

CHAPTER

11

ADVANCED USER INTERFACES





Chapter Goals

- ❑ To use layout managers to arrange user-interface components in a container
- ❑ To become familiar with common user-interface components, such as radio buttons, check boxes, and menus
- ❑ To build programs that handle events generated by user-interface components
- ❑ To browse the Java documentation effectively

In this chapter, you will learn how to use the most common user-interface components in the Java Swing toolkit, search Java documentation and handle events for interactive graphical programs.



Contents

- ❑ Layout Management
- ❑ Choices
- ❑ Menus
- ❑ Exploring the Swing Documentation
- ❑ Using Timer Events for Animations
- ❑ Mouse Events



11.1 Layout Management

- Arranging components on the screen
 - User-interface components are arranged by placing them in a Swing Container object:
 - JFrame (content pane), JPanel and JApplet
 - So far, all components have been arranged from left to right inside a JPanel container





Layout Management

- Each container has a layout manager that directs the arrangement of its components
- Three useful layout managers are:
 - 1) Border layout
 - 2) Flow layout
 - 3) Grid layout

Components are added to a container which uses a layout manager to place them



Jframe's Content Pane

- A JFrame has a content pane which is a Container where you can add components
 - Use the getContentPane method to get its reference

```
Container contentPane = getContentPane();
```

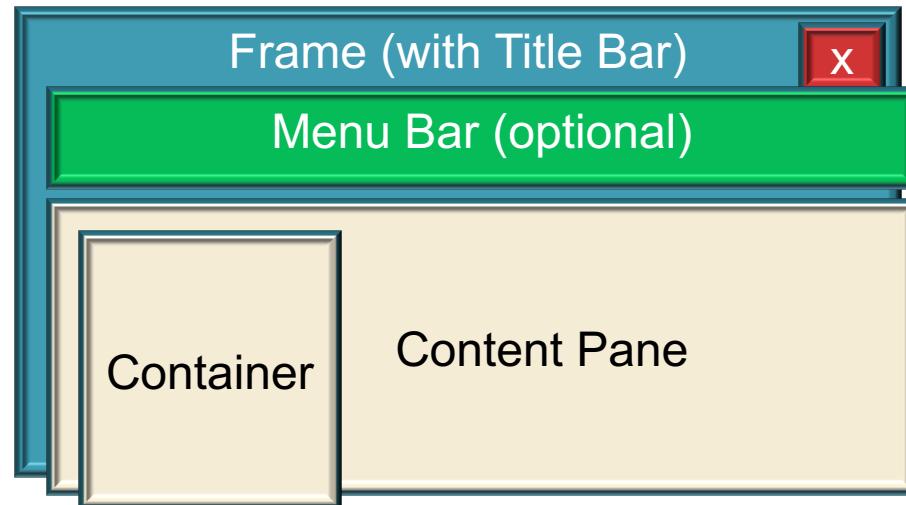
- Add components to the content pane or
 - Add a container to the content pane, then add components to the container



Jframe's Content Pane (2)

❑ Frame

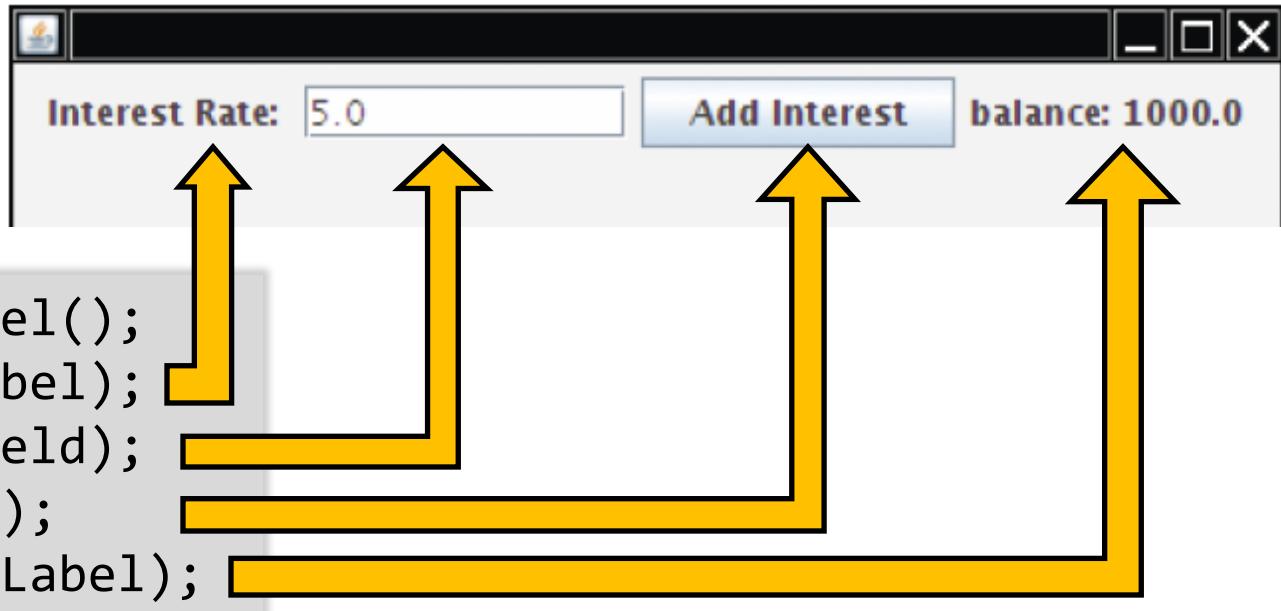
- Menu Bar
- Content Pane
 - Components
 - Container
 - Components





Flow Layout

- Components are added from left to right



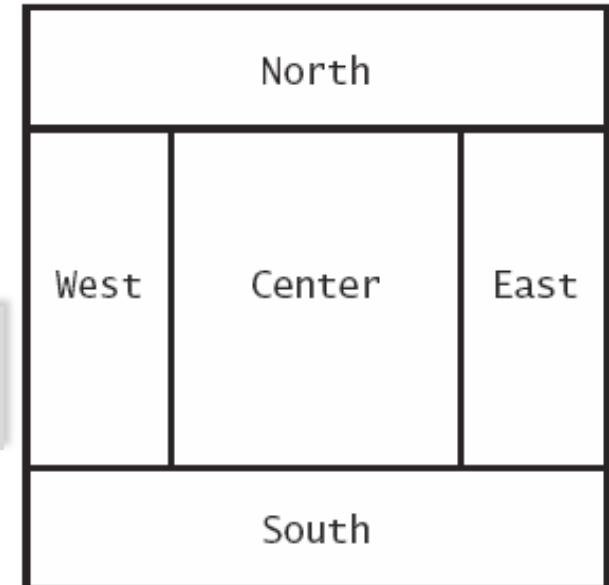
- A JPanel uses **flow layout** by default



Border Layout

- Components are placed toward areas of a container
 - NORTH, EAST, SOUTH, WEST, or CENTER
 - Specify one when adding components
 - The content pane of a JFrame uses **border layout** by default

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





Grid Layout

- ❑ Components are placed in boxes in a simple table arrangement
 - Specify the size (rows then columns) of the grid

```
JPanel buttonPanel = new JPanel();
buttonPanel.setLayout(new GridLayout(4, 3));
```

