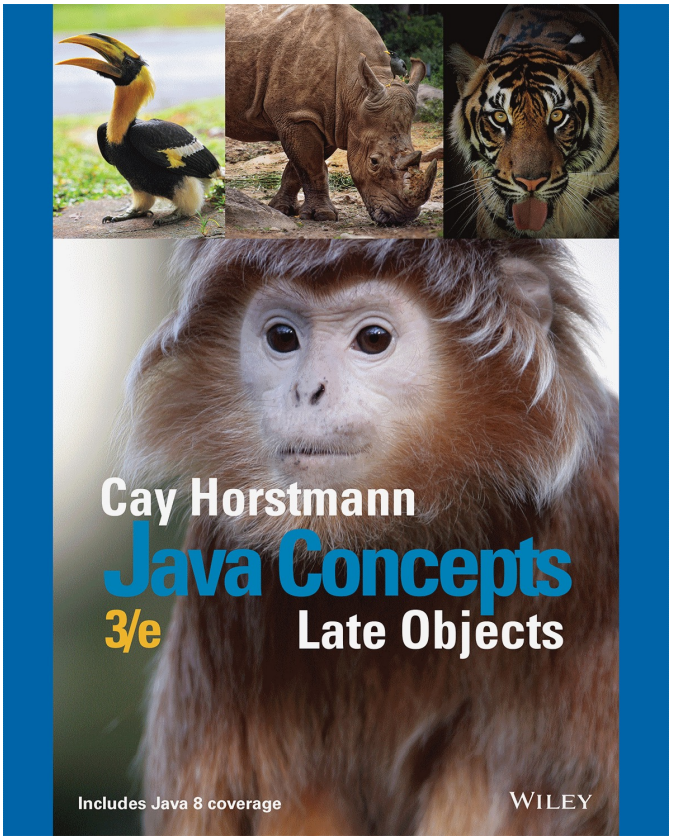
Chapter 11 - Advanced User Interfaces

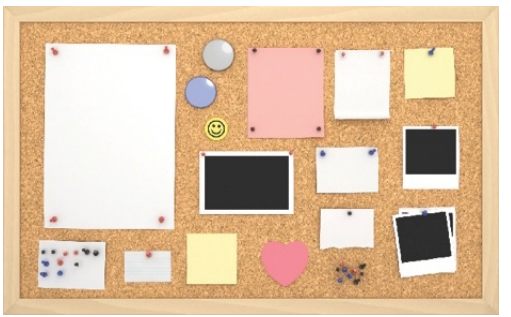


Chapter Goals

- 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

Frame Windows

- 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

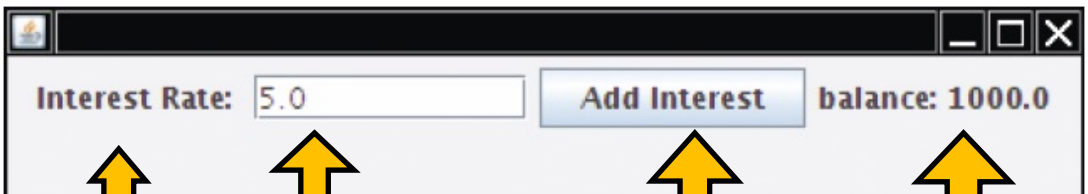
- Each container has a layout manager that directs the arrangement of its components
- Three useful layout managers are:

1) Flow layout – left to right, the default for `JPanel`
2) Border layout – five regions
3) Grid layout – rows and columns

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

Flow Layout

- Components are added from left to right
- A `JPanel` uses **flow layout** by default
- A new row is started when the current row fills

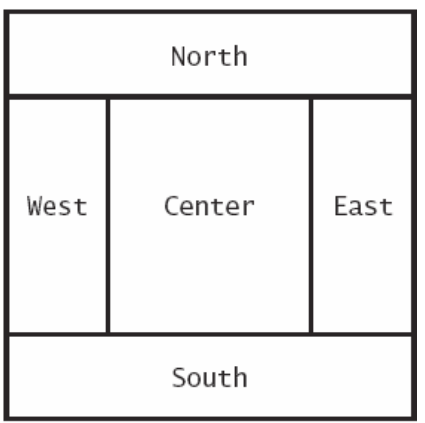



```
panel = new JPanel();
panel.add(rateLabel);
panel.add(rateField);
panel.add(button);
panel.add(resultLabel);
```

