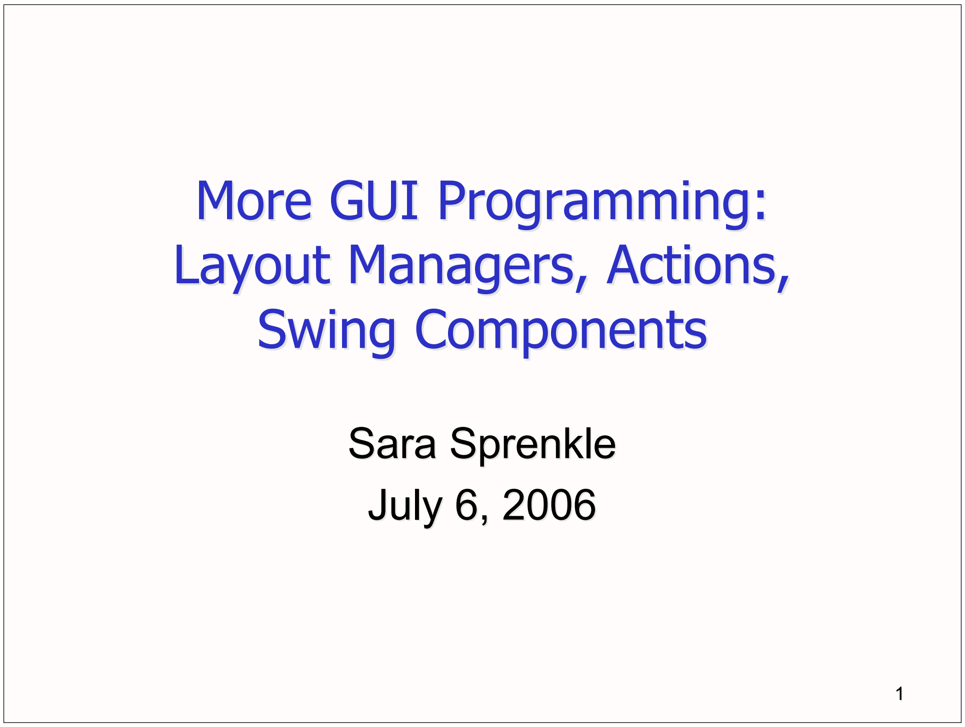
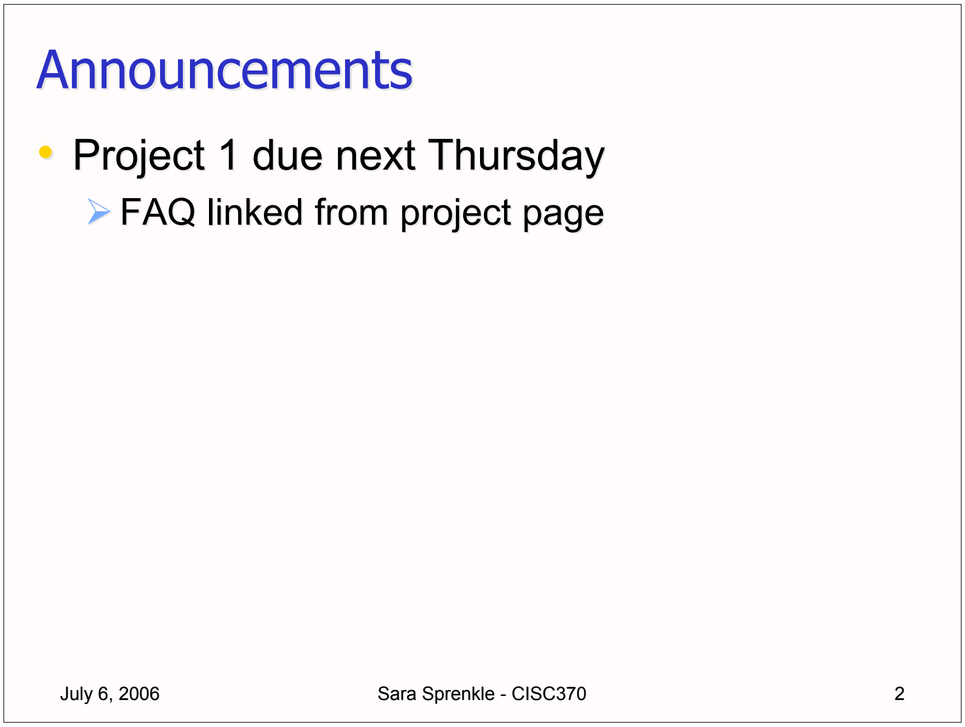


More GUI Programming: Layout Managers, Actions, Swing Components

Sara Sprenkle

July 6, 2006



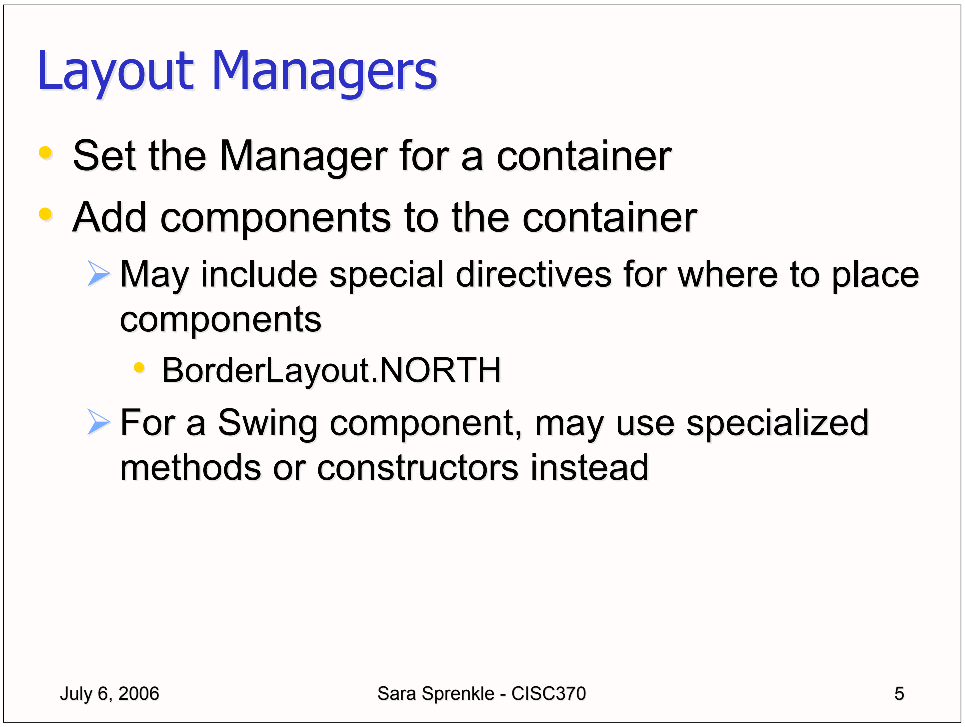
Announcements

- Project 1 due next Thursday
  - FAQ linked from project page

Review

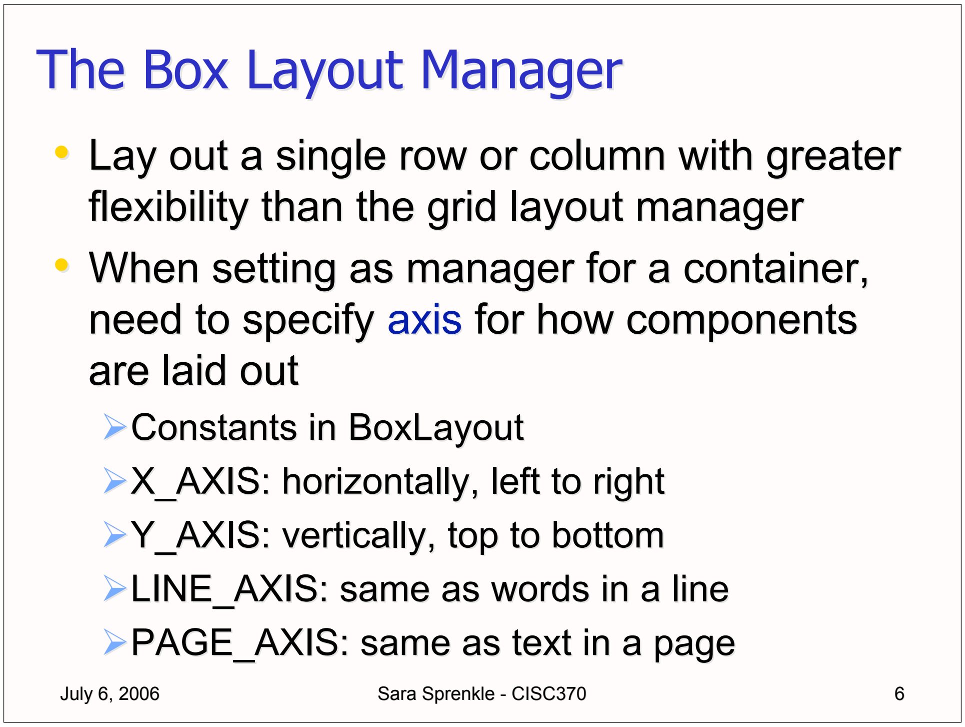
- Layout Managers
  - FlowLayout
  - BorderLayout
  - GridLayout
- Event Handling
  - Event Source
  - Listener
  - Adapter classes
- Model/Viewer/Controller
  - Design pattern

Layout Managers



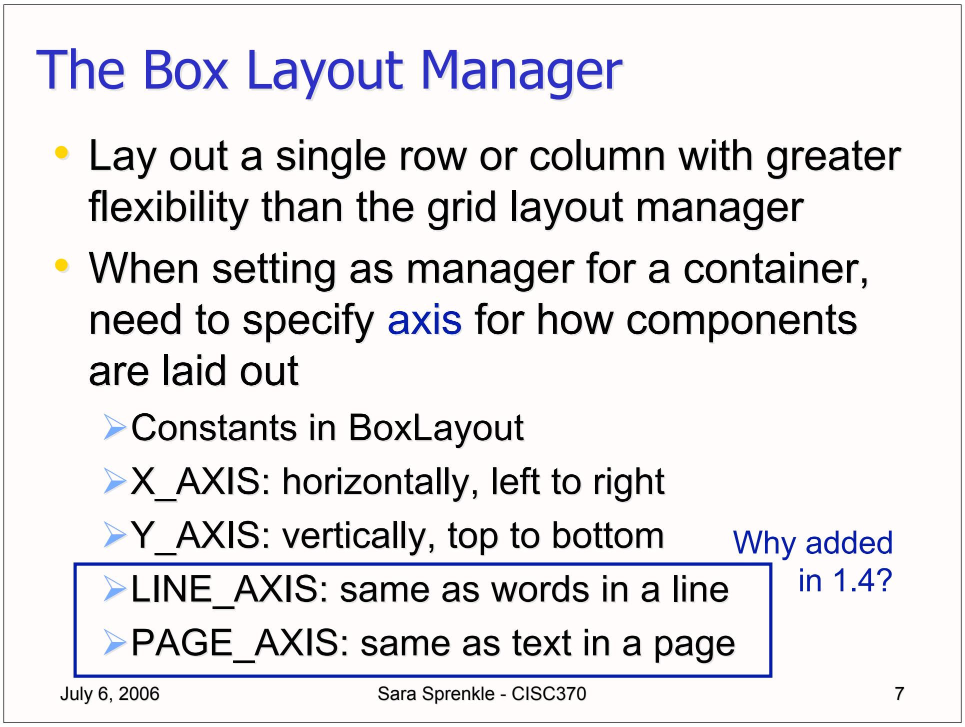
Layout Managers

- Set the Manager for a container
- Add components to the container
  - May include special directives for where to place components
    - BorderLayout.NORTH
  - For a Swing component, may use specialized methods or constructors instead



