I

GroupLayout

I.1 Introduction

Java SE 6 includes a powerful 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.

I.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 `JButton`s 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).

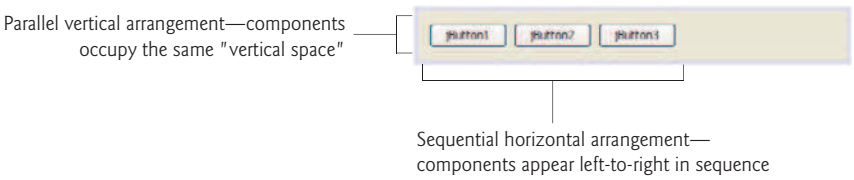

Fig. I.1 | JButtons arranged sequentially for their horizontal orientation and in parallel for their vertical orientation.

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`.

Creating 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 `ColorChooser` 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.

Adding 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 `ColorChooser` 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 `ColorChooser` class shown in **Design** view. The **Source** and **Design** buttons at the top of the `ColorChooser.java` window allow you to switch between editing the source code and designing the GUI.

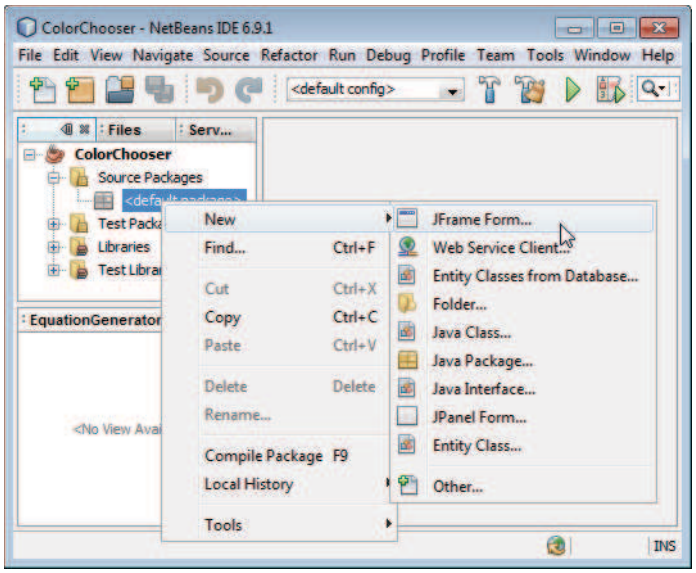

Fig. I.2 | Adding a new **JFrame Form** to the `ColorChooser` project.

Design view shows only the `ColorChooser`'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.

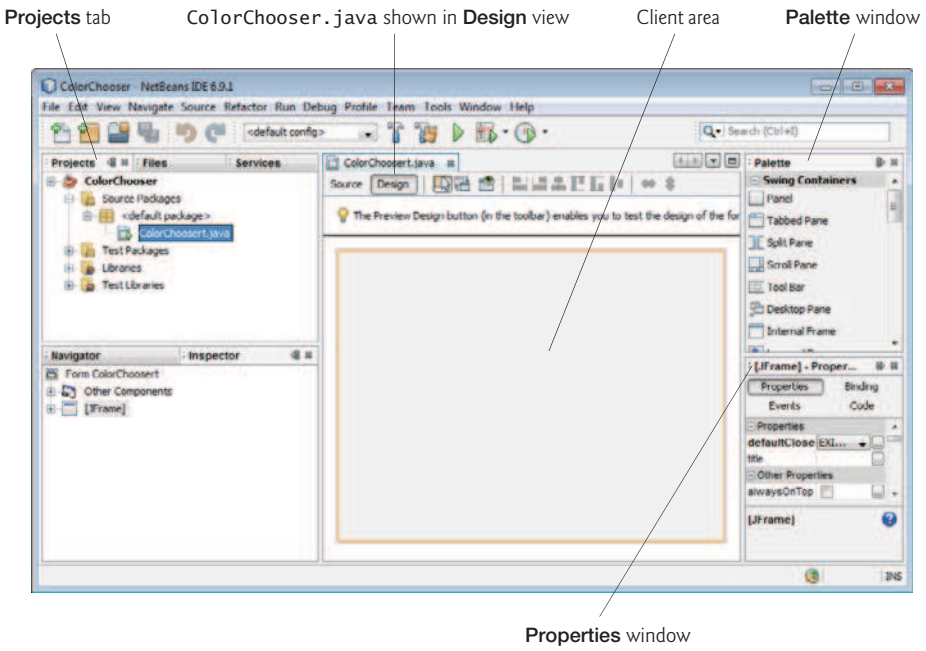

Fig. I.3 | Class `ColorChooser` shown in the NetBeans **Design** view.

