args){
    new FloodFill01();
  }//end main method
//-------------------------------------------------------//

  //Declare a large number of working variables.
  // Initialize some of them.
  JFrame explorerFrame = null;
  PictureExplorer explorer = null;
  Picture activePicture;
  Picture committedPicture;

  JPanel controlPanel = new JPanel();
  JPanel northPanel = new JPanel();
  JPanel centerPanel = new JPanel();
  JPanel southPanel = new JPanel();

  JPanel hueDeltaSliderPanel = new JPanel(new
FlowLayout(FlowLayout.RIGHT));
  JPanel satDeltaSliderPanel  = new JPanel(new
FlowLayout(FlowLayout.RIGHT));
  JPanel brightDeltaSliderPanel = new JPanel(new
FlowLayout(FlowLayout.RIGHT));

  JPanel hueSliderPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT));
  JPanel satSliderPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT));
  JPanel brightSliderPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT));

  JPanel hueDeltaLabelPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT));
  JPanel satDeltaLabelPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT));
  JPanel brightDeltaLabelPanel = new JPanel(new
FlowLayout(FlowLayout.RIGHT));

  JPanel hueLabelPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT));
  JPanel satLabelPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT));
```

```
  JPanel brightLabelPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT));


  JSlider hueDeltaSlider = new JSlider(0,100,0);
  JSlider satDeltaSlider = new JSlider(0,100,0);
  JSlider brightDeltaSlider = new JSlider(0,100,0);

  JSlider hueSlider = new JSlider(0,360,0);
  JSlider satSlider = new JSlider(0,100,100);
  JSlider brightSlider = new JSlider(0,100,100);


  JPanel northButtonPanel = new JPanel();
  JPanel centerButtonPanel = new JPanel();
  JPanel fileNamePanel = new JPanel();

  //Pre-load the input file name field with the name of
  // a specific test file.
  JTextField inputFileNameField = new JTextField(
                                "jenny-red.jpg",20);
//                                  "redeye2.jpg",20);
//                                  "ColorWheel.jpg",20);

  JButton floodFillButton = new JButton("Flood Fill");
  JButton writeButton = new JButton("Write File");
  JButton quitButton = new JButton("Quit");
  JButton undoButton = new JButton("Undo");
  JButton commitButton = new JButton("Commit");

  private double hueTarget = 0;
  private double satTarget = 0;
  private double brightTarget = 0;


  //The following constants control the selectivity of
  // the algorithm relative to whether or not the color of the incoming
pixel matches a target color. Decreasing the values makes the algorithm
more
  // selective.
  private float hueDelta = (float)0.01;
  private float satDelta = (float)0.01;
  private float brightDelta = (float)0.01;

  private Rectangle2D.Double pictureArea = null;

  //This is the color that will be used to paint during the floodfill.
  private Color newColor = Color.RED;

  //Copies of properties of the PictureExplorer object
  int xIndex = 0;
  int yIndex = 0;
  double zoomFactor = 0;

  String fileName = "no file specified";
  String outputPath = null;
  //-----------------------------------------------------//
```

```
public FloodFill01(){//constructor
  //Construct the GUI.
  controlPanel.setLayout(new BorderLayout());
  controlPanel.add(northPanel,BorderLayout.NORTH);
  controlPanel.add(centerPanel,BorderLayout.CENTER);
  controlPanel.add(southPanel,BorderLayout.SOUTH);

  northPanel.setLayout(new BorderLayout());
  northPanel.add(hueDeltaSliderPanel,BorderLayout.NORTH);
  northPanel.add(satDeltaSliderPanel,BorderLayout.CENTER);
  northPanel.add(brightDeltaSliderPanel,BorderLayout.SOUTH);