Border Layout

- Groups components into five 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);
```



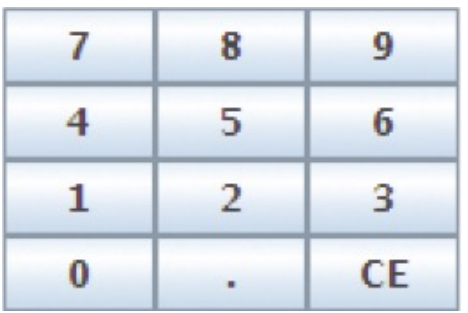
Grid Layout

- 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));
```

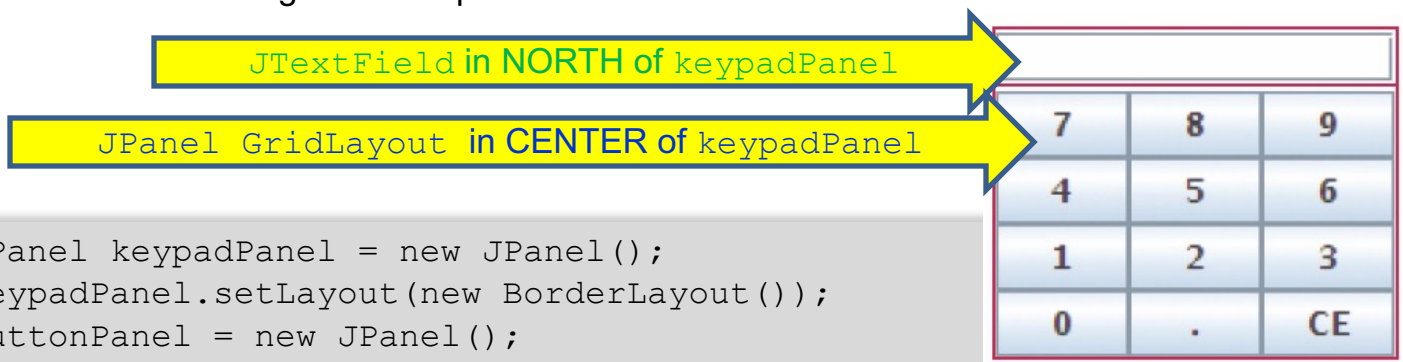
- Then add components which will be placed from the upper left, across, then down

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



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




```
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);
```

Self Check 11.1

What happens if you place two buttons in the northern position of a border layout? Try it out with a small program.

Answer: Only the second one is displayed.

Self Check 11.2

How do you add two buttons to the northern position of a frame so that they are shown next to each other?

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

Self Check 11.3

How can you stack three buttons one above the other?

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

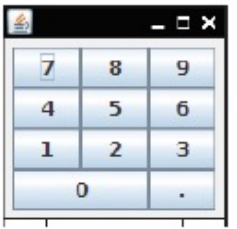
Self Check 11.4

What happens when you place one button in the northern position of a border layout and another in the center position? Try it out with a small program if you aren't sure.

Answer: The button in the north stretches horizontally to fill the width of the frame. The height of the northern area is the normal height. The center button fills the remainder of the window.

Self Check 11.5

Some calculators have a double-wide 0 button, as shown below. How can you achieve that?

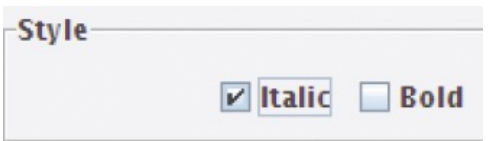


Answer: To get the double-wide button, put it in the south of a panel with border layout whose center has a 3 × 2 grid layout with the keys 7, 8, 4, 5, 1, 2. Put that panel in the west of another border layout panel whose eastern area has a 4 × 1 grid layout with the remaining keys.

