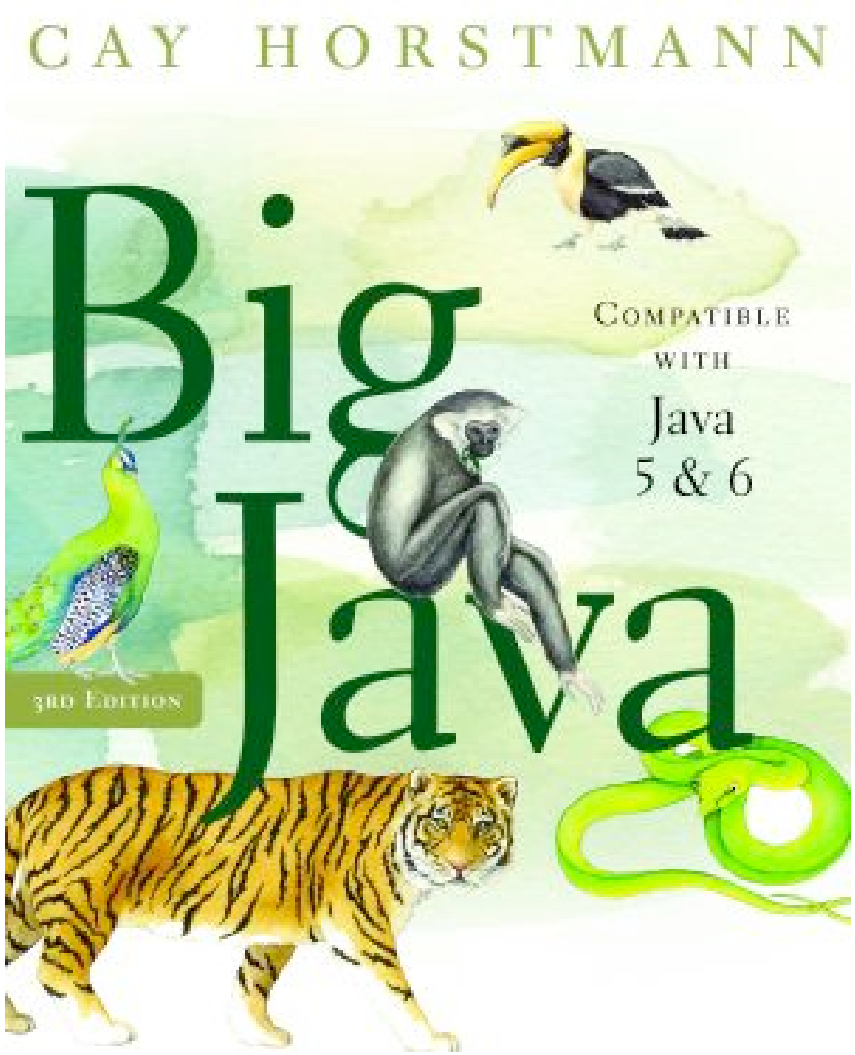


Chapter Eighteen: Graphical User Interfaces

Chapter Goals

- To understand the use of layout managers to arrange user-interface components in a container
- To become familiar with common user-interface components, such as buttons, combo boxes, text areas, and menus
- To build programs that handle events from user-interface components
- To learn how to browse the Java documentation

Layout Management

- Up to now, we have had limited control over layout of components
  - *When we used a panel, it arranged the components from the left to the right*
- User-interface components are arranged by placing them inside containers
  - *Containers can be placed inside larger containers*
- Each container has a *layout manager* that directs the arrangement of its components
- Three useful layout managers:
  - *border layout*
  - *flow layout*
  - *grid layout*

Layout Management

- By default, `JPanel` places components from left to right and starts a new row when needed
- Panel layout carried out by `FlowLayout` layout manager
- Can set other layout managers

```
panel.setLayout(new BorderLayout());
```

Border Layout

- Border layout groups container into five areas: center, north, west, south and east

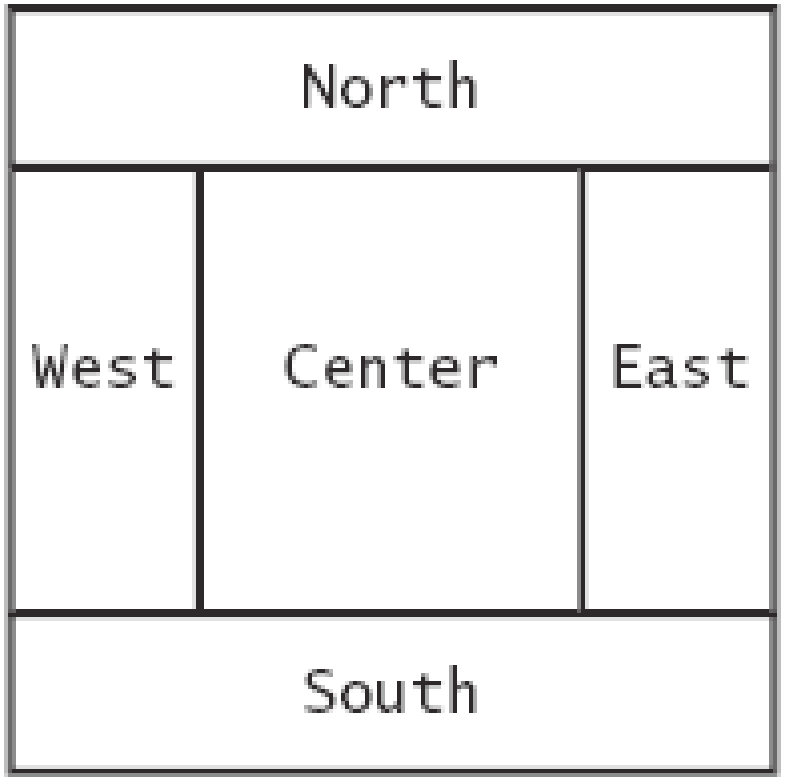


Figure 1
Components Expand to Fill Space in the Border Layout

Border Layout

- Default layout manager for a frame (technically, the frame's content pane)
- When adding a component, specify the position like this:

```
panel.add(component, BorderLayout.NORTH);
```

- Expands each component to fill the entire allotted area
  If that is not desirable, place each component inside a panel

Grid Layout

- Arranges components in a grid with a fixed number of rows and columns
- Resizes each component so that they all have same size
- Expands each component to fill the entire allotted area
- Add the components, row by row, left to right:

```
JPanel numberPanel = new JPanel();
numberPanel.setLayout(new GridLayout(4, 3));
numberPanel.add(button7);
numberPanel.add(button8);
numberPanel.add(button9);
numberPanel.add(button4);
. . .
```

Continued

Grid Layout

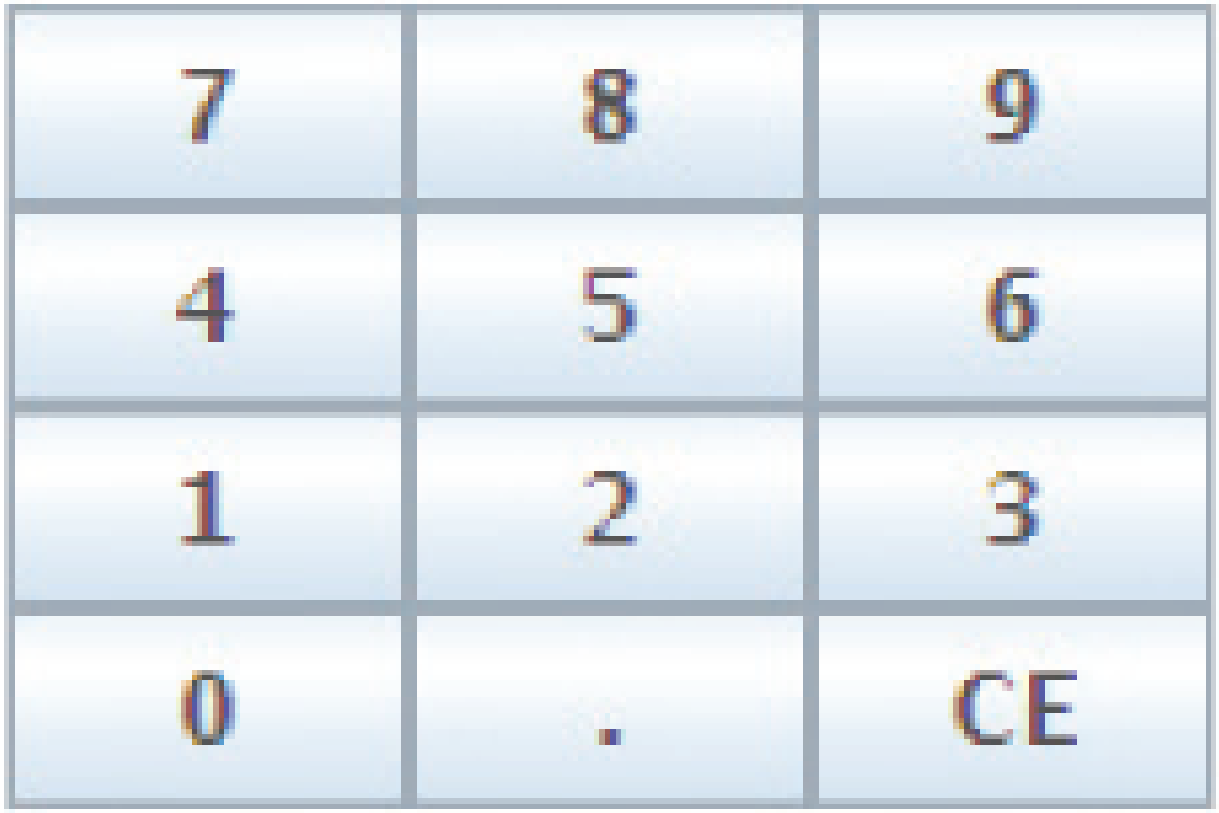


Figure 2 The Grid Layout

Grid Bag Layout

- Tabular arrangement of components
  - *Columns can have different sizes*
  - *Components can span multiple columns*
- Quite complex to use
- Not covered in the book
- Fortunately, you can create acceptable-looking layouts by nesting panels
  - *Give each panel an appropriate layout manager*
  - *Panels don't have visible borders*
  - *Use as many panels as needed to organize components*

Nesting Panels Example

Keypad from the ATM GUI in Chapter 12:

```
JPanel keypadPanel = new JPanel();
keypadPanel.setLayout(new BorderLayout());
buttonPanel = new JPanel();
buttonPanel.setLayout(new GridLayout(4, 3));
buttonPanel.add(button7);
buttonPanel.add(button8);
// . . .
keypadPanel.add(buttonPanel, BorderLayout.CENTER);
 JTextField display = new JTextField();
keypadPanel.add(display, BorderLayout.NORTH);
```

Continued

Nesting Panels Example (cont.)