  hueDeltaSliderPanel.setBackground(Color.GREEN);
  hueDeltaSliderPanel.add(hueDeltaLabelPanel);
  hueDeltaLabelPanel.add(new JLabel("Hue Tolerance"));
  hueDeltaLabelPanel.setOpaque(false);
  hueDeltaSliderPanel.add(hueDeltaSlider);
  hueDeltaSlider.setMajorTickSpacing(10);
  hueDeltaSlider.setMinorTickSpacing(2);
  hueDeltaSlider.setPaintTicks(true);
  hueDeltaSlider.setPaintLabels(true);
  hueDeltaSlider.setOpaque(false);

  satDeltaSliderPanel.setBackground(Color.GREEN);
  satDeltaSliderPanel.add(satDeltaLabelPanel);
  satDeltaLabelPanel.add(new JLabel("Saturation Tolerance"));
  satDeltaLabelPanel.setOpaque(false);
  satDeltaSliderPanel.add(satDeltaSlider);
  satDeltaSlider.setMajorTickSpacing(10);
  satDeltaSlider.setMinorTickSpacing(2);
  satDeltaSlider.setPaintTicks(true);
  satDeltaSlider.setPaintLabels(true);
  satDeltaSlider.setOpaque(false);

  brightDeltaSliderPanel.setBackground(Color.GREEN);
  brightDeltaSliderPanel.add(brightDeltaLabelPanel);
  brightDeltaSliderPanel.add(brightDeltaSlider);
  brightDeltaLabelPanel.add(new JLabel("Brightness Tolerance"));
  brightDeltaLabelPanel.setOpaque(false);
  brightDeltaSlider.setMajorTickSpacing(10);
  brightDeltaSlider.setMinorTickSpacing(2);
  brightDeltaSlider.setPaintTicks(true);
  brightDeltaSlider.setPaintLabels(true);
  brightDeltaSlider.setOpaque(false);

  centerPanel.setLayout(new BorderLayout());
  centerPanel.add(hueSliderPanel,BorderLayout.NORTH);
  centerPanel.add(satSliderPanel,BorderLayout.CENTER);
  centerPanel.add(brightSliderPanel,BorderLayout.SOUTH);

  hueSliderPanel.setBackground(Color.YELLOW);
  hueSliderPanel.add(hueLabelPanel);
  hueLabelPanel.setOpaque(false);
  hueSliderPanel.add(hueSlider);
  hueLabelPanel.add(new JLabel("New Hue"));
```

```
    hueSlider.setMajorTickSpacing(60);
    hueSlider.setMinorTickSpacing(10);
    hueSlider.setPaintTicks(true);
    hueSlider.setPaintLabels(true);
    hueSlider.setBackground(Color.RED);
    hueSlider.setForeground(Color.CYAN);

    satSliderPanel.setBackground(Color.YELLOW);
    satSliderPanel.add(satLabelPanel);
    satLabelPanel.setOpaque(false);
    satSliderPanel.add(satSlider);
    satLabelPanel.add(new JLabel("New Saturation"));
    satSlider.setMajorTickSpacing(10);
    satSlider.setMinorTickSpacing(2);
    satSlider.setPaintTicks(true);
    satSlider.setPaintLabels(true);
    satSlider.setBackground(Color.RED);
    satSlider.setForeground(Color.CYAN);

    brightSliderPanel.setBackground(Color.YELLOW);
    brightSliderPanel.add(brightLabelPanel);
    brightLabelPanel.setOpaque(false);
    brightSliderPanel.add(brightSlider);
    brightLabelPanel.add(new JLabel("New Brightness"));
    brightSlider.setMajorTickSpacing(10);
    brightSlider.setMinorTickSpacing(2);
    brightSlider.setPaintTicks(true);
    brightSlider.setPaintLabels(true);
    brightSlider.setBackground(Color.RED);
    brightSlider.setForeground(Color.CYAN);




    southPanel.setLayout(new BorderLayout());
    southPanel.add(northButtonPanel,BorderLayout.NORTH);
    southPanel.add(centerButtonPanel,BorderLayout.CENTER);
    southPanel.add(fileNamePanel,BorderLayout.SOUTH);

    northButtonPanel.setBackground(Color.BLUE);
    northButtonPanel.add(floodFillButton);

    centerButtonPanel.setBackground(Color.YELLOW);
    centerButtonPanel.add(undoButton);
    centerButtonPanel.add(commitButton);
    centerButtonPanel.add(writeButton);
    centerButtonPanel.add(quitButton);

    fileNamePanel.add(new JLabel(
                               "Enter file name here: "));
    fileNamePanel.add(inputFileNameField);