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

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

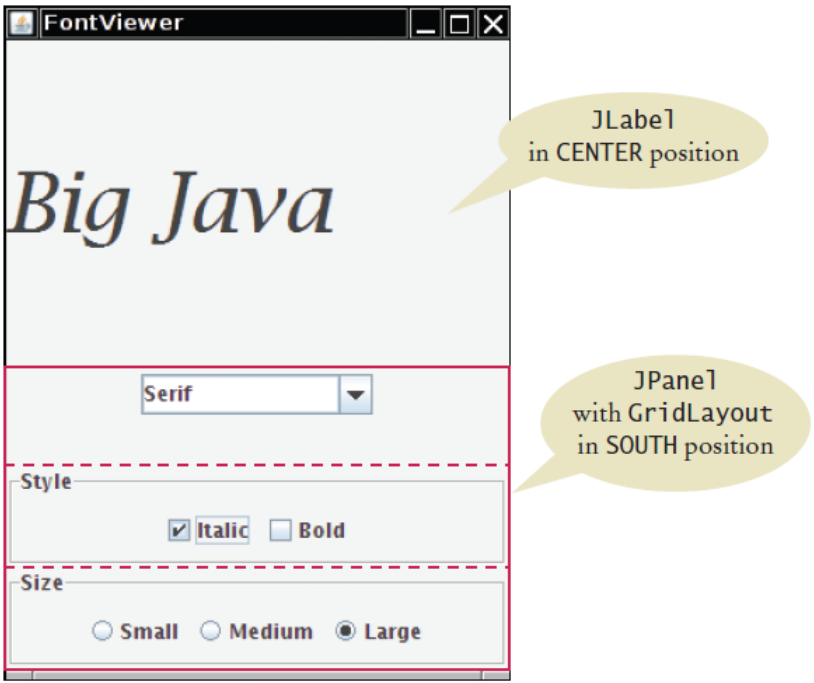

```
JRadioButton largeButton = new JRadioButton("Large");
largeButton.setSelected(true);
```

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

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

`FontViewer` Layout

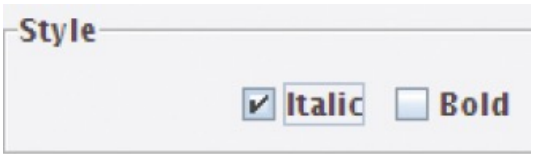
- Title Bar
- Label
  - Shows current font
- Combo Box
- Check Boxes
- Radio Buttons



Check Boxes versus Radio Buttons

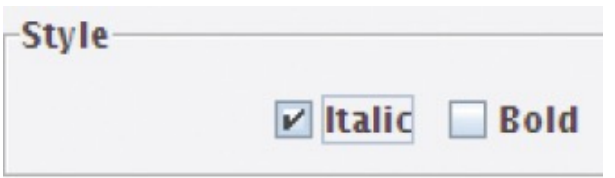
- Radio buttons and check boxes have different visual appearances
  - Radio buttons are round and show a black dot when selected

  - 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
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

`getSelectedItem` for the combo box

`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;
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

Self Check 11.6

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.

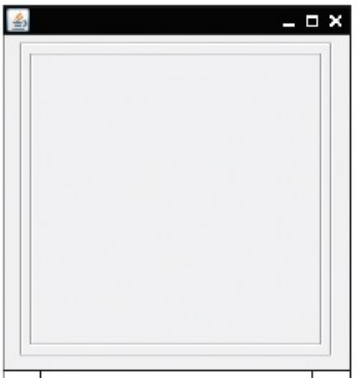
Self Check 11.7

What happens when you put two check boxes into a button group? Try it out if you are not sure.

Answer: If one of them is checked, the other one is unchecked. You should use radio buttons if that is the behavior you want.

Self Check 11.8

How can you nest two etched borders, like this?



Answer: You can't nest borders, but you can nest panels with borders:

```
JPanel p1 = new JPanel();
p1.setBorder(new EtchedBorder());
JPanel p2 = new JPanel();
p2.setBorder(new EtchedBorder());
p1.add(p2);
```

Self Check 11.9

Why do all user-interface components in the `FontFrame` 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.

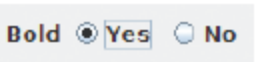
Self Check 11.10

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.

Self Check 11.11

How could the following user interface be improved?