- Then add components which will be placed from the upper left, across, then down

```
buttonPanel.add(button7);
buttonPanel.add(button8);
buttonPanel.add(button9);
buttonPanel.add(button4);
. . .
```

7	8	9
4	5	6
1	2	3
0	.	CE



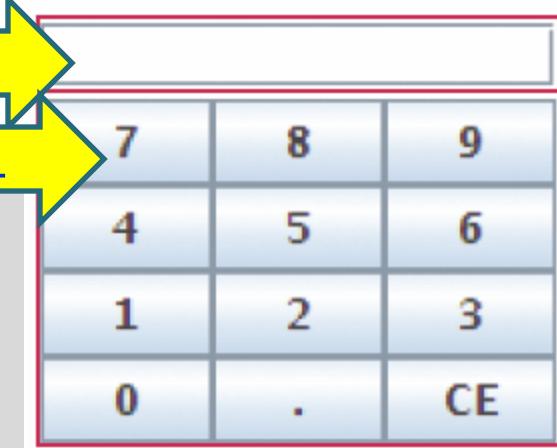
Using Nested Panels

- Create complex layouts by nesting panels
 - Give each panel an appropriate layout manager
 - Panels have invisible borders, so you can use as many panels as you need to organize components

JTextField in NORTH of keypadPanel

JPanel GridLayout in CENTER of keypadPanel

```
 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);
```





11.2 Choices

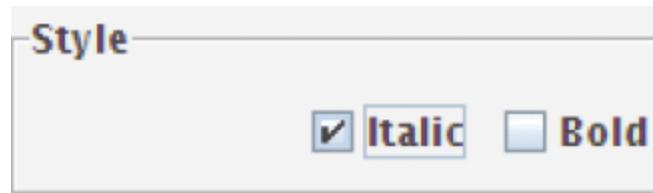
- In a modern graphical user interface program, there are commonly used devices to make different types of selections:

Radio Buttons



- For a small set of mutually exclusive choices

Check Boxes



- For a binary choice

Combo Boxes

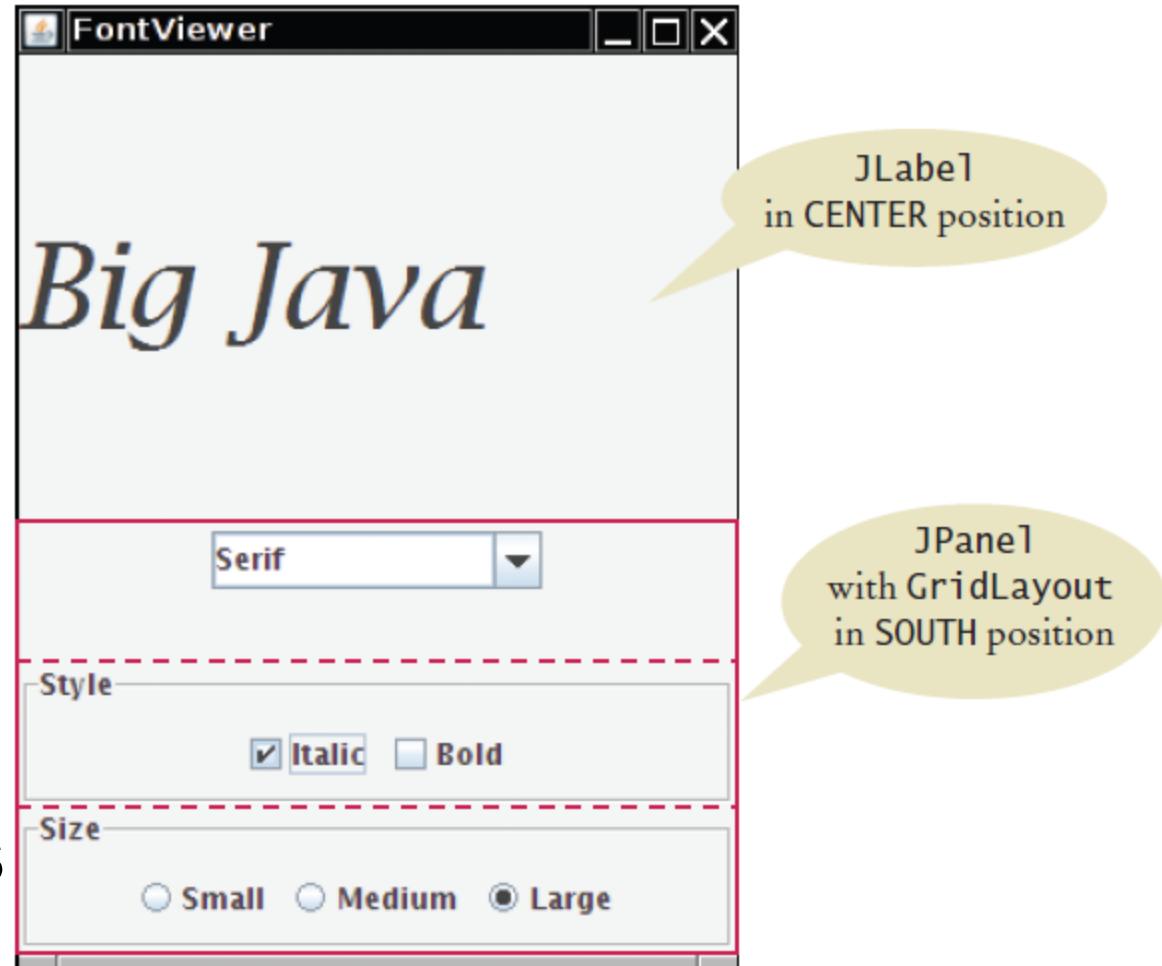


- For a large set of choices



Fontviewer Layout

- ❑ Title Bar
- ❑ Label
 - Shows current font
- ❑ Combo Box
- ❑ Check Boxes
- ❑ Radio Buttons





Radio Button Panels

- Use a panel for each set of radio buttons
 - The default border for a panel is invisible (no border)
 - You can add a border to a panel to make it visible along with a text label:



```
JPanel panel = new JPanel();
panel.add(smallButton);
panel.add(mediumButton);
panel.add(largeButton);
panel.setBorder(new TitledBorder(new EtchedBorder(),"Size"));
```

- There are a large number of border styles available
 - See the Swing documentation for more details



Grouping Radio Buttons

- Add Radio Buttons into a `ButtonGroup` so that only one button in the group is selected at a time
 - Create the `JRadioButtons` first, then add them to the `ButtonGroup`



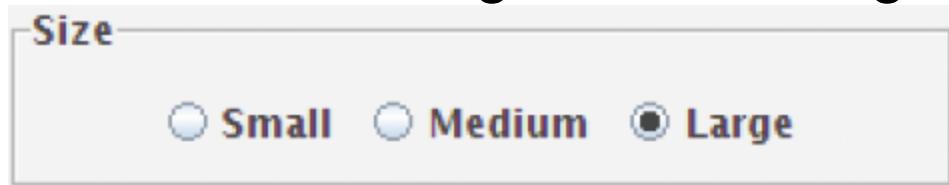
```
JRadioButton smallButton = new JRadioButton("Small");
JRadioButton mediumButton = new JRadioButton("Medium");
JRadioButton largeButton = new JRadioButton("Large");
ButtonGroup group = new ButtonGroup();
group.add(smallButton);
group.add(mediumButton);
group.add(largeButton);
```