    //Add the controlPanel to the content pane, adjust to
    // the correct size, and set the title.
    getContentPane().add(controlPanel);
    pack();
```

```
    setTitle("Copyright 2009,R.G.Baldwin");

    //Disable all user controls except for the text field
    // where the user enters the name of the input file.
    // The user controls will be enabled when the user
    // enters the name of the input file.
    hueDeltaSlider.setEnabled(false);
    satDeltaSlider.setEnabled(false);
    brightDeltaSlider.setEnabled(false);
    hueSlider.setEnabled(false);
    satSlider.setEnabled(false);
    brightSlider.setEnabled(false);
    floodFillButton.setEnabled(false);
    writeButton.setEnabled(false);
    quitButton.setEnabled(false);
    undoButton.setEnabled(false);
    commitButton.setEnabled(false);

    //Make the GUI visible and set the focus.
    setVisible(true);
    inputFileNameField.requestFocus();

    //---------------------------------------------------//
    //Register listeners on the user input components.
    //---------------------------------------------------//
    //Register a ChangeListener object on the hueDeltaSlider.
    //Each time the slider fires a ChangeEvent, this event
    // handler gets the value of the slider, casts it to type float, and
stores it in the instance variable named hueDelta.
    // The isMatch method uses the value when determining if a color match
exists.
    hueDeltaSlider.addChangeListener(
      new ChangeListener(){
        public void stateChanged(ChangeEvent e){
          hueDelta = (float)(hueDeltaSlider.getValue()/1000.0);
        }//end stateChanged
      }//end new ChangeListener
    );//end addChangeListener
    //---------------------------------------------------//
    //Register a ChangeListener object on the satDeltaSlider.
    //Each time the slider fires a ChangeEvent, this event
    // handler gets the value of the slider, casts it to type float, and
stores it in the instance variable named satDelta.
    // The isMatch method uses the value when determining if a color match
exists.
    satDeltaSlider.addChangeListener(
      new ChangeListener(){
        public void stateChanged(ChangeEvent e){
          satDelta = (float)(satDeltaSlider.getValue()/1000.0);
        }//end stateChanged
      }//end new ChangeListener
    );//end addChangeListener
    //---------------------------------------------------//
    //Register a ChangeListener object on the brightDeltaSlider.
    //Each time the slider fires a ChangeEvent, this event
    // handler gets the value of the slider, casts it to type float, and
```

