Java OOP: Using a JColorChooser object*

Richard Baldwin

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

Abstract

Learn how to use a JColorChooser object to specify a color in any one of five different ways.

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. 8)
- Summary (p. 9)
- What's next? (p. 9)
- Miscellaneous (p. 9)
- Complete program listing (p. 9)

2 Preface

This module is one of a series of modules designed to teach you about Object-Oriented Programming (OOP) using Java.

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 1.2: Sep 11, 2012 11:25 am -0500
†http://creativecommons.org/licenses/by/3.0/

2.1.1 Figures

- Figure 1 (p. 3) . Program output at startup.
- Figure 2 (p. 4) . Program output after clicking the Choose Color button.
- Figure 3 (p. 5) . Program output after selecting a reddish color and clicking the OK button.
- Figure 4 (p. 5) . Program output after clicking the Darker button.
- Figure 5 (p. 7) . The showDialog method.

2.1.2 Listings

- Listing 1 (p. 6) . Event handler for the Choose Color button.
- Listing 2 (p. 6) . Show the JColorChooser object.
- Listing 3 (p. 8) . Darkening the color.
- Listing 4 (p. 9) . Complete program listing.

3 Preview

In this module, you will learn how to use a **JColorChooser** object to specify a color in any one of five different ways:

- Swatches
- HSV
- HSL
- RGB
- CMYK

The details regarding the five different ways are not explained. Instead, an understanding of the five ways for specifying a color is left as an exercise for the student.

This module concentrates on the programming aspects of the color chooser as opposed to the aesthetic aspects of the color chooser.

You will also learn how to create brighter and darker shades of a given color.

What is a JColorChooser object?

According to the Java documentation,

"JColorChooser provides a pane of controls designed to allow a user to manipulate and select a color."

Program output at startup

A complete listing of the program discussed in this module is provided in Listing 4 (p. 9) near the end of the module.

Figure 1 (p. 3) shows the program output at startup.

The program output is a GUI containing

- three text fields, one each for red, green, and blue color values
- three labels that identify the contents of each text field
- a button labeled **Choose Color**
- two buttons labeled **Brighter** and **Darker**
- a square ***color swatch*** whose color reflects the color specified by the color component values in the text fields

Program output at startup.

Figure 1: Program output at startup.

Program output after clicking the Choose Color button.

Figure 2 (p. 4) shows the result of clicking the **Choose Color** button shown in Figure 1 (p. 3) .

The bottom image in Figure 2 (p. 4) is an object of the **JColorChooser** class.

(The color chooser object doesn't actually appear below the GUI as shown in Figure 2 (p. 4) . Instead, it appears in the upper-left corner of the screen and hides the GUI. I manually dragged it down below the GUI to produce this image.)

Program output after clicking the Choose Color button.