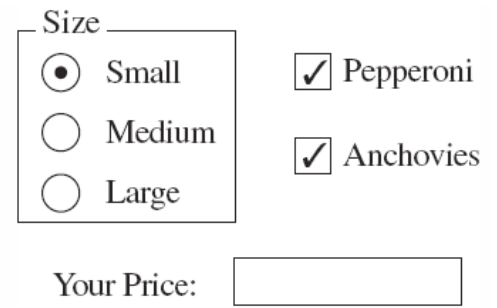


Answer: Instead of using radio buttons with two choices, use a checkbox.

Steps to Design a User Interface

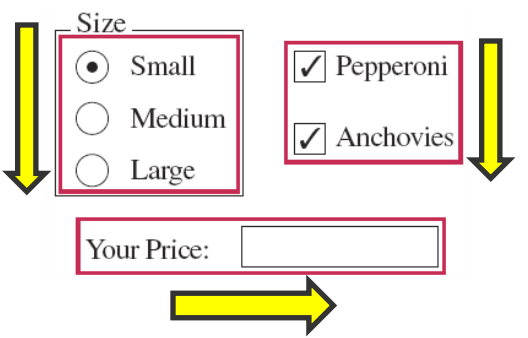
1) Make a sketch of the component layout.

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



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



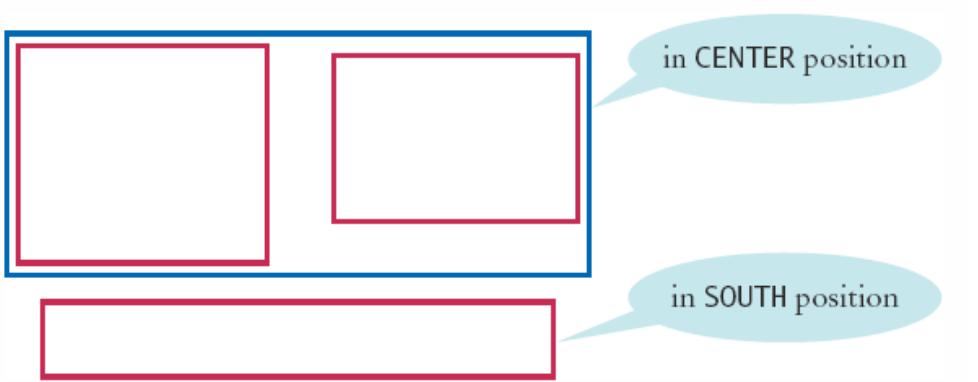
Steps to Design a User Interface

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.

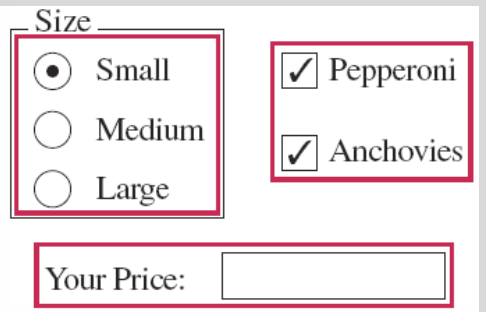
- Look at each group as one blob, and group the blobs together into larger groups, just as you grouped the components in the preceding step