```
stores it in the instance variable named brightDelta.
    // The isMatch method uses the value when determining if a color match
exists.
    brightDeltaSlider.addChangeListener(
      new ChangeListener(){
        public void stateChanged(ChangeEvent e){
          brightDelta = (float)(brightDeltaSlider.getValue()/1000.0);
        }//end stateChanged
      }//end new ChangeListener
    );//end addChangeListener
    //-------------------------------------------------//
    //Register a ChangeListener object on the hueSlider.
    //Each time the slider fires a ChangeEvent, this event
    // handler gets the value of the hue, saturation, and brightness
sliders and uses that information
    // to create and store the value of a new Color object in the instance
variable named newColor.
    // It also colors the slider background to match the new color and
colors the slider foreground to be
    // the color that is opposite of that color on the color wheel.
    hueSlider.addChangeListener(
      new ChangeListener(){
        public void stateChanged(ChangeEvent e){
          float newHue = (float)(hueSlider.getValue()/360.0);
          float newSat = (float)(satSlider.getValue()/100.0);
          float newBright = (float)(brightSlider.getValue()/100.0);
          newColor = new Color(Color.HSBtoRGB(newHue,newSat,newBright));
          //Set the slider backgrounds to the new color and set the
foreground to the opposite color on the color wheel with full saturation
and brightness.
          hueSlider.setBackground(newColor);
          hueSlider.setForeground(new
Color(Color.HSBtoRGB((float)(newHue+0.5),(float)1.0,(float)1.0)));
          satSlider.setBackground(newColor);
          satSlider.setForeground(new
Color(Color.HSBtoRGB((float)(newHue+0.5),(float)1.0,(float)1.0)));
          brightSlider.setBackground(newColor);
          brightSlider.setForeground(new
Color(Color.HSBtoRGB((float)(newHue+0.5),(float)1.0,(float)1.0)));
        }//end stateChanged
      }//end new ChangeListener
    );//end addChangeListener
    //-------------------------------------------------//
    //Register a ChangeListener object on the satSlider to provide the same
behavior as the hue slider.
    satSlider.addChangeListener(
      new ChangeListener(){
        public void stateChanged(ChangeEvent e){
          float newHue = (float)(hueSlider.getValue()/360.0);
          float newSat = (float)(satSlider.getValue()/100.0);
          float newBright = (float)(brightSlider.getValue()/100.0);
          newColor = new Color(Color.HSBtoRGB(newHue,newSat,newBright));
          hueSlider.setBackground(newColor);
          hueSlider.setForeground(new
Color(Color.HSBtoRGB((float)(newHue+0.5),(float)1.0,(float)1.0)));
          satSlider.setBackground(newColor);
```

```
          satSlider.setForeground(new
Color(Color.HSBtoRGB((float)(newHue+0.5),(float)1.0,(float)1.0)));
          brightSlider.setBackground(newColor);
          brightSlider.setForeground(new
Color(Color.HSBtoRGB((float)(newHue+0.5),(float)1.0,(float)1.0)));
        }//end stateChanged
      }//end new ChangeListener
    );//end addChangeListener
    //-----------------------------------------------------//
    //Register a ChangeListener object on the brightSlider to provide the
same behavior as the hue slider.
    brightSlider.addChangeListener(
      new ChangeListener(){
        public void stateChanged(ChangeEvent e){
          float newHue = (float)(hueSlider.getValue()/360.0);
          float newSat = (float)(satSlider.getValue()/100.0);
          float newBright = (float)(brightSlider.getValue()/100.0);
          newColor = new Color(Color.HSBtoRGB(newHue,newSat,newBright));
          hueSlider.setBackground(newColor);
          hueSlider.setForeground(new
Color(Color.HSBtoRGB((float)(newHue+0.5),(float)1.0,(float)1.0)));
          satSlider.setBackground(newColor);
          satSlider.setForeground(new
Color(Color.HSBtoRGB((float)(newHue+0.5),(float)1.0,(float)1.0)));
          brightSlider.setBackground(newColor);
          brightSlider.setForeground(new
Color(Color.HSBtoRGB((float)(newHue+0.5),(float)1.0,(float)1.0)));
        }//end stateChanged
      }//end new ChangeListener
    );//end addChangeListener
    //-----------------------------------------------------//
    //-----------------------------------------------------//

    floodFillButton.addActionListener(
      new ActionListener(){
        public void actionPerformed(ActionEvent e){
          xIndex = explorer.getXIndex();
          yIndex = explorer.getYIndex();

          //Instantiate an Rectangle2D object that includes the entire
picture.
          pictureArea = new
Rectangle2D.Double(0,0,activePicture.getWidth(),activePicture.getHeight());
          //Encapsulate one of the redeye problem pixels
          // in a PixelNode object and pass the object's
          // reference to the floodFill method.
          floodFill(
              new PixelNode(xIndex,
                            yIndex,
                            activePicture.getPixel(
                                                xIndex,
                                                yIndex)));
          undoButton.setEnabled(true);
          createNewDisplay();
        }//end action performed
      }//end newActionListener
```