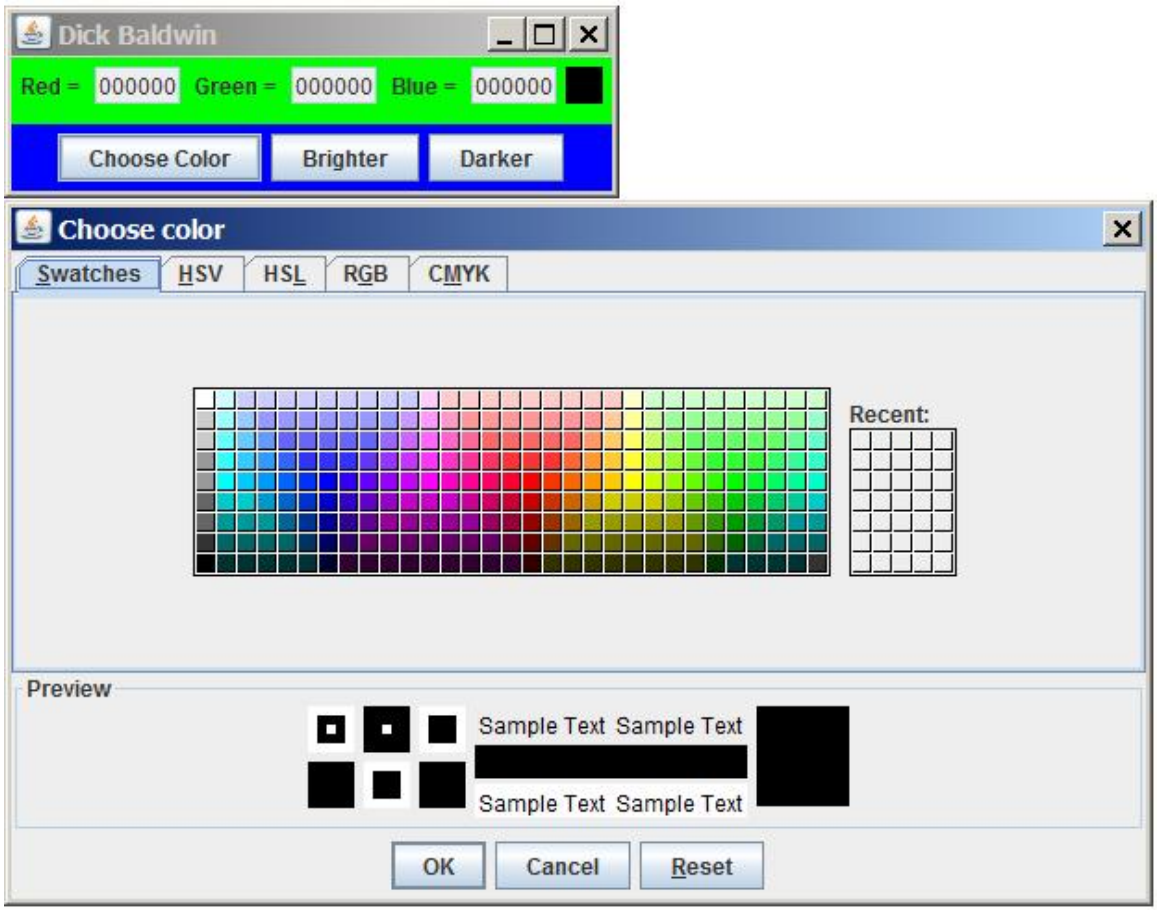

Figure 2: Program output after clicking the Choose Color button.

The initial color selection

Note in Figure 2 (p. 4) that the selected color when the color chooser appears matches the color of the square color swatch in Figure 1 (p. 3) *(black)*

Program output after selecting a reddish color and clicking the OK button.

Figure 3 (p. 5) shows the result of

- selecting a reddish color in the color chooser, and
- clicking the OK button in the color chooser.

Program output after selecting a reddish color and clicking the OK button.

Figure 3: Program output after selecting a reddish color and clicking the OK button.

New color in the GUI

Note in Figure 3 (p. 5) that

- the color selected in the color chooser now appears in the square color swatch in Figure 3 (p. 5)
- the color swatch in Figure 3 (p. 5) has a black border
- the color values in the three text fields describe the color showing in the swatch

(The color in the swatch is based on the three color values in the text fields.)

Program output after clicking the Darker button.

Figure 4 (p. 5) shows the result of clicking the **Darker** button once.

Program output after clicking the Darker button.

Figure 4: Program output after clicking the Darker button.

A darker color

The color swatch in Figure 4 (p. 5) shows a darker version of the color swatch shown in Figure 3 (p. 5) .

The numeric color values in Figure 4 (p. 5) describe the color in the color swatch, and are lower than the corresponding color values in Figure 3 (p. 5)

A brighter color

Although not demonstrated here, the color in the swatch could be made brighter by clicking the **Brighter** button.

If you click the **Darker** button enough times, the color will go to black. Similarly, if you click the **Brighter** button enough times, the color will go to white.

4 Discussion and sample code

A complete listing of the program is provided in Listing 4 (p. 9) .

Much of the code in this program is very similar to code that I explained in earlier modules. Therefore, I won't repeat those explanations here. Instead, I will concentrate on the code that is new and different.

Given that caveat, I will skip all the way down to the event handler for the **Choose Color** button.

Event handler for the Choose Color button

Listing 1 (p. 6) defines, instantiates, and registers an **ActionListener** object on the **Choose Color** button.