- Note that the button group does not place the buttons close to each other on the container



Selecting Radio Buttons

- It is customary to set one button as selected (the default) when using radio buttons
 - Use the button's `setSelected` method
 - Set the default button before making the enclosing frame visible



```
JRadioButton largeButton = new JRadioButton("Large");
largeButton.setSelected(true);
```

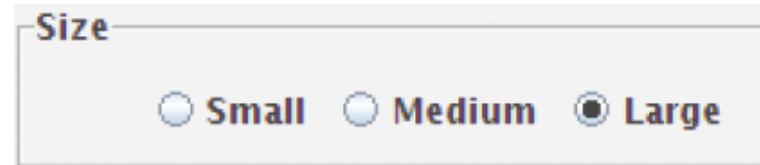
- Call the `isSelected` method of each button to find out which one it is currently selected

```
if (largeButton.isSelected())
{ size = LARGE_SIZE; }
```



Check Boxes versus Radio Buttons

- Radio buttons and check boxes have different visual appearances
 - Radio buttons are round and show a black dot when selected
 - Check boxes are square and show a check mark when selected





Check Boxes

- ❑ A check box is a user-interface component with two states: checked and unchecked
 - Use for choices that are not mutually exclusive
 - For example, text may be Italic, Bold, both or neither
 - Because check box settings do not exclude each other, you do not need to place a set of check boxes inside a button group





Selecting Check Boxes

- To setup a Check Box, use Swing JCheckBox
 - Pass the constructor the name for the check box label

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

- Call the **isSelected** method of a checkbox to find out whether it is currently selected or not

```
if (italicCheckBox.isSelected())
{ style = style + Font.ITALIC }
```



Combo Boxes

- A combo box is a combination of a list and a text field
 - Use a combo box for a large set of choices
 - Use when radio buttons would take up too much space
 - It can be either:
 - Closed (shows one selection)
 - Open, showing multiple selections
 - It can also be editable
 - Type a selection into a blank line

```
facenameCombo.setEditable();
```
 - When you click on the arrow to the right of the text field of a combo box, a list of selections drops down, and you can choose one of the items in the list





Adding and Selecting Items

- Add text ‘items’ to a combo box that will show in the list:

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

- Use the `getSelectedItem` method to return the selected item (as an Object)
 - Combo boxes can store other objects in addition to strings, so casting to a string may be required:

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



FontViewer.java

```
1 import javax.swing.JFrame;
2
3 /**
4  * This program allows the user to view font effects.
5 */
6 public class FontViewer
7 {
8     public static void main(String[] args)
9     {
10         JFrame frame = new FontFrame();
11         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12         frame.setTitle("FontViewer");
13         frame.setVisible(true);
14     }
15 }
```

Instantiates a FontFrame object

Sets close event handler, Title bar, and sets the frame to visible



FontFrame.java (1)

```
35  /**
36   * Constructs the frame.
37  */
38 public FontFrame()
39 {
40     // Construct text sample
41     label = new JLabel("Big Java");
42     add(label, BorderLayout.CENTER);
43
44     // This listener is shared among all components
45     listener = new ChoiceListener();
46
47     createControlPanel();
48     setLabelFont();
49     setSize(FRAME_WIDTH, FRAME_HEIGHT);
50 }
51
52 class ChoiceListener implements ActionListener
53 {
54     public void actionPerformed(ActionEvent event)
55     {
56         setLabelFont();
57     }
58 }
```

Events from all components of the frame use the same listener

createControlPanel helper method builds the GUI

ChoiceListener is an inner class of the constructor of FontFrameViewer



FontFrame.java (2)

```
60  /**
61   * Creates the control panel to change the font.
62  */
63  public void createControlPanel()
64  {
65      JPanel facenamePanel = createComboBox();
66      JPanel sizeGroupPanel = createCheckboxes();
67      JPanel styleGroupPanel = createRadioButtons();
68
69      // Line up component panels
70
71      JPanel controlPanel = new JPanel();
72      controlPanel.setLayout(new GridLayout(3, 1));
73      controlPanel.add(facenamePanel);
74      controlPanel.add(sizeGroupPanel);
75      controlPanel.add(styleGroupPanel);
76
77      // Add panels to content pane
78
79      add(controlPanel, BorderLayout.SOUTH);
80  }
```

Uses helper methods to build each Panel

Adds each panel to the controlPanel



setLabelFont Method (1)

```
152 /**
153  Gets user choice for font name, style, and size
154  and sets the font of the text sample.
155 */
156 public void setLabelFont()
157 {
158     // Get font name
159     String facename = (String) facenameCombo.getSelectedItem();
160
161     // Get font style      getSelectedItem for the combo box
162
163     int style = 0;
164     if (italicCheckBox.isSelected())
165     {
166         style = style + Font.ITALIC;
167     }
168     if (boldCheckBox.isSelected())
169     {
170         style = style + Font.BOLD;
171     }
```

Queries components of the frame
and sets the font name and style

isSelected for check boxes



setLabelFont Method (2)

```
173 // Get font size  
174  
175 int size = 0;  
176  
177 final int SMALL_SIZE = 24;  
178 final int MEDIUM_SIZE = 36;  
179 final int LARGE_SIZE = 48; isSelectedItem for radio buttons  
180  
181 if (smallButton.isSelected()) { size = SMALL_SIZE; }  
182 else if (mediumButton.isSelected()) { size = MEDIUM_SIZE; }  
183 else if (largeButton.isSelected()) { size = LARGE_SIZE; }  
184  
185 // Set font of text field  
186  
187 label.setFont(new Font(facename, style, size));  
188 label.repaint();  
189 }  
190 }
```

Queries components of the frame and sets the font size based on radio button selection

isSelectedItem for radio buttons

Calls **setFont** with face, style and size



Steps to Design a User Interface

1) Make a sketch of the component layout.

- Draw all the buttons, labels, text fields, and borders on a sheet of graph paper

Size

<input checked="" type="radio"/>	Small
<input type="radio"/>	Medium
<input type="radio"/>	Large

Pepperoni

Anchovies

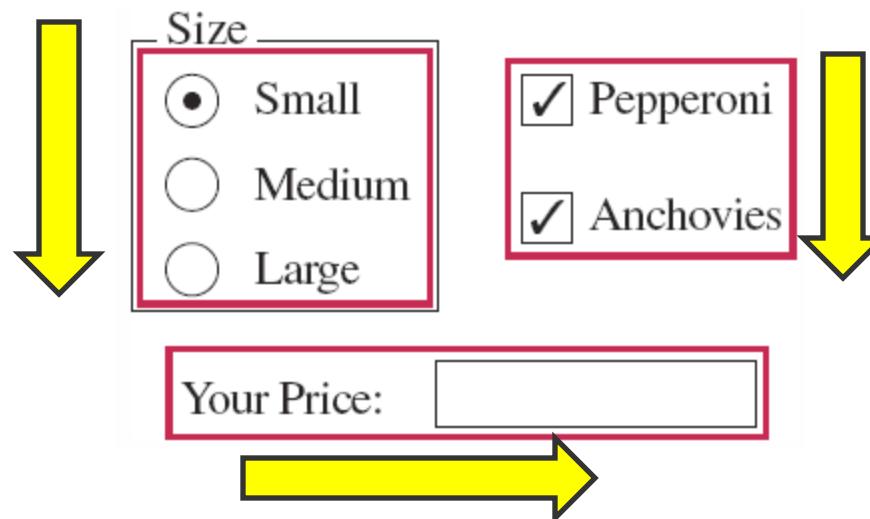
Your Price:



Steps to Design a User Interface

2) Find groupings of adjacent components with the same layout.

