

I

GroupLayout

1.1 Introduction

Java SE 6 includes a powerful new layout manager called **GroupLayout**, which is the default layout manager in the NetBeans IDE (www.netbeans.org). In this appendix, we overview GroupLayout, then demonstrate how to use the NetBeans IDE's **Matisse GUI designer** to create a GUI using GroupLayout to position the components. NetBeans generates the GroupLayout code for you automatically. Though you can write GroupLayout code by hand, in most cases you'll use a GUI design tool like the one provided by NetBeans to take advantage of GroupLayout's power. For more details on GroupLayout, see the list of web resources at the end of this appendix.

1.2 GroupLayout Basics

Chapters 14 and 25 presented several layout managers that provide basic GUI layout capabilities. We also discussed how to combine layout managers and multiple containers to create more complex layouts. Most layout managers do not give you precise control over the positioning of components. In Chapter 25, we discussed the **GridBagLayout**, which provides more precise control over the position and size of your GUI components. It allows you to specify the horizontal and vertical position of each component, the number of rows and columns each component occupies in the grid, and how components grow and shrink as the size of the container changes. This is all specified at once with a **GridBagConstraints** object. Class **GroupLayout** is the next step in layout management. GroupLayout is more flexible, because you can specify the horizontal and vertical layouts of your components independently.

Sequential and Parallel Arrangements

Components are arranged either sequentially or in parallel. The three **JButtons** in Fig. I.1 are arranged with **sequential horizontal orientation**—they appear left to right in sequence. Vertically, the components are arranged in parallel, so, in a sense, they “occupy

the same vertical space.” Components can also be arranged sequentially in the vertical direction and in parallel in the horizontal direction, as you’ll see in Section I.3. To prevent overlapping components, components with parallel vertical orientation are normally arranged with sequential horizontal orientation (and vice versa).

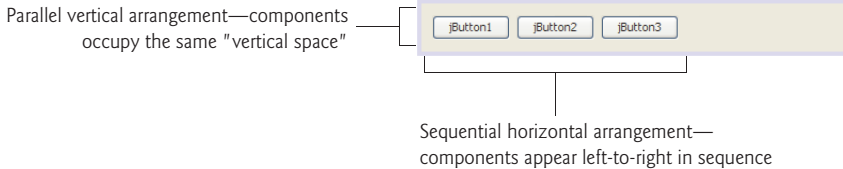


Fig. I.1 | JButtons arranged sequentially for their horizontal orientation and in parallel for their vertical orientation.

Groups and Alignment

To create more complex user interfaces, `GroupLayout` allows you to create **groups** that contain sequential or parallel elements. Within a group you can have GUI components, other groups and gaps. Placing a group within another group is similar to building a GUI using nested containers, such as a `JPanel` that contains other `JPanel`s, which in turn contain GUI components.

When you create a group, you can specify the **alignment** of the group’s elements. Class `GroupLayout` contains four constants for this purpose—`LEADING`, `TRAILING`, `CENTER` and `BASELINE`. The constant `BASELINE` applies only to vertical orientations. In horizontal orientation, the constants `LEADING`, `TRAILING` and `CENTER` represent left justified, right justified and centered, respectively. In vertical orientation, `LEADING`, `TRAILING` and `CENTER` align the components at their tops, bottoms or vertical centers, respectively. Aligning components with `BASELINE` indicates they should be aligned using the baseline of the font for the components’ text. For more information about font baselines, see Section 15.4.

Spacing

`GroupLayout` by default uses the recommended GUI design guidelines of the underlying platform for spacing between components. The `addGap` method of `GroupLayout` nested classes `GroupLayout.Group`, `GroupLayout.SequentialGroup` and `GroupLayout.ParallelGroup` allows you to control the spacing between components.

Sizing Components

By default, `GroupLayout` uses each component’s `getMinimumSize`, `getMaximumSize` and `getPreferredSize` methods to help determine the component’s size. You can override the default settings.