Listing 1: Event handler for the Choose Color button.

```
    chooseButton.addActionListener(
  new ActionListener(){
    public void actionPerformed(ActionEvent e){
      Color selColor = JColorChooser.showDialog(
                chooseButton,"Choose color",new Color(
                   redInt,greenInt,blueInt));
      if(selColor != null){
        //Don't change the color if the user cancels
        // out.
        redField.setText("" + selColor.getRed());
        greenField.setText("" + selColor.getGreen());
        blueField.setText("" + selColor.getBlue());
      }//end if

    }//end action performed
  }//end new ActionListener
);//end addActionListener
```

Show the JColorChooser object

Listing 2 (p. 6) shows the code, *(extracted from Listing 1* (p. 6)) that causes the **color chooser** to appear on the screen.

Listing 2: Show the JColorChooser object.

```
Color selColor = JColorChooser.showDialog(
                chooseButton,
                "Choose color",
                new Color(redInt,greenInt,blueInt));
```

Call the static showDialog method

Listing 2 (p. 6) calls the static **showDialog** method of the **JColorChooser** class.

Figure 5 (p. 7) provides information for the **showDialog** method.

The showDialog method.

```
public static Color showDialog(
                      Component component,
                      String title,
                      Color initialColor)
                        throws HeadlessException

Shows a modal color-chooser dialog and blocks
until the dialog is hidden. If the user presses
the "OK" button, then this method hides/disposes
the dialog and returns the selected color. If
the user presses the "Cancel" button or closes
the dialog without pressing "OK", then this
method hides/disposes the dialog and returns null.

Parameters:
  component - the parent Component for the dialog
  title - the String containing the dialog's title
  initialColor - the initial Color set when the
    color-chooser is shown
Returns:
  the selected color or null if the user opted out
```

Figure 5: The showDialog method.

A straightforward explanation

The behavior of the dialog is explained in Figure 5 (p. 7) and shouldn't require further explanation.

Parameters to the showDialog method

A comparison of Listing 2 (p. 6) and Figure 5 (p. 7) shows how the color chooser is integrated into the program.

Perhaps the most important aspects of that integration are the *third parameter* and the *return value* .

The third parameter

The third parameter is an anonymous **Color** object that matches the color specified by the text fields in the GUI, which determine the color of the swatch.

Thus, the initial color for the color chooser matches the color of the swatch in the GUI when the color chooser first appears.

The return value

The return value from the color chooser is a reference to an object of type **Color** , which is saved in a variable named **selColor** .

Processing the return value

Returning to the code in Listing 1 (p. 6) ,

- if the return value is not null,
- the color components of the return value are converted to strings and
- stored in the red, green, and blue text fields.

What happens next?

You learned in an earlier module that if the values stored in any of the three text fields changes for any reason, **DocumentListener** objects registered on the three text fields cause the color of the swatch to change to match the new values of the color components.

Thus, the color that is chosen by the user from the color chooser modifies the values in the text fields causing the color of the swatch in the GUI to match the chosen color.

Darkening and brightening the color

Listing 3 (p. 8) shows an **ActionListener** object being registered on the button labeled **Darker** in Figure 1 (p. 3) .

Listing 3: Darkening the color.

```
    darkerButton.addActionListener(
  new ActionListener(){
    public void actionPerformed(ActionEvent e){
      Color color = new Color(
           redInt,greenInt,blueInt).darker();
      redField.setText("" + color.getRed());
      greenField.setText("" + color.getGreen());
      blueField.setText("" + color.getBlue());
    }//end action performed
  }//end newActionListener
);//end addActionListener
```

The darkening action listener

This code creates a new **Color** object that matches the color values in the three text fields.

Then it calls the **darker** method on that **Color** object causing its color to be darkened.

Then it extracts the color components from the darker **Color** object and stores those values in the text fields.

This, in turn, causes the color swatch to change to match the new color values.

A brightening event handler

A similar event handler is registered on the button labeled **Brighter** in Figure 1 (p. 3) .

This code, which you can view in Listing 4 (p. 9) calls the **brighter** method instead of the **darker** method on the **Color** object.

End of story

That concludes the explanation of material that is new and different in this program.

5 Run the program

I encourage you to copy the code from Listing 4 (p. 9) . 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

In this module, you will learned how to use a **JColorChooser** object to specify a color in any one of five different ways.

You also learned how to create brighter and darker shades of a given color.

7 What's next?

In the next module, you will learn how to write an editor program that you can use to modify the colors in an image on a pixel-by-pixel basis.