```
    );//end addActionListener
    //-----------------------------------------------//

    writeButton.addActionListener(
      new ActionListener(){
        public void actionPerformed(ActionEvent e){
            activePicture.write(outputPath);
        }//end action performed
      }//end newActionListener
    );//end addActionListener
    //-----------------------------------------------//

    //Note that the Quit button and the JFrame close
    // button are designed to behave the same way: save
    // the file and terminate the program.
    quitButton.addActionListener(
      new ActionListener(){
        public void actionPerformed(ActionEvent e){
          activePicture.write(outputPath);
          System.exit(0);
        }//end action performed
      }//end newActionListener
    );//end addActionListener
    //-----------------------------------------------//

    //when the Undo button is clicked, the activePicture
    // is discarded and replaced by the committedPicture.
    // Thus, everything is undone back to the most recent
    // commit operation.
    undoButton.addActionListener(
      new ActionListener(){
        public void actionPerformed(ActionEvent e){
          activePicture = new Picture(committedPicture);
          undoButton.setEnabled(false);
          createNewDisplay();
        }//end action performed
      }//end newActionListener
    );//end addActionListener
    //-----------------------------------------------//

    //When the Commit button is clicked, the activePicture
    // is committed for the long term and cannot be
    // undone by clicking the undoButton.
    commitButton.addActionListener(
      new ActionListener(){
        public void actionPerformed(ActionEvent e){
          committedPicture = new Picture(activePicture);
          undoButton.setEnabled(false);
          createNewDisplay();
        }//end action performed
      }//end newActionListener
    );//end addActionListener
    //-----------------------------------------------//

    addWindowListener(
      new WindowAdapter(){
```

```
      public void windowClosing(WindowEvent e){
        activePicture.write(outputPath);
        System.exit(0);
      }//end windowClosing
    }//end new WindowAdapter
  );//end addWindowListener
  //-----------------------------------------------//

  inputFileNameField.addActionListener(
    new ActionListener(){
      public void actionPerformed(ActionEvent e){
        fileName = inputFileNameField.getText();

        activePicture = new Picture(fileName);
        committedPicture = new Picture(activePicture);

        //Because color corruption can occur when
        // writing jpg files in Java, the following code
        // makes a copy of the input file as a bmp file.
        // All further processing and writing is done in
        // bmp format. The characters BAK are
        // inserted in the output file immediately
        // before the extension. The final output is a
        // bmp file, which can be converted back to a
        // jpg file using an image-utility program such
        // as Lview.
        String inputPath = new File(fileName).
                                    getAbsolutePath();
        int posDot = inputPath.lastIndexOf('.');
        outputPath = inputPath.substring(0,posDot)
                                          + "BAK.bmp";
        activePicture.write(outputPath);

        explorer =
                  new PictureExplorer(new Picture(activePicture));
        explorerFrame = explorer.getFrame();
        explorerFrame.setDefaultCloseOperation(
                   WindowConstants.DO_NOTHING_ON_CLOSE);
        setDefaultCloseOperation(
                   WindowConstants.DO_NOTHING_ON_CLOSE);

        //Cause the GUI to be located immediately below
        // the PictureExplorer object.
        setLocation(0,explorerFrame.getHeight());

        //Enable the user input controls.
        hueDeltaSlider.setEnabled(true);
        satDeltaSlider.setEnabled(true);
        brightDeltaSlider.setEnabled(true);
        hueSlider.setEnabled(true);
        satSlider.setEnabled(true);
        brightSlider.setEnabled(true);
        writeButton.setEnabled(true);
        quitButton.setEnabled(true);
        floodFillButton.setEnabled(true);
        commitButton.setEnabled(true);
```

```
        }//end action performed
      }//end newActionListener
    );//end addActionListener
    //-----------------------------------------------------//