Steps to Design a User Interface

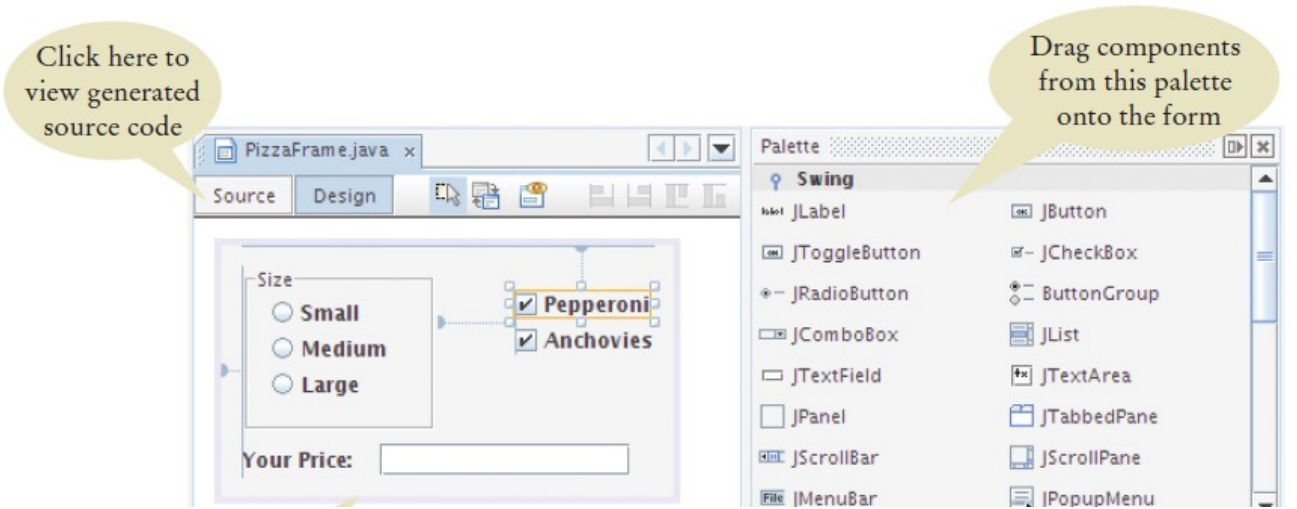
5) Write the code to generate the layout

```
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);
```



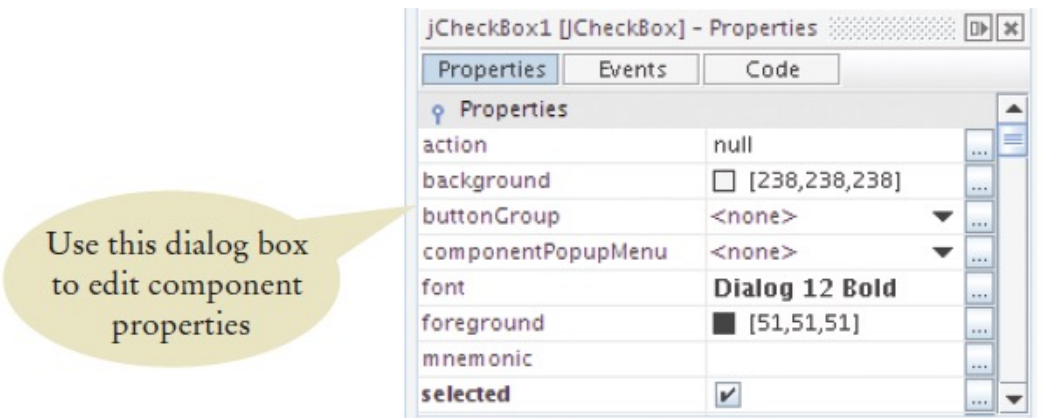
Use a GUI Builder

- 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`



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



Menus

- A frame can contain a menu bar
  - Menu items can be added to each Menu or subMenu

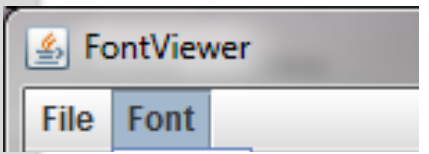
```
public class MyFrame extends JFrame
{
  public MyFrame()
  {
    JMenuBar menuBar = new JMenuBar();
    setJMenuBar(menuBar);
    . . .
  }
  . . .
}
```

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 `JMenu` objects to the `MenuBar`

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



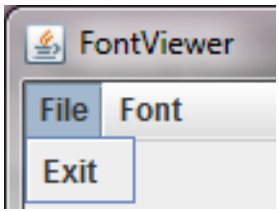
  - A menu contains submenus and menu items
    - A menu item has no further submenus
    - You add menu items and submenus with the `add` method

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

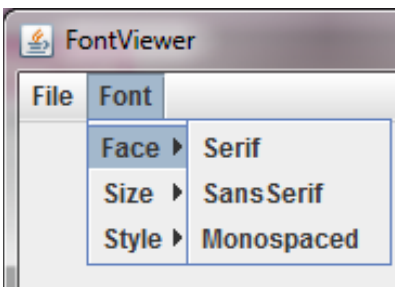
Menu Item Events

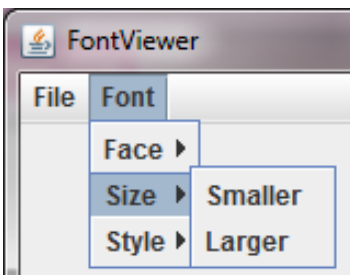
- Add action listeners to each Menu item