The Box Layout Manager

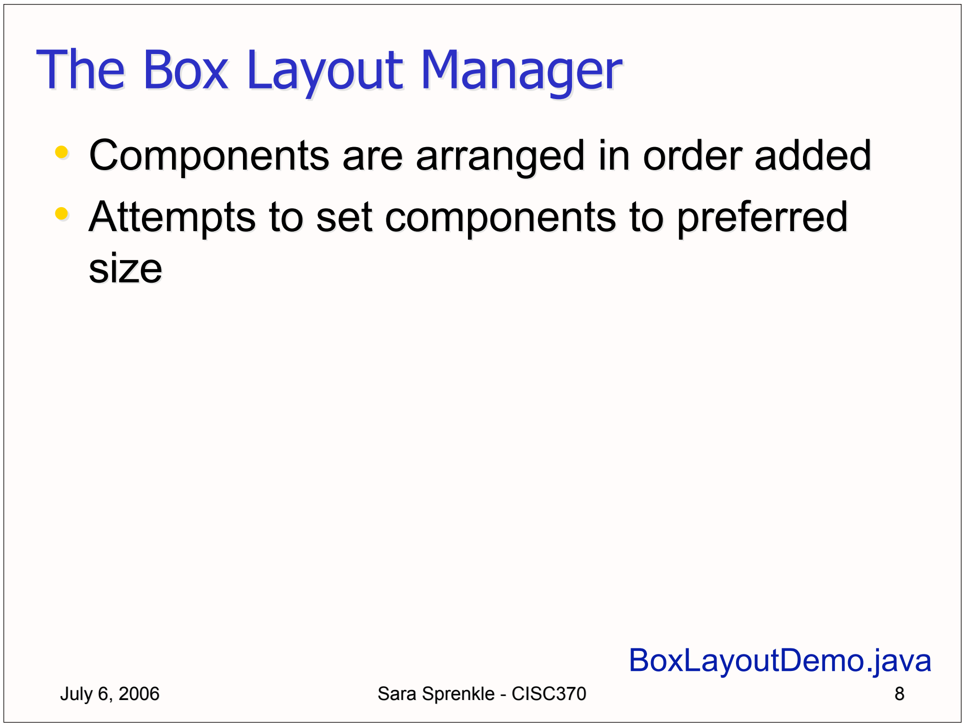
- Lay out a single row or column with greater flexibility than the grid layout manager
- When setting as manager for a container, need to specify axis for how components are laid out
  - Constants in BoxLayout
  - X_AXIS: horizontally, left to right
  - Y_AXIS: vertically, top to bottom
  - LINE_AXIS: same as words in a line
  - PAGE_AXIS: same as text in a page



The Box Layout Manager

- Lay out a single row or column with greater flexibility than the grid layout manager
- When setting as manager for a container, need to specify axis for how components are laid out
  - Constants in BoxLayout
  - X_AXIS: horizontally, left to right
  - Y_AXIS: vertically, top to bottom
  - LINE_AXIS: same as words in a line
  - PAGE_AXIS: same as text in a page

Why added in 1.4?



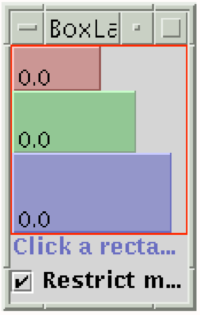
The Box Layout Manager

- Components are arranged in order added
- Attempts to set components to preferred size

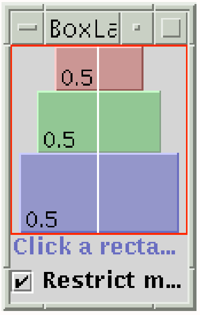
BoxLayoutDemo.java

AlignmentX

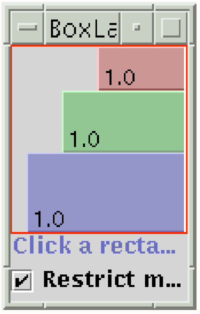
left



center



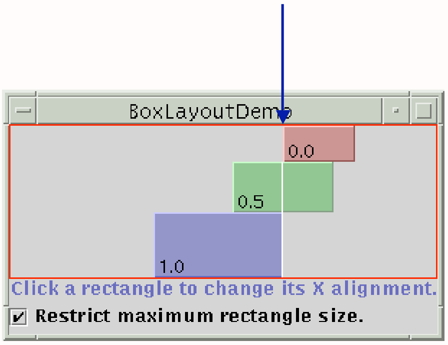
right



- Demonstrates alignment of boxes of different sizes

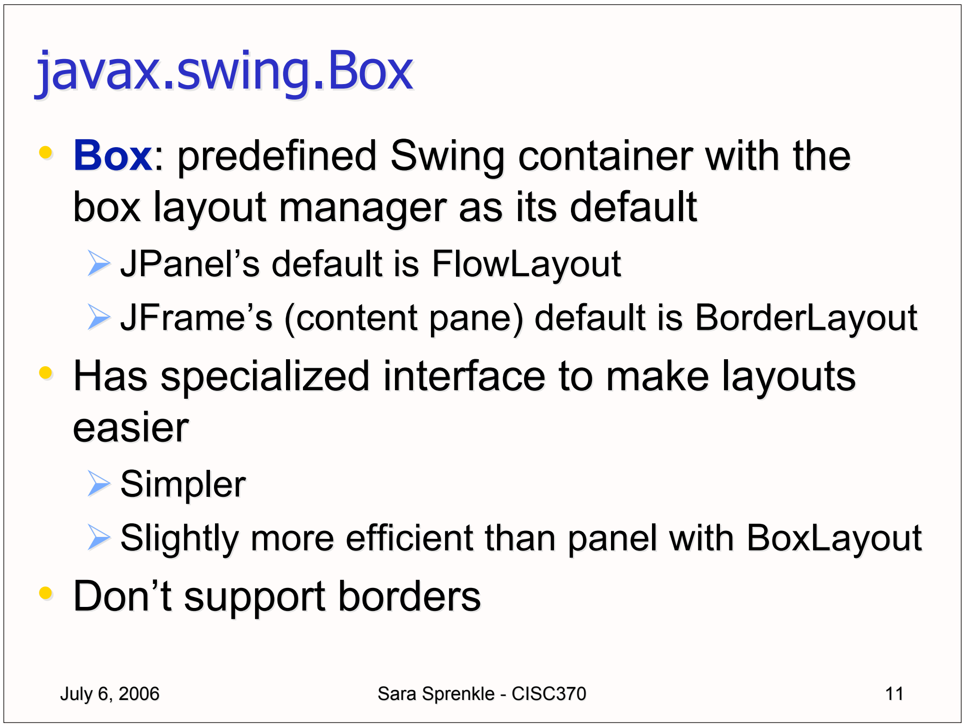
AlignmentX: Mixed Alignments

One alignment point



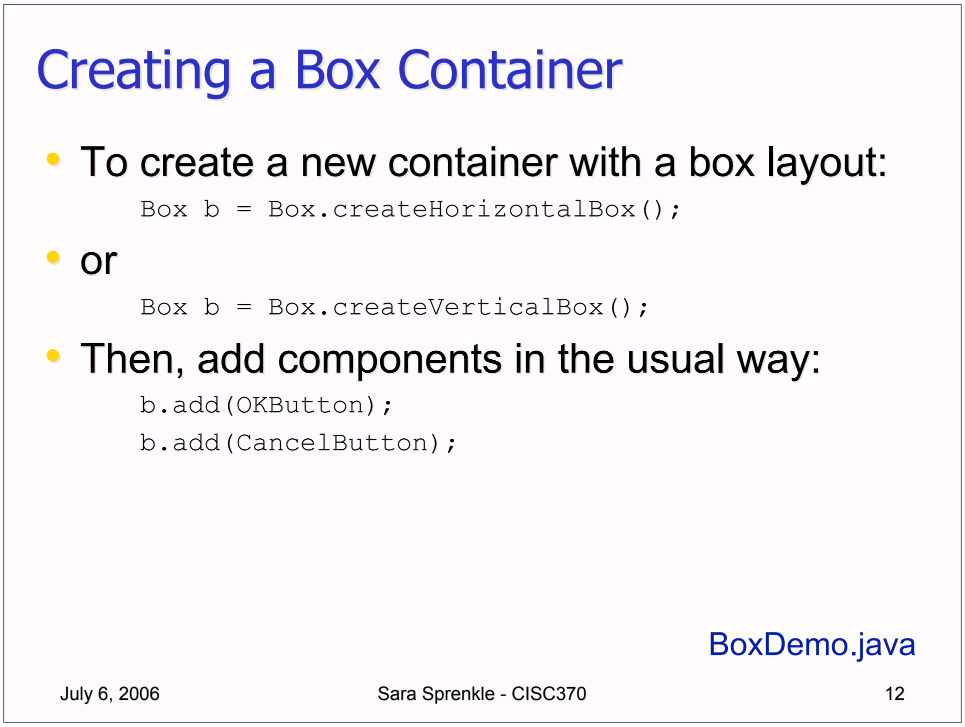
Click a rectangle to change its X alignment.