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

  //This method uses a flood fill algorithm to change the color of all
pixels connected to an anchor pixel selected by the user
  // when the colors of the connected pixels match the
  // color of the anchor pixel. The color is deemed to match if it matches
to within the tolerances specified in the method named isMatch.
  //The algorithm is very fast and reasonably memory efficient. Two data
structures are used. A TreeSet object
  // containing references to Point objects will expand to the number of
pixels in the picture. A LinkList object used for a queue will expand to a
few
  // hundred references to objects of type PixelNode.
  private void floodFill(PixelNode firstPixelNode){

System.out.println(hueDelta);

    //This counter is used to prevent the method from
    // going into a seemingly endless loop if the picture is very large. If
the
    // algorithm hasn't terminated by the time the limit
    // is reached, the method simply prints a message and terminates.
    long cnt = 0;
    long cntLim = 1000000;

    //The following variable is used to keep track of the maximum queue
size for information purposes.
    int maxQueueSize = 0;

    //Declare some working variables.
    LinkedList <PixelNode>queue = new LinkedList<PixelNode>();
    MyComparator myComparator = new MyComparator();
    TreeSet <Point>visited = new TreeSet<Point>(myComparator);
    Pixel westPixel = null;
    Pixel eastPixel = null;
    Pixel northPixel = null;
    Pixel southPixel = null;
    Pixel anchorPixel = null;
    int anchorX = 0;
    int anchorY = 0;
    Color currentColor = null;
    PixelNode anchorNode = null;
    float[] hsbvals = new float[3];
    int red = 0;
    int green = 0;
    int blue = 0;

    //Add the incoming PixelNode object to the queue. A
    // PixelNode object contains a reference to a Pixel
```

```
    // object and the coordinates of the Pixel object.
    //This pixel was selected by the user. It will serve as the first
anchor
    // for testing its west, east, north, and south
    // neighbors to determine if they have matching colors.
    queue.add(firstPixelNode);

    anchorPixel = firstPixelNode.pixel;
    currentColor = anchorPixel.getColor();
    red = currentColor.getRed();
    green = currentColor.getGreen();
    blue = currentColor.getBlue();

    //Get and save the color of the first anchor pixel in the HSB color
model. This will be the basis for comparison with the colors of the other
pixels.
    Color.RGBtoHSB(red,green,blue,hsbvals);
    hueTarget = hsbvals[0];
    satTarget = hsbvals[1];
    brightTarget = hsbvals[2];

    //Loop and process all of the PixelNode objects in the queue.
    while(queue.size() > 0){
      //Get a reference to the first PixelNode element in the queue and
remove it in the process. This node contains the pixel that will be used as
an
      // anchor pixel during this iteration.
      anchorNode = queue.removeFirst();

      //Extract the Pixel object from the node and get its color.
      anchorPixel = anchorNode.pixel;
      currentColor = anchorPixel.getColor();
/*
      //Get the RGB colors. They are used to negate the color of the pixel.
      red = currentColor.getRed();
      green = currentColor.getGreen();
      blue = currentColor.getBlue();

      //Negate the color of the anchor pixel.
      anchorPixel.setColor(new Color(255-red,255-green,255-blue));
*/
//      anchorPixel.setColor(Color.GREEN);
      anchorPixel.setColor(newColor);

      //Get the coordinates of the anchor pixel.
      anchorX = anchorNode.pixelX;
      anchorY = anchorNode.pixelY;

      //If the pixels to the west, east, north, and south of the anchor
pixel are inside the bounds of the picture, try to add their coordinates to
      // the TreeSet object referred to by visited. If the add method
returns true for the coordinates of a particular pixel, that pixel hasn't
      // previously been visited. In that case, process it. Otherwise,
ignore that pixel and go on to the pixel to the east of the anchor pixel.
      //Processing a pixel consists of testing the color of the pixel to
see if it matches the color of the anchor pixel. If the colors match,
```

