LECTURE 12

ADVANCED USER INTERFACES

Ian Stavness, CMPT 270, 2015-16 Term 1

Lecture Goals

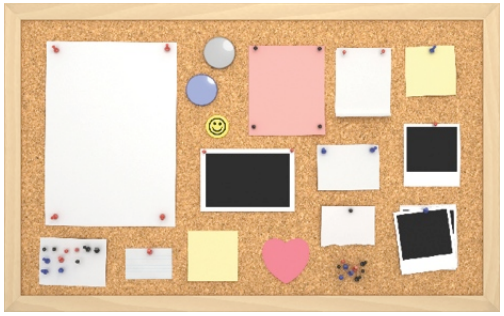
- To use layout managers to arrange user-interface components in a container
- To become familiar with common user-interface components, such as radio buttons, check boxes, and menus
- To build programs that handle events generated by user-interface components
- To browse the Java documentation effectively

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

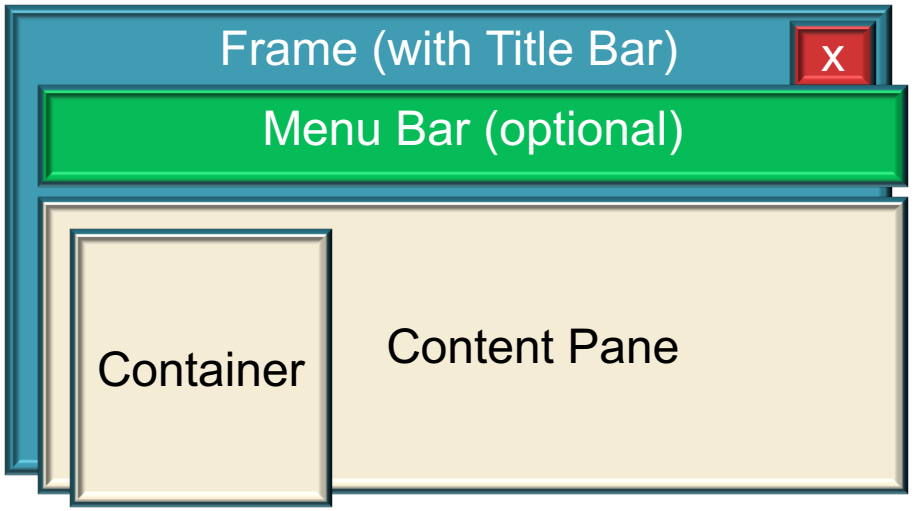
- A `JFrame` has a content pane which is a `Container` where you can add components
  - Use the `getContentPane` method to get its reference

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

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

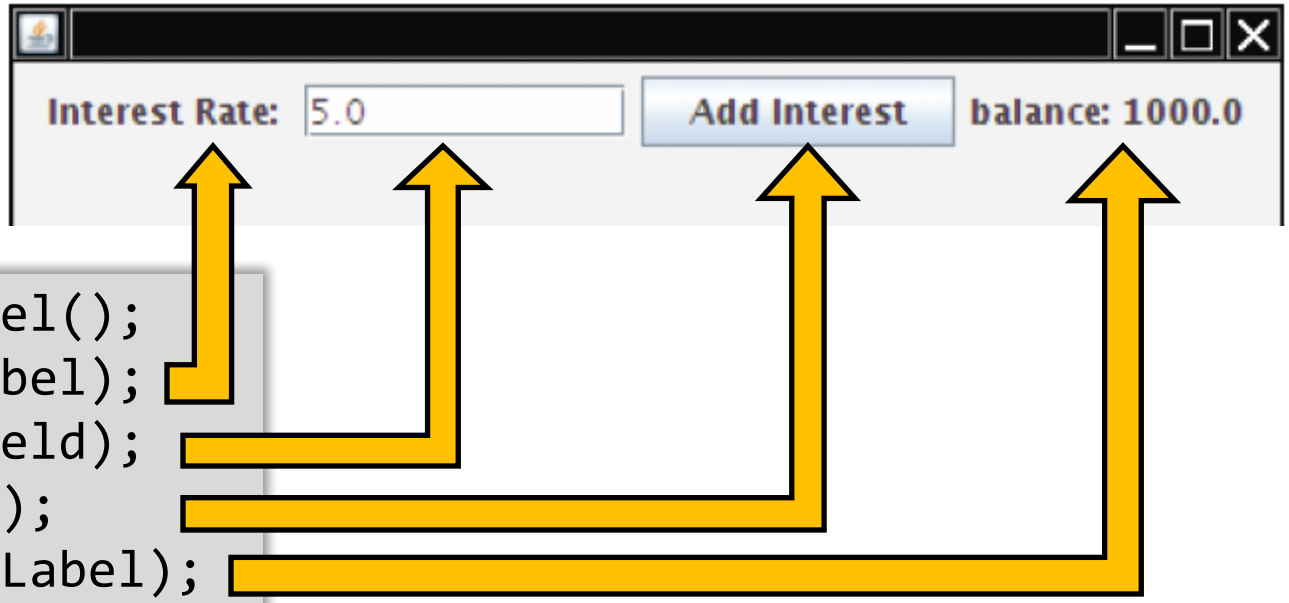
Jframe's Content Pane (2)