☑ Restrict maximum rectangle size.



javax.swing.Box

- **Box**: predefined Swing container with the box layout manager as its default
  - JPanel’s default is FlowLayout
  - JFrame’s (content pane) default is BorderLayout
- Has specialized interface to make layouts easier
  - Simpler
  - Slightly more efficient than panel with BoxLayout
- Don’t support borders



Creating a Box Container

- To create a new container with a box layout:

```
Box b = Box.createHorizontalBox();
```

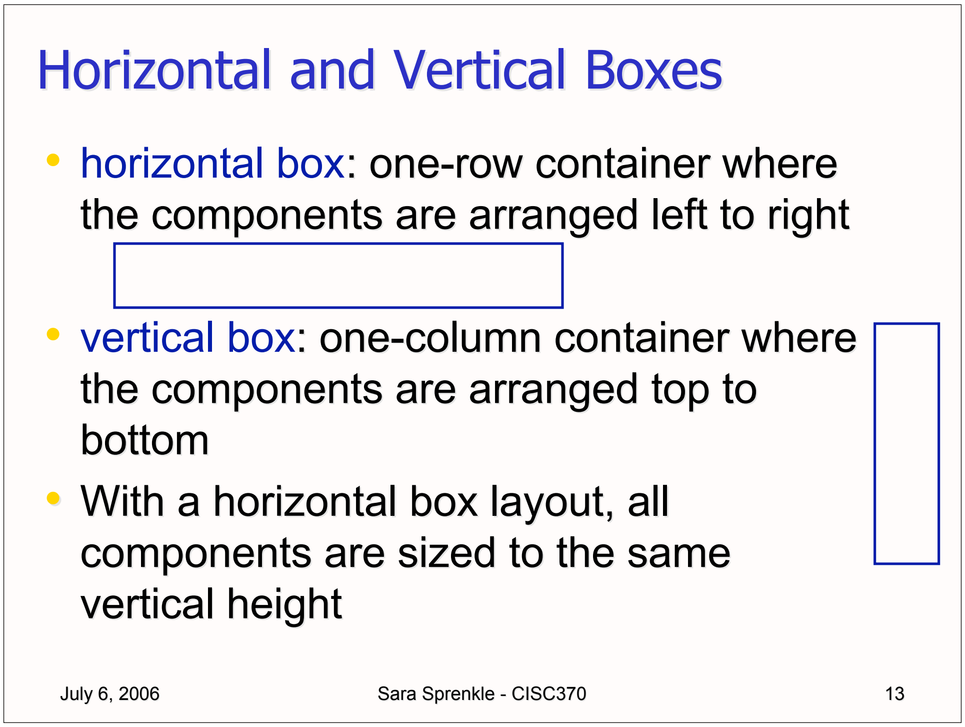
- or

```
Box b = Box.createVerticalBox();
```

- Then, add components in the usual way:

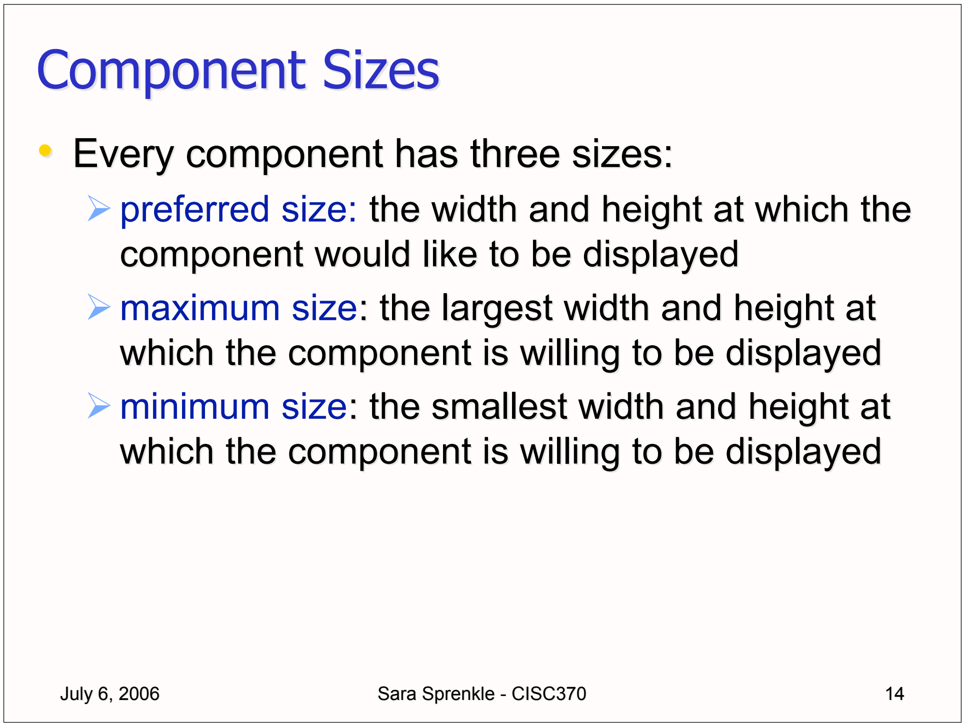
```
b.add(OKButton);
b.add(CancelButton);
```

BoxDemo.java



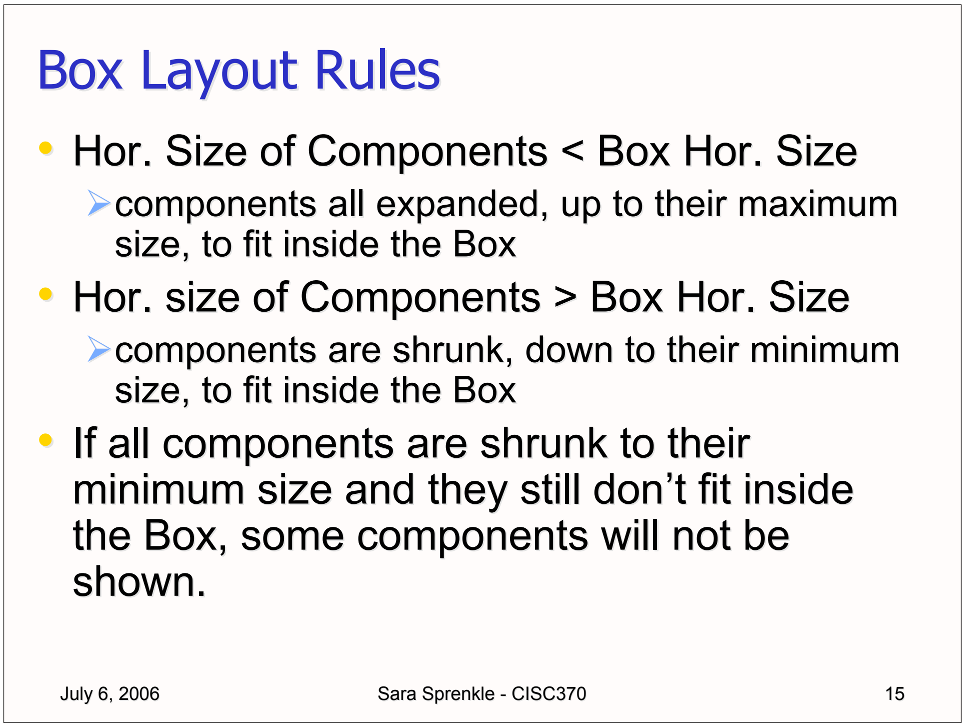
Horizontal and Vertical Boxes

- horizontal box: one-row container where the components are arranged left to right
- vertical box: one-column container where the components are arranged top to bottom
- With a horizontal box layout, all components are sized to the same vertical height



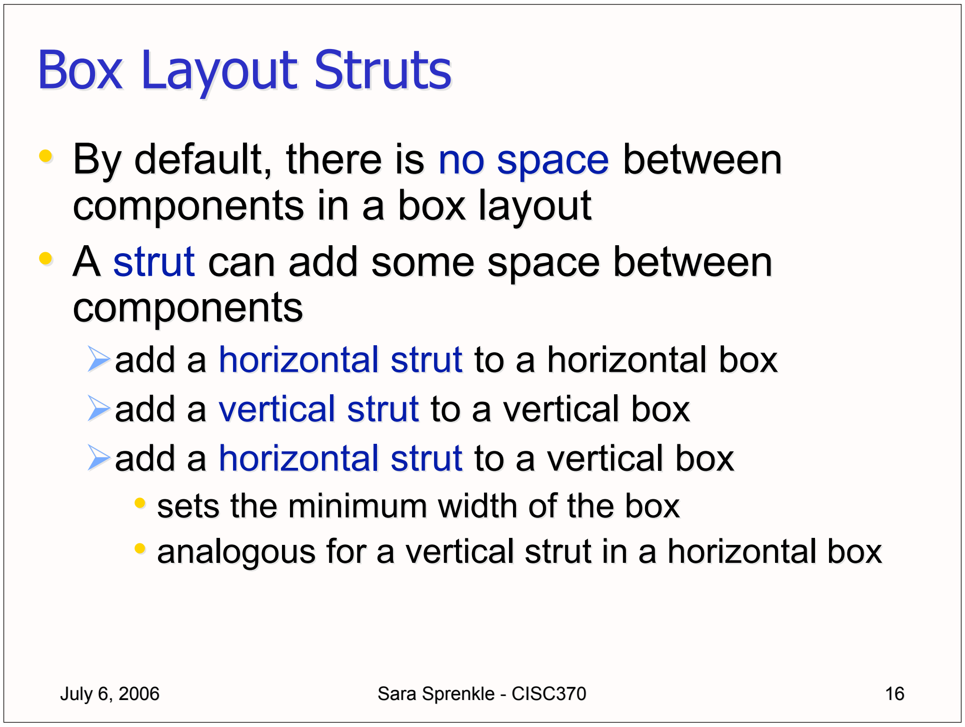
Component Sizes

- Every component has three sizes:
  - preferred size: the width and height at which the component would like to be displayed
  - maximum size: the largest width and height at which the component is willing to be displayed
  - minimum size: the smallest width and height at which the component is willing to be displayed



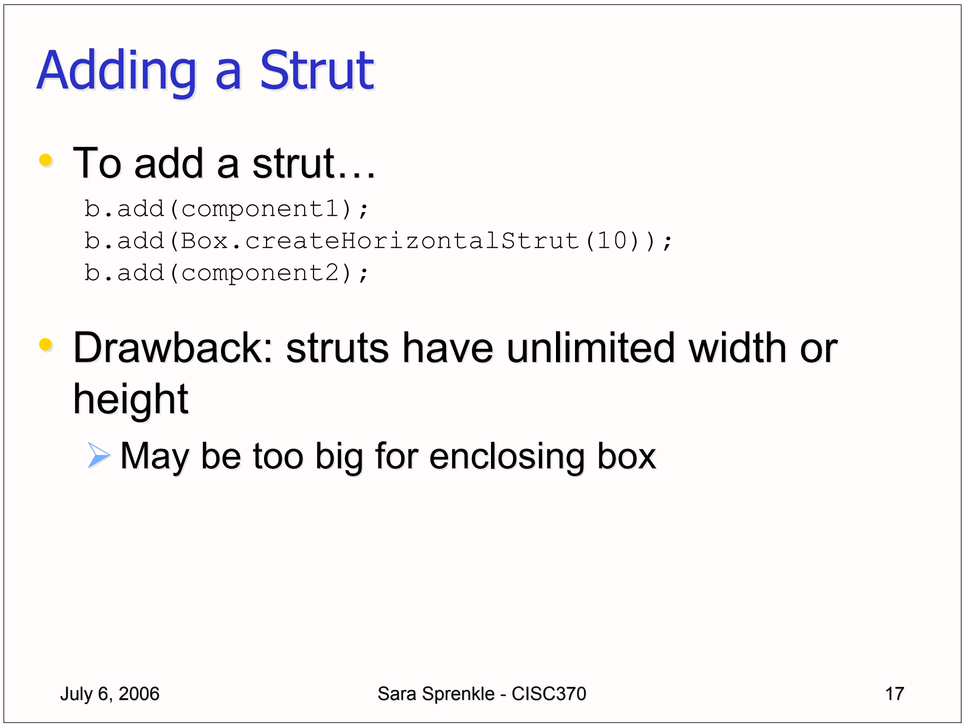
Box Layout Rules

- Hor. Size of Components < Box Hor. Size
  - components all expanded, up to their maximum size, to fit inside the Box
- Hor. size of Components > Box Hor. Size
  - components are shrunk, down to their minimum size, to fit inside the Box
- If all components are shrunk to their minimum size and they still don’t fit inside the Box, some components will not be shown.