- Start by looking at adjacent components that are arranged top to bottom or left to right





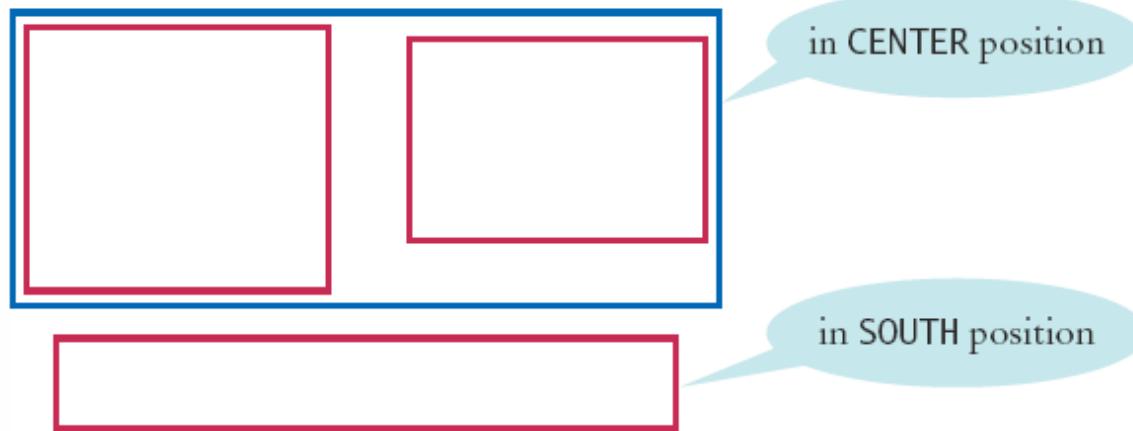
Steps to Design a User Interface

3) Identify layouts for each group.

- For horizontal components, use flow Layout
- For vertical components, use a grid layout with one column

4) Group the groups together.

- Look at each group as one blob, and group the blobs together into larger groups, just as you grouped the components in the preceding step





Steps to Design a User Interface

5) Write the code to generate the layout

```
JPanel radioButtonPanel = new JPanel();
radioButtonPanel.setLayout(new GridLayout(3, 1));
radioButton.setBorder(new TitledBorder(new EtchedBorder(), "Size"));
radioButtonPanel.add(smallButton);
radioButtonPanel.add(mediumButton);
radioButtonPanel.add(largeButton);
```

```
JPanel checkBoxPanel = new JPanel();
checkBoxPanel.setLayout(new GridLayout(2, 1));
checkBoxPanel.add(pepperoniButton());
checkBoxPanel.add(anchoviesButton());
```

```
JPanel pricePanel = new JPanel(); // Uses FlowLayout by default
pricePanel.add(new JLabel("Your Price:"));
pricePanel.add(priceTextField);
JPanel centerPanel = new JPanel(); // Uses FlowLayout
centerPanel.add(radioButtonPanel);
centerPanel.add(checkBoxPanel); // Frame uses BorderLayout by default
add(centerPanel, BorderLayout.CENTER);
add(pricePanel, BorderLayout.SOUTH);
```

Size

- Small
- Medium
- Large

Pepperoni

Anchovies

Your Price:



Use a GUI Builder



- A GUI Builder allows you to drag and drop components onto a panel and generates the code for you
- Try the free NetBeans development environment, available from <http://netbeans.org>
 - Uses new Java 6 GroupLayout

The screenshot shows the NetBeans IDE interface. On the left, there's a code editor window titled "PizzaFrame.java" with tabs for "Source" and "Design". A speech bubble points to the "Source" tab with the text "Click here to view generated source code". In the center, a preview window displays a Java Swing application frame. Inside the frame, there's a group of radio buttons labeled "Size" with options "Small", "Medium", and "Large". To the right of this is another group of checkboxes with "Pepperoni" and "Anchovies" checked. Below these controls is a text input field labeled "Your Price:". On the right side of the interface is a "Palette" window titled "Swing" which lists various Java Swing components with their corresponding icons.

Click here to view generated source code

PizzaFrame.java

Source Design

Palette

Swing

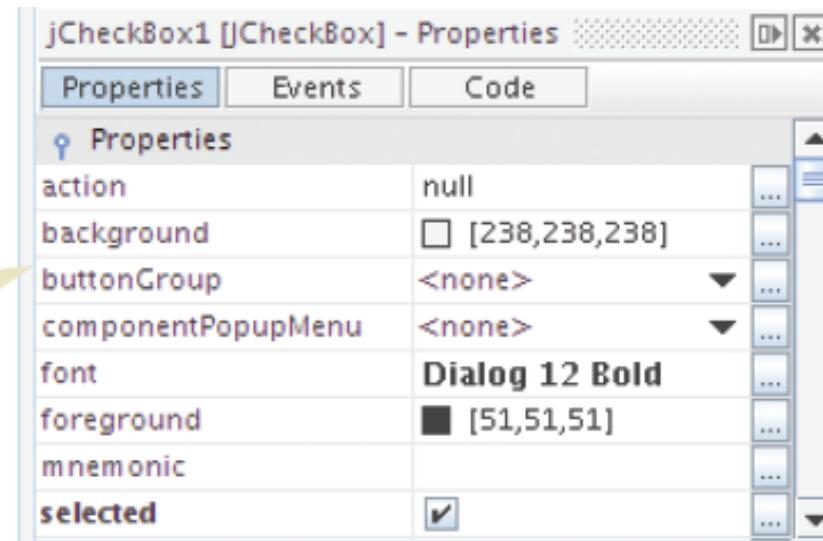
- JLabel
- JToggleButton
- JRadioButton
- JComboBox
- JTextField
- JPanel
- JScrollBar
- JMenuBar
- JButton
- JCheckBox
- ButtonGroup
- JList
- JTextArea
- JTabbedPane
- JScrollPane
- JPopupMenu



GUI Builder Component Properties

- You can configure properties of each component
 - Select a component on the screen (JCheckBox1 in this example)
 - Select Properties, set color, font, default state...
- You can setup event handlers by picking the event to process and providing just the code
 - Select an event under the Events tab
 - Write the code under the Code tab