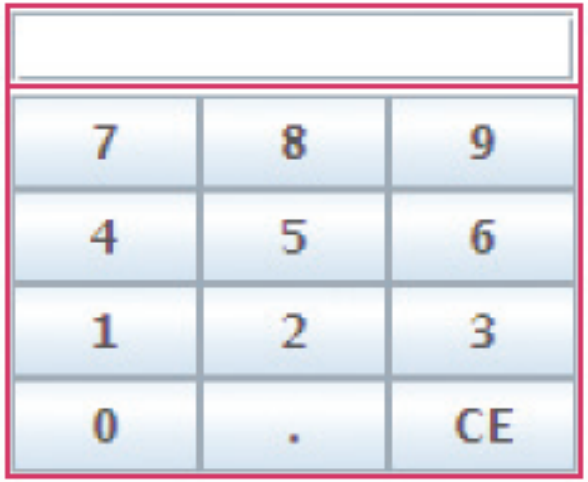


Figure 3 Nesting Panels

Self Check 18.1

How do you add two buttons to the north area of a frame?

Answer: First add them to a panel, then add the panel to the north end of a frame.

Self Check 18.2

How can you stack three buttons on top of each other?

Answer: Place them inside a panel with a `GridLayout` that has three rows and one column.

Choices

- Radio buttons
- Check boxes
- Combo boxes

Figure 4
A Combo Box, Check Boxes, and Radio Buttons

Radio Buttons

- For a small set of mutually exclusive choices, use radio buttons or a combo box
- In a radio button set, only one button can be selected at a time
- When a button is selected, previously selected button in set is automatically turned off

Continued

Radio Buttons (cont.)

- In previous figure, font sizes are mutually exclusive:

```
JRadioButton smallButton = new JRadioButton("Small");
JRadioButton mediumButton = new JRadioButton("Medium");
JRadioButton largeButton = new JRadioButton("Large");

// Add radio buttons into a ButtonGroup so that
// only one button in group is on at any time
ButtonGroup group = new ButtonGroup();
group.add(smallButton);
group.add(mediumButton);
group.add(largeButton);
```

Radio Buttons

- Button group does not place buttons close to each other on container
- It is your job to arrange buttons on screen
- `isSelected`: called to find out if a button is currently selected or not

```
if(largeButton.isSelected()) size = LARGE_SIZE
```

- Call `setSelected(true)` on a radio button in group before making the enclosing frame visible

Borders

- Place a border around a panel to group its contents visually
- `EtchedBorder`: three-dimensional etched effect
- Can add a border to any component, but most commonly to panels:

```
JPanel panel = new JPanel();
panel.setBorder(new EtchedBorder());
```

- `TitledBorder`: a border with a title

```
panel.setBorder(new TitledBorder(new EtchedBorder(),
       "Size"));
```

Check Boxes

- Two states: checked and unchecked
- Use one checkbox for a binary choice
- Use a group of check boxes when one selection does not exclude another
- Example: "bold" and "italic" in previous figure
- Construct by giving the name in the constructor:

```
JCheckBox italicCheckBox = new JCheckBox("Italic");
```

- Don't place into a button group

Combo Boxes

- For a large set of choices, use a combo box
  - *Uses less space than radio buttons*
- "Combo": combination of a list and a text field
  - *The text field displays the name of the current selection*

Figure 5
An Open Combo Box

Combo Boxes

- If combo box is editable, user can type own selection
  - *Use* `setEditable` *method*
- Add strings with `addItem` method:

```
JComboBox facenameCombo = new JComboBox();
facenameCombo.addItem("Serif");
facenameCombo.addItem("SansSerif");
. . .
```

- Get user selection with getSelectedItem (return type is Object)

```
String selectedString
      = (String) facenameCombo.getSelectedItem();
```

- Select an item with `setSelectedItem`

Radio Buttons, Check Boxes, and Combo Boxes

- They generate an `ActionEvent` whenever the user selects an item
- An example: `ChoiceFrame`

Continued

Radio Buttons, Check Boxes, and Combo Boxes (cont.)

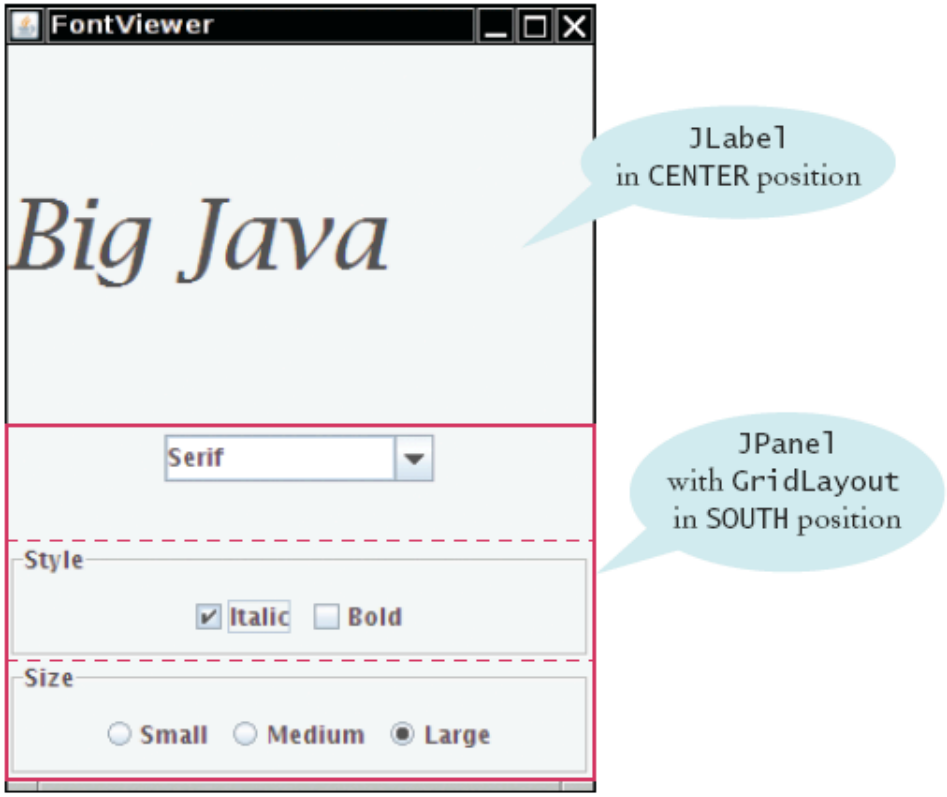


Figure 6
The Components of the FontViewerFrame

- *All components notify the same listener object*
- *When user clicks on any component, we ask each component for its current content*
- *Then redraw text sample with the new font*