Box Layout Struts

- By default, there is no space between components in a box layout
- A strut can add some space between components
  - add a horizontal strut to a horizontal box
  - add a vertical strut to a vertical box
  - add a horizontal strut to a vertical box
    - sets the minimum width of the box
    - analogous for a vertical strut in a horizontal box

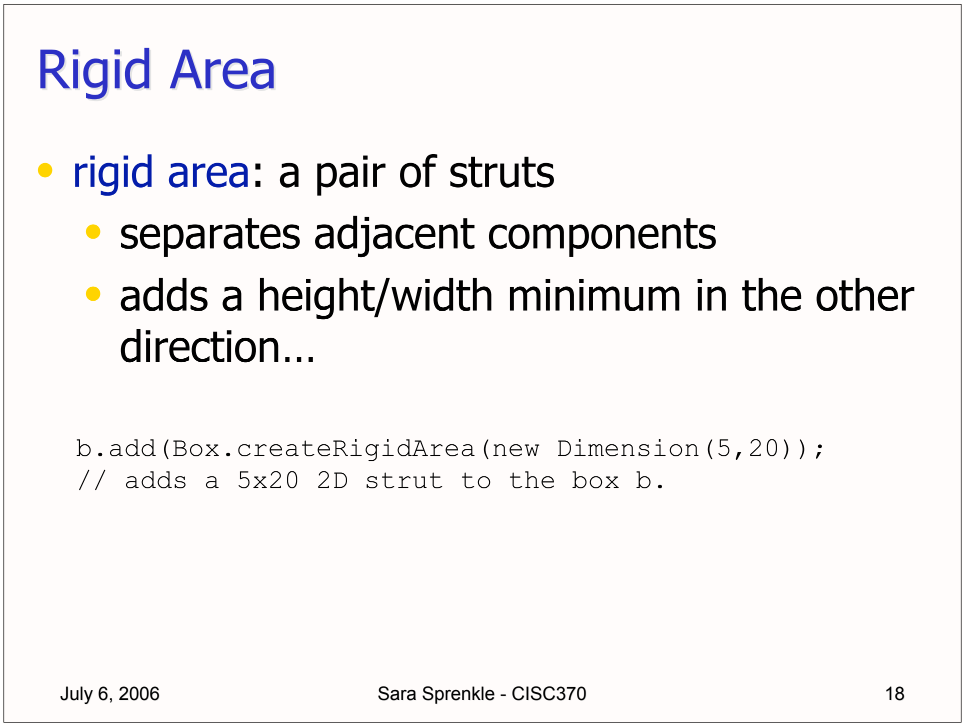


Adding a Strut

- To add a strut…

```
b.add(component1);
b.add(Box.createHorizontalStrut(10));
b.add(component2);
```

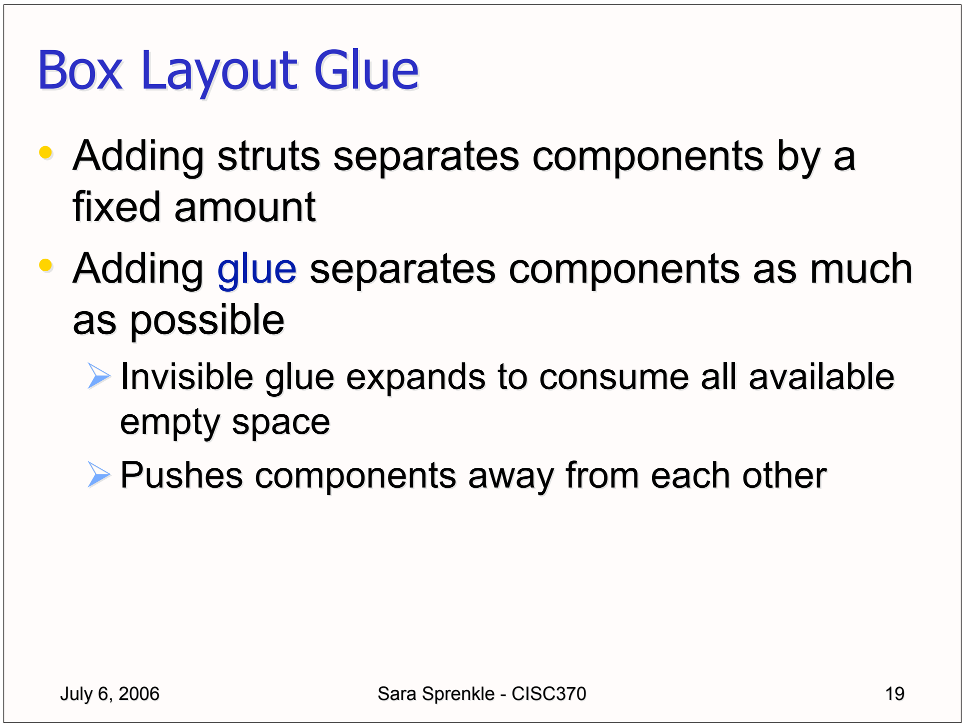
- Drawback: struts have unlimited width or height
    - May be too big for enclosing box



Rigid Area

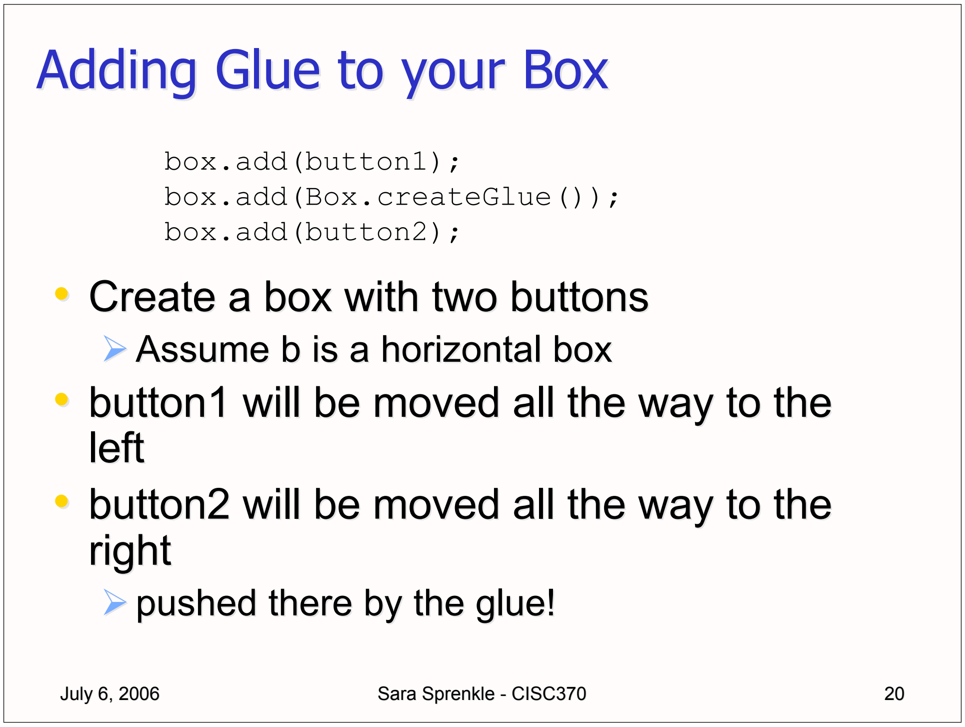
- rigid area: a pair of struts
    - separates adjacent components
    - adds a height/width minimum in the other direction…

```
b.add(Box.createRigidArea(new Dimension(5,20));
// adds a 5x20 2D strut to the box b.
```



Box Layout Glue

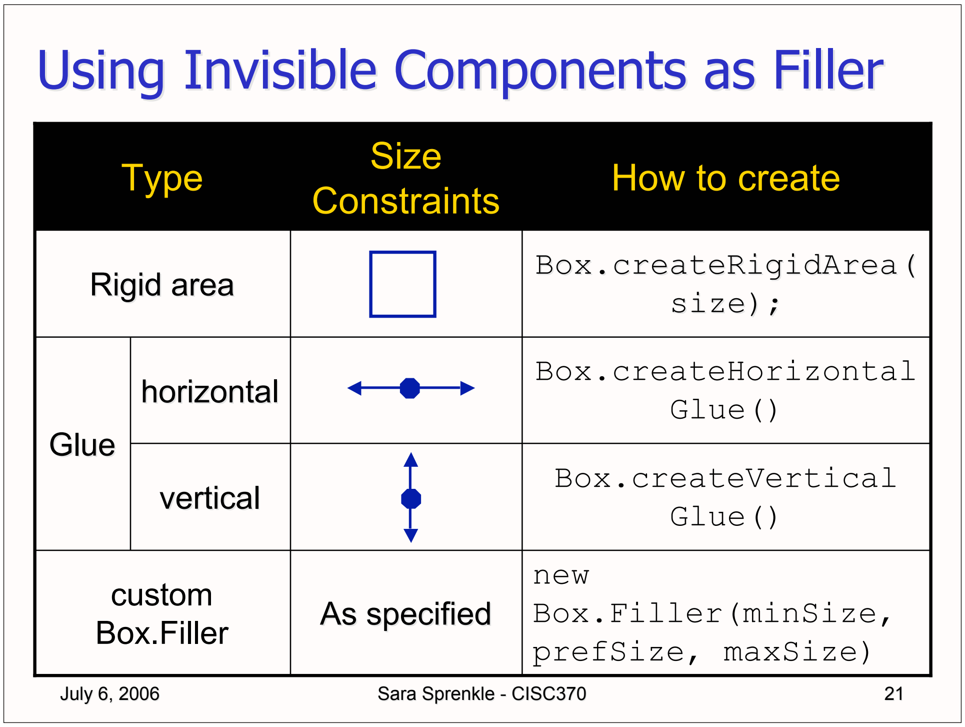
- Adding struts separates components by a fixed amount
- Adding glue separates components as much as possible
  - Invisible glue expands to consume all available empty space
  - Pushes components away from each other