Use this dialog box
to edit component
properties

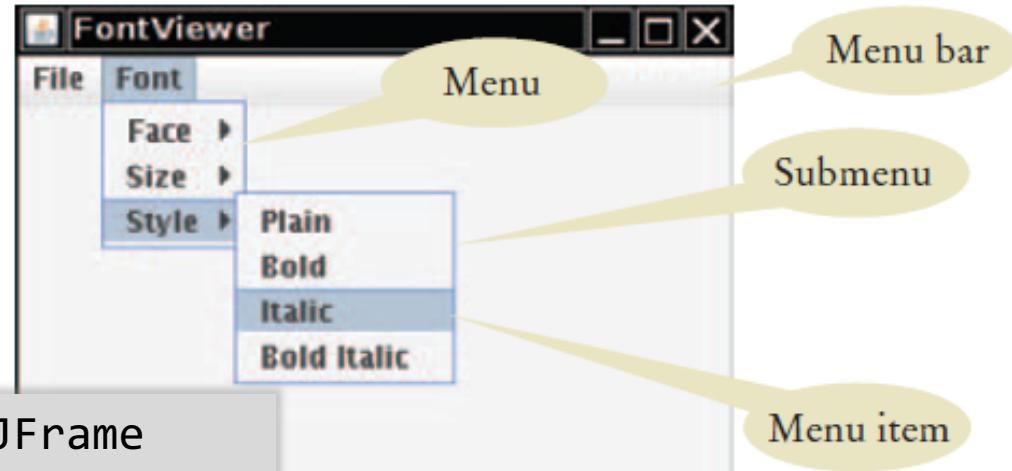




11.3 Menus

- A frame can contain a menu bar

- Menu items can be added to each Menu or subMenu



```
public class MyFrame extends JFrame  
{  
    public MyFrame()  
    {  
        JMenuBar menuBar = new JMenuBar();  
        setJMenuBar(menuBar);  
        . . .  
    }  
    . . .  
}
```

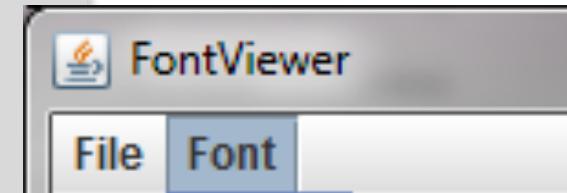
Instantiate a menu bar, then add it to the frame with the `setJMenuBar` method.



MenuBar and Menu Items

- The MenuBar contains Menus
 - The container for the top-level Menu items is called a MenuBar
 - Add JMenuItem objects to the MenuBar

```
JMenuBar menuBar = new JMenuBar();
JMenu fileMenu = new JMenu("File");
JMenu fontMenu = new JMenu("Font");
menuBar.add(fileMenu);
menuBar.add(fontMenu);
```



- A Menu contains SubMenus and Menu items
 - A Menu item has no further SubMenus
 - You add Menu items and SubMenus with the add method



Menu Item Events

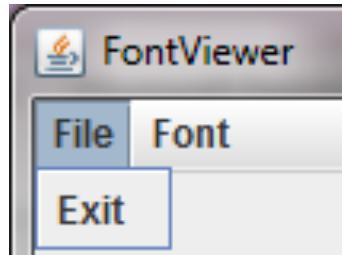
- ❑ Menu items generate action events when selected
 - Add action listeners only to menu items
 - Not to menus or the menu bar
 - When the user clicks on a menu name and a submenu opens, no action event is sent





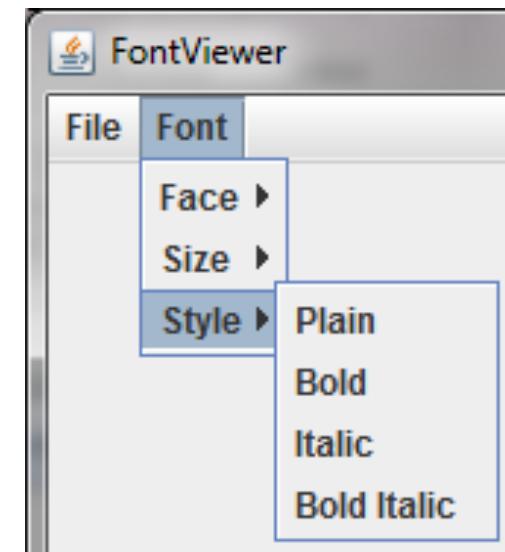
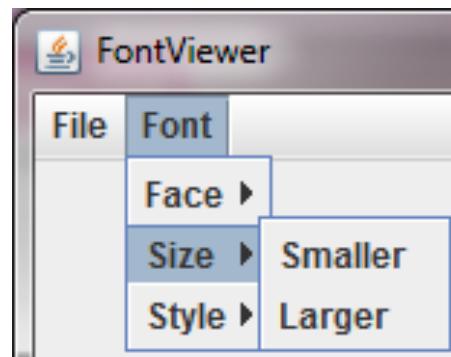
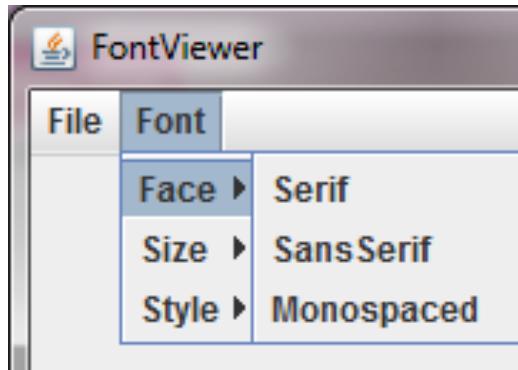
Menu Item Events

- Add action listeners to each Menu item



```
ActionListener listener = new ExitItemListener();  
exitItem.addActionListener(listener);
```

- The listener is customized for each Menu item

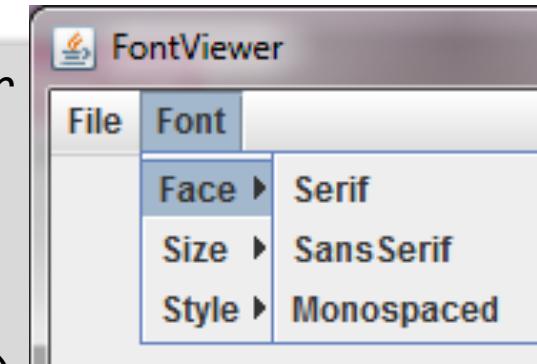




Example Menu Item Listener

- ❑ **createFaceEvent ActionListener Tasks**
 - Takes a String parameter variable (name of font face)
 - 1. Set the current face name to the menu item text
 - 2. Make a new font from the current face, size, and style, and apply it to the label

```
class FaceItemListener implements ActionListener
{
    private String name;
    public FaceItemListener(String newName)
    { name = newName; }
    public void actionPerformed(ActionEvent event)
    {
        faceName = name; // Sets instance variable of frame class
        setLabelFont();
    }
}
```





FaceEvent ActionListener (1)

- ❑ Install the listener object with appropriate name:

```
public JMenuItem createFaceItem(String name)
{
    JMenuItem item = new JMenuItem(name);
    ActionListener listener = new FaceItemListener(name);
    item.addActionListener(listener);
    return item;
}
```

- ❑ Not Optimal: Use a local inner class instead
 - When we move the declaration of the inner class inside the createFaceItem method, the actionPerformed method can access the name parameter variable directly (rather than passing it)



FaceEvent ActionListener (2)

□ Listener Inner Class version

```
public JMenuItem createFaceItem(final String name)
// Final variables can be accessed from an inner class method
{
    class FaceItemListener implements ActionListener
    {
        public void actionPerformed(ActionEvent event)
        {
            facename = name; // Accesses the local variable name
            setLabelFont();
        }
    }
    JMenuItem item = new JMenuItem(name);
    ActionListener listener = new FaceItemListener();
    item.addActionListener(listener);
    return item;
}
```