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



Flow Layout

- Components are added from left to right

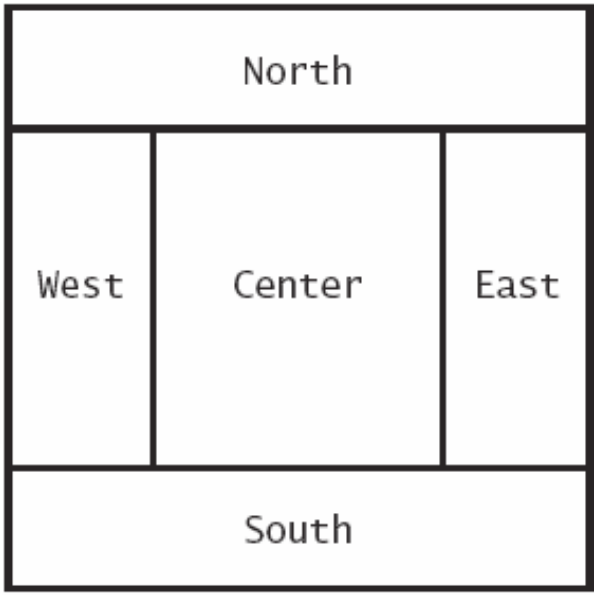



```
panel = new JPanel();
panel.add(rateLabel);
panel.add(rateField);
panel.add(button);
panel.add(resultLabel);
```

  - 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);
```

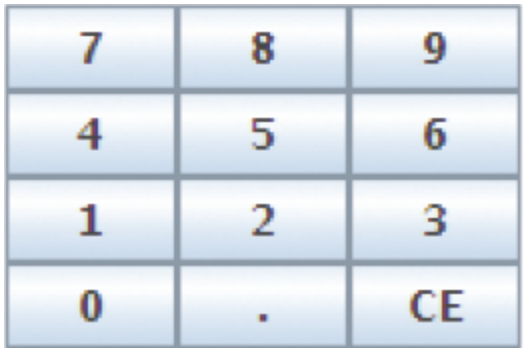
Grid Layout

- 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);
. . .
```