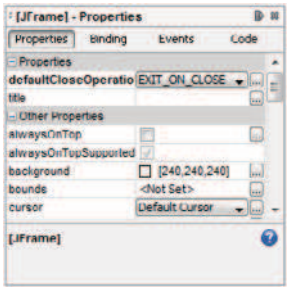

Fig. I.4 | **Properties** window with buttons that enable you to configure various aspects of the component.

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're 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. Next, rename the `JSliders` to `redJSlider`, `greenJSlider` and `blueJSlider`. To do so, 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 rename 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`.

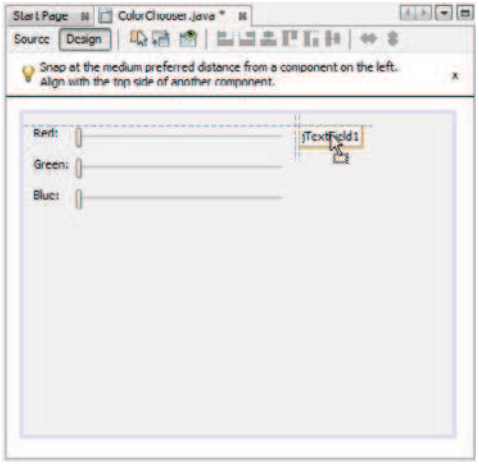

Fig. I.5 | Positioning the first `JTextField`.

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

Drag three **Label**s (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`. To align each **Label**, **Slider** and **Text Field** nicely, you can select them by dragging the mouse across all three and use the alignment buttons at the top of the Design window.

Next, 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 black (the initially selected RGB color). Finally, drag the bottom-right border of the client area toward the top-left of the **Design** area until you see the snap-to lines that

show the recommended client area dimensions (which are based on the components in the client area) as shown in Fig. I.8.

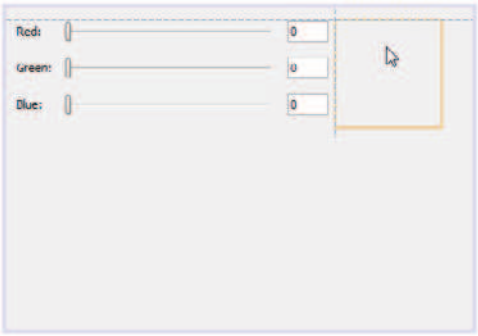

Fig. I.7 | Positioning the `JPanel`.

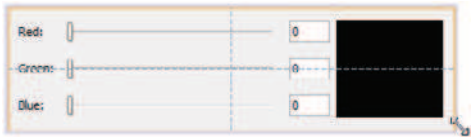

Fig. I.8 | Setting the height of the client area.

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 following the class's constructor. In each `JSlider` event handler set the corresponding `JTextField` to the new value of the `JSlider`, then call method `changeColor`. 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. You can now run the program to see it in action. Drag each slider and watch the `colorJPanel`'s background color change.