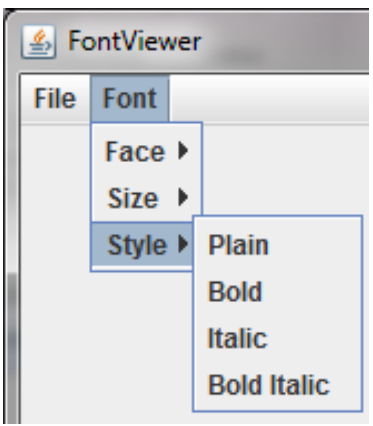



```
ActionListener listener = new ExitItemListener();
exitItem.addActionListener(listener);
```

- The listener is customized for each Menu item

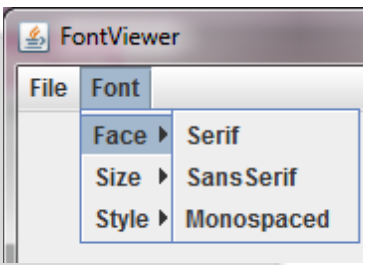






Example Menu Item Listener

- `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

Creates the Font menu and adds submenus using helper methods

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

Font Face submenu

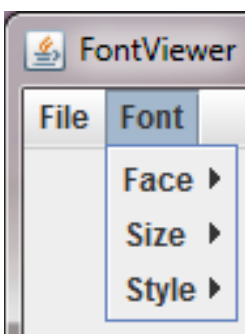
```
87    public JMenu createFaceMenu()
88    {
89       JMenu menu = new JMenu("Face");
90       menu.add(createFaceItem("Serif"));
91       menu.add(createFaceItem("SansSerif"));
92       menu.add(createFaceItem("Monospaced"));
93       return menu;
94    }
```

Font Size submenu

```
100   public JMenu createSizeMenu()
101   {
102      JMenu menu = new JMenu("Size");
103      menu.add(createSizeItem("Smaller", -1));
104      menu.add(createSizeItem("Larger", 1));
105      return menu;
106   }
```

Font Style submenu

```
112   public JMenu createStyleMenu()
113   {
114      JMenu menu = new JMenu("Style");
115      menu.add(createStyleItem("Plain", Font.PLAIN));
116      menu.add(createStyleItem("Bold", Font.BOLD));
```



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       }
138
139
140
141
142
143
152       public JMenuItem createSizeItem(String name, final int increment)
153       {
154          class SizeItemListener implements ActionListener
155          {
156             public void actionPerformed(ActionEvent event)
157             {
158                fontsize = fontsize + increment;
159                setLabelFont();
160             }
161          }
162
163          JMenuItem item = new JMenuItem(name);
164          ActionListener listener = new SizeItemListener();
165          item.addActionListener(listener);
166          return item;
167       }
```

Inner class listener

Each Menu Item handles its own events

Self Check 11.12

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 11.13

Can you add a menu item directly to the menu bar? Try it out. What happens?

Answer: Yes, you can—`JMenuItem` is a subclass of `JMenu`. The item shows up on the menu bar. When you click on it, its listener is called. But the behavior feels unnatural for a menu bar and is likely to confuse users.

Self Check 11.14

Why is the `increment` parameter variable in the `createSizeItem` method declared as `final`?

Answer: Using the `final` modifier enables the code to compile with versions prior to Java 8. In those versions, it was necessary to declare the parameter variable as `final` so that it could be accessed in a method of an inner class.

Self Check 11.15

Why can't the `createFaceItem` method simply set the `faceName` instance variable, like this:

```
class FaceItemListener implements ActionListener
{
   public void actionPerformed(ActionEvent event)
   {
      setLabelFont();
   }
}

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

Answer: Then the `faceName` variable is set when the menu item is added to the menu, not when the user selects the menu.