Classes of the Font Choice Program

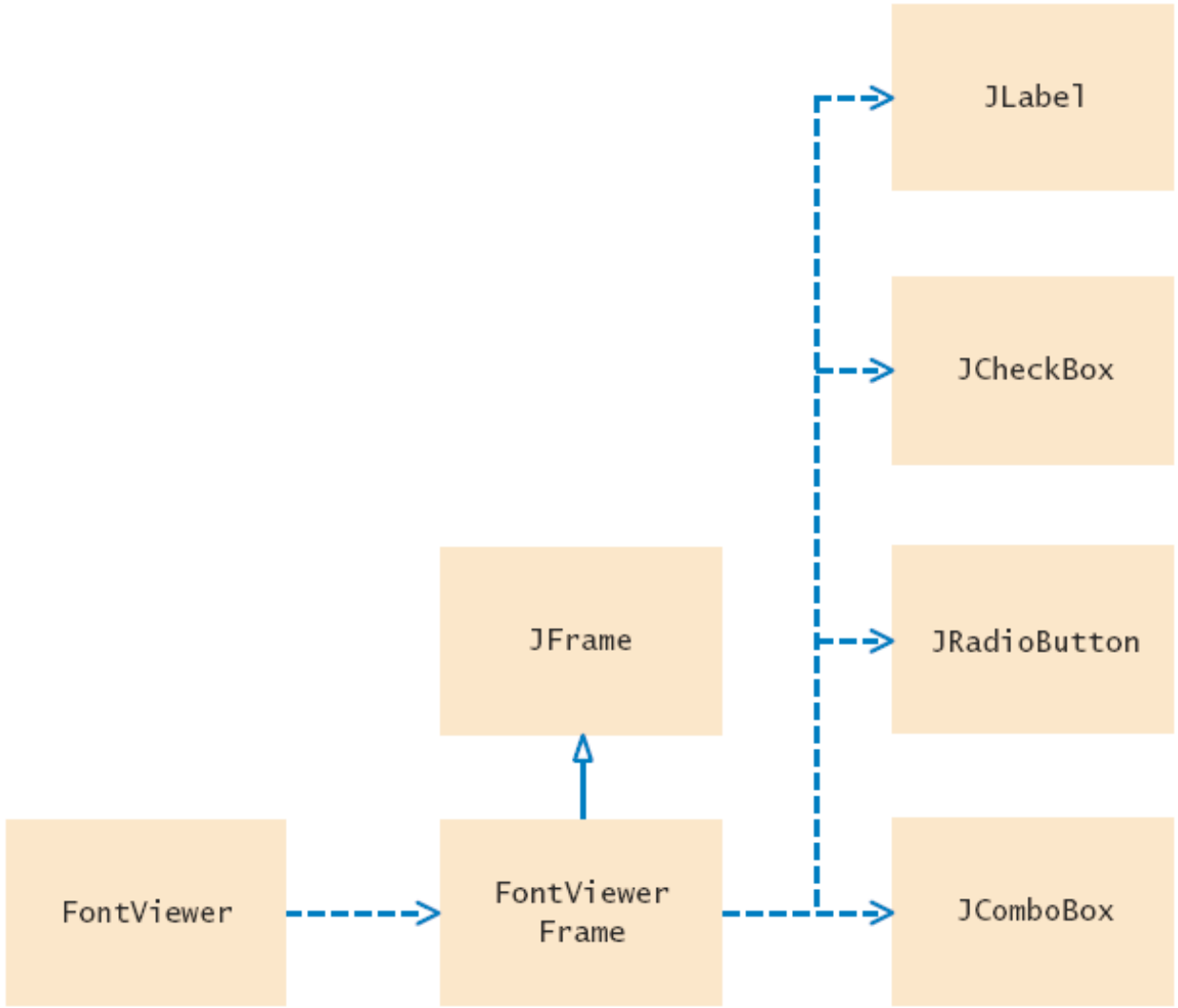


Figure 7
Classes of the Font Viewer Program

ch18/choice/FontViewer.java

```
01: import javax.swing.JFrame;
02:
03: /**
04:    This program allows the user to view font effects.
05: */
06: public class FontViewer
07: {
08:    public static void main(String[] args)
09:    {
10:       JFrame frame = new FontViewerFrame();
11:       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12:       frame.setTitle("FontViewer");
13:       frame.setVisible(true);
14:    }
15: }
16:
```

ch18/ choice/FontViewerFrame.java

```
001: import java.awt.BorderLayout;
002: import java.awt.Font;
003: import java.awt.GridLayout;
004: import java.awt.event.ActionEvent;
005: import java.awt.event.ActionListener;
006: import javax.swing.ButtonGroup;
007: import javax.swing.JButton;
008: import javax.swing.JCheckBox;
009: import javax.swing.JComboBox;
010: import javax.swing.JFrame;
011: import javax.swing.JLabel;
012: import javax.swing.JPanel;
013: import javax.swing.JRadioButton;
014: import javax.swing.border.EtchedBorder;
015: import javax.swing.border.TitledBorder;
016:
017: /**
018:    This frame contains a text field and a control panel
019:    to change the font of the text.
020: */
```

Continued

ch18/ choice/FontViewerFrame.java (cont.)

```
021: public class FontViewerFrame extends JFrame
022: {
023:    /**
024:       Constructs the frame.
025:    */
026:    public FontViewerFrame()
027:    {
028:       // Construct text sample
029:       sampleField = new JLabel("Big Java");
030:       add(sampleField, BorderLayout.CENTER);
031:
032:       // This listener is shared among all components
033:       class ChoiceListener implements ActionListener
034:       {
035:          public void actionPerformed(ActionEvent event)
036:          {
037:             setSampleFont();
038:          }
039:       }
040:
```

Continued

ch18/ choice/FontViewerFrame.java (cont.)

```
041:        listener = new ChoiceListener();
042:
043:        createControlPanel();
044:        setSampleFont();
045:        setSize(FRAME_WIDTH, FRAME_HEIGHT);
046:     }
047:
048:     /**
049:        Creates the control panel to change the font.
050:     */
051:     public void createControlPanel()
052:     {
053:        JPanel facenamePanel = createComboBox();
054:        JPanel sizeGroupPanel = createCheckBoxes();
055:        JPanel styleGroupPanel = createRadioButtons();
056:
057:        // Line up component panels
058:
059:        JPanel controlPanel = new JPanel();
060:        controlPanel.setLayout(new GridLayout(3, 1));
061:        controlPanel.add(facenamePanel);
```

Continued

ch18/ choice/FontViewerFrame.java (cont.)

```
062:        controlPanel.add(sizeGroupPanel);
063:        controlPanel.add(styleGroupPanel);
064:
065:        // Add panels to content pane
066:
067:        add(controlPanel, BorderLayout.SOUTH);
068:     }
069:
070:     /**
071:        Creates the combo box with the font style choices.
072:        @return the panel containing the combo box
073:     */
074:     public JPanel createComboBox()
075:     {
076:        facenameCombo = new JComboBox();
077:        facenameCombo.addItem("Serif");
078:        facenameCombo.addItem("SansSerif");
079:        facenameCombo.addItem("Monospaced");
080:        facenameCombo.setEditable(true);
081:        facenameCombo.addActionListener(listener);
082:
```

Continued

ch18/ choice/FontViewerFrame.java (cont.)

```
083:        JPanel panel = new JPanel();
084:        panel.add(facenameCombo);
085:        return panel;
086:     }
087:
088:     /**
089:        Creates the check boxes for selecting bold and italic styles.
090:        @return the panel containing the check boxes
091:     */
092:     public JPanel createCheckBoxes()
093:     {
094:        italicCheckBox = new JCheckBox("Italic");
095:        italicCheckBox.addActionListener(listener);
096:
097:        boldCheckBox = new JCheckBox("Bold");
098:        boldCheckBox.addActionListener(listener);
099:
100:       JPanel panel = new JPanel();
101:       panel.add(italicCheckBox);
102:       panel.add(boldCheckBox);
103:       panel.setBorder
104:          (new TitledBorder(new EtchedBorder(), "Style"));
```

Continued

ch18/ choice/FontViewerFrame.java (cont.)

```
105:
106:       return panel;
107:    }
108:
109:    /**
110:       Creates the radio buttons to select the font size
111:       @return the panel containing the radio buttons
112:    */
113:    public JPanel createRadioButtons()
114:    {
115:       smallButton = new JRadioButton("Small");
116:       smallButton.addActionListener(listener);
117:
118:       mediumButton = new JRadioButton("Medium");
119:       mediumButton.addActionListener(listener);
120:
121:       largeButton = new JRadioButton("Large");
122:       largeButton.addActionListener(listener);
123:       largeButton.setSelected(true);
124:
125:       // Add radio buttons to button group
126:
```

Continued

ch18/ choice/FontViewerFrame.java (cont.)

```
127:        ButtonGroup group = new ButtonGroup();
128:        group.add(smallButton);
129:        group.add(mediumButton);
130:        group.add(largeButton);
131:
132:        JPanel panel = new JPanel();
133:        panel.add(smallButton);
134:        panel.add(mediumButton);
135:        panel.add(largeButton);
136:        panel.setBorder
137:              (new TitledBorder(new EtchedBorder(), "Size"));
138:
139:        return panel;
140:     }
141:
142:     /**
143:        Gets user choice for font name, style, and size
144:        and sets the font of the text sample.
145:     */
146:     public void setSampleFont()
147:     {
```

Continued

ch18/ choice/FontViewerFrame.java (cont.)

```
148:         // Get font name
149:         String facename
150:               = (String) facenameCombo.getSelectedItem();
151:
152:         // Get font style
153:
154:         int style = 0;
155:         if (italicCheckBox.isSelected())
156:            style = style + Font.ITALIC;
157:         if (boldCheckBox.isSelected())
158:            style = style + Font.BOLD;
159:
160:         // Get font size
161:
162:         int size = 0;
163:
164:         final int SMALL_SIZE = 24;
165:         final int MEDIUM_SIZE = 36;
166:         final int LARGE_SIZE = 48;
167:
```

Continued

ch18/ choice/FontViewerFrame.java (cont.)

```
168:         if (smallButton.isSelected())
169:            size = SMALL_SIZE;
170:         else if (mediumButton.isSelected())
171:            size = MEDIUM_SIZE;
172:         else if (largeButton.isSelected())
173:            size = LARGE_SIZE;
174:
175:         // Set font of text field
176:
177:         sampleField.setFont(new Font(facename, style, size));
178:         sampleField.repaint();
179:      }
180:
181:      private JLabel sampleField;
182:      private JCheckBox italicCheckBox;
183:      private JCheckBox boldCheckBox;
184:      private JRadioButton smallButton;
185:      private JRadioButton mediumButton;
186:      private JRadioButton largeButton;
187:      private JComboBox facenameCombo;
188:      private ActionListener listener;
189:
```

Continued

ch18/ choice/FontViewerFrame.java (cont.)

```
190:    private static final int FRAME_WIDTH = 300;
191:    private static final int FRAME_HEIGHT = 400;
192: }
```

Self Check 18.3

What is the advantage of a `JComboBox` over a set of radio buttons? What is the disadvantage?