Method `initComponents` (lines 39–162) 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 41–93 construct and initialize the GUI components. Lines

95–161 specify the layout of those components using `GroupLayout`. Lines 108–136 specify the horizontal group and lines 137–159 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 `changeColor` method in lines 25–30. 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 166, 172 and 178), then calls method `changeColor` (lines 167, 173 and 179) to update the `colorJPanel`'s background color. Method `changeColor` gets the current value of each `JSlider` (lines 28–29) and uses these values as the arguments to the `Color` constructor to create a new `Color`.

```
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   * To change this template, choose Tools | Templates
3   * and open the template in the editor.
4   */
5
6  /*
7   * ColorChooser.java
8   *
9   * Created on Feb 8, 2011, 9:20:27 AM
10  */
11
12 /**
13  *
14  * @author Paul Deitel
15  */
16 public class ColorChooser extends javax.swing.JFrame {
17
18     /** Creates new form ColorChooser */
19     public ColorChooser() {
20         initComponents();
21     }
22
23     // changes the colorJPanel's background color based on the current
24     // values of the JSliders
25     public void changeColor()
26     {
```

Fig. I.10 | `ColorChooser` class that uses `GroupLayout` for its GUI layout. (Part 1 of 6.)

```
27          colorJPanel.setBackground( new java.awt.Color(
28             redJSlider.getValue(), greenJSlider.getValue(),
29             blueJSlider.getValue() ) );
30       } // end method changeColor
31
32       /** This method is called from within the constructor to
33        * initialize the form.
34        * WARNING: Do NOT modify this code. The content of this method is
35        * always regenerated by the Form Editor.
36        */
37       @SuppressWarnings("unchecked")
38      // <editor-fold defaultstate="collapsed" desc="Generated Code">
39      private void initComponents() {
40
41         redJSlider = new javax.swing.JSlider();
42         greenJSlider = new javax.swing.JSlider();
43         blueJSlider = new javax.swing.JSlider();
44         redJLabel = new javax.swing.JLabel();
45         greenJLabel = new javax.swing.JLabel();
46         blueJLabel = new javax.swing.JLabel();
47         redJTextField = new javax.swing.JTextField();
48         greenJTextField = new javax.swing.JTextField();
49         blueJTextField = new javax.swing.JTextField();
50         colorJPanel = new javax.swing.JPanel();
51
52         setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
53
54         redJSlider.setMaximum(255);
55         redJSlider.setValue(0);
56         redJSlider.addChangeListener(new javax.swing.event.ChangeListener()
{
57            public void stateChanged(javax.swing.event.ChangeEvent evt) {
58               redJSliderStateChanged(evt);
59            }
60         });
61
62         greenJSlider.setMaximum(255);
63         greenJSlider.setValue(0);
64         greenJSlider.addChangeListener(new
javax.swing.event.ChangeListener() {
65            public void stateChanged(javax.swing.event.ChangeEvent evt) {
66               greenJSliderStateChanged(evt);
67            }
68         });
69
70         blueJSlider.setMaximum(255);
71         blueJSlider.setValue(0);
72        blueJSlider.addChangeListener(new javax.swing.event.ChangeListener()
{
73            public void stateChanged(javax.swing.event.ChangeEvent evt) {
74               blueJSliderStateChanged(evt);
75            }
76         });
```

Fig. I.10 | `ColorChooser` class that uses `GroupLayout` for its GUI layout. (Part 2 of 6.)

```
77
78          redJLabel.setText("Red:");
79
80          greenJLabel.setText("Green:");
81
82          blueJLabel.setText("Blue:");
83
84          redJTextField.setColumns(4);
85          redJTextField.setText("0");
86
87          greenJTextField.setColumns(4);
88          greenJTextField.setText("0");
89
90          blueJTextField.setColumns(4);
91          blueJTextField.setText("0");
92
93          colorJPanel.setBackground(new java.awt.Color(0, 0, 0));
94
95          javax.swing.GroupLayout colorJPanelLayout = new
javax.swing.GroupLayout(colorJPanel);
96          colorJPanel.setLayout(colorJPanelLayout);
97          colorJPanelLayout.setHorizontalGroup(
98
colorJPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADIN
G)
99             .addGap(0, 100, Short.MAX_VALUE)
100         );
101         colorJPanelLayout.setVerticalGroup(
102
colorJPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADIN
G)
103            .addGap(0, 91, Short.MAX_VALUE)
104         );
105
106         javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
107         getContentPane().setLayout(layout);
108         layout.setHorizontalGroup(
109
layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
110            .addGroup(layout.createSequentialGroup()
111
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING
)
112                  .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
113                     .addContainerGap()
114                     .addComponent(redJLabel)
115                     .addGap(20, 20, 20)
116                     .addComponent(redJSlider,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
```