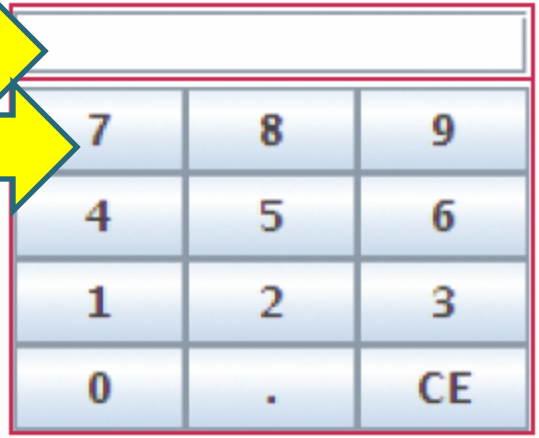


Using Nested Panels

- 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);
```

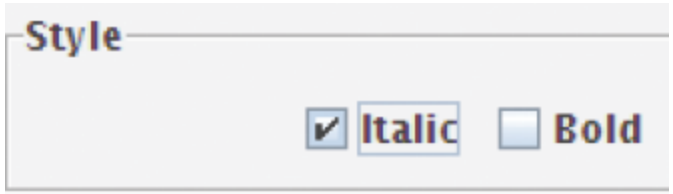
11.2 Choices

- 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



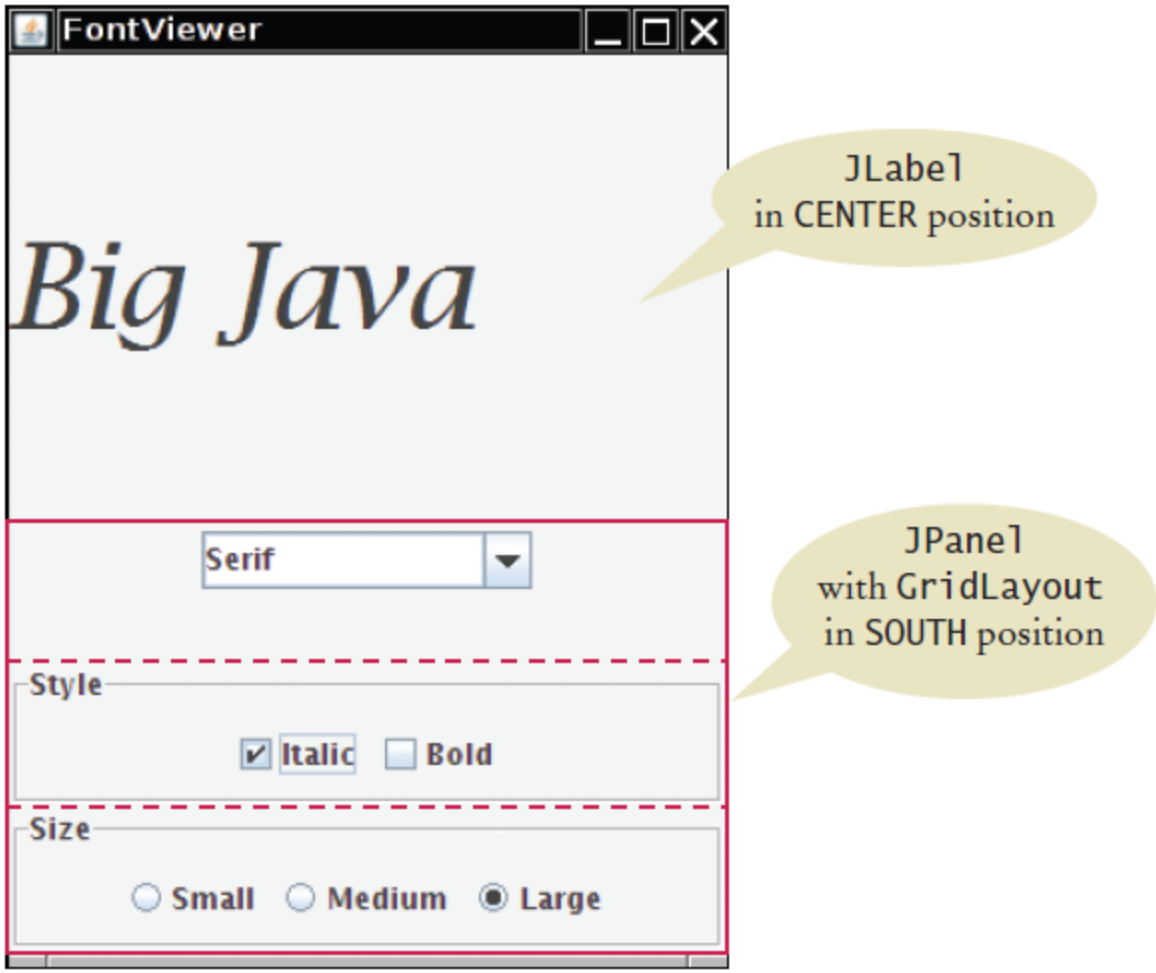
  - For a binary choice

Combo Boxes

  - For a large set of choices

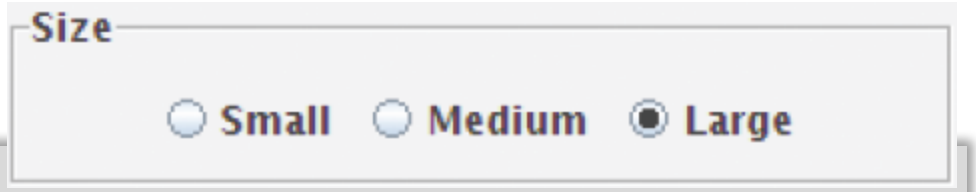
FontViewer Layout

- 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

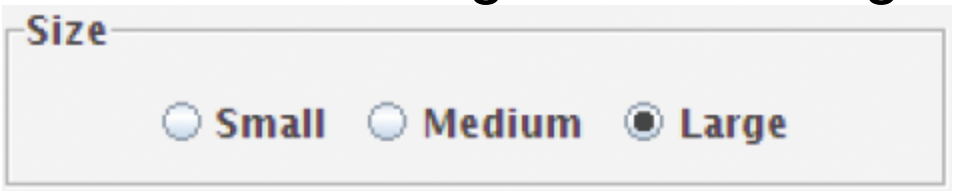
- Add Radio Buttons into a `ButtonGroup` so that only one button in the group is selected at a time
  - Create the `JRadioButtons` first, then add them to the `ButtonGroup`


```
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



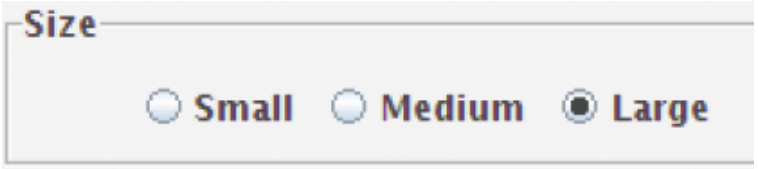

```
JRadioButton largeButton = new JRadioButton("Large");
largeButton.setSelected(true);
```

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

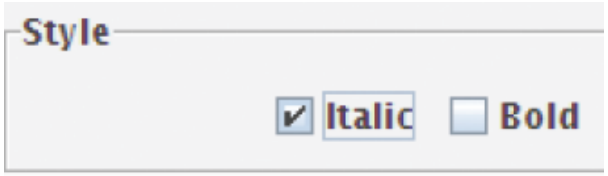
```
if (largeButton.isSelected())
   { size = LARGE_SIZE; }
```