Answer: If you have many options, a set of radio buttons takes up a large area. A combo box can show many options without using up much space. But the user cannot see the options as easily.

Self Check 18.4

Why do all user interface components in the `FontViewerFrame` class share the same listener?

Answer: When any of the component settings is changed, the program simply queries all of them and updates the label.

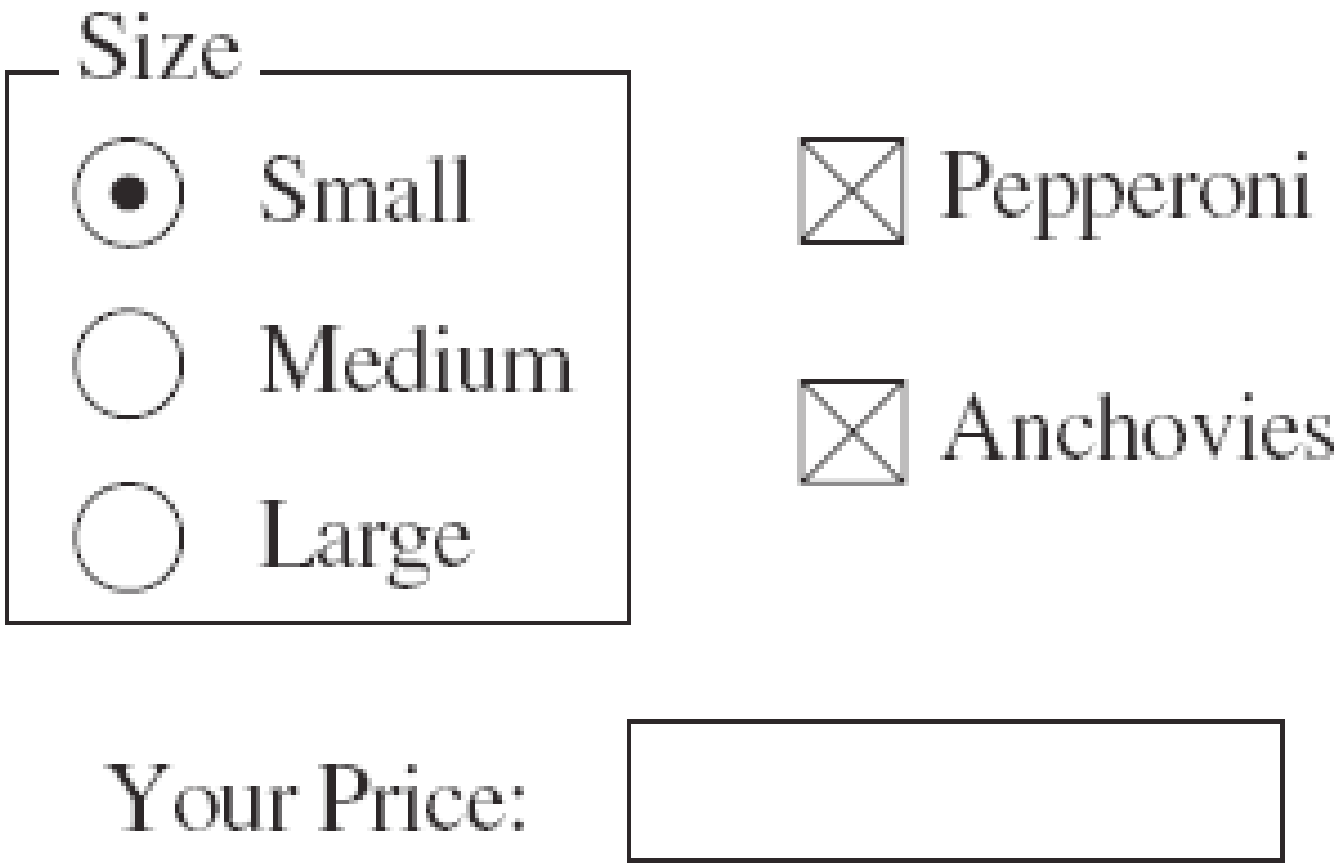
Self Check 18.5

Why was the combo box placed inside a panel? What would have happened if it had been added directly to the control panel?

Answer: To keep it from growing too large. It would have grown to the same width and height as the two panels below it.

How To 18.1 Laying Out a User Interface

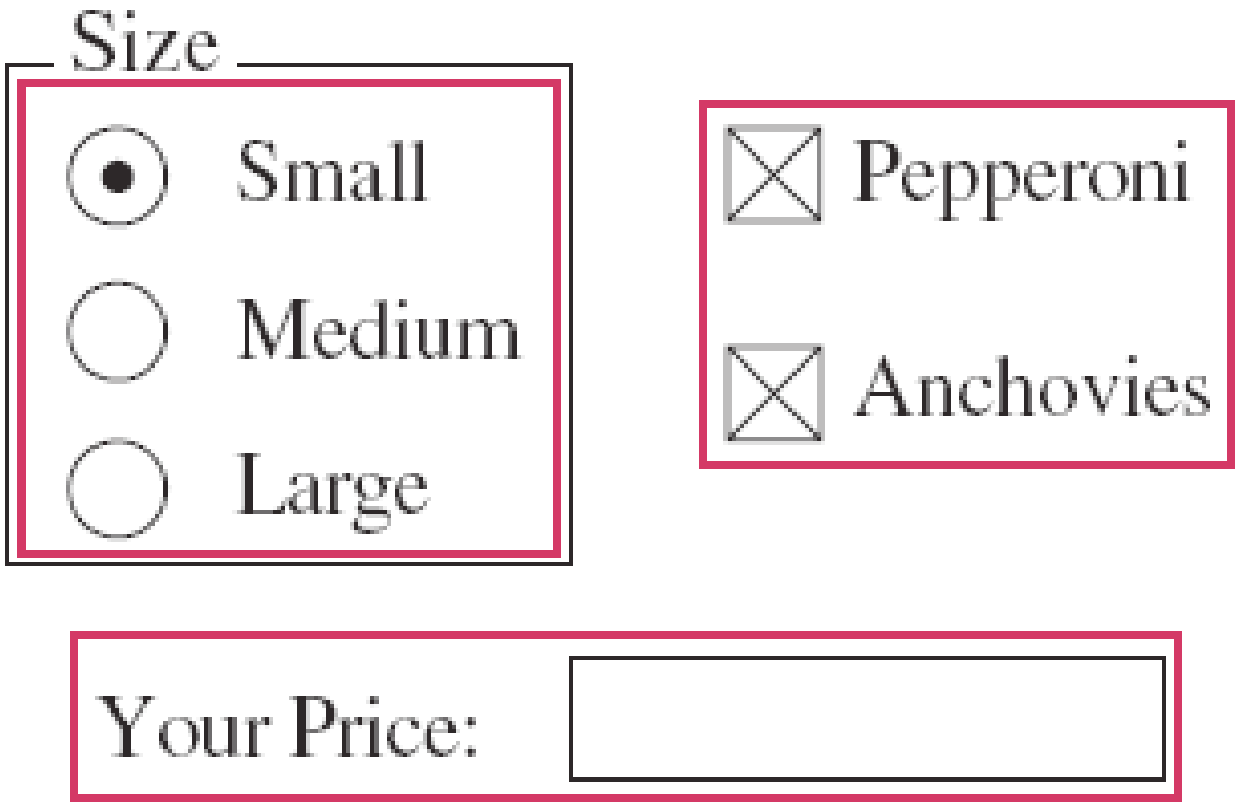
Step 1: Make a sketch of your desired component layout



Continued

How To 18.1 Laying Out a User Interface (cont.)

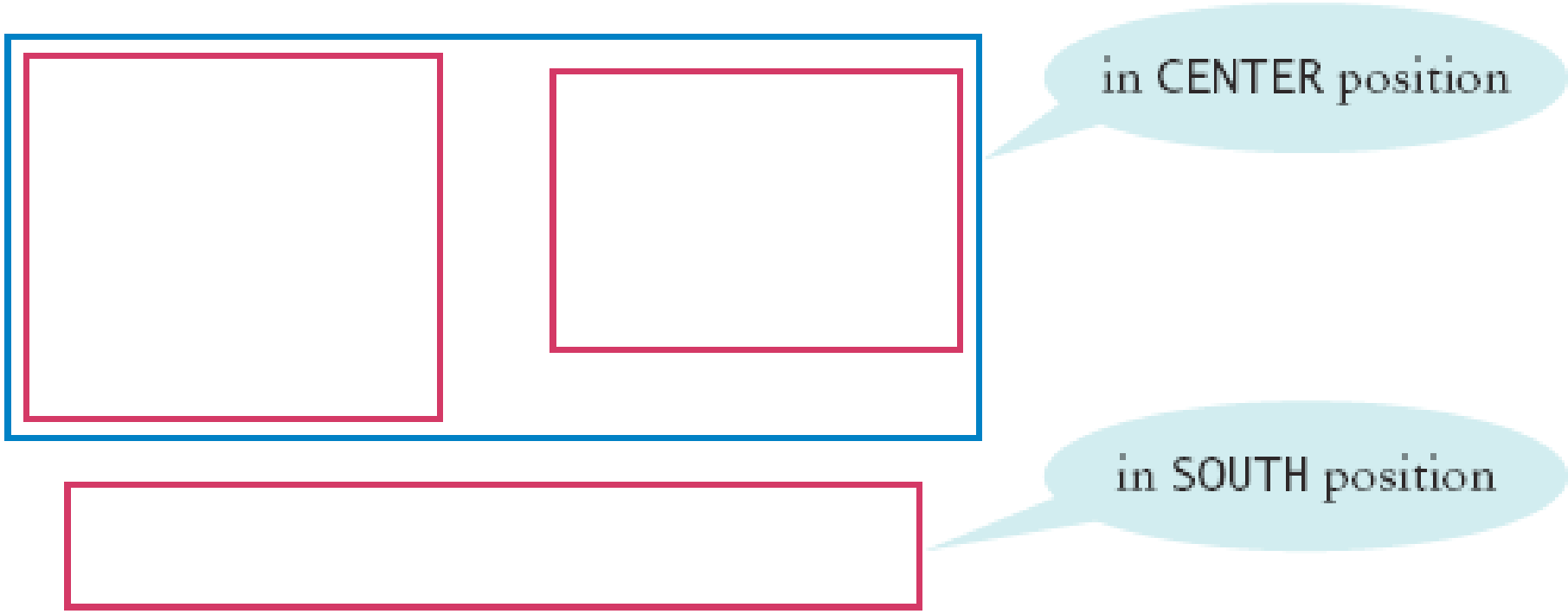
Step 2: Find groupings of adjacent components with the same layout



Continued

How To 18.1 Laying Out a User Interface (cont.)