FontViewer2.java - Menu

```
25  /**
26   * Constructs the frame.
27  */
28 public FontFrame2()
29 {
30     // Construct text sample
31     label = new JLabel("Big Java");
32     add(label, BorderLayout.CENTER);
33
34     // Construct menu
35     JMenuBar menuBar = new JMenuBar();
36     setJMenuBar(menuBar);
37     menuBar.add(createFileMenu());
38     menuBar.add(createFontMenu());
39
40     facename = "Serif";
41     fontsize = 24;
42     fontstyle = Font.PLAIN;
43
44     setLabelFont();
45     setSize(FRAME_WIDTH, FRAME_HEIGHT);
46 }
```



Creates the top level menu bar with and adds it to the frame using the `setJMenuBar` method

Creates the File and Font menus using helper methods



FontViewer2.java – File Menu

```
48 class ExitItemListener implements ActionListener  
49 {  
50     public void actionPerformed(ActionEvent event)  
51     {  
52         System.exit(0);  
53     }  
54 }
```

Inner class of frame handles
File Menu Exit event

```
60     public JMenu createFileMenu()  
61     {  
62         JMenu menu = new JMenu("File");  
63         JMenuItem exitItem = new JMenuItem("Exit");  
64         ActionListener listener = new ExitItemListener();  
65         exitItem.addActionListener(listener);  
66         menu.add(exitItem);  
67         return menu;  
68     }
```

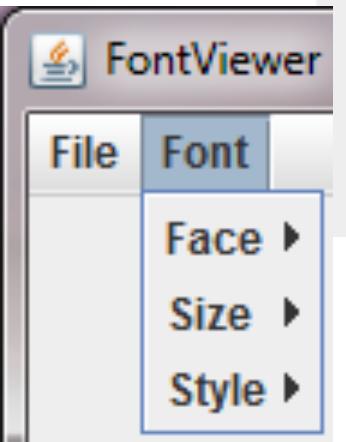
Creates the File menu, adds a
menu item Exit, instantiates the
inner class ExitItemListener,
registers it, and adds exitItem to
the menu



FontViewer2.java – Submenus

```
74     public JMenu createFontMenu()  
75     {  
76         JMenu menu = new JMenu("Font");  
77         menu.add(createFaceMenu());  
78         menu.add(createSizeMenu());  
79         menu.add(createStyleMenu());  
80         return menu;  
81     }
```

Creates the Font menu and adds submenus using helper methods



```
87     public JMenu createFaceMenu()  
88     {  
89         JMenu menu = new JMenu("Face");  
90         menu.add(createFaceItem("Serif"));  
91         menu.add(createFaceItem("SansSerif"));
```

Font Face submenu

```
92     public JMenu createSizeMenu()  
93     {  
94         JMenu menu = new JMenu("Size");  
95         menu.add(createSizeItem("Smaller", -1));  
96         menu.add(createSizeItem("Larger", 1));  
97         return menu;
```

Font Size submenu

```
98     public JMenu createStyleMenu()  
99     {  
100        JMenu menu = new JMenu("Style");  
101        menu.add(createStyleItem("Plain", Font.PLAIN));  
102        menu.add(createStyleItem("Bold", Font.BOLD));
```

Font Style submenu



FontViewer2.java – Listeners

```
128     public JMenuItem createFaceItem(final String name)
129     {
130         class FaceItemListener implements ActionListener
131         {
132             public void actionPerformed(ActionEvent event)
133             {
134                 facename = name;
135                 setLabelFont();
136             }
137             public JMenuItem createSizeItem(String name, final int increment)
138             {
139                 class SizeItemListener implements ActionListener
140                 {
141                     public void actionPerformed(ActionEvent event)
142                     {
143                         fontsize = fontsize + increment;
144                         setLabelFont();
145                     }
146                 }
147             }
148             JMenuItem item = new JMenuItem(name);
149             ActionListener listener = new SizeItemListener();
150             item.addActionListener(listener);
151             return item;
152         }
153     }
```

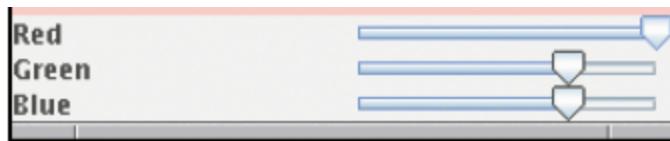
Inner class listener

Each Menu Item handles
its own events



11.4 Exploring Swing Documentation

- ❑ You should learn to navigate the API documentation to find out more about user-interface components
 - Examples so far has only used basic features of Swing
 - The purpose of this section is to show you how you can use the documentation to your advantage without becoming overwhelmed
- ❑ Example: Use sliders to set colors of red, green and blue





Swing Documentation

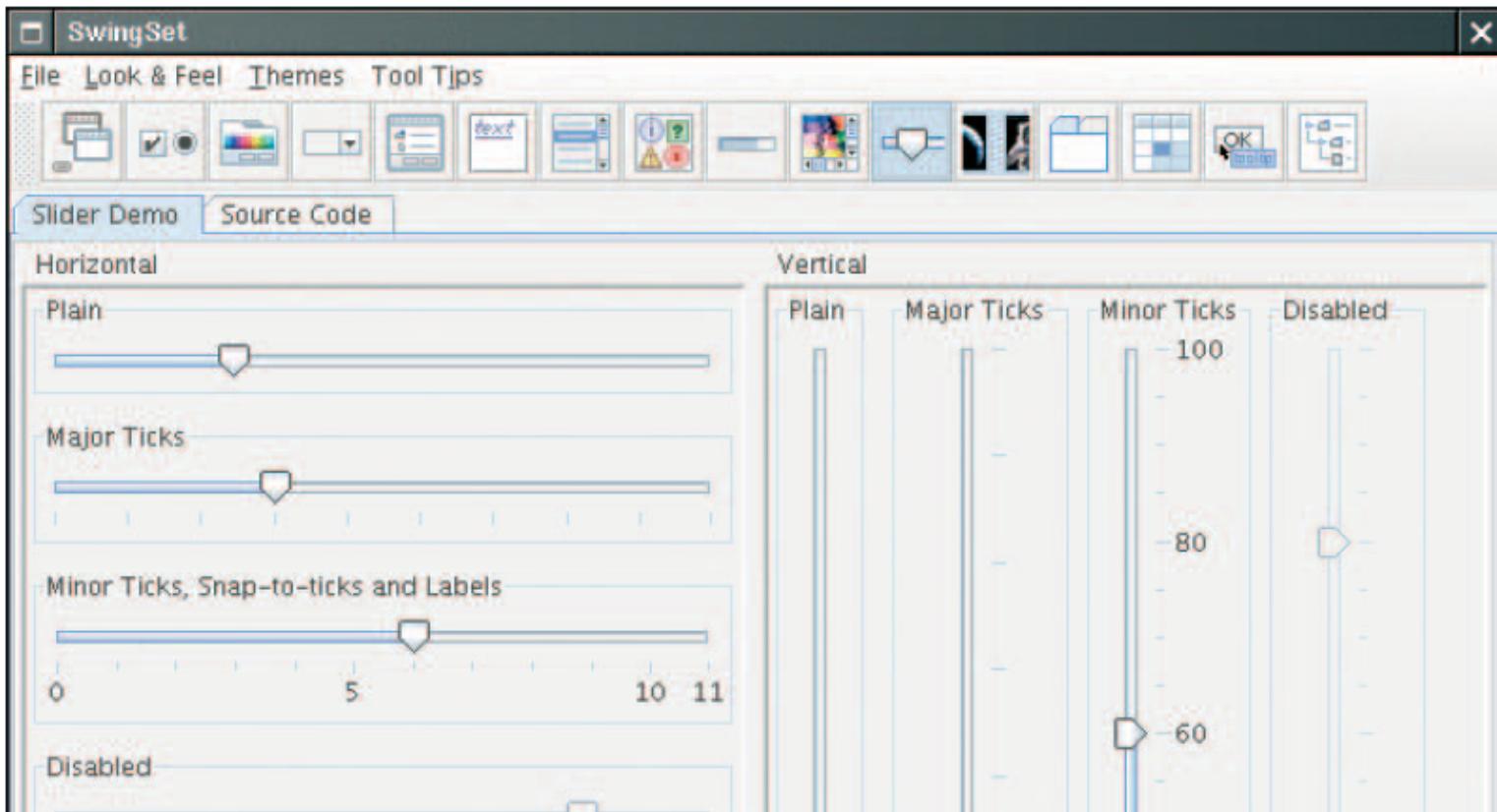
- ❑ The Swing user-interface toolkit provides a large set of components, including the Slider
 - How do you know if there is a slider?
 - Buy a book that illustrates all Swing components
 - Run the sample application included in the Java Development Kit that shows off all Swing components
 - Look at the names of all of the classes that start with J and decide that JSlider may be a good candidate