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

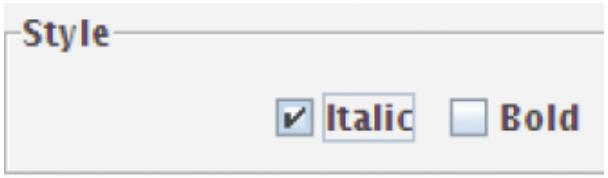


  - Check boxes are square and show a check mark when selected



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

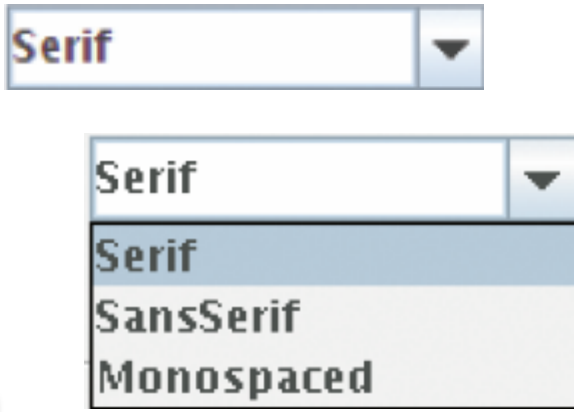
```
JCheckBox italicCheckBox = new JCheckBox("Italic");
```

- 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
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

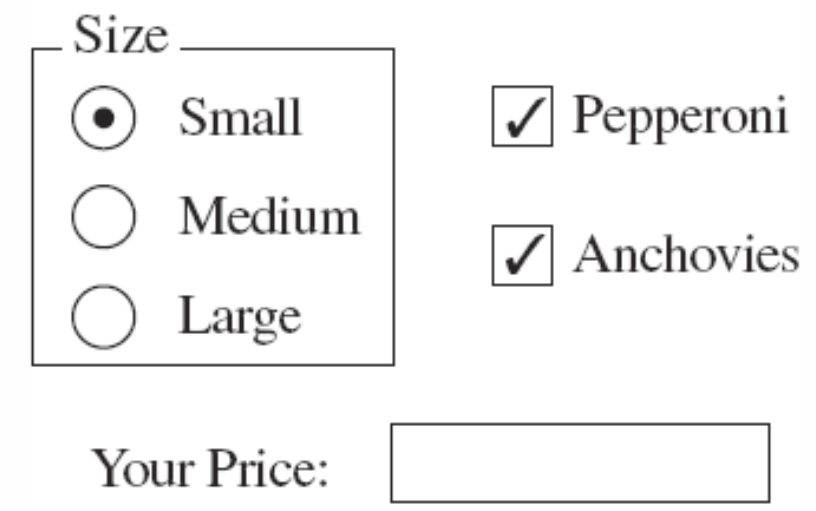
`isSelectedItem` for radio buttons

Calls `setFont` with face, style and size

Steps to Design a User Interface

1) Make a sketch of the component layout.

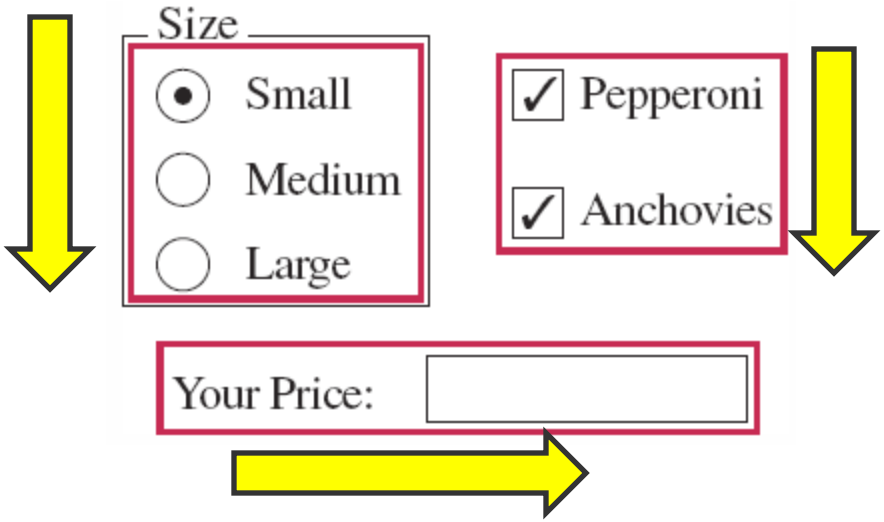
- Draw all the buttons, labels, text fields, and borders on a sheet of graph paper