Step 3: Identify layouts for each group
Step 4: Group the groups together



Step 5: Write the code to generate the layout

GUI Builder

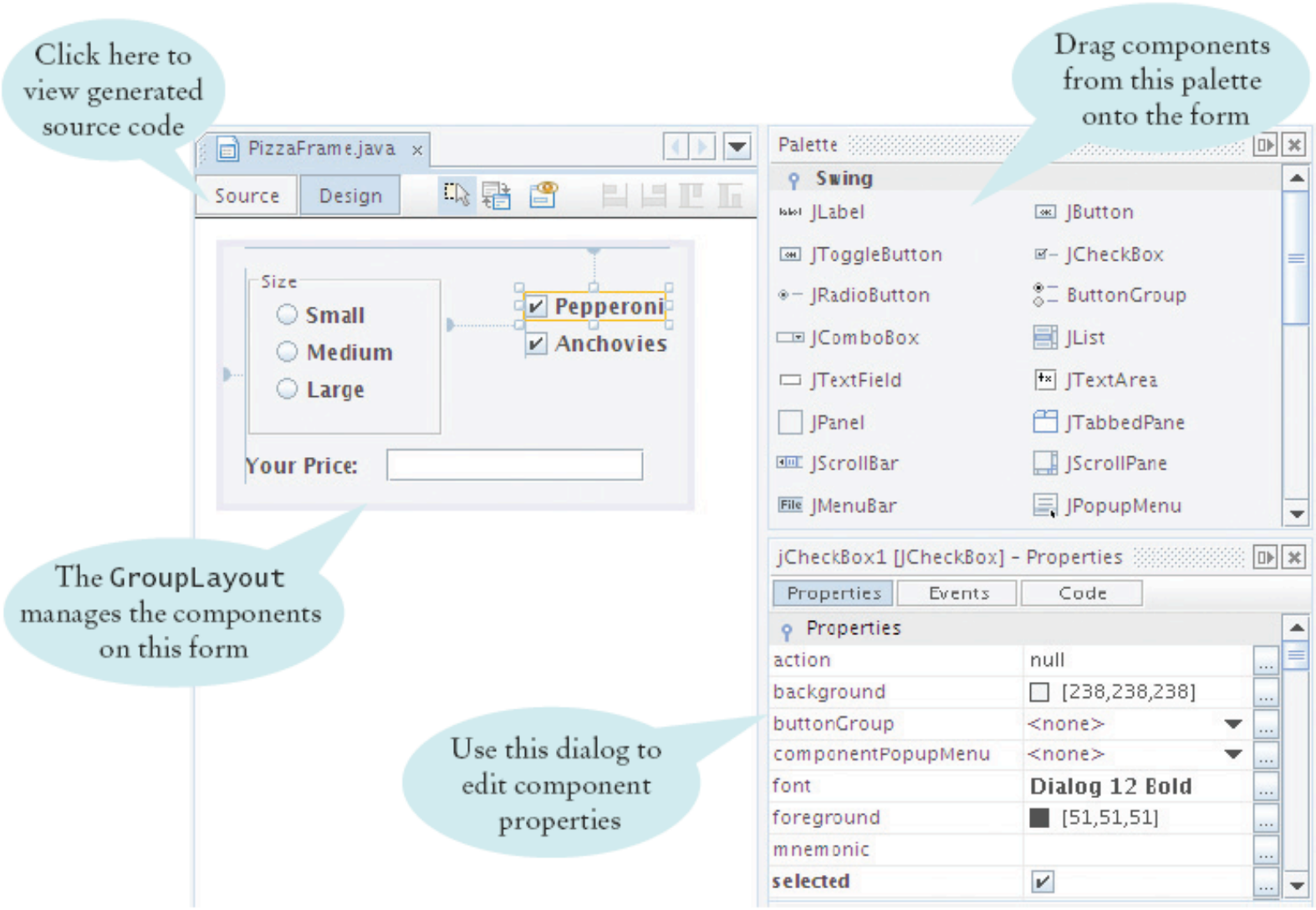


Figure 8 A GUI Builder

Menus

- A frame contains a menu bar
- The menu bar contains menus
- A menu contains submenus and menu items

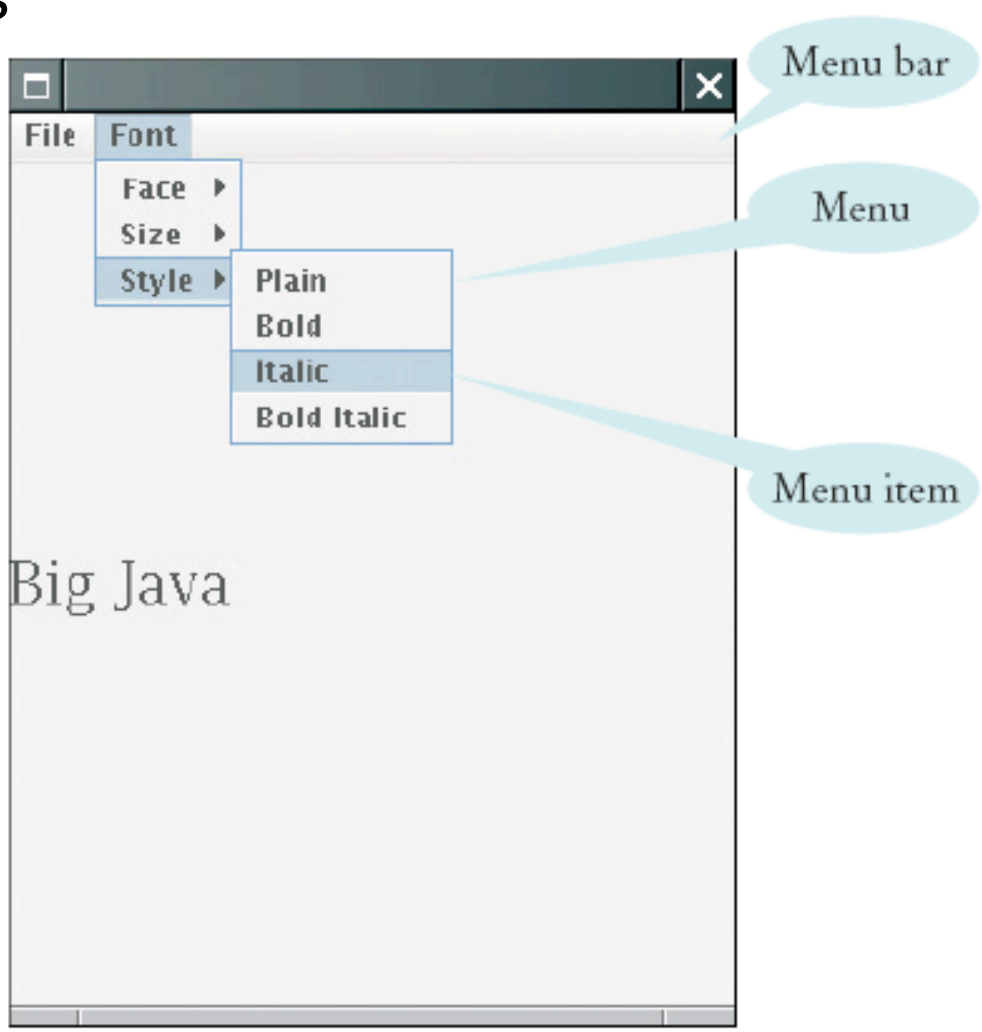


Figure 9
Pull-Down Menus

Menu Items

- Add menu items and submenus with the `add` method: `JMenuItem`

```
fileExitItem = new JMenuItem("Exit");
fileMenu.add(fileExitItem);
```

- A menu item has no further submenus
- Menu items generate action events
- Add a listener to each menu item:
  `fileExitItem.addActionListener(listener);`
- Add action listeners only to menu items, not to menus or the menu bar