I.3 Building a ColorChooser

We now present a `ColorChooser` application to demonstrate the `GroupLayout` layout manager. The application consists of three `JSlider` objects, each representing the values from 0 to 255 for specifying the red, green and blue values of a color. The selected values for each `JSlider` will be used to display a filled rectangle of the specified color. We build

the application using NetBeans. For an more detailed introduction to developing GUI applications in the NetBeans IDE, see www.netbeans.org/kb/trails/matisse.html.

Create a New Project

Begin by opening a new NetBeans project. Select **File > New Project...** In the **New Project** dialog, choose **Java** from the **Categories** list and **Java Application** from the **Projects** list then click **Next >**. Specify **CoolorChooser** as the project name and uncheck the **Create Main Class** checkbox. You can also specify the location of your project in the **Project Location** field. Click **Finish** to create the project.

Add a New Subclass of JFrame to the Project

In the IDE's **Projects** tab just below the **File** menu and toolbar (Fig. I.2), expand the **Source Packages** node. Right-click the **<default package>** node that appears and select **New > JFrame Form**. In the **New JFrame Form** dialog, specify **CoolorChooser** as the class name and click **Finish**. This subclass of **JFrame** will display the application's GUI components. The NetBeans window should now appear similar to Fig. I.3 with the **CoolorChooser** class shown in **Design** view. The **Source** and **Design** buttons at the top of the **CoolorChooser.java** window allow you to switch between editing the source code and designing the GUI.

Design view shows only the **CoolorChooser**'s client area (i.e., the area that will appear inside the window's borders). To build a GUI visually, you can drag GUI components from the **Palette** window onto the client area. You can configure the properties of each component by selecting it, then modifying the property values that appear in the **Properties** window (Fig. I.3). When you select a component, the **Properties** window displays three buttons—**Properties**, **Bindings**, **Events**, **Code** (see Fig. I.4)—that enable you to configure various aspects of the component.

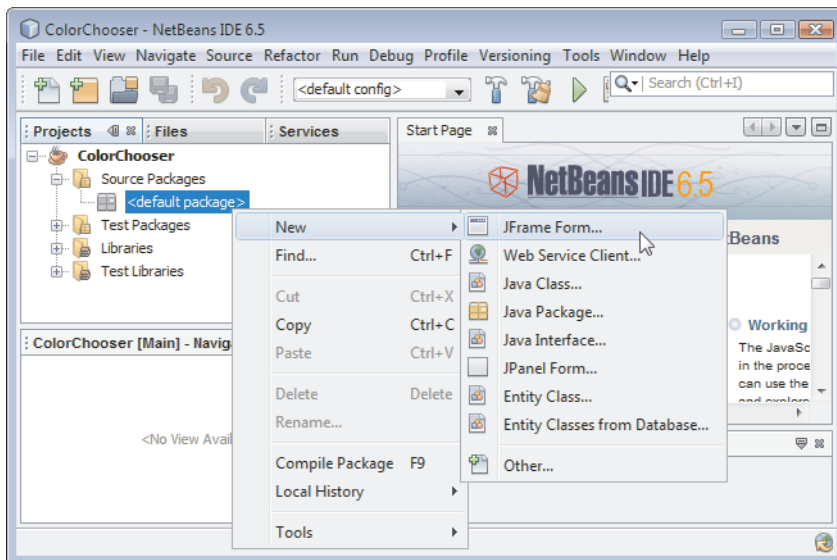


Fig. I.2 | Adding a new **JFrame Form** to the **CoolorChooser** project.