Steps to Design a User Interface

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

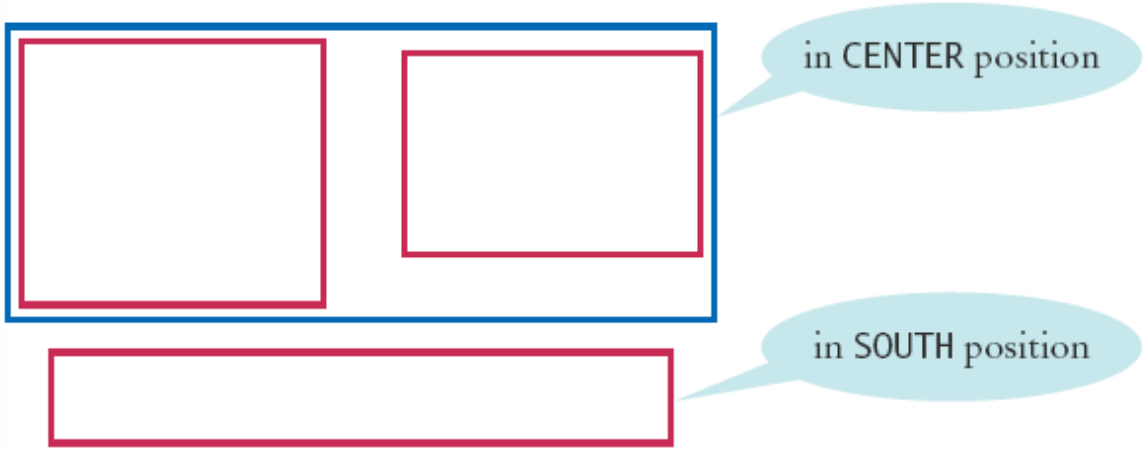
- Start by looking at adjacent components that are arranged top to bottom or left to right



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

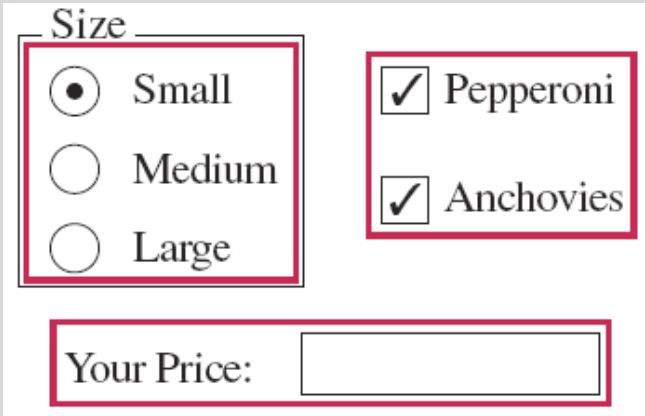
4) Group the groups together.
   - Look at each group as one blob, and group the blobs together into larger groups, just as you grouped the components in the preceding step



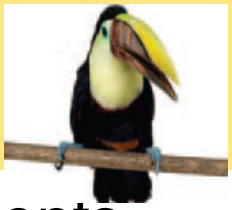
Steps to Design a User Interface

5) Write the code to generate the layout

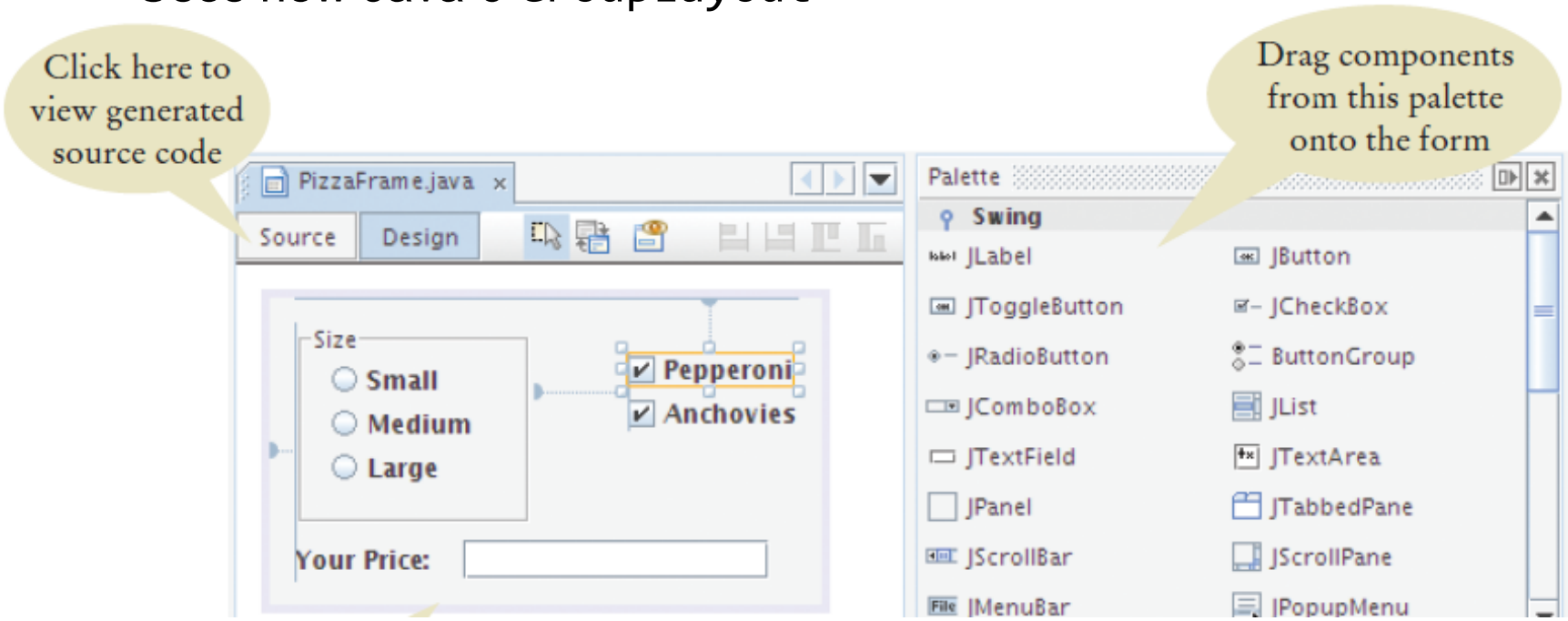
```
JPanel radioButtonPanel = new JPanel();
radioButtonPanel.setLayout(new GridLayout(3, 1));
radioButton.setBorder(new TitledBorder(new EtchedBorder(), "Size"));
radioButtonPanel.add(smallButton);
radioButtonPanel.add(mediumButton);
radioButtonPanel.add(largeButton);

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);
```