A Sample Program

- Builds up a small but typical menu
- Traps action events from menu items
- To keep program readable, use a separate method for each menu or set of related menus
  - *createFaceItem: creates menu item to change the font face*
  - *createSizeItem*
  - *createStyleItem*

ch18/menu/FontViewer2.java

```
01: import javax.swing.JFrame;
02:
03: /**
04:    This program allows the user to view font effects.
05: */
06: public class FontViewer
07: {
08:    public static void main(String[] args)
09:    {
10:       JFrame frame = new FontViewerFrame();
11:       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12:       frame.setTitle("FontViewer");
13:       frame.setVisible(true);
14:    }
15: }
16:
```

ch18/menu/FontViewer2Frame.java

```
001: import java.awt.BorderLayout;
002: import java.awt.Font;
003: import java.awt.GridLayout;
004: import java.awt.event.ActionEvent;
005: import java.awt.event.ActionListener;
006: import javax.swing.ButtonGroup;
007: import javax.swing.JButton;
008: import javax.swing.JCheckBox;
009: import javax.swing.JComboBox;
010: import javax.swing.JFrame;
011: import javax.swing.JLabel;
012: import javax.swing.JPanel;
013: import javax.swing.JRadioButton;
014: import javax.swing.border.EtchedBorder;
015: import javax.swing.border.TitledBorder;
016:
017: /**
018:    This frame contains a text field and a control panel
019:    to change the font of the text.
020: */
```

Continued

ch18/menu/FontViewer2Frame.java (cont.)

```
021: public class FontViewerFrame extends JFrame
022: {
023:    /**
024:       Constructs the frame.
025:    */
026:    public FontViewerFrame()
027:    {
028:       // Construct text sample
029:       sampleField = new JLabel("Big Java");
030:       add(sampleField, BorderLayout.CENTER);
031:
032:       // This listener is shared among all components
033:       class ChoiceListener implements ActionListener
034:       {
035:          public void actionPerformed(ActionEvent event)
036:          {
037:             setSampleFont();
038:          }
039:       }
040:
```

Continued

ch18/menu/FontViewer2Frame.java (cont.)

```
041:        listener = new ChoiceListener();
042:
043:        createControlPanel();
044:        setSampleFont();
045:        setSize(FRAME_WIDTH, FRAME_HEIGHT);
046:     }
047:
048:     /**
049:        Creates the control panel to change the font.
050:     */
051:     public void createControlPanel()
052:     {
053:        JPanel facenamePanel = createComboBox();
054:        JPanel sizeGroupPanel = createCheckBoxes();
055:        JPanel styleGroupPanel = createRadioButtons();
056:
057:        // Line up component panels
058:
059:        JPanel controlPanel = new JPanel();
060:        controlPanel.setLayout(new GridLayout(3, 1));
061:        controlPanel.add(facenamePanel);
```

Continued

ch18/menu/FontViewer2Frame.java (cont.)

```
062:          controlPanel.add(sizeGroupPanel);
063:          controlPanel.add(styleGroupPanel);
064:
065:          // Add panels to content pane
066:
067:          add(controlPanel, BorderLayout.SOUTH);
068:       }
069:
070:       /**
071:          Creates the combo box with the font style choices.
072:          @return the panel containing the combo box
073:       */
074:       public JPanel createComboBox()
075:       {
076:          facenameCombo = new JComboBox();
077:          facenameCombo.addItem("Serif");
078:          facenameCombo.addItem("SansSerif");
079:          facenameCombo.addItem("Monospaced");
080:          facenameCombo.setEditable(true);
081:          facenameCombo.addActionListener(listener);
082:
```

Continued

ch18/menu/FontViewer2Frame.java (cont.)

```
083:        JPanel panel = new JPanel();
084:        panel.add(facenameCombo);
085:        return panel;
086:     }
087:
088:     /**
089:        Creates the check boxes for selecting bold and italic styles.
090:        @return the panel containing the check boxes
091:     */
092:     public JPanel createCheckBoxes()
093:     {
094:        italicCheckBox = new JCheckBox("Italic");
095:        italicCheckBox.addActionListener(listener);
096:
097:        boldCheckBox = new JCheckBox("Bold");
098:        boldCheckBox.addActionListener(listener);
099:
100:       JPanel panel = new JPanel();
101:       panel.add(italicCheckBox);
102:       panel.add(boldCheckBox);
103:       panel.setBorder
104:          (new TitledBorder(new EtchedBorder(), "Style"));
```

Continued

ch18/menu/FontViewer2Frame.java (cont.)

```
105:
106:          return panel;
107:       }
108:
109:       /**
110:          Creates the radio buttons to select the font size
111:          @return the panel containing the radio buttons
112:       */
113:       public JPanel createRadioButtons()
114:       {
115:          smallButton = new JRadioButton("Small");
116:          smallButton.addActionListener(listener);
117:
118:          mediumButton = new JRadioButton("Medium");
119:          mediumButton.addActionListener(listener);
120:
121:          largeButton = new JRadioButton("Large");
122:          largeButton.addActionListener(listener);
123:          largeButton.setSelected(true);
124:
125:          // Add radio buttons to button group
126:
```

Continued

ch18/menu/FontViewer2Frame.java (cont.)

```
127:        ButtonGroup group = new ButtonGroup();
128:        group.add(smallButton);
129:        group.add(mediumButton);
130:        group.add(largeButton);
131:
132:        JPanel panel = new JPanel();
133:        panel.add(smallButton);
134:        panel.add(mediumButton);
135:        panel.add(largeButton);
136:        panel.setBorder
137:              (new TitledBorder(new EtchedBorder(), "Size"));
138:
139:        return panel;
140:     }
141:
142:     /**
143:        Gets user choice for font name, style, and size
144:        and sets the font of the text sample.
145:     */
146:     public void setSampleFont()
147:     {
```

Continued

ch18/menu/FontViewer2Frame.java (cont.)

```
148:         // Get font name
149:         String facename
150:               = (String) facenameCombo.getSelectedItem();
151:
152:         // Get font style
153:
154:         int style = 0;
155:         if (italicCheckBox.isSelected())
156:            style = style + Font.ITALIC;
157:         if (boldCheckBox.isSelected())
158:            style = style + Font.BOLD;
159:
160:         // Get font size
161:
162:         int size = 0;
163:
164:         final int SMALL_SIZE = 24;
165:         final int MEDIUM_SIZE = 36;
166:         final int LARGE_SIZE = 48;
167:
```

Continued

ch18/menu/FontViewer2Frame.java (cont.)

```
168:         if (smallButton.isSelected())
169:            size = SMALL_SIZE;
170:         else if (mediumButton.isSelected())
171:            size = MEDIUM_SIZE;
172:         else if (largeButton.isSelected())
173:            size = LARGE_SIZE;
174:
175:         // Set font of text field
176:
177:         sampleField.setFont(new Font(facename, style, size));
178:         sampleField.repaint();
179:      }
180:
181:      private JLabel sampleField;
182:      private JCheckBox italicCheckBox;
183:      private JCheckBox boldCheckBox;
184:      private JRadioButton smallButton;
185:      private JRadioButton mediumButton;
186:      private JRadioButton largeButton;
187:      private JComboBox facenameCombo;
188:      private ActionListener listener;
```

Continued

ch18/menu/FontViewer2Frame.java (cont.)

```
189:
190:     private static final int FRAME_WIDTH = 300;
191:     private static final int FRAME_HEIGHT = 400;
192: }
```

Self Check 18.6

Why do `JMenu` objects not generate action events?

Answer: When you open a menu, you have not yet made a selection. Only `JMenuItem` objects correspond to selections.

Self Check 18.7

Why is the name parameter in the createFaceItem method declared as final?

Answer: The parameter variable is accessed in a method of an inner class.

Exploring the Swing Documentation

- For more sophisticated effects, explore the Swing documentation
- The documentation can be quite intimidating at first glance
- Next example will show how to use the documentation to your advantage

Example: A Color Viewer

- It should be fun to mix your own colors, with a slider for the red, green, and blue values

Figure 10
A Color Viewer

Example: A Color Viewer

- How do you know if there is a slider?
  - *Buy a book that illustrates all Swing components*
  - *Run sample application included in the JDK that shows off all Swing components*
  - *Look at the names of all of the classes that start with J*
    - `JSlider` seems like a good candidate
- Next, ask a few questions:
  - *How do I construct a `JSlider`?*
  - *How can I get notified when the user has moved it?*
  - *How can I tell to which value the user has set it?*
- After mastering sliders, you can find out how to set tick marks, etc.

