A Temperature and Tint Editing Program for Digital Photos

Published: March 19, 2009
By Richard G. Baldwin

Java Programming Notes # 361

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

Complete program listing

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

Listing 1. Source code for the program named TemperatureTint01.

```
/*File TemperatureTint01 Copyright 2009 R.G.Baldwin
Need to update all comments. However, the code has been
tested and confirmed to work.


The purpose of the program is to show you how to perform
HSB color editing on your digital photos.

The program allows the user to specify an image file to be
edited. The user is then presented with a display of the
image being edited and a GUI containing three sliders, a
Write button, and a text field for entry of the file name.

The sliders are labeled Hue, Sat, and Bright. The labels
are abbreviations for Hue, Saturation, and Brightness,
which are the words behind the nomenclature for the HSB
color model.

The user can change any combination of hue, saturation,
and brightness of the image by adjusting the sliders.

At any point, the user can click a Write button on the GUI
and write the processed image in its current state into a
backup file. (Note that only the most recent five backup
files are saved.) When the user terminates the program,
the final processed image is written into a final output
file.

The template program requires access to Ericson's
multimedia library.

Input files of type jpg, bmp, png are supported. The input
image file is not modified.

If the input file is in the same directory as the program
files, only the name and extension must be entered into
the text field. Otherwise, a complete path and file name
with extension must be entered.

Note, that this  program was designed to teach
programming and image processing concepts. It was not
designed to compete on a speed or convenience basis with
commercially-available photograph processing programs such
as Adobe's Photoshop Elements.

The GUI initially appears in the upper-left corner of the
screen. At this point, the sliders and the buttons are all
disabled. When the user enters the name of the input file,
a display of the image contained in that file appears in
the upper-left corner of the screen and the GUI is
relocated to a position immediately below the display.

When the GUI is relocated to the position immediately
below the display, the sliders and the buttons are all
```

```
enabled. and the text field is disabled. The width of the
GUI is changed to match the width of the display if
possible.

Clicking the large X in the upper-right corner of the
display does not terminate the program. It simply hides
the display, which is of no practical use.

The program is terminated by clicking the large X in the
upper-right corner of the GUI. Before terminating, the
program writes an output file containing the final state
of the display in the same format as the input file. The
name of the output file is the same as the name of the
input file except that the word FINAL is inserted
immediately ahead of the extension. The final output file
(and all of the backup files) are written into the same
directory from which the image file was originally read.

Tested using Windows Vista Home Premium Edition,
Java 1.6x, and the version of Ericson's multimedia library
contained in bookClasses10-1-07.zip.
**********************************************************/

import java.awt.Graphics;
import java.awt.Image;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.FlowLayout;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JSlider;
import javax.swing.JLabel;
import javax.swing.JButton;
import javax.swing.JTextField;
import javax.swing.event.ChangeListener;
import javax.swing.event.ChangeEvent;
import javax.swing.WindowConstants;

import java.io.File;

public class TemperatureTint01 extends JFrame{
  //The following constants are used to configure the GUI.
  // You can control whether or not the sliders and
  // buttons appear in the GUI, along with the contents of
  // various labels by changing the contents of the
  // following constants.

  //SLIDERS
  //Change the following boolean values to false to
  // selectively exclude sliders
  private final boolean sliderAPanelInclude = true;
```

```
  private final boolean sliderBPanelInclude = true;
  private final boolean sliderCPanelInclude = true;
  private final boolean sliderDPanelInclude = true;
  private final boolean sliderEPanelInclude = true;

  //Change the following int values to change the limits
  // and initial positions of the sliders.
  private final int sliderAMin = 0;//minimum
  private final int sliderAMax = 360;//maximum
  private final int sliderAInit = 0;//initial value

  private final int sliderBMin = 0;
  private final int sliderBMax = 400;
  private final int sliderBInit = 100;

  private final int sliderCMin = 0;
  private final int sliderCMax = 200;
  private final int sliderCInit = 100;

  private final int sliderDMin = -100;
  private final int sliderDMax = 100;
  private final int sliderDInit = 0;

  private final int sliderEMin = -100;
  private final int sliderEMax = 100;
  private final int sliderEInit = 0;

  //Change the following int values to change the tick
  // spacing on the sliders.