Use a GUI Builder

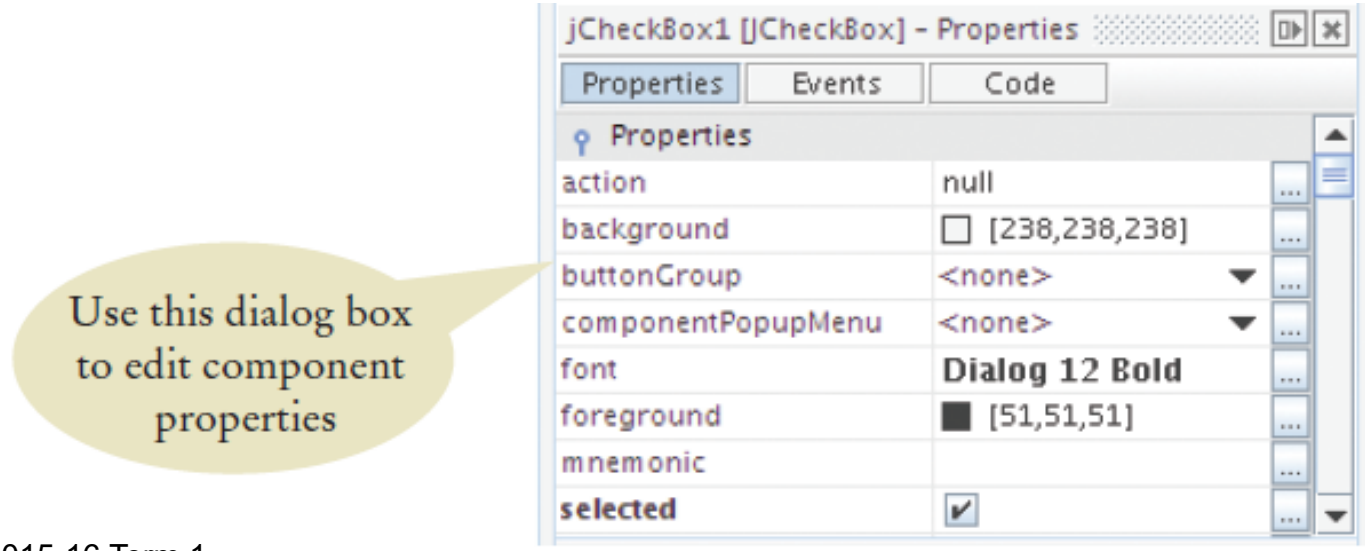


- A GUI Builder allows you to drag and drop components onto a panel and generates the code for you
- Try the free NetBeans development environment, available from http://netbeans.org
  - Uses new Java 6 `GroupLayout`



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



11.3 Menus

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

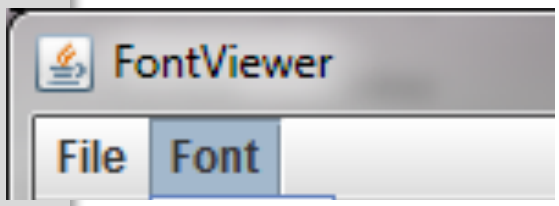

```
public class MyFrame extends JFrame
{
  public MyFrame()
  {
    JMenuBar menuBar = new JMenuBar();
    setJMenuBar(menuBar);
    . . .
  }
  . . .
}
```