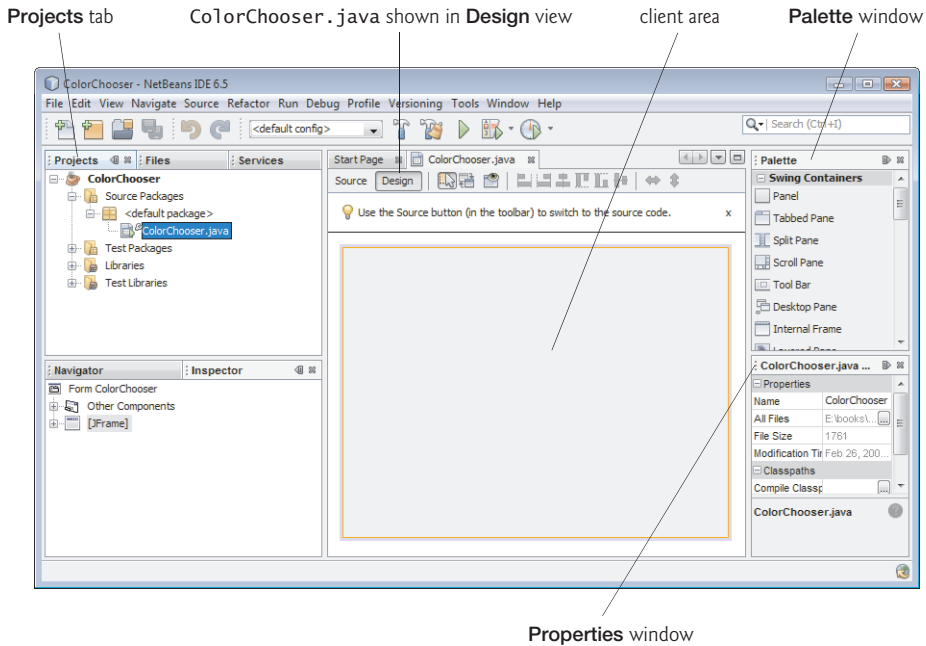


Fig. I.3 | Class ColorChooser shown in the NetBeans Design view.

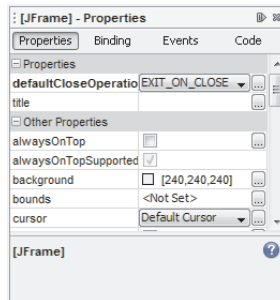


Fig. I.4 | Properties window with buttons that enable you to configure various aspects of the component.

Build the GUI

Drag three **Sliders** (objects of class `JSlider`) from the **Palette** onto the `JFrame` (you may need to scroll through the **Palette**). As you drag components near the edges of the client area or near other components, NetBeans displays **guide lines** (Fig. I.5) that show you the recommended distances and alignments between the component you are dragging, the edges of the client area and other components. As you follow the steps to build the GUI, use the guide lines to arrange the components into three rows and three columns as in Fig. I.6. Use the **Properties** window to rename the `JSlider`s to `redJSlider`, `greenJSlider` and `blueJSlider`. Select the first `JSlider`, then click the **Code** button in the **Properties** window and change the **Variable Name** property to `redSlider`. Repeat this process to re-

name the other two JSliders. Then, click the **Properties** button in the **Properties** window, select each JSlider and change its **maximum** property to 255 so that it will produce values in the range 0–255, and change its **value** property to 0 so the JSlider's thumb will initially be at the left of the JSlider.

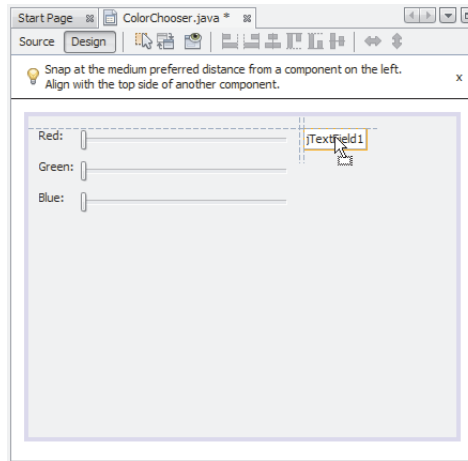


Fig. I.5 | Positioning the first JTextField.