8 Miscellaneous

This section contains a variety of miscellaneous information.

NOTE: **Housekeeping material**

- Module name: Java OOP: Using a JColorChooser object
- File: Java3128.htm
- Published: 09/11/12

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 listing

A complete listing of the source code discussed in this module is shown in Listing 4 (p. 9) .

Listing 4: Complete program listing.

```
/*File Prob14 Copyright 2012 R.G.Baldwin

Old material:
This program services document events on the contents of
text fields containing color values.

The values are used to create a color swatch that displays
the color indicated by the color values in the red, green,
and blue text fields.
```

```
New material:
Demonstrates how to create and use a JColorChooser dialog.
Also demonstrates use of the darker and brighter methods.
Also demonstrates how to cause JTextField objects to be
non-editable.
*******************************************************/
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.JLabel;
import javax.swing.WindowConstants;
import javax.swing.event.DocumentListener;
import javax.swing.event.DocumentEvent;
import javax.swing.border.LineBorder;
import java.awt.Color;
import java.awt.GridLayout;
import java.awt.Dimension;
import javax.swing.JColorChooser;

public class Prob14{
  public static void main(String[] args){
    new Prob14Runner();
  }//end main method
}//end class Prob14
//=======================================================//

class Prob14Runner extends JFrame{

  private JPanel controlPanel = new JPanel();
  private JPanel colorPanel = new JPanel();
  private JPanel buttonPanel = new JPanel();
  private JPanel colorIndicatorPanel = new JPanel();

  private JTextField redField = new JTextField("000000");
  private JTextField greenField =
                                new JTextField("000000");
  private JTextField blueField = new JTextField("000000");

  private int redInt = 0;
  private int greenInt = 0;
  private int blueInt = 0;

  private JButton chooseButton =
                              new JButton("Choose Color");
  private JButton brighterButton =
                                 new JButton("Brighter");
  private JButton darkerButton = new JButton("Darker");
```

```
  //-----------------------------------------------------//

  public Prob14Runner(){//constructor

    setDefaultCloseOperation(
                          WindowConstants.EXIT_ON_CLOSE);

    controlPanel.setLayout(new GridLayout(2,1));
    controlPanel.add(colorPanel);
    controlPanel.add(buttonPanel);

    colorPanel.setBackground(Color.GREEN);
    colorPanel.add(new JLabel("Red = "));
    colorPanel.add(redField);
    colorPanel.add(new JLabel(" Green = "));
    colorPanel.add(greenField);
    colorPanel.add(new JLabel(" Blue = "));
    colorPanel.add(blueField);
    colorPanel.add(colorIndicatorPanel);

    redField.setEditable(false);
    greenField.setEditable(false);
    blueField.setEditable(false);

    colorIndicatorPanel.setBorder(
                          new LineBorder(Color.black,1));
    colorIndicatorPanel.setPreferredSize(
                                  new Dimension(20,20));

    buttonPanel.setBackground(Color.BLUE);
    buttonPanel.add(chooseButton);
    buttonPanel.add(brighterButton);
    buttonPanel.add(darkerButton);

    //Color the swatch for the first time.
    paintColorSwatch();

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

    //Make the GUI visible
    setVisible(true);

    //-----------------------------------------------------//
    //Register listeners on the user input components.
    //-----------------------------------------------------//
    chooseButton.addActionListener(
      new ActionListener(){
```

```
      public void actionPerformed(ActionEvent e){
        Color selColor = JColorChooser.showDialog(
                  chooseButton,"Choose color",new Color(
                     redInt,greenInt,blueInt));
        if(selColor != null){
          //Don't change the color if the user cancels
          // out.
          redField.setText("" + selColor.getRed());
          greenField.setText(
                              "" + selColor.getGreen());
          blueField.setText("" + selColor.getBlue());
        }//end if

      }//end action performed
    }//end new ActionListener
  );//end addActionListener
  //---------------------------------------------------//

  darkerButton.addActionListener(
    new ActionListener(){
      public void actionPerformed(ActionEvent e){
        Color color = new Color(
             redInt,greenInt,blueInt).darker();
        redField.setText("" + color.getRed());
        greenField.setText("" + color.getGreen());
        blueField.setText("" + color.getBlue());
      }//end action performed
    }//end newActionListener
  );//end addActionListener
  //---------------------------------------------------//