Adding Glue to your Box

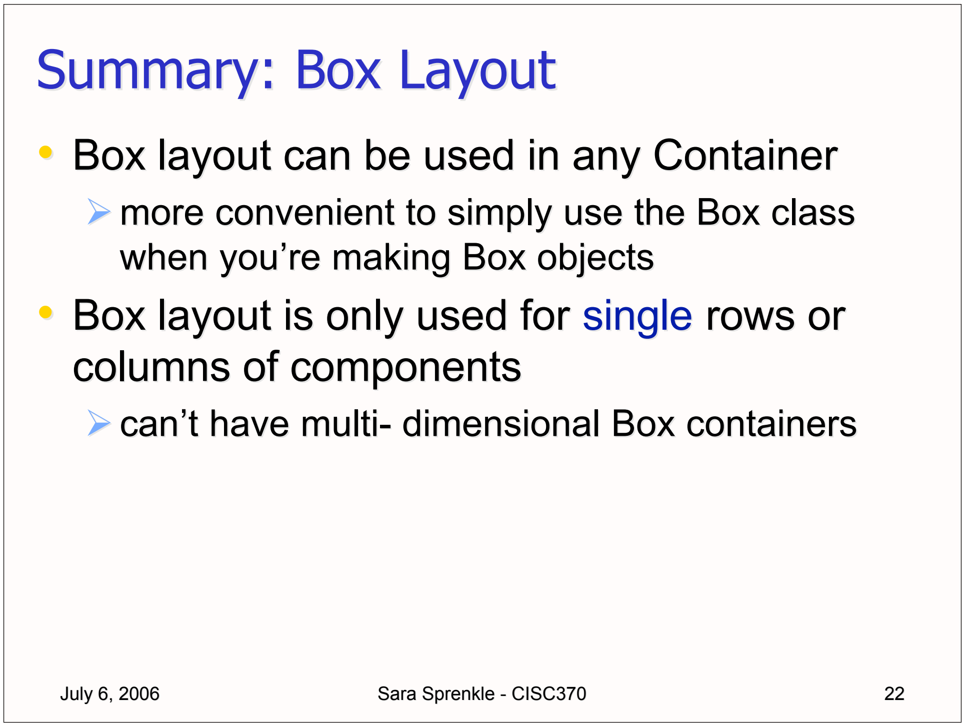
```
box.add(button1);
box.add(Box.createGlue());
box.add(button2);
```

- Create a box with two buttons
  - Assume b is a horizontal box
- button1 will be moved all the way to the left
- button2 will be moved all the way to the right
  - pushed there by the glue!



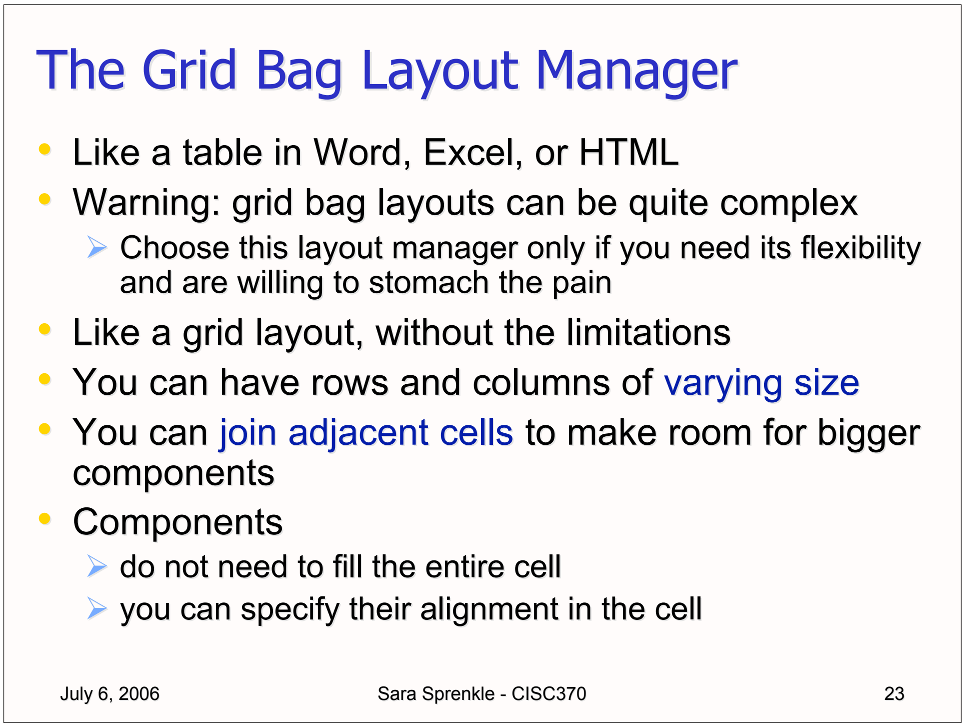
Using Invisible Components as Filler

Type		Size Constraints	How to create
Rigid area		□	`Box.createRigidArea(size);`
Glue	horizontal	↔	`Box.createHorizontalGlue()`
	vertical	↕	`Box.createVerticalGlue()`
custom Box.Filler		As specified	`new Box.Filler(minSize, prefSize, maxSize)`



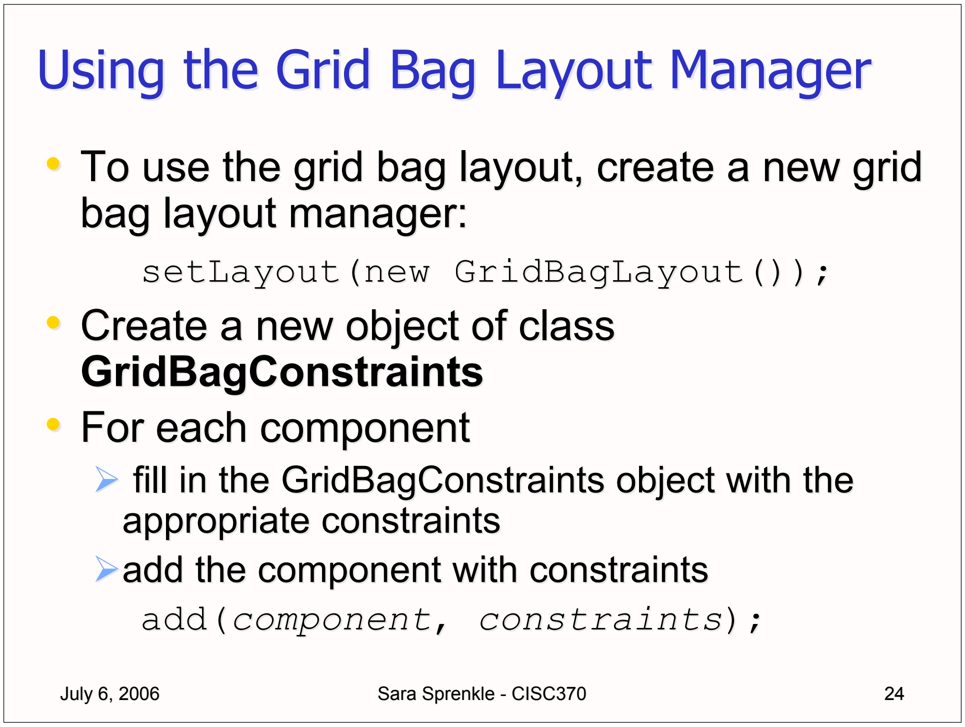
Summary: Box Layout

- Box layout can be used in any Container
  - more convenient to simply use the Box class when you're making Box objects
- Box layout is only used for single rows or columns of components
  - can't have multi- dimensional Box containers



The Grid Bag Layout Manager

- Like a table in Word, Excel, or HTML
- Warning: grid bag layouts can be quite complex
  - Choose this layout manager only if you need its flexibility and are willing to stomach the pain
- Like a grid layout, without the limitations
- You can have rows and columns of varying size
- You can join adjacent cells to make room for bigger components
- Components
  - do not need to fill the entire cell
  - you can specify their alignment in the cell



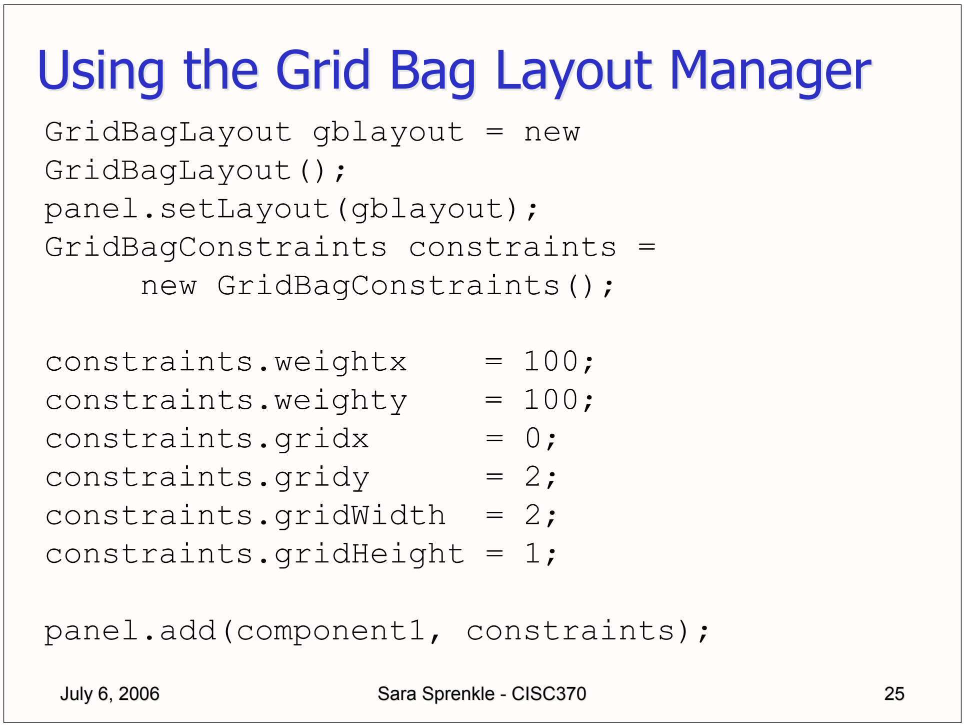
Using the Grid Bag Layout Manager

- To use the grid bag layout, create a new grid bag layout manager:

```
setLayout(new GridBagLayout());
```