  private final int sliderAMajorTickSpacing = 60;
  private final int sliderAMinorTickSpacing = 10;
  private final int sliderBMajorTickSpacing = 50;
  private final int sliderBMinorTickSpacing = 10;
  private final int sliderCMajorTickSpacing = 50;
  private final int sliderCMinorTickSpacing = 10;
  private final int sliderDMajorTickSpacing = 50;
  private final int sliderDMinorTickSpacing = 10;
  private final int sliderEMajorTickSpacing = 50;
  private final int sliderEMinorTickSpacing = 10;

  //Change these string values to change the labels
  // displayed to the left of the sliders.
  private final String sliderALabel = "Hue";
  private final String sliderBLabel = "Saturation";
  private final String sliderCLabel = "Brightness";
  private final String sliderDLabel = "Temperature";
  private final String sliderELabel = "Green Tint";


  //BUTTONS
  //Change the following value to false to exclude all
  // three buttons as a group.
  //None of the three buttons were used.
  private final boolean buttonPanelInclude = false;

  //Change the following values to false to selectively
```

```
  // exclude individual buttons
  private final boolean buttonAInclude = true;
  private final boolean buttonBInclude = true;
  private final boolean buttonCInclude = true;

  //Change these string values to change the text on the
  // buttons
  private final String buttonALabel = "buttonA";
  private final String buttonBLabel = "buttonB";
  private final String buttonCLabel = "buttonC";


  //MISCELLANEOUS
  //Change this string to change the text to the left of
  // the text field.
  private final JLabel fileNameLabel =
                               new JLabel("File Name: ");

  //Change this string to change the path and name of the
  // default input file.
  private final String defaultFileName =
                                 "TemperatureTint01.jpg";
  //-----------------------------------------------------//

  //Components for construction of the GUI.
  private final JPanel mainPanel = new JPanel();

  private final JPanel northPanel = new JPanel();
  private final JPanel centerPanel = new JPanel();
  private final JPanel southPanel = new JPanel();

  private final JPanel sliderAPanel =
            new JPanel(new FlowLayout(FlowLayout.RIGHT));
  private final JPanel sliderBPanel =
            new JPanel(new FlowLayout(FlowLayout.RIGHT));
  private final JPanel sliderCPanel =
            new JPanel(new FlowLayout(FlowLayout.RIGHT));
  private final JPanel sliderDPanel =
            new JPanel(new FlowLayout(FlowLayout.RIGHT));
  private final JPanel sliderEPanel =
            new JPanel(new FlowLayout(FlowLayout.RIGHT));

  private final JPanel buttonPanel = new JPanel();

  private final JButton writeButton =
                                    new JButton("Write");
  private final JButton buttonA =
                               new JButton(buttonALabel);
  private final JButton buttonB =
                               new JButton(buttonBLabel);
  private final JButton buttonC =
                               new JButton(buttonCLabel);

  //This text field is preloaded with the name of a test
  // file to make testing and debugging easier.
  private JTextField fileNameField =
```

```
                          new JTextField(defaultFileName);

  //Change the int values at the beginning of the program
  // to reconfigure these sliders.
  private final JSlider sliderA =
           new JSlider(sliderAMin,sliderAMax,sliderAInit);
  private final JSlider sliderB =
           new JSlider(sliderBMin,sliderBMax,sliderBInit);
  private final JSlider sliderC =
           new JSlider(sliderCMin,sliderCMax,sliderCInit);
  private final JSlider sliderD =
           new JSlider(sliderDMin,sliderDMax,sliderDInit);
  private final JSlider sliderE =
           new JSlider(sliderEMin,sliderEMax,sliderEInit);

  //A reference to the original Picture object will be
  // stored here.