Fig. I.6 | Layout of the JLabels, JSliders and JTextFields.

Drag three **Labels** (objects of class JLabel) from the **Palette** to the JFrame to label each JSlider with the color it represents. Name the JLabels redJLabel, greenJLabel and blueJLabel, respectively. Each JLabel should be placed to the left of the corresponding JSlider (Fig. I.6). Change each JLabel's **text** property either by double clicking the JLabel and typing the new text, or by selecting the JLabel and changing the **text** property in the **Properties** window.

Add a **Text Field** (an object of class JTextField) next to each of the JSliders to display the value of the slider. Name the JTextFields redJTextField, greenJTextField and blueJTextField, respectively. Change each JTextField's **text** property to 0 using the same techniques as you did for the JLabels. Change each JTextField's **columns** property to 4.

Double click the border of the client area to display the **Set Form Designer Size** dialog and change the first number (which represents the width) to 410, then click **OK**. This makes the client area wide enough to accommodate the **Panel** (an object of class JPanel) you'll add next. Finally, add a **Panel** named colorJPanel to the right of this group of components. Use the guide lines as shown in Fig. I.7 to place the JPanel. Change this JPanel's background color to display the selected color. Finally, drag the bottom border of the client area toward the top of the **Design** area until you see the snap-to line that shows the

recommended height of the client area (based on the components in the client area) as shown in Fig. I.8.

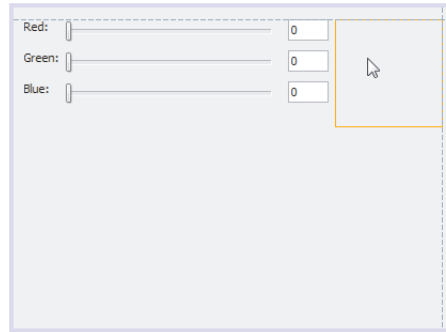


Fig. I.7 | Positioning the JPanel1.



Fig. I.8 | Setting the height of the client area.

Editing the Source Code and Adding Event Handlers

The IDE automatically generated the GUI code, including methods for initializing components and aligning them using the `GroupLayout` layout manager. We must add the desired functionality to the components' event handlers. To add an event handler for a component, right click it and position the mouse over the **Events** option in the pop-up menu. You can then select the category of event you wish to handle and the specific event within that category. For example, to add the `JSlider` event handlers for this example, right click each `JSlider` and select **Events > Change > stateChanged**. When you do this, NetBeans adds a `ChangeListener` to the `JSlider` and switches from **Design** view to **Source** view where you can place code in the event handler. Use the **Design** button to return to **Design** view and repeat the preceding steps to add the event handlers for the other two `JSliders`. To complete the event handlers, first add the method in Fig. I.9. In each `JSlider` event handler set the corresponding `TextField` to the new value of the `JSlider`, then call method `changeColor`. Finally, in the constructor after the call to `initComponents`, add the line

```
colorJPanel.setBackground( java.awt.Color.BLACK );
```

Figure I.10 shows the completed `ColorChooser` class as it's generated in NetBeans. We did not restyle the code to match our coding conventions that you've seen throughout the book.

```
1  // changes the colorJPanel's background color based on the current
2  // values of the JSliders
3  public void changeColor()
4  {
5      colorJPanel.setBackground( new java.awt.Color(
6          redJSlider.getValue(), greenJSlider.getValue(),
7          blueJSlider.getValue() ) );
8  } // end method changeColor
```