```
      // encapsulate the pixel in a PixelNode object and add it to the
queue. The PixelNode object will emerge from the queue later and be used
      // as an anchor pixel to test its neighbors in the same way. Note
that a match in this case means that they match
      // to within the tolerances used by the isMatch method.
      //Get coordinate of west pixel.
      int west = anchorX - 1;
      //Confirm that the west pixel is in the picture and hasn't been
visited before.
      if(pictureArea.contains(west,anchorY) && visited.add(new
Point(west,anchorY))){
        //Get the pixel at these coordinates.
        westPixel = activePicture.getPixel(west,anchorY);
        if(isMatch(westPixel)){
          //The color matches. Encapsulate the pixel in a PixelNode object
and add it to the queue.
          queue.add(new PixelNode(west,anchorY,westPixel));
          //Keep track of maximum queue size for information purposes only.
          if(queue.size() > maxQueueSize){maxQueueSize = queue.size();}
        }//end if
      }//end if

      //Apply the same logic to the pixel to the east of the anchor pixel.
      int east = anchorX + 1;
      if(pictureArea.contains(east,anchorY) && visited.add(new
Point(east,anchorY))){
        eastPixel = activePicture.getPixel(east,anchorY);
        if(isMatch(eastPixel)){
          queue.add(new PixelNode(east,anchorY,eastPixel));
          if(queue.size() > maxQueueSize){maxQueueSize = queue.size();}
        }//end if
      }//end if

      //Apply the same logic to the pixel to the north of the anchor pixel.
      int north = anchorY - 1;
      if(pictureArea.contains(anchorX,north) && visited.add(new
Point(anchorX,north))){
        northPixel = activePicture.getPixel(anchorX,north);
        if(isMatch(northPixel)){
          queue.add(new PixelNode(anchorX,north,northPixel));
          if(queue.size() > maxQueueSize){maxQueueSize = queue.size();}
        }//end if
      }//end if

      //Apply the same logic to the pixel to the south of the anchor pixel.
      int south = anchorY + 1;
      if(pictureArea.contains(anchorX,south) && visited.add(new
Point(anchorX,south))){
        southPixel = activePicture.getPixel(anchorX,south);
        if(isMatch(southPixel)){
          queue.add(new PixelNode(anchorX,south,southPixel));
          if(queue.size() > maxQueueSize){maxQueueSize = queue.size();}
        }//end if
      }//end if

      //Terminate the loop if the number of iterations is excessive.
```

```
    if(cnt++ > cntLim){
      System.out.println("Iteration limit exceeded - terminating");
      break;
    }//end if
  }//end while loop

  System.out.println("Pixels visited:" + visited.size());
  System.out.println("Maximum queue size: " + maxQueueSize);

}//end floodFill
//-----------------------------------------------------//

  private boolean isMatch(Pixel pixel){


  float[] hsbvals = new float[3];
  Color color = pixel.getColor();
  int red = color.getRed();
  int green = color.getGreen();
  int blue = color.getBlue();
  Color.RGBtoHSB(red,green,blue,hsbvals);

  //This code deals with the fact that red wraps at 0 and 1.
  if(((hsbvals[0] > (hueTarget - hueDelta)) &&
      (hsbvals[0] < (hueTarget + hueDelta)) ||
      (hsbvals[0] < (hueTarget - 1 + hueDelta)) ||
      (hsbvals[0] > (hueTarget + 1 - hueDelta))) &&
      (hsbvals[1] > (satTarget - satDelta)) &&
      (hsbvals[1] < (satTarget + satDelta)) &&
      (hsbvals[2] < (brightTarget + brightDelta))){
    return true;
  }//end if
  return false;
}//end isMatch
//-----------------------------------------------------//

private void createNewDisplay(){
        zoomFactor = explorer.getZoomFactor();
        String zoomString = "100%";
        if(zoomFactor == 0.25){
          zoomString = "25%";
        }else if(zoomFactor == 0.50){
          zoomString = "50%";
        }else if(zoomFactor == 0.75){
          zoomString = "75%";
        }else if(zoomFactor == 1.0){
          zoomString = "100%";
        }else if(zoomFactor == 1.5){
          zoomString = "150%";
        }else if(zoomFactor == 2.0){
          zoomString = "200%";
        }else if(zoomFactor == 5.0){
          zoomString = "500%";
        }else{
          zoomString = "100%";//in case no match
        }//end else
```