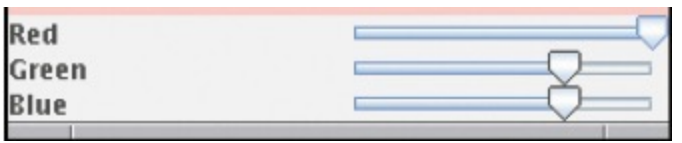
Self Check 11.16

In this program, the font specification (name, size, and style) is stored in instance variables. Why was this not necessary in the program of the previous section?

Answer: In the previous program, the user-interface components effectively served as storage for the font specification. Their current settings were used to construct the font. But a menu doesn't save settings; it just generates an action.

Exploring Swing Documentation

- You should learn to navigate the API documentation to find out more about user-interface components
  - Examples so far have 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

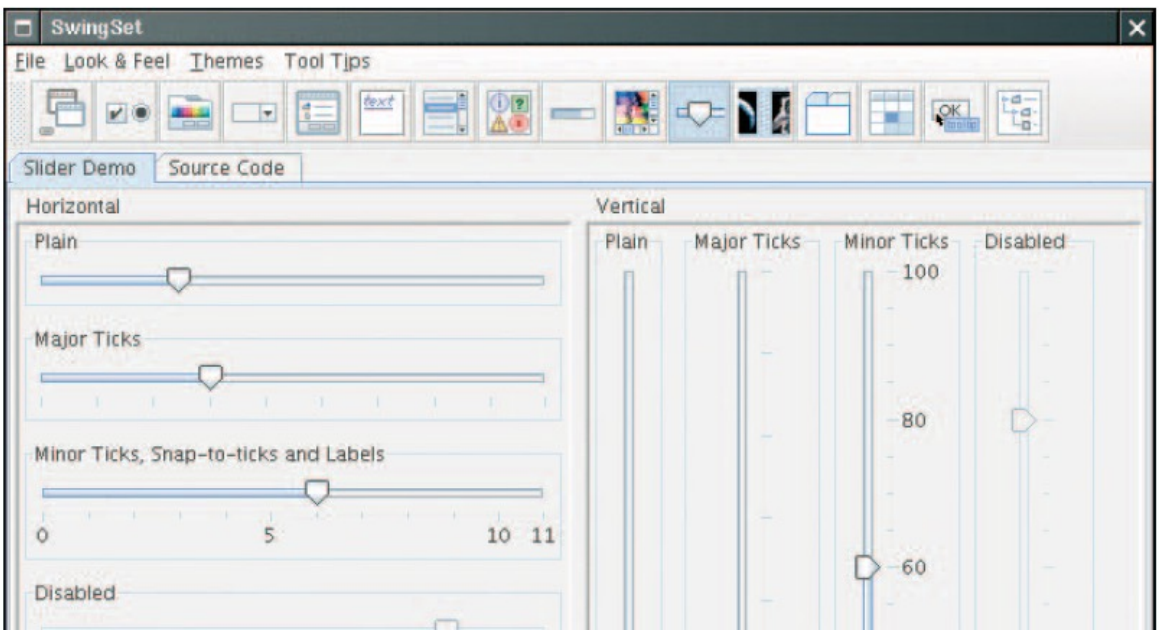


Swing Documentation

- 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

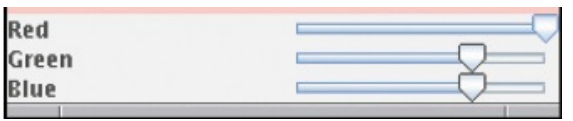
SwingSet Demo Examples

- 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?

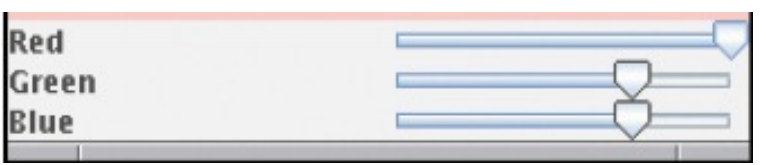


- When you look at the documentation of the `JSlider` class, you will probably not be happy
  - There are over 50 methods in the `JSlider` class and over 250 inherited methods, and some of the method descriptions look downright scary
- Concentrate on what you will need:

  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
- 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

Self Check 11.17

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 11.18

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.