Fig. I.9 | Method that changes the `colorJPanel`'s background color based on the values of the three `JSliders`.

```
1  /*
2   * ColorChooser.java
3   *
4   * Created on Feb 26, 2009, 11:05:35 AM
5   */
6  /**
7   *
8   * @author paul
9   */
10 public class ColorChooser extends javax.swing.JFrame
11 {
12     /** Creates new form ColorChooser */
13     public ColorChooser()
14     {
15         initComponents();
16         colorJPanel.setBackground( java.awt.Color.BLACK );
17     }
18
19     // changes the colorJPanel's background color based on the current
20     // values of the JSliders
21     public void changeColor()
22     {
23         colorJPanel.setBackground( new java.awt.Color(
24             redJSlider.getValue(), greenJSlider.getValue(),
25             blueJSlider.getValue() ) );
26     } // end method changeColor
27
28     /** This method is called from within the constructor to
29      * initialize the form.
30      * WARNING: Do NOT modify this code. The content of this method is
31      * always regenerated by the Form Editor.
32      */
33     @SuppressWarnings("unchecked")
34     // <editor-fold defaultstate="collapsed" desc="Generated Code">
35     private void initComponents() {
36
37         redJSlider = new javax.swing.JSlider();
38         greenJSlider = new javax.swing.JSlider();
39         blueJSlider = new javax.swing.JSlider();
40         redJLabel = new javax.swing.JLabel();
```

Fig. I.10 | `ColorChooser` class that uses `GroupLayout` for its GUI layout. (Part I of 5.)

```

41     greenJLabel = new javax.swing.JLabel();
42     blueJLabel = new javax.swing.JLabel();
43     redJTextField = new javax.swing.JTextField();
44     greenJTextField = new javax.swing.JTextField();
45     blueJTextField = new javax.swing.JTextField();
46     colorJPanel = new javax.swing.JPanel();
47
48     setDefaultCloseOperation(
javax.swing.WindowConstants.EXIT_ON_CLOSE);
49
50     redJSlider.setMaximum(255);
51     redJSlider.setValue(0);
52     redJSlider.addChangeListener(
new javax.swing.event.ChangeListener() {
53         public void stateChanged(javax.swing.event.ChangeEvent evt) {
54             redJSliderStateChanged(evt);
55         }
56     });
57
58     greenJSlider.setMaximum(255);
59     greenJSlider.setValue(0);
60     greenJSlider.addChangeListener(
new javax.swing.event.ChangeListener() {
61         public void stateChanged(javax.swing.event.ChangeEvent evt) {
62             greenJSliderStateChanged(evt);
63         }
64     });
65
66     blueJSlider.setMaximum(255);
67     blueJSlider.setValue(0);
68     blueJSlider.addChangeListener(
new javax.swing.event.ChangeListener() {
69         public void stateChanged(javax.swing.event.ChangeEvent evt) {
70             blueJSliderStateChanged(evt);
71         }
72     });
73
74     redJLabel.setText("Red:");
75
76     greenJLabel.setText("Green:");
77
78     blueJLabel.setText("Blue:");
79
80     redJTextField.setColumns(4);
81     redJTextField.setText("0");
82
83     greenJTextField.setColumns(4);
84     greenJTextField.setText("0");
85
86     blueJTextField.setColumns(4);
87     blueJTextField.setText("0");
88
89     javax.swing.GroupLayout colorJPanelLayout =
new javax.swing.GroupLayout(colorJPanel);