Fig. I.10 | `ColorChooser` class that uses `GroupLayout` for its GUI layout. (Part 3 of 6.)

```
117
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
118                     .addComponent(redJTextField,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
119                 .addGroup(layout.createSequentialGroup()
120                     .addContainerGap()
121                     .addComponent(greenJLabel)
122
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
123                     .addComponent(greenJSlider,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
124                     .addGap(10, 10, 10)
125                     .addComponent(greenJTextField,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
126                 .addGroup(layout.createSequentialGroup()
127                     .addContainerGap()
128                     .addComponent(blueJLabel)
129                     .addGap(19, 19, 19)
130                     .addComponent(blueJSlider,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
131
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
132                     .addComponent(blueJTextField,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)))
133
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
134                 .addComponent(colorJPanel,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
135                 .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
136         );
137         layout.setVerticalGroup(
138
layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
139             .addGroup(layout.createSequentialGroup()
140                 .addContainerGap()
141
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILIN
G, false)
142                     .addComponent(colorJPanel,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
143                     .addGroup(javax.swing.GroupLayout.Alignment.LEADING,
layout.createSequentialGroup()
144
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.CENTER)
```

Fig. I.10 | `ColorChooser` class that uses `GroupLayout` for its GUI layout. (Part 4 of 6.)

```
145                         .addComponent(redJSlider,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
146                         .addComponent(redJTextField,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
147                         .addComponent(redJLabel))
148
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
149
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.CENTER)
150                         .addComponent(greenJSlider,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
151                         .addComponent(greenJLabel)
152                         .addComponent(greenJTextField,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
153
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
154
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.CENTER)
155                         .addComponent(blueJLabel)
156                         .addComponent(blueJSlider,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
157                         .addComponent(blueJTextField,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))))
158                 .addContainerGap())
159         );
160
161         pack();
162      }// </editor-fold>
163
164       private void redJSliderStateChanged(javax.swing.event.ChangeEvent evt)
165       {
166          redJTextField.setText( String.valueOf( redJSlider.getValue() ) );
167          changeColor();
168       }
169
170       private void greenJSliderStateChanged(javax.swing.event.ChangeEvent
evt)
171       {
172          greenJTextField.setText( String.valueOf( greenJSlider.getValue() )
);
173          changeColor();
174       }
175
176       private void blueJSliderStateChanged(javax.swing.event.ChangeEvent
evt)
177       {
178          blueJTextField.setText( String.valueOf( blueJSlider.getValue() ) );
```

Fig. I.10 | `ColorChooser` class that uses `GroupLayout` for its GUI layout. (Part 5 of 6.)

```
179          changeColor();
180      }
181
182      /**
183      * @param args the command line arguments
184      */
185      public static void main(String args[]) {
186          java.awt.EventQueue.invokeLater(new Runnable() {
187              public void run() {
188                  new ColorChooser().setVisible(true);
189              }
190          });
191      }
192
193     // Variables declaration - do not modify
194     private javax.swing.JLabel blueJLabel;
195     private javax.swing.JSlider blueJSlider;
196     private javax.swing.JTextField blueJTextField;
197     private javax.swing.JPanel colorJPanel;
198     private javax.swing.JLabel greenJLabel;
199     private javax.swing.JSlider greenJSlider;
200     private javax.swing.JTextField greenJTextField;
201     private javax.swing.JLabel redJLabel;
202     private javax.swing.JSlider redJSlider;
203     private javax.swing.JTextField redJTextField;
204     // End of variables declaration
205
206 }
```

Fig. I.10 | `ColorChooser` class that uses `GroupLayout` for its GUI layout. (Part 6 of 6.)

I.4 GroupLayout Web Resources

`download.oracle.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.

`www.developer.com/java/ent/article.php/3589961`
Tutorial: "Building Java GUIs with Matisse: A Gentle Introduction," by Dick Wall.