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 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);
```



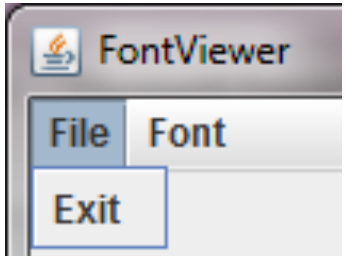
  - A Menu contains SubMenus and Menu items
    - A Menu item has no further SubMenus
    - You add Menu items and SubMenus with the add method

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

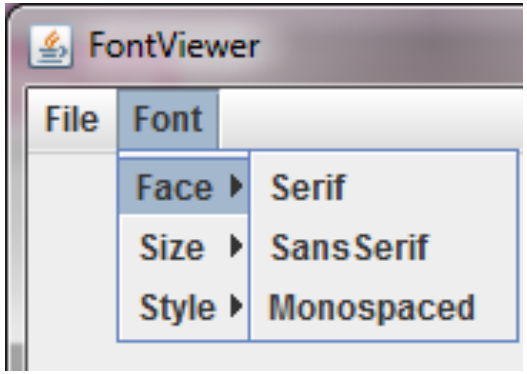
Menu Item Events

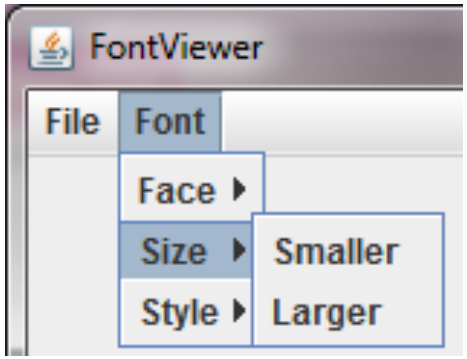
- Add action listeners to each Menu item

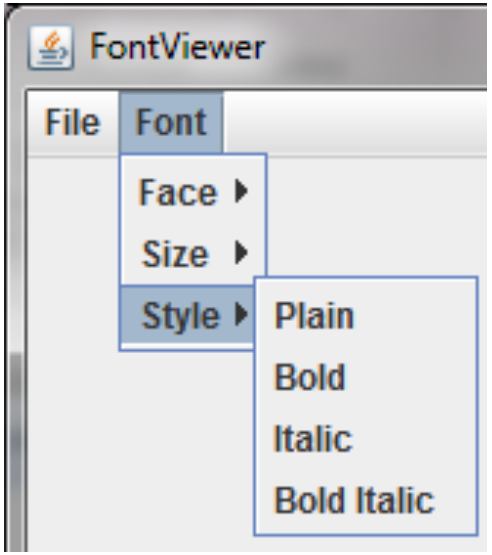



```
ActionListener listener = new ExitItemListener();
exitItem.addActionListener(listener);
```

- The listener is customized for each Menu item



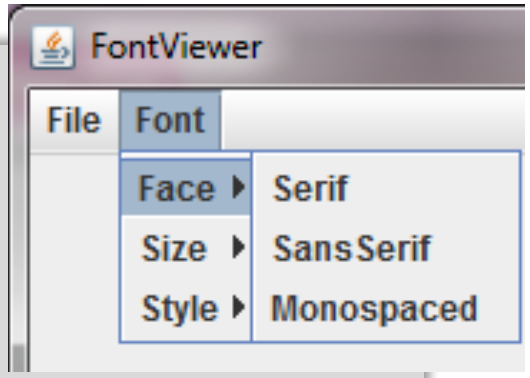




Example Menu Item Listener

- `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

```
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"));
```

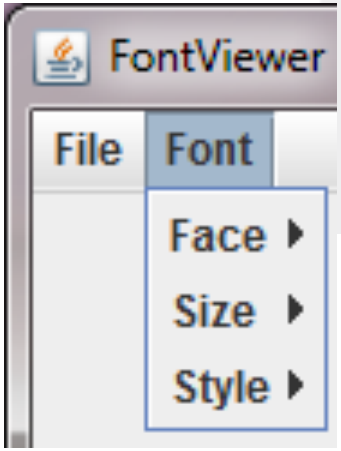
Font Face submenu

```
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;
```

Font Size submenu

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

Font Style submenu



FontViewer2.java – Listeners

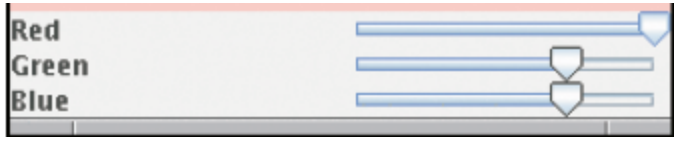
```
128    public JMenuItem createFaceItem(final String name)
129    {
130       class FaceItemListener implements ActionListener
131       {
132          public void actionPerformed(ActionEvent event)
133          {
134             facename = name;
135             setLabelFont();
136    152    public JMenuItem createSizeItem(String name, final int increment)
137    153    {
138    154       class SizeItemListener implements ActionListener
139    155       {
140    156          public void actionPerformed(ActionEvent event)
141    157          {
142    158             fontsize = fontsize + increment;
143    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