```

Fig. I.10 | ColorChooser class that uses GroupLayout for its GUI layout. (Part 2 of 5.)

```

90         colorJPanel.setLayout(colorJPanelLayout);
91         colorJPanelLayout.setHorizontalGroup(
92             colorJPanelLayout.createParallelGroup(
93                 javax.swing.GroupLayout.Alignment.LEADING)
94                 .addGap(0, 100, Short.MAX_VALUE)
95             );
96         colorJPanelLayout.setVerticalGroup(
97             colorJPanelLayout.createParallelGroup(
98                 javax.swing.GroupLayout.Alignment.LEADING)
99                 .addGap(0, 100, Short.MAX_VALUE)
100             );
101         javax.swing.GroupLayout layout =
102         new javax.swing.GroupLayout(getContentPane());
103         getContentPane().setLayout(layout);
104         layout.setHorizontalGroup(
105             layout.createParallelGroup(
106                 javax.swing.GroupLayout.Alignment.LEADING)
107                 .addGroup(layout.createSequentialGroup()
108                     .addContainerGap()
109                     .addGroup(layout.createParallelGroup(
110                         javax.swing.GroupLayout.Alignment.LEADING)
111                         .addComponent(greenJLabel)
112                         .addComponent(blueJLabel)
113                         .addComponent(redJLabel))
114                     .addPreferredGap(
115                         javax.swing.LayoutStyle.ComponentPlacement.RELATED)
116                     .addGroup(layout.createParallelGroup(
117                         javax.swing.GroupLayout.Alignment.LEADING)
118                         .addGroup(layout.createSequentialGroup()
119                             .addComponent(blueJSlider,
120                                 javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
121                                 javax.swing.GroupLayout.PREFERRED_SIZE)
122                             .addPreferredGap(
123                                 javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
124                             .addComponent(blueJTextField,
125                                 javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
126                                 javax.swing.GroupLayout.PREFERRED_SIZE))
127                         .addGroup(layout.createSequentialGroup()
128                             .addComponent(greenJSlider,
129                                 javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
130                                 javax.swing.GroupLayout.PREFERRED_SIZE)
131                             .addPreferredGap(
132                                 javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
133                             .addComponent(greenJTextField,
134                                 javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
135                                 javax.swing.GroupLayout.PREFERRED_SIZE))
136                         .addGroup(layout.createSequentialGroup()
137                             .addComponent(redJSlider,
138                                 javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
139                                 javax.swing.GroupLayout.PREFERRED_SIZE)
140                             .addPreferredGap(
141                                 javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

```

Fig. I.10 | ColorChooser class that uses GroupLayout for its GUI layout. (Part 3 of 5.)

```

123         .addComponent(redJTextField,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)))
124         .addPreferredGap(
javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
125         .addComponent(colorJPanel,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
126         .addContainerGap()
127     );
128     layout.setVerticalGroup(
129         layout.createParallelGroup(
javax.swing.GroupLayout.Alignment.LEADING)
130         .addGroup(layout.createSequentialGroup()
131             .addContainerGap()
132             .addGroup(layout.createParallelGroup(
javax.swing.GroupLayout.Alignment.LEADING)
133                 .addComponent(colorJPanel,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
134                 .addGroup(layout.createSequentialGroup()
135                     .addGroup(layout.createParallelGroup(
javax.swing.GroupLayout.Alignment.LEADING)
136                         .addComponent(redJSlider,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
137                         .addComponent(redJLabel)
138                         .addComponent(redJTextField,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
139                     .addPreferredGap(
javax.swing.LayoutStyle.ComponentPlacement.RELATED)
140                     .addGroup(layout.createParallelGroup(
javax.swing.GroupLayout.Alignment.LEADING)
141                         .addComponent(greenJSlider,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
142                         .addComponent(greenJLabel)
143                         .addComponent(greenJTextField,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
144                     .addPreferredGap(
javax.swing.LayoutStyle.ComponentPlacement.RELATED)
145                     .addGroup(layout.createParallelGroup(
javax.swing.GroupLayout.Alignment.LEADING)
146                         .addComponent(blueJTextField,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
147                         .addComponent(blueJSlider,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
148                         .addComponent(blueJLabel))))))