  private Picture picture = null;
  //A reference to a modified copy of the original
  // Picture object will be stored here.
  private Picture display = null;

  //Miscellaneous working variables.
  private Image image = null;
  private Graphics graphics = null;

  private Pixel pixel = null;
  private int red = 0;
  private int green = 0;
  private int blue = 0;
  private int writeCounter = 0;

  private Pixel[] pixels = null;

  private String fileName = null;
  private String outputPath = null;
  private String extension = null;
  //------------------------------------------------------//

  public static void main(String[] args){
    new TemperatureTint01();
  }//end main method
  //------------------------------------------------------//

  public TemperatureTint01(){//constructor
    //All close operations are handled in a WindowListener
    // object.
    setDefaultCloseOperation(
                     WindowConstants.DO_NOTHING_ON_CLOSE);

    //Put decorations on the sliders. Change the constants
    // at the beginning of the program to control major
    // and minor tick spacing.
    sliderA.setMajorTickSpacing(sliderAMajorTickSpacing);
    sliderA.setMinorTickSpacing(sliderAMinorTickSpacing);
    sliderA.setPaintTicks(true);
```

```
    sliderA.setPaintLabels(true);

    sliderB.setMajorTickSpacing(sliderBMajorTickSpacing);
    sliderB.setMinorTickSpacing(sliderBMinorTickSpacing);
    sliderB.setPaintTicks(true);
    sliderB.setPaintLabels(true);

    sliderC.setMajorTickSpacing(sliderCMajorTickSpacing);
    sliderC.setMinorTickSpacing(sliderCMinorTickSpacing);
    sliderC.setPaintTicks(true);
    sliderC.setPaintLabels(true);

    sliderD.setMajorTickSpacing(sliderDMajorTickSpacing);
    sliderD.setMinorTickSpacing(sliderDMinorTickSpacing);
    sliderD.setPaintTicks(true);
    sliderD.setPaintLabels(true);

    sliderE.setMajorTickSpacing(sliderEMajorTickSpacing);
    sliderE.setMinorTickSpacing(sliderEMinorTickSpacing);
    sliderE.setPaintTicks(true);
    sliderE.setPaintLabels(true);

    //Construct the GUI working generally from the top
    // down.
    mainPanel.setLayout(new BorderLayout());

    mainPanel.add(northPanel,BorderLayout.NORTH);
    mainPanel.add(centerPanel,BorderLayout.CENTER);
    mainPanel.add(southPanel,BorderLayout.SOUTH);

    northPanel.setLayout(new BorderLayout());

    //Add sliders if the constants at the beginning of the
    // program are true.
    if(sliderAPanelInclude)
          northPanel.add(sliderAPanel,BorderLayout.NORTH);
    if(sliderBPanelInclude)
         northPanel.add(sliderBPanel,BorderLayout.CENTER);
    if(sliderCPanelInclude)
          northPanel.add(sliderCPanel,BorderLayout.SOUTH);

    centerPanel.setLayout(new BorderLayout());

    //Add more sliders if the constants at the beginning
    // of the program are true.
    if(sliderDPanelInclude)
         centerPanel.add(sliderDPanel,BorderLayout.NORTH);
    if(sliderEPanelInclude)
        centerPanel.add(sliderEPanel,BorderLayout.CENTER);

    //Add a panel containing from one to three buttons if
    // the constant at the beginning of the program is
    // true.
    if(buttonPanelInclude)
          centerPanel.add(buttonPanel,BorderLayout.SOUTH);
```

```
    //Add buttons if the constants at the beginning of the
    // program are true.
    if(buttonAInclude) buttonPanel.add(buttonA);
    if(buttonBInclude) buttonPanel.add(buttonB);
    if(buttonCInclude) buttonPanel.add(buttonC);

    //These components should always be added.