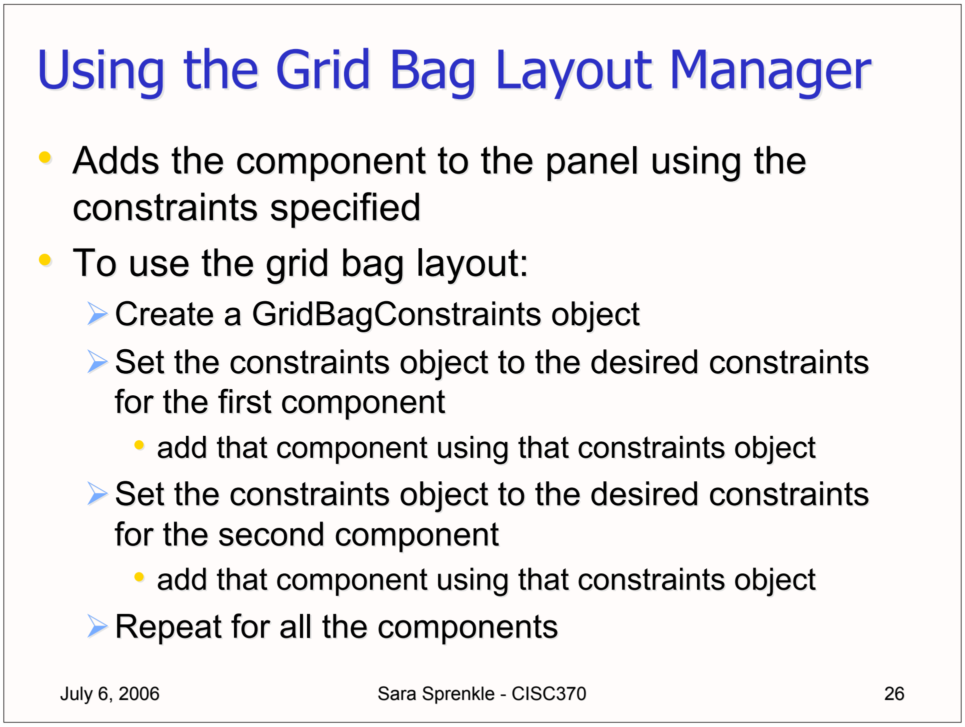
- Create a new object of class **GridBagConstraints**
- For each component
  - fill in the GridBagConstraints object with the appropriate constraints
  - add the component with constraints

```
add(component, constraints);
```



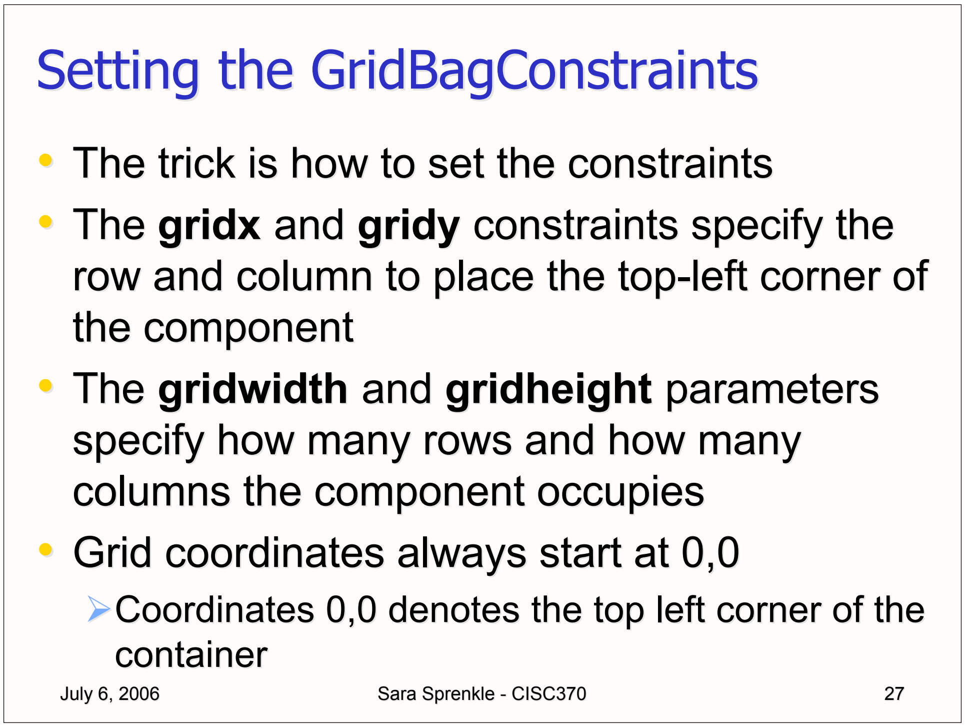
Using the Grid Bag Layout Manager

```
GridBagLayout gblayout = new
GridBagLayout();
panel.setLayout(gblayout);
GridBagConstraints constraints =
     new GridBagConstraints();

constraints.weightx    = 100;
constraints.weighty    = 100;
constraints.gridx      = 0;
constraints.gridy      = 2;
constraints.gridWidth  = 2;
constraints.gridHeight = 1;

panel.add(component1, constraints);
```



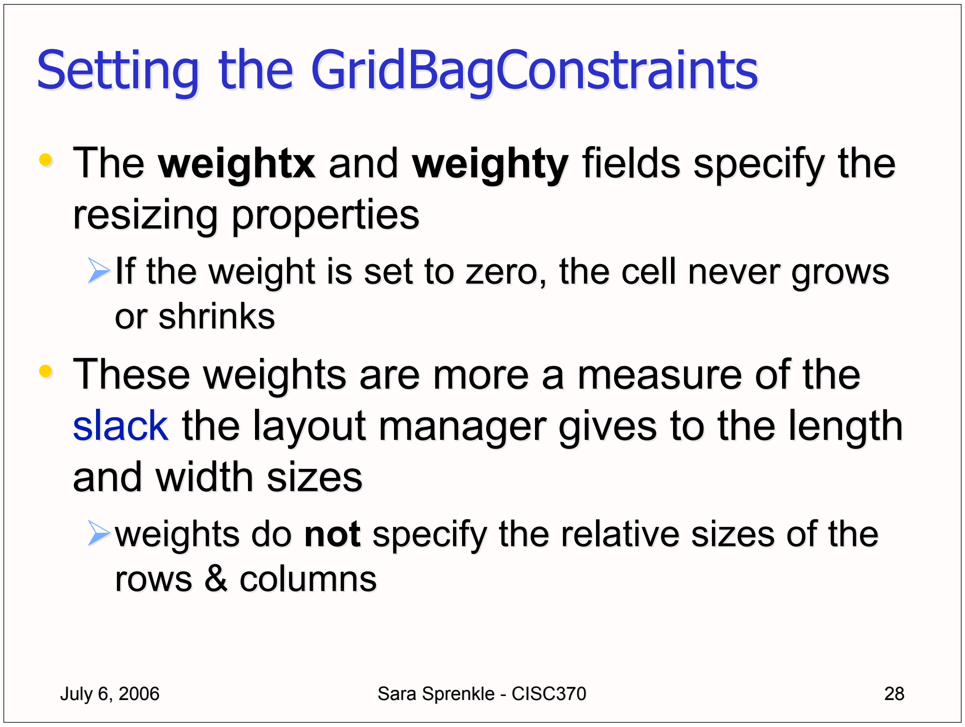
Using the Grid Bag Layout Manager

- Adds the component to the panel using the constraints specified
- To use the grid bag layout:
  - Create a GridBagConstraints object
  - Set the constraints object to the desired constraints for the first component
    - add that component using that constraints object
  - Set the constraints object to the desired constraints for the second component
    - add that component using that constraints object
  - Repeat for all the components



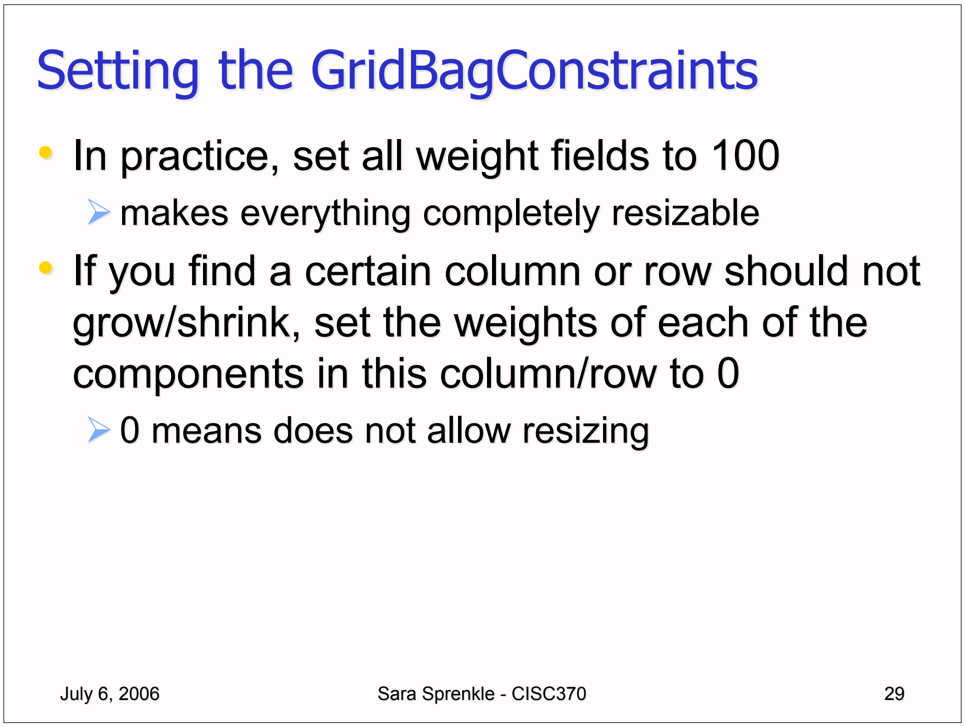
Setting the GridBagConstraints

- The trick is how to set the constraints
- The **gridx** and **gridy** constraints specify the row and column to place the top-left corner of the component
- The **gridwidth** and **gridheight** parameters specify how many rows and how many columns the component occupies
- Grid coordinates always start at 0,0
  - Coordinates 0,0 denotes the top left corner of the container