```
          //Dispose of the existing explorer and create a
          // new one.
          explorerFrame.dispose();
          explorer = new PictureExplorer(
                                        new Picture(activePicture));
          //Get reference to the new frame
          explorerFrame = explorer.getFrame();
          explorerFrame.setDefaultCloseOperation(
                    WindowConstants.DO_NOTHING_ON_CLOSE);

          //Now set the state of the new explorer.
          //Simulate a mouse pressed event in the picture
          // to set the cursor and the text in the
          // coordinate fields.
          explorer.mousePressed(new MouseEvent(
                          new JButton("dummy component"),
                          MouseEvent.MOUSE_PRESSED,
                          (long)0,
                          0,
                          xIndex,
                          yIndex,
                          0,
                          false));

          //Simulate an action event on the zoom menu to
          // set the zoom.
          explorer.actionPerformed(new ActionEvent(
                            explorer,
                            ActionEvent.ACTION_PERFORMED,
                            zoomString));

  }//end create new display
  //=================================================//

  //This member class is used to encapsulate a reference
  // to a Pixel object and the coordinates of the physical pixel
represented by the Pixel object. This makes it possible to
  // process the Pixel object and the coordinates of the pixel as a single
element in a LinkedList object.
  private class PixelNode{
    public int pixelX;
    public int pixelY;
    public Pixel pixel;

    PixelNode(int pixelX,int pixelY,Pixel pixel){
      this.pixelX = pixelX;
      this.pixelY = pixelY;
      this.pixel = pixel;
    }//end constructor
    //-------------------------------------------------//
  }//end class PixelNode
  //===============================================================

  //This class defines a Comparator object used by the TreeSet object to
sort the Point data when it is added to the tree.
```

```
  //See http://www.breakitdownblog.com/generic-comparators-in-java/ for
help with generics format.
  private class MyComparator implements Comparator<Point>{
    public int compare(Point obj1,Point obj2){
      //Eliminate the need to deal with two separate values for sorting
purposes by converting a pair of coordinates into a single integer value
that
      // is guaranteed to be unique so long as the dimensions of the
picture are less than 100000 pixels.
      int obj1Data = 100000*((Point)obj1).x + ((Point)obj1).y;
      int obj2Data = 100000*((Point)obj2).x + ((Point)obj2).y;
      if(obj1Data < obj2Data){
        return -1;
      }else if(obj1Data == obj2Data){
        return 0;
      }else{
        return 1;
      }//end else
    }//end compare
  }//end class MyPoint
}//end class FloodFill01
```

Copyright

About the author

Richard Baldwin *is a college professor (at Austin Community College in Austin, TX) and private consultant whose primary focus is object-oriented programming using Java and other OOP languages.*

Richard has participated in numerous consulting projects and he frequently provides onsite training at the high-tech companies located in and around Austin, Texas. He is the author of Baldwin's Programming Tutorials, which have gained a worldwide following among experienced and aspiring programmers. He has also published articles in JavaPro magazine.

In addition to his programming expertise, Richard has many years of practical experience in Digital Signal Processing (DSP). His first job after he earned his Bachelor's degree was doing DSP in the Seismic Research Department of Texas Instruments. (TI is still a world leader in DSP.) In the following years, he applied his programming and DSP expertise to other interesting areas including sonar and underwater acoustics.

Richard holds an MSEE degree from Southern Methodist University and has many years of experience in the application of computer technology to real-world problems.

Baldwin@DickBaldwin.com

-end-