    southPanel.add(writeButton);
    southPanel.add(fileNameLabel);
    southPanel.add(fileNameField);

    sliderAPanel.add(new JLabel(sliderALabel));
    sliderAPanel.add(sliderA);

    sliderBPanel.add(new JLabel(sliderBLabel));
    sliderBPanel.add(sliderB);

    sliderCPanel.add(new JLabel(sliderCLabel));
    sliderCPanel.add(sliderC);

    sliderDPanel.add(new JLabel(sliderDLabel));
    sliderDPanel.add(sliderD);

    sliderEPanel.add(new JLabel(sliderELabel));
    sliderEPanel.add(sliderE);

    //Disable the sliders and the buttons until the
    // user enters the file name.
    sliderA.setEnabled(false);
    sliderB.setEnabled(false);
    sliderC.setEnabled(false);
    sliderD.setEnabled(false);
    sliderE.setEnabled(false);

    buttonA.setEnabled(false);
    buttonB.setEnabled(false);
    buttonC.setEnabled(false);
    writeButton.setEnabled(false);

    //Set the size of the GUI and display it in the upper-
    // left corner of the screen. It will be moved later
    // to a position immediately below the display of the
    // picture.
    getContentPane().add(mainPanel);
    pack();//Set to overall preferred size.
    setVisible(true);

    //Move the focus to the text field to make it easy
    // for the user to enter the name of the input file.
    fileNameField.requestFocus();
    //-----------------------------------------------//

    //Register a listener on the text field. When the user
    // enters the file name in the text field, set
    // everything up properly so that the program will
    // function as an event-driven picture-manipulation
```

```
    // program until the user clicks the large X in the
    // upper-right of the GUI.
    fileNameField.addActionListener(
      new ActionListener(){
        public void actionPerformed(ActionEvent e){
          //Disable the text field to prevent the user
          // from entering anything else in it.
          fileNameField.setEnabled(false);
          fileNameLabel.setEnabled(false);

          //Get the file name from the text field and use
          // it to create a new Picture object. Display my
          // name in the image.
          fileName = fileNameField.getText();
          picture = new Picture(fileName);
          picture.addMessage("Dick Baldwin",10,20);

          //Get information that will be used to write the
          // output files.
          String inputPath = new File(fileName).
                                       getAbsolutePath();
          int posDot = inputPath.lastIndexOf('.');
          outputPath = inputPath.substring(0,posDot);
          //Write the first copy of the output backup
          // file before any processing is done.
          picture.write(outputPath
                      + "BAK" + writeCounter++ + ".bmp");

          //Get filename extension. It will be used later
          // to write the final output file.
          extension = inputPath.substring(posDot);

          //Decorate the GUI.
          setTitle("Copyright 2009, R.G.Baldwin");

          //Create the picture that will be processed.
          // Note that the original image file is not
          // modified by this program.
          int pictureWidth = picture.getWidth();
          int pictureHeight = picture.getHeight();
          display = new Picture(
                             pictureWidth,pictureHeight);

          //Draw the initial display.
          graphics = display.getGraphics();
          graphics.drawImage(picture.getImage(),0,0,null);
          display.show();

          //Adjust the width of the GUI to match the width
          // of the display if possible. Then relocate the
          // GUI to a position immediately below the
          // display.
          //Establish the preferred size now that the
          // input file name has been entered.
          pack();
          int packedHeight = getHeight();
```

```
        int packedWidth = getWidth();
        if((pictureWidth + 7) >= packedWidth){
          //Make the width of the GUI the same as the
          // width of the display.
          setSize(pictureWidth + 7,packedHeight);
        }//Else, just leave the GUI at its current size.
        //Put the GUI in its new location immediately
        // below the display.
        setLocation(0,pictureHeight + 30);

        //Enable the sliders and the buttons.
        sliderA.setEnabled(true);
        sliderB.setEnabled(true);
        sliderC.setEnabled(true);
        sliderD.setEnabled(true);
        sliderE.setEnabled(true);

        buttonA.setEnabled(true);
        buttonB.setEnabled(true);
        buttonC.setEnabled(true);

        writeButton.setEnabled(true);

      }//end actionPerformed
    }//end new ActionListener
  );//end addActionListener
  //-----------------------------------------------//

  //Register an ActionListener on the writeButton.