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 JComponent
 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     }
```

```
32     public void moveRectangleBy(int dx, int dy)
33     {
34        xLeft = xLeft + dx;
35        yTop = yTop + dy;
36        repaint();
37     }
38  }
```

`repaint` method invokes the `paintComponent` method

`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.

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

Self Check 11.19

Why does a timer require a listener object?

Answer: The timer needs to call some method whenever the time interval expires. It calls the `actionPerformed` method of the listener object.

Self Check 11.20

How can you make the rectangle move backwards?

Answer: Call `scene.moveRectangleBy(-1, -1)` in the `actionPerformed` method.

Self Check 11.21

You can cut the timer delay in half (to 50 milliseconds between ticks), or you can double the distance by which the rectangle moves, by calling `scene.moveRectangleBy(2, 2)`.

Answer:

Self Check 11.22

How can you make a car move instead of a rectangle?

Answer: The component class would need to draw a car at positon (x, y) instead of a rectangle.

Self Check 11.23

How can you make two rectangles move in parallel in the scene?

Answer: There are two entirely different ways:

a. Add a second `RectangleComponent` to the frame, using a grid layout. Change the `actionPerformed` method of the `TimerListener` to call `moveRectangleBy` on both components.

b. Draw a second rectangle in the `paintComponent` method of `RectangleComponent`.

Self Check 11.24

What would happen if you omitted the call to `repaint` in the `moveRectangleBy` method?

Answer: The moved rectangles won't be painted, and the rectangle will appear to be stationary until the frame is repainted for an external reason.

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
`mousePressed`	A mouse button has been pressed on a component
`mouseReleased`	A mouse button has been released on a component
`mouseClicked`	The mouse has been clicked on a component
`mouseEntered`	The mouse enters a component
`mouseExited`	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

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

```
 8  public class RectangleComponent2 extends JComponent
 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     }
```

```
32     public void moveRectangleTo(int x, int y)
33     {
34        xLeft = x;
35        yTop = y;
36        repaint();
37     }
38  }
```

`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);
22       }
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
```

Some useful methods of the `event` object tell us where the mouse was!

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

RectangleFrame2.java

```
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 methods
25       public void mouseReleased(MouseEvent event) {}
26       public void mouseClicked(MouseEvent event) {}
27       public void mouseEnt
28       public void mouseExi
29    }
```

`Scene` is the name of the `RectangleComponent2` object

```
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 }
```

Self Check 11.25

Why was the `moveRectangleBy` method in `RectangleComponent2` replaced with a `moveRectangleTo` method?

Answer: Because you know the current mouse position, not the amount by which the mouse has moved.

Self Check 11.26

Why must the `MousePressListener` class supply five methods?

Answer: It implements the `MouseListener` interface, which has five methods.

Self Check 11.27

How could you change the behavior of the program so that a new rectangle is added whenever the mouse is clicked?

Answer: The `RectangleComponent2` class needs to keep track of the locations of multiple rectangles. It can do that with an array list of `Point` or `Rectangle` objects. The `paintComponent` method needs to draw them all. Replace the `moveRectangleTo` method with an `addRectangleAt` method that adds a rectangle at a given (*x*, *y*) position.

Special Topic (1)

- Keyboard Events
  - You may need to process keystrokes such as arrow keys in some applications by adding a key listener

```
class MyKeyListener implements KeyListener
{
   public void keyPressed(KeyEvent event)
   {
       String key = KeyStroke
              .getKeyStrokeForEvent(event)
              .toString();
       key = key.replace("pressed ", "");
       Process the key.
    }
    public void keyReleased(KeyEvent e) {}
    public void keyTyped(KeyEvent e) {}
}
```

`keyReleased` and `keyTyped` events are ignored

Special Topic (2)

- Keyboard Events
  - Create the `KeyListener` and add it to the component
  - The component calls `setFocusable()` in order to receive key events

```
KeyListener listener = new MyKeyListener();
scene.addKeyListener(listener);
scene.setFocusable(true);
```

Special Topic

- Event Adapters
  - An alternative to implementing the `MouseListener` interface and coding each of its five methods, is to extend the `MouseAdapter` class
  - All the methods in `MouseAdapter` class do nothing
  - The advantage is this allows you to override only the methods that are relevant to your application

```
class MouseClickListener extends MouseAdapter
{
   public void mouseClicked(MouseEvent event)
   {
      Mouse click action
   }
}
```