SwingSet Demo Examples

❑ Use the online demo for examples

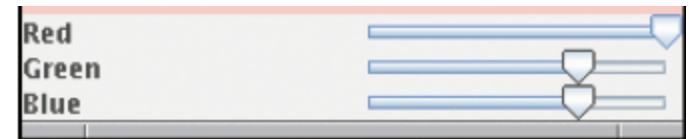
- http://java.sun.com/products/plugin/1.3.1_01a/demos/jfc/SwingSet2/SwingSet2Plugin.html (or Google Java SwingSet Demo)
- Use the Source Code tab to see how it works





JSlider Documentation and Use

- Next, you need to ask yourself 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?



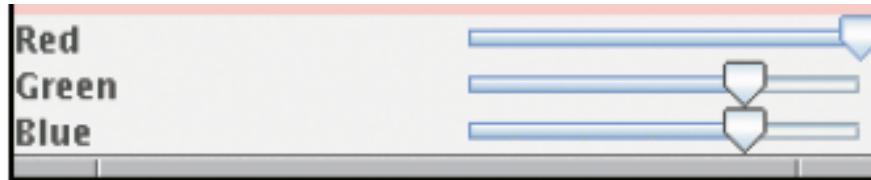
- When you look at the documentation of the `JSlider` class, you will probably not be happy.
 - There are over 50 methods in the `JSlider` class and over 250 inherited methods, and some of the method descriptions look downright scary
- Concentrate on what you will need
 - Constructors Event handling Get the value



JSlider Constructor Choice

□ Constructor Options

- We want a value between 0 and 255
- Find one that will do what we need:
 - `public JSlider()`
 - Creates a horizontal slider with the range 0 to 100 and an initial value of 50.
 - `public JSlider(BoundedRangeModel brm)`
 - Creates a horizontal slider using the specified BoundedRangeModel
 - `public JSlider(int min, int max, int value)`
 - Creates a horizontal slider using the specified min, max, and value





JSlider Event Handling

- Goal: Add a change event listener to each slider
 - There is no addActionListener method. That makes sense. Adjusting a slider seems different from clicking a button, and Swing uses a different event type for these events
- That leaves a couple of possible options:
 - `public void addChangeListener(ChangeListener l)`
 - Looks familiar... Like `AddActionListener` !
 - What is a `ChangeListener`? Like an `ActionListener`, but for a slider!
 - Has a `stateChanged` event instead of an `ActionPerformed` event
 - `void stateChanged(ChangeEvent e)`
 - Called whenever the user adjusts the slider... Perfect!
 - What is a `ChangeEvent`? We won't need it – it says which slider



JSlider Event Handling

- So the plan is:

- Setup three sliders and one `ChangeListener` object
- Use `AddChangeListener`, passing our `ChangeListener` to each slider
- In the `stateChanged` method, check the values of the colors



JSlider Example

```
16  private JPanel colorPanel;
17  private JSlider redSlider;
18  private JSlider greenSlider;
19  private JSlider blueSlider;
20
21  public ColorFrame()
22  {
23      colorPanel = new JPanel();
24
25      add(colorPanel, BorderLayout.CENTER);
26      createControlPanel();
27      setSampleColor();
28      setSize(FRAME_WIDTH, FRAME_HEIGHT);
29  }
30
31  class ColorListener implements ChangeListener
32  {
33      public void stateChanged(ChangeEvent event)
34      {
35          setSampleColor();
36      }
37  }
38
```

Constructor of ColorFrame calls
createControlPanel helper method

Setup inner class to handle slider
event stateChanged and call helper
method setSampleColor()



JSlider createControlPanel

```
39 public void createControlPanel()
40 {
41     ChangeListener listener = new ColorListener();
42
43     redSlider = new JSlider(0, 255, 255);
44     redSlider.addChangeListener(listener);
45
46     greenSlider = new JSlider(0, 255, 175);
47     greenSlider.addChangeListener(listener);
48
49     blueSlider = new JSlider(0, 255, 175);
50     blueSlider.addChangeListener(listener);
51
52     JPanel controlPanel = new JPanel();
53     controlPanel.setLayout(new GridLayout(3, 2));
54
55     controlPanel.add(new JLabel("Red"));
56     controlPanel.add(redSlider);
57
58     controlPanel.add(new JLabel("Green"));
59     controlPanel.add(greenSlider);
60
61     controlPanel.add(new JLabel("Blue"));
62     controlPanel.add(blueSlider);
63
64     add(controlPanel, BorderLayout.SOUTH);
65 }
```

Instantiates one listener, and registers it for each slider

Adds each slider to the controlPanel, and then adds controlPanel to the frame



JSlider setSampleColor

```
67  /**
68   * Reads the slider values and sets the panel to
69   * the selected color.
70  */
71  public void setSampleColor()
72  {
73      // Read slider values
74
75      int red = redSlider.getValue();
76      int green = greenSlider.getValue();
77      int blue = blueSlider.getValue();
78
79      // Set panel background to selected color
80
81      colorPanel.setBackground(new Color(red, green, blue));
82      colorPanel.repaint();
83  }
84 }
```

Read each slider with the
getValue method

Set a new background
color for the panel using
the new color values.