```

Fig. I.10 | ColorChooser class that uses GroupLayout for its GUI layout. (Part 4 of 5.)

```

149         .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
150     );
151
152     pack();
153     ///  

154
155     private void redJSliderStateChanged(javax.swing.event.ChangeEvent
evt) {
156         redJTextField.setText( String.valueOf( redJSlider.getValue() ) );
157         changeColor();
158     }
159
160     private void greenJSliderStateChanged(javax.swing.event.ChangeEvent
evt) {
161         greenJTextField.setText( String.valueOf( greenJSlider.getValue() )
);
162         changeColor();
163     }
164
165     private void blueJSliderStateChanged(javax.swing.event.ChangeEvent
evt) {
166         blueJTextField.setText( String.valueOf( blueJSlider.getValue() ) );
167         changeColor();
168     }
169
170     /**
171      * @param args the command line arguments
172      */
173     public static void main( String args[] )
174     {
175         java.awt.EventQueue.invokeLater( new Runnable()
176         {
177             public void run()
178             {
179                 new ColorChooser().setVisible( true );
180             }
181         } );
182     }
183
184
185     ///  

186     private javax.swing.JLabel blueJLabel;
187     private javax.swing.JSlider blueJSlider;
188     private javax.swing.JTextField blueJTextField;
189     private javax.swing.JPanel colorJPanel;
190     private javax.swing.JLabel greenJLabel;
191     private javax.swing.JSlider greenJSlider;
192     private javax.swing.JTextField greenJTextField;
193     private javax.swing.JSlider redJSlider;
194     private javax.swing.JTextField redJTextField;
195     private javax.swing.JLabel redJLabel;
196     ///  

197 }

```

Fig. I.10 | ColorChooser class that uses GroupLayout for its GUI layout. (Part 5 of 5.)

Method `initComponents` (lines 35–153) was entirely generated by NetBeans based on your interactions with the GUI designer. This method contains the code that creates and formats the GUI. Lines 37–87 construct and initialize the GUI components. Lines 89–153 specify the layout of those components using `GroupLayout` . Lines 102–127 specify the horizontal group and lines 128–150 specify the vertical group. Notice how complex the code is. More and more software development is done with tools that generate complex code like this, saving you the time and effort of doing it yourself.

We manually added the statement that changes the `colorJPanel` 's background color in line 16 and the `changeColor` method in lines 21–26. When the user moves the thumb on one of the `JSliders` , the `JSlider` 's event handler sets the text in its corresponding `JTextField` to the `JSlider` 's new value (lines 156, 161 and 166), then calls method `changeColor` (lines 157, 162 and 167) to update the `colorJPanel` 's background color. Method `changeColor` gets the current value of each `JSlider` (lines 24–25) and uses these values as the arguments to the `Color` constructor to create a new `Color` .

I.4 GroupLayout Web Resources

weblogs.java.net/blog/tpavek/archive/2006/02/getting_to_know_1.html

Part 1 of Tomas Pavek's `GroupLayout` blog post overviews `GroupLayout` theory behind.

weblogs.java.net/blog/tpavek/archive/2006/03/getting_to_know.html

Part 2 of Tomas Pavek's `GroupLayout` blog post presents a complete GUI implemented with `GroupLayout` .

java.sun.com/javase/6/docs/api/javax/swing/GroupLayout.html

API documentation for class `GroupLayout` .

wiki.java.net/bin/view/Javadesktop/GroupLayoutExample

Provides an Address Book demo of a GUI built manually with `GroupLayout` with source code.

weblogs.java.net/blog/claudio/archive/nb-layouts.html

Flash-based `GroupLayout` Tutorial.

www.developer.com/java/ent/article.php/3589961

Tutorial: "Building Java GUIs with Matisse: A Gentle Introduction," by Dick Wall.

[myeclipseide.com/enterpriseworkbench/help/index.jsp?](http://myeclipseide.com/enterpriseworkbench/help/index.jsp?topic=/com.genuitec.eclipse.dehory.doc/doc/quickstart/index.html)

topic=/com.genuitec.eclipse.dehory.doc/doc/quickstart/index.html

Tutorial: "Matisse4MyEclipse—Swing RCP Development Quickstart," from MyEclipse. Introduces a version of the Matisse GUI designer for the Eclipse IDE.