The Swing Demo Set

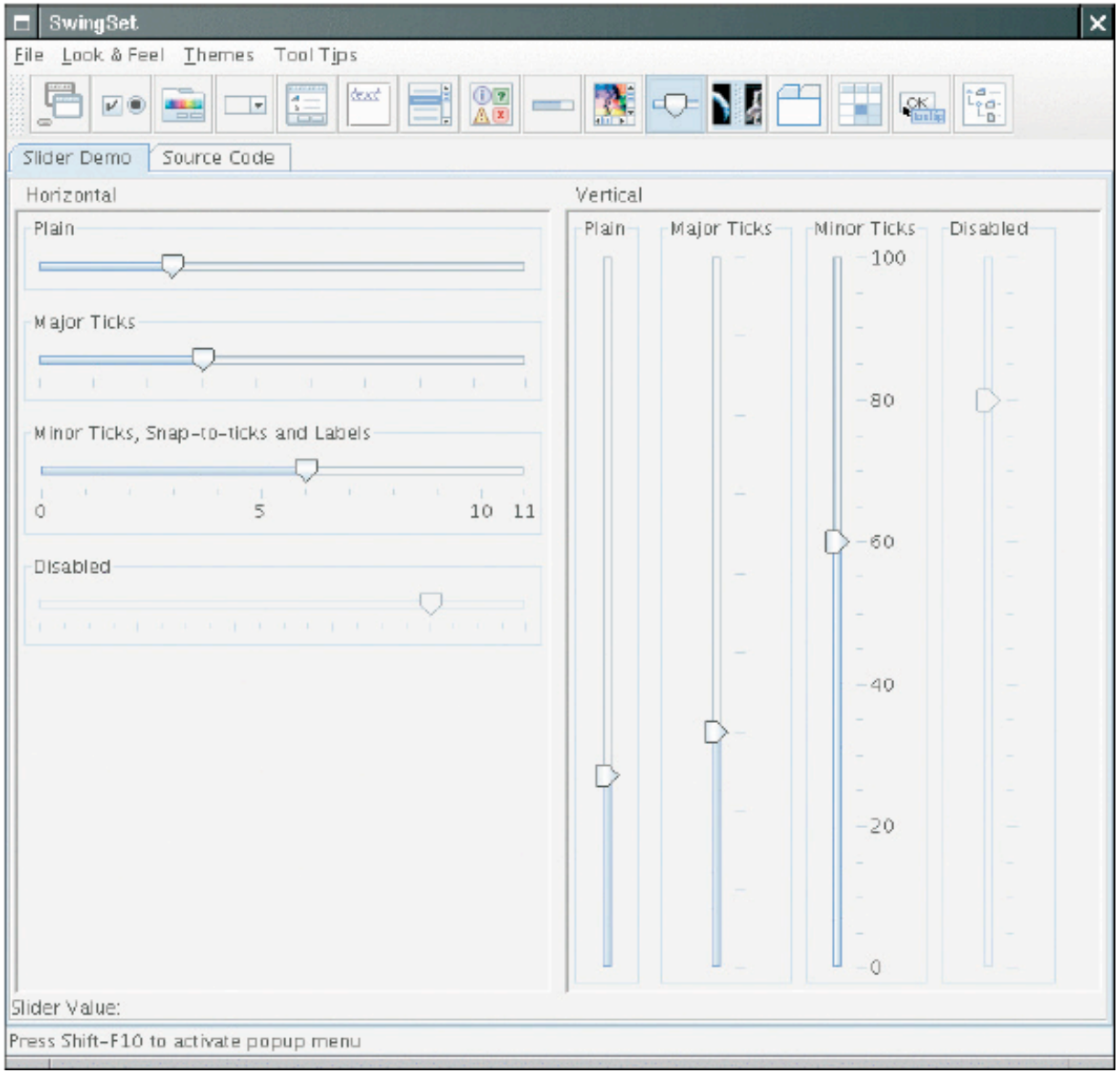


Figure 11
The SwingSet Demo

Example: A Color Viewer

- There are over 50 methods in `JSlider` class and over 250 inherited methods
- Some method descriptions look scary

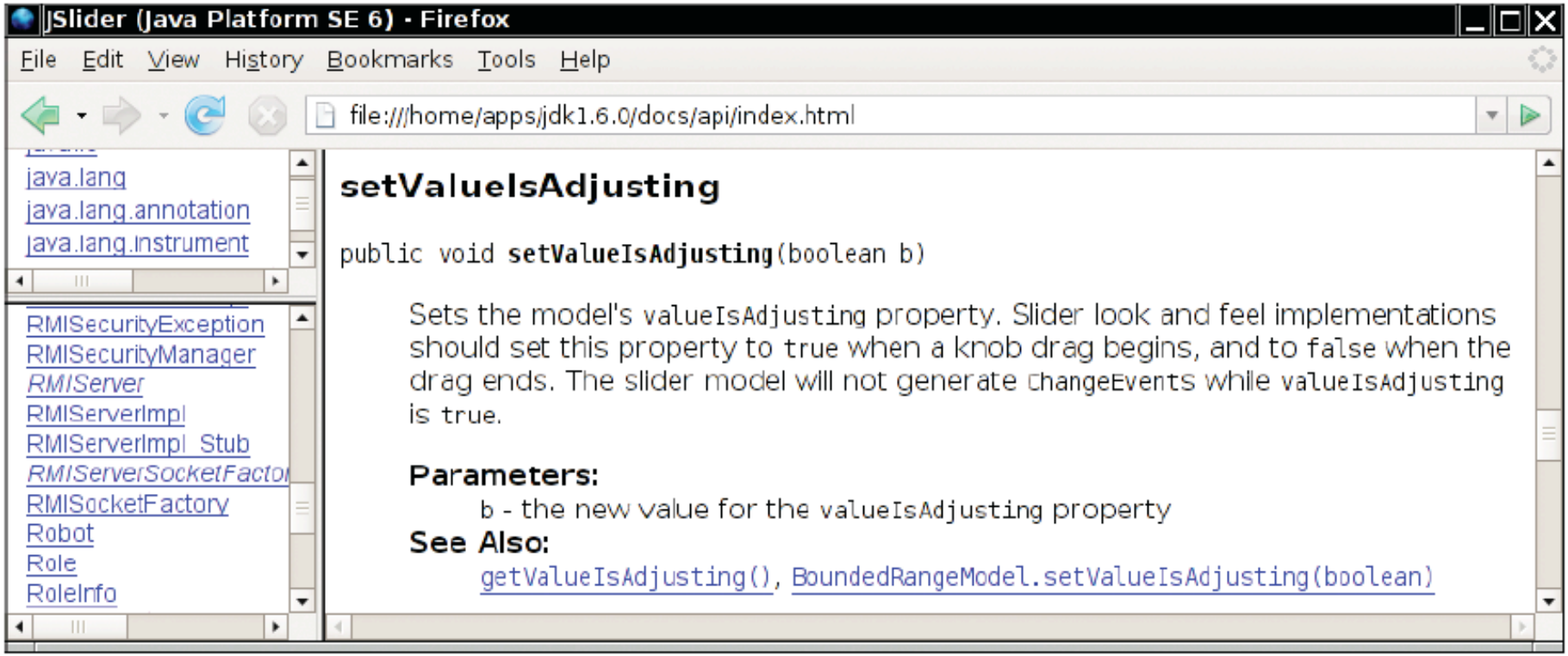


Figure 12 A Mysterious Method Description from the API Documentation

Continued

Example: A Color Viewer (cont.)

- Develop the ability to separate fundamental concepts from ephemeral minutiae

How do I construct a `JSlider`?

- Look at the Java version 6 API documentation
- There are six constructors for the `JSlider` class
- Learn about one or two of them
- Strike a balance somewhere between the trivial and the bizarre
- Too limited:

```
public JSlider()
```

  - *Creates a horizontal slider with the range 0 to 100 and an initial value of 50*

- Bizarre:

```
public JSlider(BoundedRangeModel brm)
```

  - *Creates a horizontal slider using the specified `BoundedRangeModel`*

Continued

How do I construct a `JSlider`? (cont.)

- Useful:

```
public JSlider(int min, int max, int value)
```

  - *Creates a horizontal slider using the specified* `min`, `max`, *and* `value`

How can I get notified when the user has moved a `JSlider`?

- There is no `addActionListener` method
- There is a method

```
public void addChangeListener(ChangeListener l)
```

- Click on the `ChangeListener` link to learn more
- It has a single method:

```
void stateChanged(ChangeEvent e)
```

- Apparently, method is called whenever user moves the slider
- What is a `ChangeEvent`?
  - *It inherits `getSource` method from superclass `EventObject`*
  - *`getSource`: tells us which component generated this event*

How can I tell to which value the user has set a `JSlider`?

- Now we have a plan:
  - *Add a change event listener to each slider*
  - *When slider is changed, `stateChanged` method is called*
  - *Find out the new value of the slider*
  - *Recompute color value*
  - *Repaint color panel*
- Need to get the current value of the slider
- Look at all the methods that start with get; you find:

```
public int getValue()
```