  // Each time the user clicks the button, a backup bmp
  // file containing the current state of the display is
  // written into the directory from which the original
  // picture was read. The five most recent backup files
  // are saved. The names of the backup files are the
  // same as the name of the input file except that BAKn
  // is inserted immediately ahead of the extension
  // where n is a digit ranging from 0 to 4. The value
  // of n rolls over at 4 and starts back at 0.
  writeButton.addActionListener(
    new ActionListener(){
      public void actionPerformed(ActionEvent e){
        display.write(outputPath
                      + "BAK" + writeCounter++ + ".bmp");
        //Reset the writeCounter if it exceeds 4 to
        // conserve disk space.
        if(writeCounter > 4){
          writeCounter = 0;
        }//end if
      }//end action performed
    }//end newActionListener
  );//end addActionListener
  //-----------------------------------------------//

  //Register an ActionListener on buttonA. This button
  // and this listener are here for future expansion.
  // For demonstration purposes, make the computer beep
```

```
    // when the user clicks any of the three buttons.
    buttonA.addActionListener(
      new ActionListener(){
        public void actionPerformed(ActionEvent e){
          getToolkit().getDefaultToolkit().beep();
        }//end action performed
      }//end newActionListener
    );//end addActionListener
    //----------------------------------------------------//
    //Register an ActionListener on buttonB. This button
    // and this listener are here for future expansion.
    buttonB.addActionListener(
      new ActionListener(){
        public void actionPerformed(ActionEvent e){
          getToolkit().getDefaultToolkit().beep();
        }//end action performed
      }//end newActionListener
    );//end addActionListener
    //----------------------------------------------------//
    //Register an ActionListener on buttonC. This button
    // and this listener are here for future expansion.
    buttonC.addActionListener(
      new ActionListener(){
        public void actionPerformed(ActionEvent e){
          getToolkit().getDefaultToolkit().beep();
        }//end action performed
      }//end newActionListener
    );//end addActionListener
    //----------------------------------------------------//

    //Register a WindowListener that will respond when the
    // user clicks the large X in the upper-right corner
    // of the GUI. This event handler will write the final
    // state of the display into an output file of the
    // same type as the original input file. The name will
    // be the same except that the word FINAL will be
    // inserted immediately ahead of the extension.
    addWindowListener(
      new WindowAdapter(){
        public void windowClosing(WindowEvent e){
          display.write(outputPath + "FINAL" + extension);
          System.exit(0);
        }//end windowClosing
      }//end new WindowAdapter
    );//end addWindowListener
    //----------------------------------------------------//

    //Register a ChangeListener object on sliderA.
    //Each time sliderA fires a ChangeEvent, this event
    // handler creates a new display as a copy of the
    // original picture and calls a method named
    // handleSliderABandC. You can modify the code in this
    // listener, the listeners for the other sliders, and
    // the listeners for the buttons to customize the
    // behavior of the program to meet your needs.
    sliderA.addChangeListener(
```

```
      new ChangeListener(){
        public void stateChanged(ChangeEvent e){
          //Draw a new copy of the picture on the display.
          graphics = display.getGraphics();
          graphics.drawImage(picture.getImage(),0,0,null);
          handleAllSliders();
        }//end stateChanged
      }//end new ChangeListener
    );//end addChangeListener
    //-----------------------------------------------//
    //Register a ChangeListener object on sliderB.
    //Each time sliderB fires a ChangeEvent, this event
    // handler creates a new display as a copy of the
    // original picture and calls a method named
    // handleSliderABandC.
    sliderB.addChangeListener(
      new ChangeListener(){
        public void stateChanged(ChangeEvent e){
          //Draw a new copy of the picture on the display.
          graphics = display.getGraphics();
          graphics.drawImage(picture.getImage(),0,0,null);
          handleAllSliders();
        }//end stateChanged
      }//end new ChangeListener
    );//end addChangeListener
    //-----------------------------------------------//
    //Register a ChangeListener object on sliderC.