  brighterButton.addActionListener(
    new ActionListener(){
      public void actionPerformed(ActionEvent e){
        Color color = new Color(
           redInt,greenInt,blueInt).brighter();
        redField.setText("" + color.getRed());
        greenField.setText("" + color.getGreen());
        blueField.setText("" + color.getBlue());
      }//end action performed
    }//end newActionListener
  );//end addActionListener
  //---------------------------------------------------//

  //Register a document listener on the red text field.
  // This listener will respond when the contents of
  // the text field are modified either by the program
  // or by the user.
  redField.getDocument().addDocumentListener(
    new DocumentListener(){
      public void changedUpdate(DocumentEvent e){
```

```
        //Empty method - not needed
      }//end changedUpdate

      public void removeUpdate(DocumentEvent e){
        try{
          redInt = Integer.parseInt(
                                redField.getText());
          if((redInt >= 0) && (redInt <= 255)){
            paintColorSwatch();
          }//end if
        }catch(Exception ex){
          //do nothing on exception
        }//end catch
      }//end removeUpdate

      public void insertUpdate(DocumentEvent e){
        try{
          redInt = Integer.parseInt(
                                redField.getText());
          if((redInt >= 0) && (redInt <= 255)){
            paintColorSwatch();
          }//end if
        }catch(Exception ex){
          //do nothing on exception
        }//end catch
      }//end insertUpdate

    }//end new DocumentListener
  );//end addDocumentListener
  //---------------------------------------------------//

  //Register a document listener on the green text
  // field. Essentially the same as the above.
  greenField.getDocument().addDocumentListener(
    new DocumentListener(){
      public void changedUpdate(DocumentEvent e){}

      public void removeUpdate(DocumentEvent e){
        try{
          greenInt = Integer.parseInt(
                              greenField.getText());
          if((greenInt >= 0) && (greenInt <= 255))
          {
            paintColorSwatch();
          }//end if
        }catch(Exception ex){
          //do nothing on exception
        }//end catch
      }//end removeUpdate

      public void insertUpdate(DocumentEvent e){
```

```
        try{
          greenInt = Integer.parseInt(
                              greenField.getText());
          if((greenInt >= 0) && (greenInt <= 255))
          {
            paintColorSwatch();
          }//end if
        }catch(Exception ex){
          //do nothing on exception
        }//end catch
      }//end insertUpdate

    }//end new DocumentListener
  );//end addDocumentListener
  //---------------------------------------------------//

  //Register a document listener on the blue text
  // field. Essentially the same as the above.
  blueField.getDocument().addDocumentListener(
    new DocumentListener(){
      public void changedUpdate(DocumentEvent e){}

      public void removeUpdate(DocumentEvent e){
        try{
          blueInt = Integer.parseInt(
                              blueField.getText());
          if((blueInt >= 0) && (blueInt <= 255)){
            paintColorSwatch();
          }//end if
        }catch(Exception ex){
          //do nothing on exception
        }//end catch
      }//end removeUpdate

      public void insertUpdate(DocumentEvent e){
        try{
          blueInt = Integer.parseInt(
                              blueField.getText());
          if((blueInt >= 0) && (blueInt <= 255)){
            paintColorSwatch();
          }//end if
        }catch(Exception ex){
          //do nothing on exception
        }//end catch
      }//end insertUpdate

    }//end new DocumentListener
  );//end addDocumentListener
  //---------------------------------------------------//
}//end constructor
//---------------------------------------------------//
```

```
  //The purpose of this method is to color a swatch
  // located next to the RGB color values.
  private void paintColorSwatch(){
    colorIndicatorPanel.setBackground(
                      new Color(redInt,greenInt,blueInt));
  }//end paintColorSwatch

  //-----------------------------------------------------//

  //The purpose of this method is to absorb any exceptions
  // that may be thrown by the parseInt method in order
  // to avoid having the program abort. In the event that
  // an exception is thrown, this method simply returns an
  // int value of 0;
  private int goParseInt(String string){
    int result = 0;
    try{
      result = Integer.parseInt(string);
    }catch(Exception e){
      result = 0;
    }//end catch
    return result;
  }//end goParseInt

  //-----------------------------------------------------//


}//end class Prob14Runner
```

-end-