  - *Returns the slider's value*

The Components of the `ColorViewerFrame`

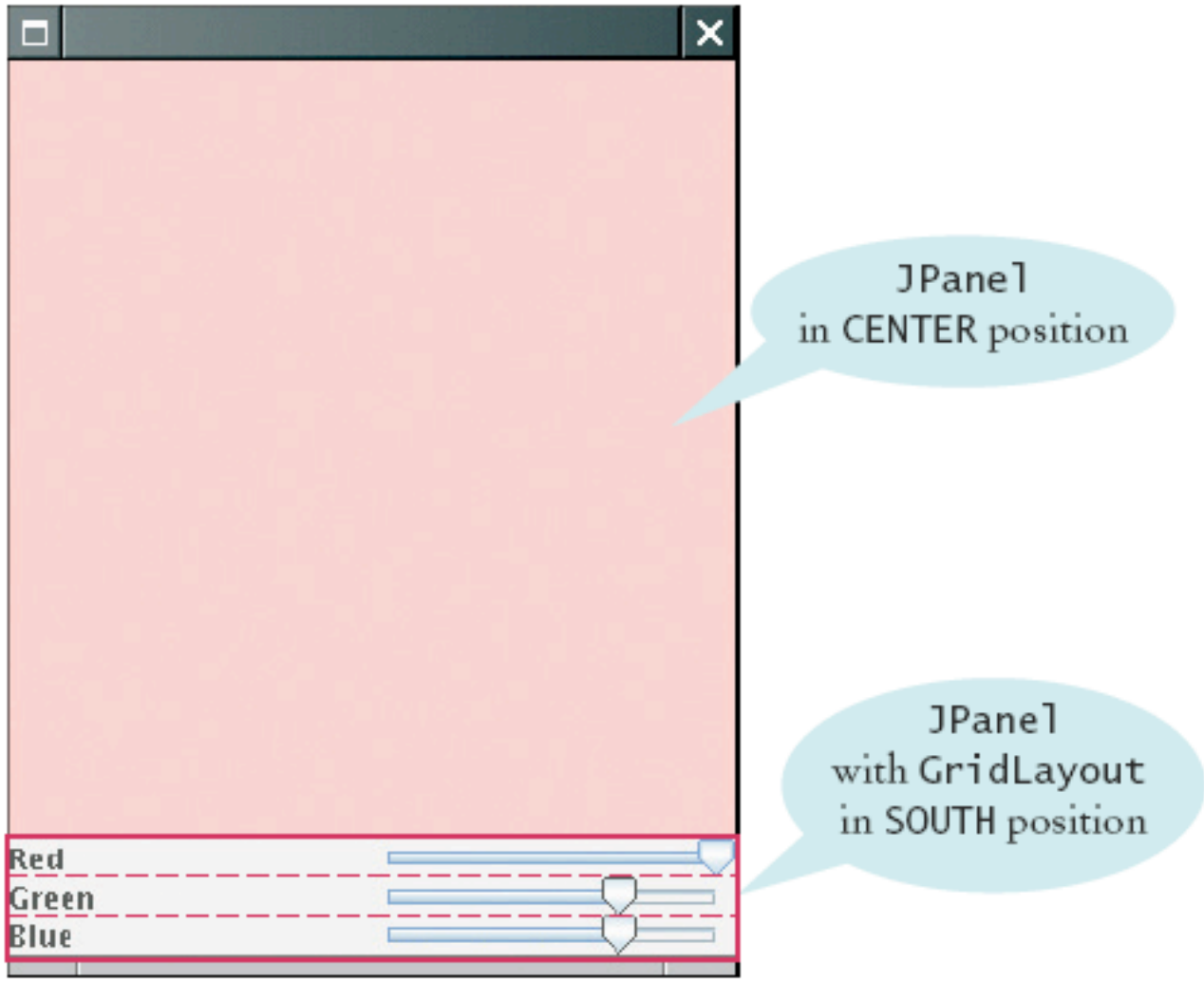


Figure 13
The Components of the `ColorViewerFrame`

Classes of the Color Viewer Program

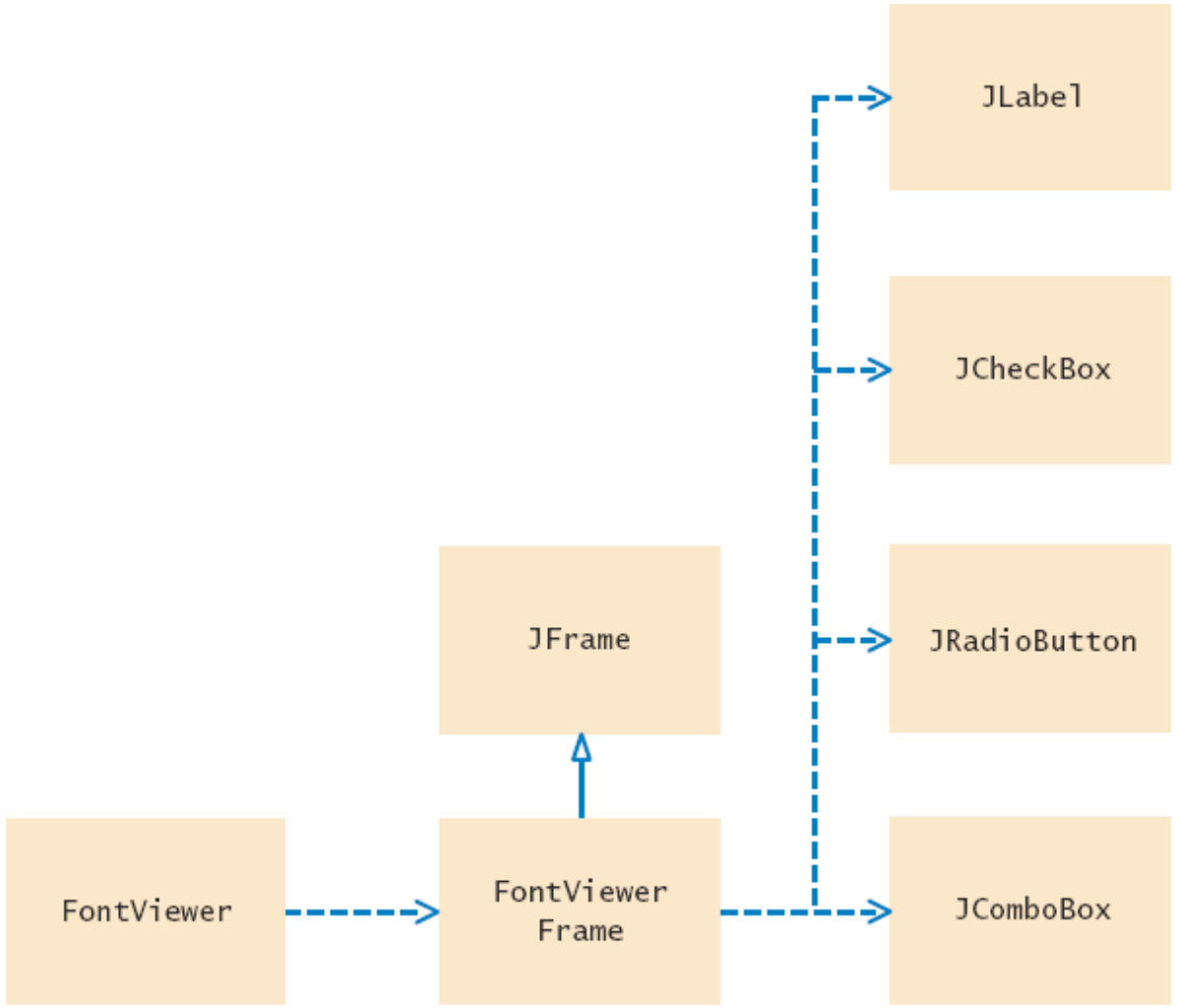


Figure 7
Classes of the Font Viewer Program

ch18/slider/ColorViewer.java

```
01: import javax.swing.JFrame;
02:
03: public class ColorViewer
04: {
05:    public static void main(String[] args)
06:    {
07:       ColorViewerFrame frame = new ColorViewerFrame();
08:       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
09:       frame.setVisible(true);
10:    }
11: }
12:
```

ch18/slider/ColorViewerFrame.java

```
01: import java.awt.BorderLayout;
02: import java.awt.Color;
03: import java.awt.GridLayout;
04: import javax.swing.JFrame;
05: import javax.swing.JLabel;
06: import javax.swing.JPanel;
07: import javax.swing.JSlider;
08: import javax.swing.event.ChangeListener;
09: import javax.swing.event.ChangeEvent;
10:
11: public class ColorViewerFrame extends JFrame
12: {
13:    public ColorViewerFrame()
14:    {
15:       colorPanel = new JPanel();
16:
17:       add(colorPanel, BorderLayout.CENTER);
18:       createControlPanel();
19:       setSampleColor();
20:       setSize(FRAME_WIDTH, FRAME_HEIGHT);
21:    }
```

Continued

ch18/slider/ColorViewerFrame.java (cont.)

```
22:
23:    public void createControlPanel()
24:    {
25:       class ColorListener implements ChangeListener
26:       {
27:          public void stateChanged(ChangeEvent event)
28:          {
29:             setSampleColor();
30:          }
31:       }
32:
33:       ChangeListener listener = new ColorListener();
34:
35:       redSlider = new JSlider(0, 255, 255);
36:       redSlider.addChangeListener(listener);
37:
38:       greenSlider = new JSlider(0, 255, 175);
39:       greenSlider.addChangeListener(listener);
40:
41:       blueSlider = new JSlider(0, 255, 175);
42:       blueSlider.addChangeListener(listener);
43:
```

Continued

ch18/slider/ColorViewerFrame.java (cont.)

```
44:         JPanel controlPanel = new JPanel();
45:         controlPanel.setLayout(new GridLayout(3, 2));
46:
47:         controlPanel.add(new JLabel("Red"));
48:         controlPanel.add(redSlider);
49:
50:         controlPanel.add(new JLabel("Green"));
51:         controlPanel.add(greenSlider);
52:
53:         controlPanel.add(new JLabel("Blue"));
54:         controlPanel.add(blueSlider);
55:
56:         add(controlPanel, BorderLayout.SOUTH);
57:     }
58:
59:     /**
60:        Reads the slider values and sets the panel to
61:        the selected color.
62:     */
63:     public void setSampleColor()
64:     {
65:        // Read slider values
66:
```

Continued

ch18/slider/ColorViewerFrame.java (cont.)

```
67:         int red = redSlider.getValue();
68:         int green = greenSlider.getValue();
69:         int blue = blueSlider.getValue();
70:
71:         // Set panel background to selected color
72:
73:         colorPanel.setBackground(new Color(red, green, blue));
74:         colorPanel.repaint();
75:      }
76:
77:      private JPanel colorPanel;
78:      private JSlider redSlider;
79:      private JSlider greenSlider;
80:      private JSlider blueSlider;
81:
82:      private static final int FRAME_WIDTH = 300;
83:      private static final int FRAME_HEIGHT = 400;
84: }
```

Self Check 18.8

Suppose you want to allow users to pick a color from a color dialog box. Which class would you use? Look in the API documentation.

Answer: `JColorChooser`.

Self Check 18.9

Why does a slider emit change events and not action events?

Answer: Action events describe one-time changes, such as button clicks. Change events describe continuous changes.