    //Each time sliderC fires a ChangeEvent, this event
    // handler creates a new display as a copy of the
    // original picture and calls a method named
    // handleSliderABandC.
    sliderC.addChangeListener(
      new ChangeListener(){
        public void stateChanged(ChangeEvent e){
          //Draw a new copy of the picture on the display.
          graphics = display.getGraphics();
          graphics.drawImage(picture.getImage(),0,0,null);
          handleAllSliders();
        }//end stateChanged
      }//end new ChangeListener
    );//end addChangeListener
    //-----------------------------------------------//

    //Register a ChangeListener object on sliderD.
    //Each time sliderD fires a ChangeEvent, this event
    // handler creates a new display as a copy of the
    // original picture and calls a method named
    // handleSliderD. Note that this is a different
    // method than the method called by the three
    // previous listener objects.
    sliderD.addChangeListener(
      new ChangeListener(){
        public void stateChanged(ChangeEvent e){
          //Draw a new copy of the picture on the display.
          graphics = display.getGraphics();
          graphics.drawImage(picture.getImage(),0,0,null);
```

```
          handleAllSliders();
        }//end stateChanged
      }//end new ChangeListener
    );//end addChangeListener
    //-----------------------------------------------//

    //This slider not used.
    //Register a ChangeListener object on sliderE.
    //Each time sliderE fires a ChangeEvent, this event
    // handler causes the computer to beep solely to
    // demonstrate that the slider is alive. Note that the
    // beep occurs at the end of the slider travel.
    sliderE.addChangeListener(
      new ChangeListener(){
        public void stateChanged(ChangeEvent e){
          //Draw a new copy of the picture on the display.
          graphics = display.getGraphics();
          graphics.drawImage(picture.getImage(),0,0,null);
          handleAllSliders();
        }//end stateChanged
      }//end new ChangeListener
    );//end addChangeListener
    //-----------------------------------------------//

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

  /*
  This method makes it possible for the user to perform
  HSB color editing, adjust the color temperature, and
  adjust the green tint on a digital photo. The method is
  called each time any one of the the five sliders fires
  a ChangeEvent. Events are fired when the user moves the
  sliders.

  Immediately before this method is called, a new
  display is created, which is a copy of the original
  picture. This method operates only on the display. It
  does not modify the original picture.

  Each time this method is called, it gets the value of
  each slider and uses that value to modify the
  corresponding hue, saturation, and brightness property
  for every pixel in the image currently residing in the
  display. Then it uses the RGB color model to adjust the
  color temperature and the green tint.

  Basically, for each pixel, the method does the
  following:
  1. Gets the red, green, and blue values from the native
  RGB color model data for the pixel.
  2. Converts the RGB values into HSB values.
  3. Modifies the HSB values independently of one another
  using the current slider values.
  4. Converts the modified HSB values back to RGB values.
  5. Uses the RGB values to adjust the color temperature.
```

```
  6. Uses the RGB values to adjust the green tint.
  7. Uses the RGB values to set the modified color into
  the pixel.

  The method is synchronized to eliminate the possiblilty
  that it may be called on two threads concurrently.
  */
  private synchronized void handleAllSliders(){
    pixels = display.getPixels();
    float[] hsbvals = new float[3];//HSB pixel values.
    int sliderValue = 0;

    //Process every pixel in the image using the same
    // algorithm.
    for(int cnt = 0;cnt < pixels.length;cnt++){
      //Get the red, green, and blue values for the
      // current pixel.
      red = pixels[cnt].getRed();
      green = pixels[cnt].getGreen();
      blue = pixels[cnt].getBlue();

      //Get the three HSB color model values that
      // correspond to the current pixel color expressed
      // in the RGB color model. When the method returns,
      // the three HSB values have been placed in the
      // three-element array referred to by hsbvals in the
      // order hue, saturation, and brightness.