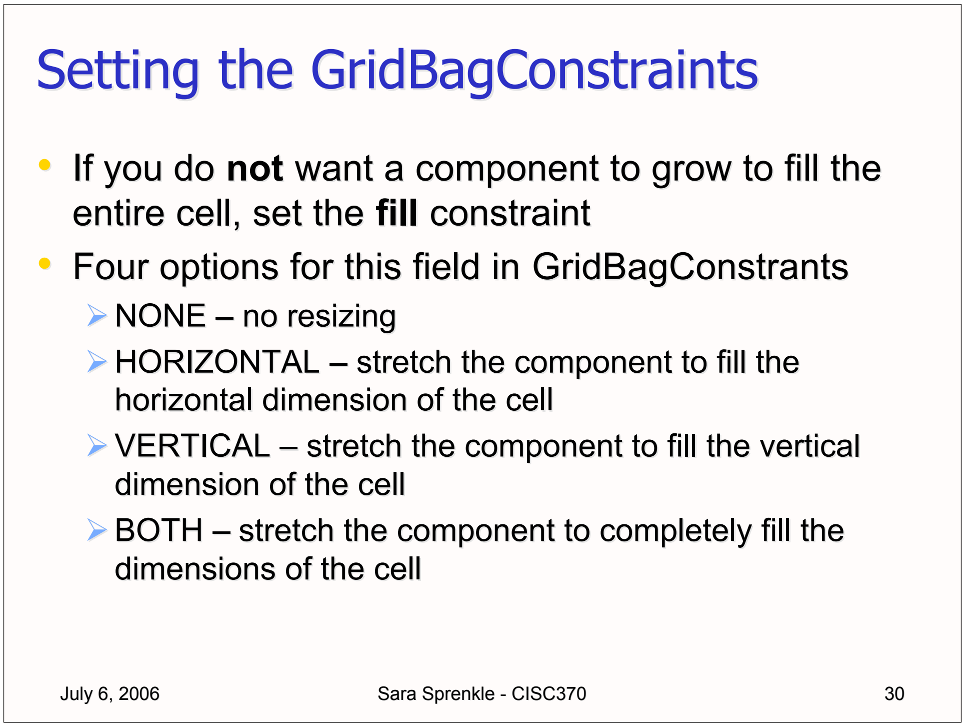
Setting the GridBagConstraints

- The **weightx** and **weighty** fields specify the resizing properties
  - If the weight is set to zero, the cell never grows or shrinks
- These weights are more a measure of the slack the layout manager gives to the length and width sizes
  - weights do **not** specify the relative sizes of the rows & columns



Setting the GridBagConstraints

- In practice, set all weight fields to 100
  - makes everything completely resizable
- If you find a certain column or row should not grow/shrink, set the weights of each of the components in this column/row to 0
  - 0 means does not allow resizing



Setting the GridBagConstraints

- If you do **not** want a component to grow to fill the entire cell, set the **fill** constraint
- Four options for this field in GridBagConstrants
  - NONE – no resizing
  - HORIZONTAL – stretch the component to fill the horizontal dimension of the cell
  - VERTICAL – stretch the component to fill the vertical dimension of the cell
  - BOTH – stretch the component to completely fill the dimensions of the cell

Setting the GridBagConstraints

- If the component does not completely fill the cell, set the **anchor** constraint:
  - GridBagConstraints.CENTER – center the component in the cell (default value)
  - GridBagConstraints.NORTH – center the component in the cell horizontally and place it at the top in the vertical direction
  - GridBagConstraints.NORTHEAST – top in the vertical direction and all the way to the left
  - and so forth…

How to Make a Good Grid Bag Layout Design

- Sketch out the panel
- Find a grid such that small components are each contained in a cell and larger components span multiple cells
- Label the rows and columns 0,1,2,…
  - gives you the gridx, gridy, gridwidth and gridheight values for each component
- For each component, does it need to be aligned in the cell?
  - If so, set fill and anchor

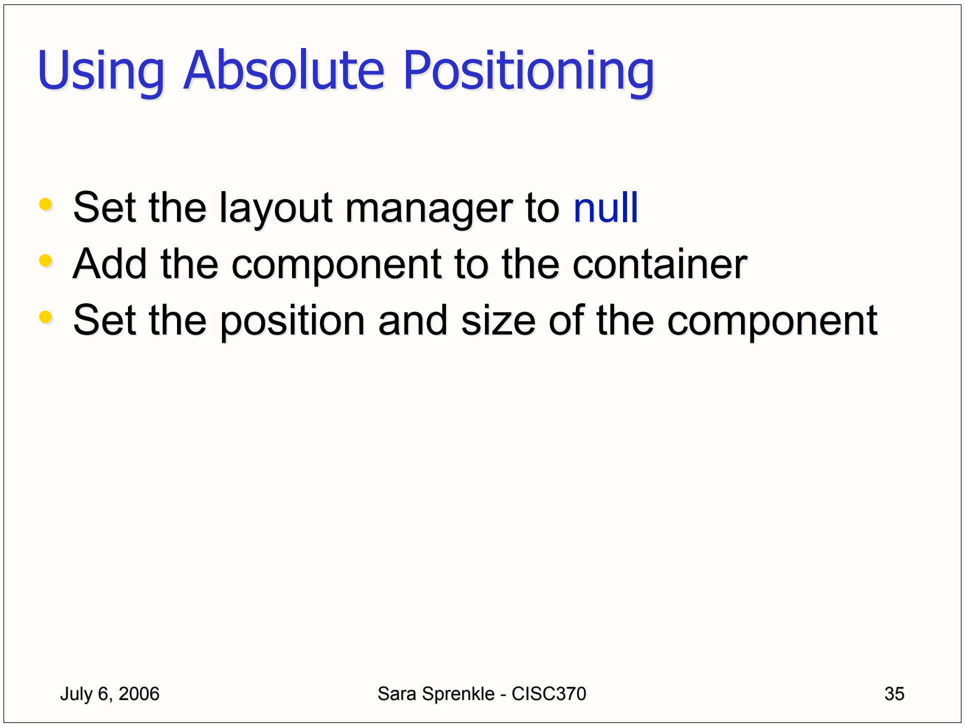
How to Make a Good Grid Bag Layout Design

- Set all weights to 100
  - If you do not want a row or column to be resized, set the weightx/weighty to 0 in *all* components in the row/column
- Write the code, triple-checking all the constraints
- Compile and run
- Find the errors in your constraints, fix them, and repeat

GridBagLayoutDemo.java

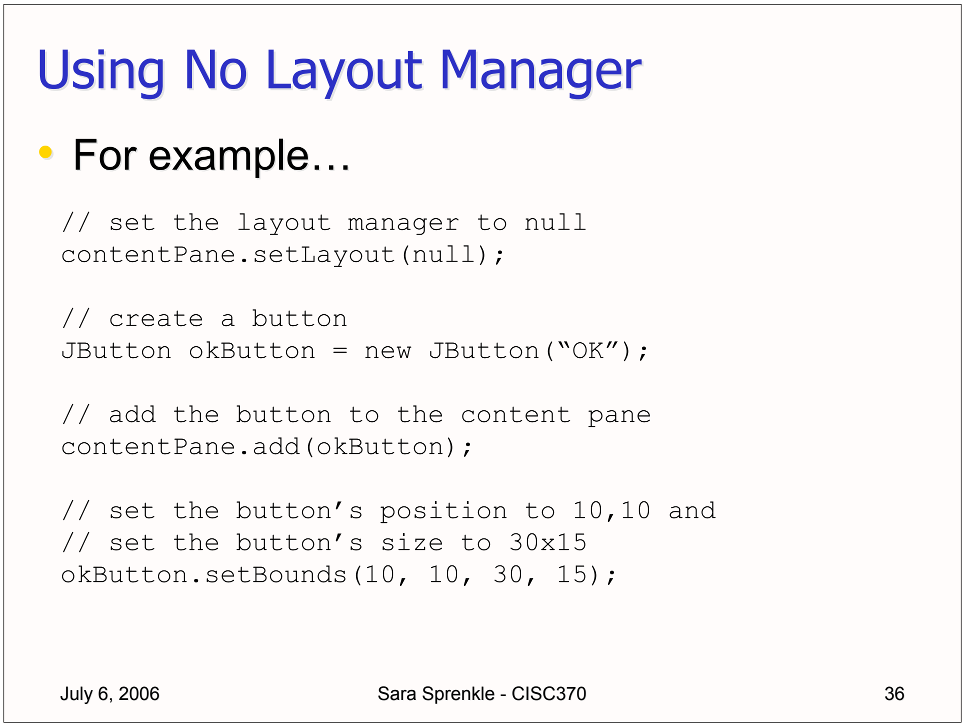
Absolute Positioning

- It is possible to not use a layout manager
  - Called Absolute Positioning
- If you do this, you need to provide a fixed (or *absolute*) position and size for each component
- Doesn’t adjust well when top-level container is resized
- Doesn’t adapt well to different users and systems
  - Fonts, locales, etc.



Using Absolute Positioning

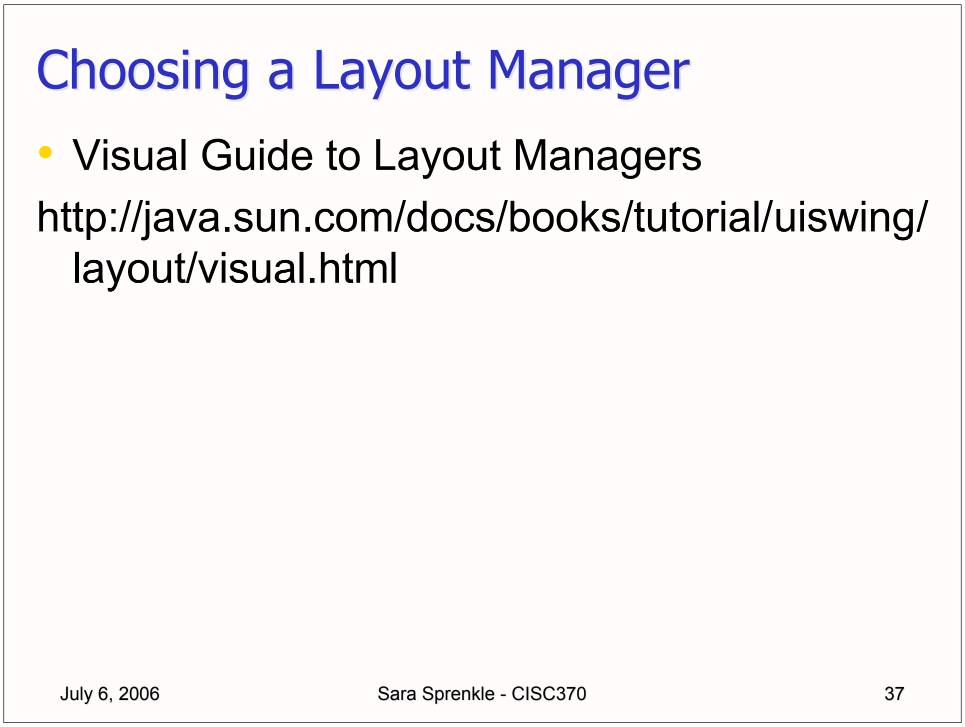
- Set the layout manager to null
- Add the component to the container
- Set the position and size of the component