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

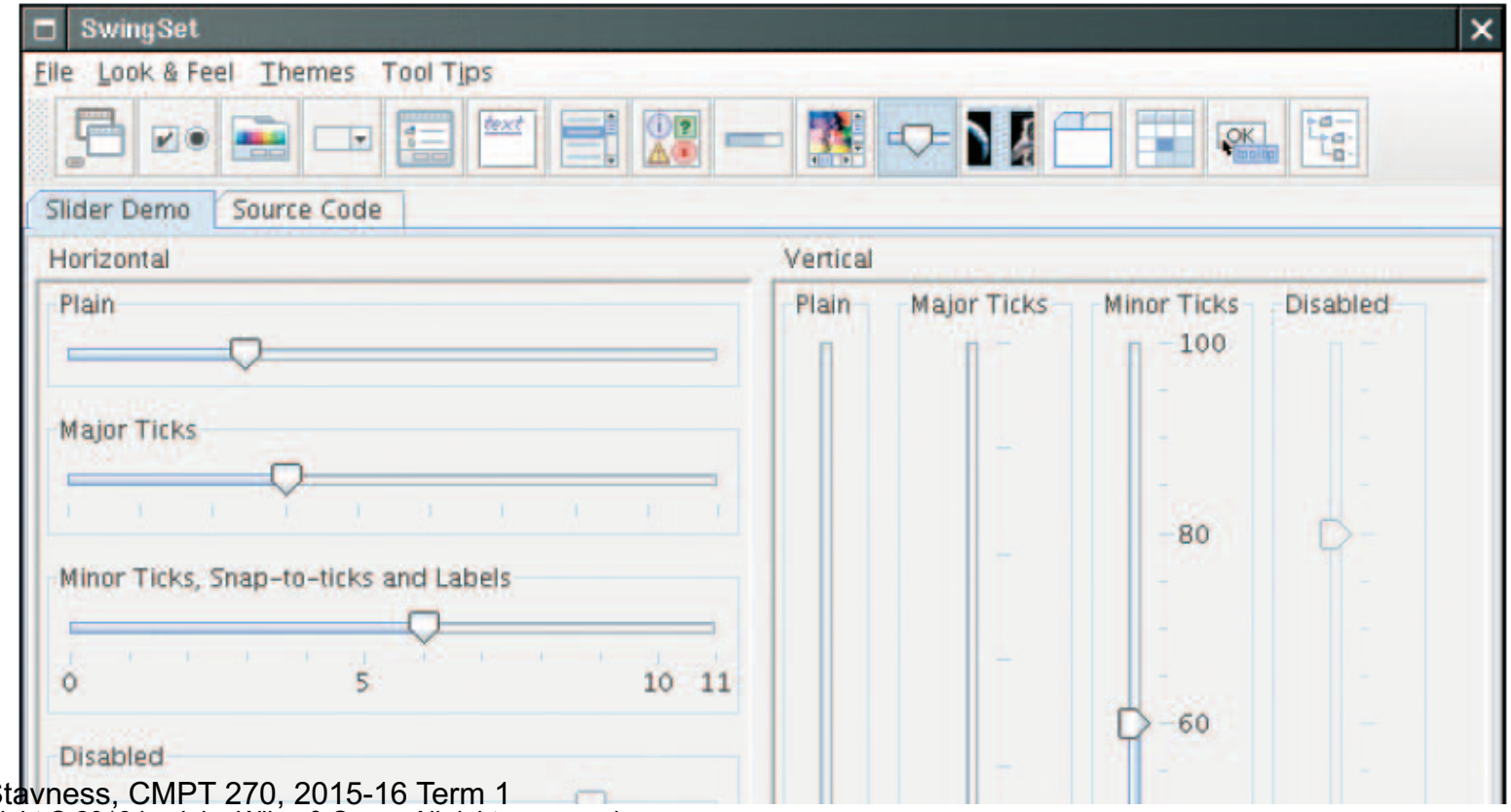


Swing Documentation

- 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

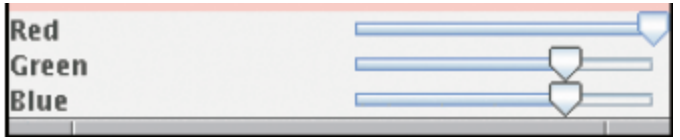
SwingSet Demo Examples

- 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?

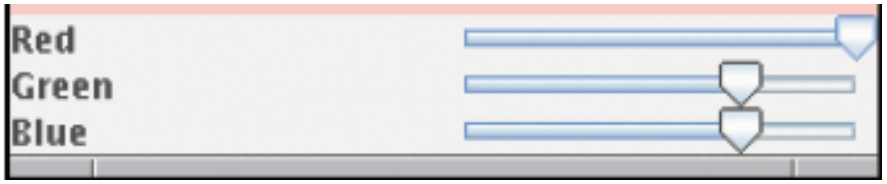


- When you look at the documentation of the `JSlider` class, you will probably not be happy.
  - There are over 50 methods in the `JSlider` class and over 250 inherited methods, and some of the method descriptions look downright scary

- 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.