      Color.RGBtoHSB(red,green,blue,hsbvals);

      //Modify the hue value for the pixel based on the
      // current value of the Hue slider. Note that this
      // statement performs addition instead of
      // multiplication.
      hsbvals[0] = (float)(hsbvals[0] +
                           sliderA.getValue()/360.0);//hue

      //Modify the saturation value for the pixel based
      // on the current value of the Sat slider. Note
      // that multiplication is used here.
      hsbvals[1] = (float)(hsbvals[1] *
                    sliderB.getValue()/100.0);//saturation
      //If the computed value is greter than 1.0, clip it
      // at 1.0.
      if(hsbvals[1] > 1.0) hsbvals[1] = (float)1.0;

      //Modify the brightness value for the pixel based
      // on the current value of the Bright slider. Once
      // again, multiplication is used here.
      hsbvals[2] = (float)(hsbvals[2] *
                    sliderC.getValue()/100.0);//brightness
      if(hsbvals[2] > 1.0) hsbvals[2] = (float)1.0;

      //Convert the HSB color values back into RGB color
      // values.
      int color = Color.HSBtoRGB(
                        hsbvals[0],hsbvals[1],hsbvals[2]);
```

```
    //Adjust the color temperature using the RGB model.
    // Increase the red and decrease the blue when the
    // slider moves to the left of zero. Increase the
    // blue and decrease the red when it moves to the
    // right of zero.

    //Get a Color object based on the int color value
    // that was computed above.
    Color newColor = new Color(color);
    red = newColor.getRed();
    blue = newColor.getBlue();
    green = newColor.getGreen();

    //Set the slider limit to 100 and divide the slider
    // value by 500 so that the total change is only
    // 20 percent when the pointer is at the end of the
    // slider. Each of the 100 slider steps represents
    // a 0.2-percent change in color temperature.
    //Don't change the colors when the slider value
    // is 0.
    sliderValue = sliderD.getValue();
    if(sliderValue < 0){
      sliderValue = -sliderValue;//Change sign
      //Increase the value of red.
      red = red + (int)(red * sliderValue/500.0);
      //Decrease the value of blue by the same percent.
      blue = blue - (int)(blue * sliderValue/500.0);
      if(red > 255)red = 255;//Clip at 255.
    }else if(sliderValue > 0){
      //Decrease the value of red.
      red = red - (int)(red * sliderValue/500.0);
      //Increase the value of blue by the same percent.
      blue = blue + (int)(blue * sliderValue/500.0);
      if(blue > 255)blue = 255;//Clip at 255.
    }//end else

    //Adjust the green tint  Don't change the colors
    // when the slider value is 0.
    sliderValue = sliderE.getValue();
    if(sliderValue < 0){
      sliderValue = -sliderValue;//Change sign
      //Decrease green and increase both red and blue
      // by half the percentage
      green = green - (int)(green * sliderValue/500.0);
      red = red + (int)(red * sliderValue/1000.0);
      blue = blue + (int)(blue * sliderValue/1000.0);
      if(red > 255)red = 255;//Clip at 255.
      if(blue > 255)blue = 255;//Clip at 255.
    }else if(sliderValue > 0){
      //Increase green and decrease both red and blue
      // by half the percentage.
      green = green + (int)(green * sliderValue/500.0);
      red = red - (int)(red * sliderValue/1000.0);
      blue = blue - (int)(blue * sliderValue/1000.0);
```

```
        if(green > 255)green = 255;//Clip at 255.
      }//end else

      //Use the RGB color values to set the color of
      // the pixel.
      newColor = new Color(red,green,blue);
      pixels[cnt].setColor(newColor);
    }//end for loop

    //Repaint the display.
    display.repaint();

  }//end handleSliderABandC
  //-----------------------------------------------------//

}//end class TemperatureTint01
```

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-