11.5 Using Timer Events for Animation

- In this section, we will study timer events and use them to implement simple animations
- javax.swing provides a handy Timer class

A Swing timer notifies a listener with each “tick”.





javax.swing Timer class

❑ javax.swing.Timer

- Can generate a series of events at even time intervals
- Specify the frequency of the events and an object of a class that implements the ActionListener interface

```
class MyListener implements ActionListener
{
    public void actionPerformed(ActionEvent event)
    {
        // Listener action (executed at each timer event)
    }
}
MyListener listener = new MyListener();
Timer t = new Timer(interval, listener);
t.start();
```

Timer events will begin after the start method is called



Animated Rectangle Example

```
8  public class RectangleComponent extends JPanel
9  {
10     private static final int RECTANGLE_WIDTH = 20;
11     private static final int RECTANGLE_HEIGHT = 30;
12
13     private int xLeft;
14     private int yTop;
15
16     public RectangleComponent()
17     {
18         xLeft = 0;
19         yTop = 0;
20     }
21
22     public void paintComponent(Graphics g)
23     {
24         g.fillRect(xLeft, yTop, RECTANGLE_WIDTH, RECTANGLE_HEIGHT);
25     }
26
27     public void moveRectangleBy(int dx, int dy)
28     {
29         xLeft = xLeft + dx;
30         yTop = yTop + dy;
31         repaint();
32     }
33 }
```

repaint method invokes the
paintComponent method



RectangleFrame.java

```
9  public class RectangleFrame extends JFrame
10 {
11     private static final int FRAME_WIDTH = 300;
12     private static final int FRAME_HEIGHT = 400;
13
14     private RectangleComponent scene;
15
16     class TimerListener implements ActionListener
17     {
18         public void actionPerformed(ActionEvent event)
19         {
20             scene.moveRectangleBy(1, 1);
21         }
22     }
23
24     public RectangleFrame()
25     {
26         scene = new RectangleComponent();
27         add(scene);
28
29         setSize(FRAME_WIDTH, FRAME_HEIGHT);
30
31         ActionListener listener = new TimerListener();
32
33         final int DELAY = 100; // Milliseconds between timer ticks
34         Timer t = new Timer(DELAY, listener);
35         t.start();
36     }
37 }
```

Inner class TimerListener moves the rectangle

TimerListener is called every 100ms by the Timer object



repaint and paintComponent

- When you make a change to the data, the component is not automatically painted with the new data
- You must call the repaint method of the component, either in the event handler or in the component's mutator methods
- Your component's paintComponent method will then be invoked with an appropriate Graphics object

Do not call the paintComponent method directly.



11.6 Mouse Events

- ❑ If you write programs that show drawings, and you want users to manipulate the drawings with a mouse, then you need to process mouse events

In Swing, a mouse event is a notification of a mouse click by the program user.



- ❑ But, mouse events are more complex than button clicks or timer ticks:



Mouse Events

- A mouse listener must implement all five methods of the `MouseListener` interface

method	Triggering event
<code>mousePressed</code>	A mouse button has been pressed on a component
<code>mouseReleased</code>	A mouse button has been released on a component
<code>mouseClicked</code>	The mouse has been clicked on a component
<code>mouseEntered</code>	The mouse enters a component
<code>mouseExited</code>	The mouse exits a component

- The `mousePressed` and `mouseReleased` methods are called whenever a mouse button is pressed or released



MouseListener Methods

- ❑ If a button is pressed and released in quick succession, and the mouse has not moved, then the `mouseClicked` method is called as well.
- ❑ The `mouseEntered` and `mouseExited` methods can be used to paint a user-interface component in a special way whenever the mouse is pointing inside it.

```
public interface MouseListener
{
    void mousePressed(MouseEvent event);
    void mouseReleased(MouseEvent event);
    void mouseClicked(MouseEvent event);
    void mouseEntered(MouseEvent event);
    void mouseExited(MouseEvent event);
}
```



Implementing MouseListener

- Another difference is that you use the `addMouseListener` method instead of `addActionListener` to add it to a component:

```
public class MyMouseListener implements MouseListener
{
    // Implements five methods
}
MouseListener listener = new MyMouseListener();
component.addMouseListener(listener);
```

- In our sample program, a user clicks on a component containing a rectangle. Whenever the mouse button is pressed, the rectangle is moved to the mouse location.



RectangleComponent2.java

```
8 public class RectangleComponent2 extends JPanel
9 {
10     private static final int RECTANGLE_WIDTH = 20;
11     private static final int RECTANGLE_HEIGHT = 30;
12
13     private int xLeft;
14     private int yTop;
15
16     public RectangleComponent2()
17     {
18         xLeft = 0;
19         yTop = 0;
20     }
21
22     public void paintComponent(Graphics g)
23     {
24         g.fillRect(xLeft, yTop, RECTANGLE_WIDTH, RECTANGLE_HEIGHT);
25     }
26
27
28     public void moveRectangleTo(int x, int y)
29     {
30         xLeft = x;
31         yTop = y;
32         repaint();
33     }
34 }
```

- Modified moveRectangleTo method takes mouse location
- Now we need a mouse listener to call it

repaint method invokes the
paintComponent method



The MouseListener Methods

- mousePressed is the only method we want to handle, but we need to write them all (even though they are empty):

```
15  class MousePressListener implements MouseListener
16  {
17      public void mousePressed(MouseEvent event)
18      {
19          int x = event.getX();
20          int y = event.getY();
21          scene.moveRectangleTo(x, y);    Some useful methods of the event
22      }                                object tell us where the mouse was!
23
24      // Do-nothing methods
25      public void mouseReleased(MouseEvent event) {}
26      public void mouseClicked(MouseEvent event) {}
27      public void mouseEntered(MouseEvent event) {}
28      public void mouseExited(MouseEvent event) {}
29  }
30
```

- Where did it get that `scene` reference?
 - Must be an inner class of a method with that reference variable!



RectangleFrame2.java (the rest)

```
8  public class RectangleFrame2 extends JFrame
9  {
10     private static final int FRAME_WIDTH = 300;
11     private static final int FRAME_HEIGHT = 400;
12
13     private RectangleComponent2 scene;
14
15     class MousePressListener implements MouseListener
16     {
17         public void mousePressed(MouseEvent event)
18         {
19             int x = event.getX();
20             int y = event.getY();
21             scene.moveRectangleTo(x, y);
22         }
23
24         // Do-nothing
25         public void mouseClicked(MouseEvent event)
26         {
27         }
28
29     }
30
31     public RectangleFrame2()
32     {
33         scene = new RectangleComponent2();
34         add(scene);
35
36         MouseListener listener = new MousePressListener();
37         scene.addMouseListener(listener);
38
39         setSize(FRAME_WIDTH, FRAME_HEIGHT);
40     }
41 }
```

Scene is the name of the RectangleComponent2 object



Summary: Containers and Layouts

- ❑ 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 are the border layout, flow layout, and grid layout.
 - When adding a component to a container with the border layout, specify the NORTH, EAST, SOUTH, WEST, or CENTER position
- ❑ The content pane of a frame has a border layout by default
- ❑ A panel has a flow layout by default



Summary: Swing Components

- ❑ For a small set of mutually exclusive choices, use a group of radio buttons or a combo box
- ❑ Add radio buttons into a `ButtonGroup` so that only one button in the group is on at any time
- ❑ You can place a border around a panel to group its contents visually
- ❑ For a binary choice, use a check box
- ❑ For a large set of choices, use a combo box
- ❑ Radio buttons, check boxes, and combo boxes generate action events, just as buttons do



Summary: Swing Menus

- A frame may contain a menu bar
 - The menu bar contains menus
 - A menu contains submenus and menu items
- Menu items generate action events
- You should learn to navigate the API documentation to find out more about user-interface components



Summary: Timers, Events

❑ Timers and Animation

- A timer generates action events at fixed intervals.
- To make an animation, the timer listener should update and repaint a component several times per second.

❑ Mouse Events

- You use a mouse listener to capture mouse events.