Using No Layout Manager

- For example…

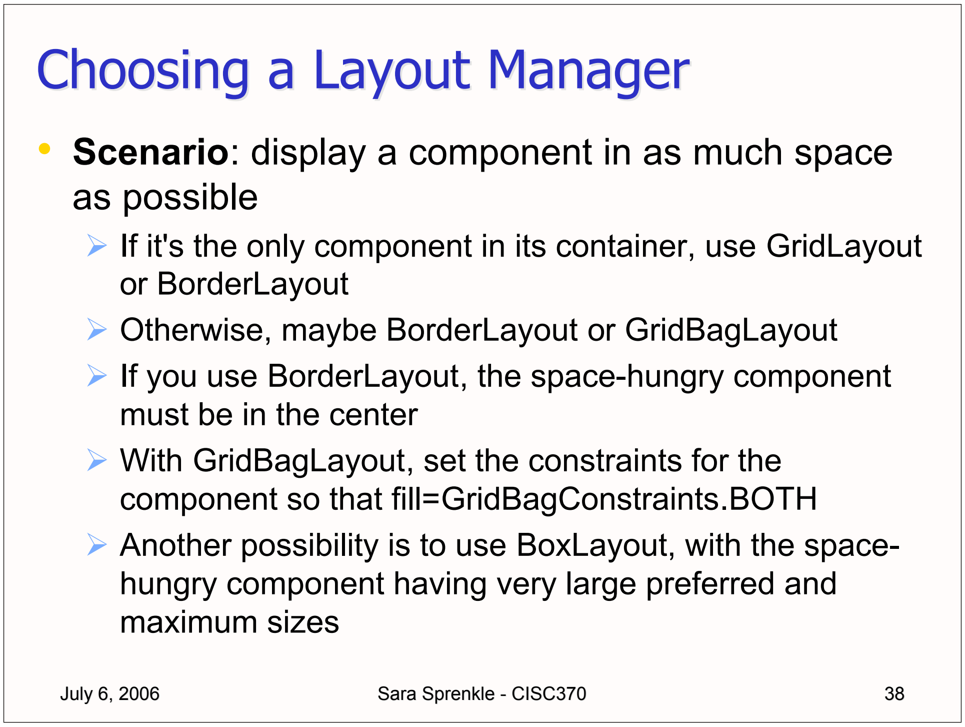
```
// set the layout manager to null
contentPane.setLayout(null);

// create a button
JButton okButton = new JButton(“OK”);

// add the button to the content pane
contentPane.add(okButton);

// set the button’s position to 10,10 and
// set the button’s size to 30x15
okButton.setBounds(10, 10, 30, 15);
```



Choosing a Layout Manager

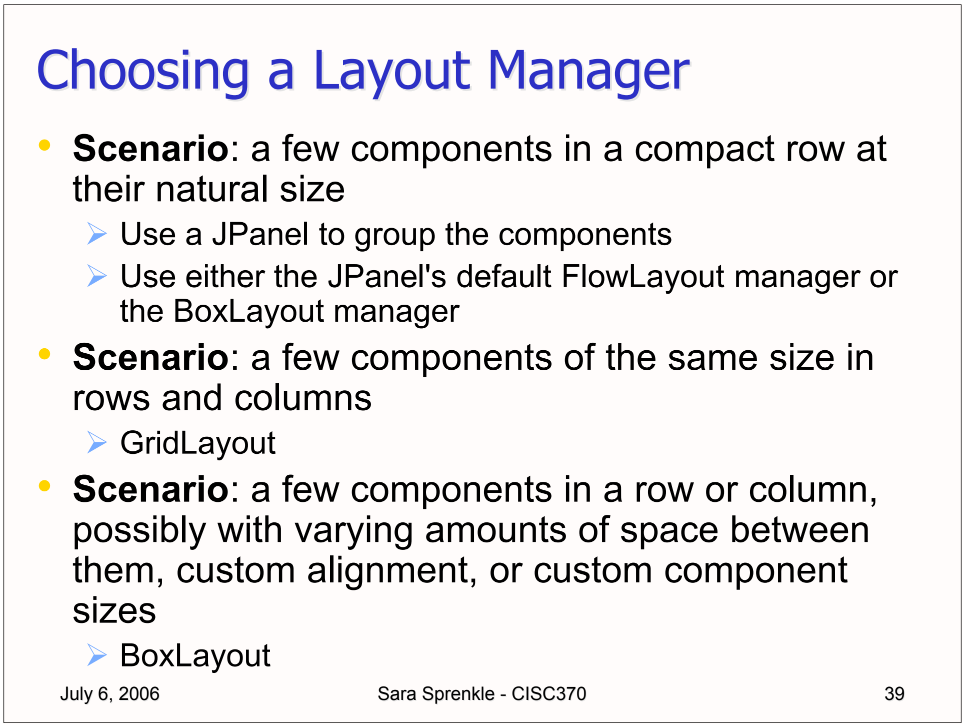
- Visual Guide to Layout Managers

http://java.sun.com/docs/books/tutorial/uiswing/layout/visual.html



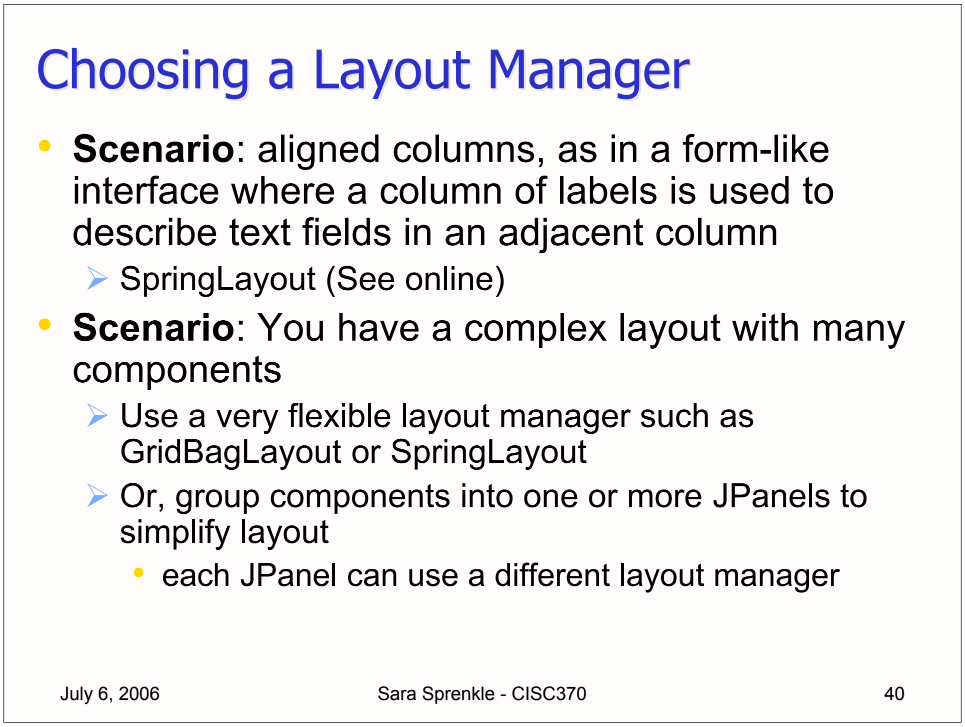
Choosing a Layout Manager

- **Scenario**: display a component in as much space as possible
  - If it's the only component in its container, use GridLayout or BorderLayout
  - Otherwise, maybe BorderLayout or GridBagLayout
  - If you use BorderLayout, the space-hungry component must be in the center
  - With GridBagLayout, set the constraints for the component so that fill=GridBagConstraints.BOTH
  - Another possibility is to use BoxLayout, with the space-hungry component having very large preferred and maximum sizes



Choosing a Layout Manager

- **Scenario**: a few components in a compact row at their natural size
  - Use a JPanel to group the components
  - Use either the JPanel's default FlowLayout manager or the BoxLayout manager
- **Scenario**: a few components of the same size in rows and columns
  - GridLayout
- **Scenario**: a few components in a row or column, possibly with varying amounts of space between them, custom alignment, or custom component sizes
  - BoxLayout



Choosing a Layout Manager

- **Scenario**: aligned columns, as in a form-like interface where a column of labels is used to describe text fields in an adjacent column
  - SpringLayout (See online)
- **Scenario**: You have a complex layout with many components
  - Use a very flexible layout manager such as GridBagLayout or SpringLayout
  - Or, group components into one or more JPanels to simplify layout
    - each JPanel can use a different layout manager

Event Handling

Actions

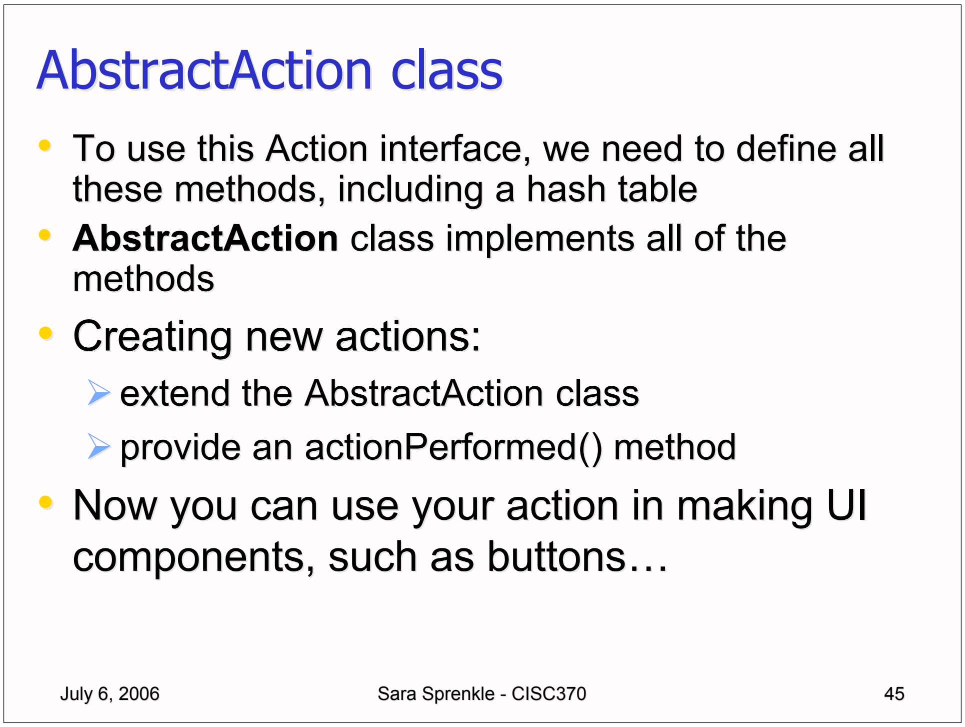
- Often, a program has more than one way to cause a certain action to occur
  - click on a menu option, click a button, press a certain key sequence invokes same method
- The Swing package provides a mechanism to support this in the **Action** interface
  - Action listener
  - Centralized state handling for events

The Action Interface

```
public interface Action
{
     void actionPerformed(ActionEvent event);
     void setEnabled(boolean b);
     boolean isEnabled();
     void putValue(String key, Object value);
     Object getValue(String key);
     void addPropertyChangeListener
          (PropertyChangeListener l);
     void removePropertyChangeListener
          (PropertyChangeListener l);
}
```

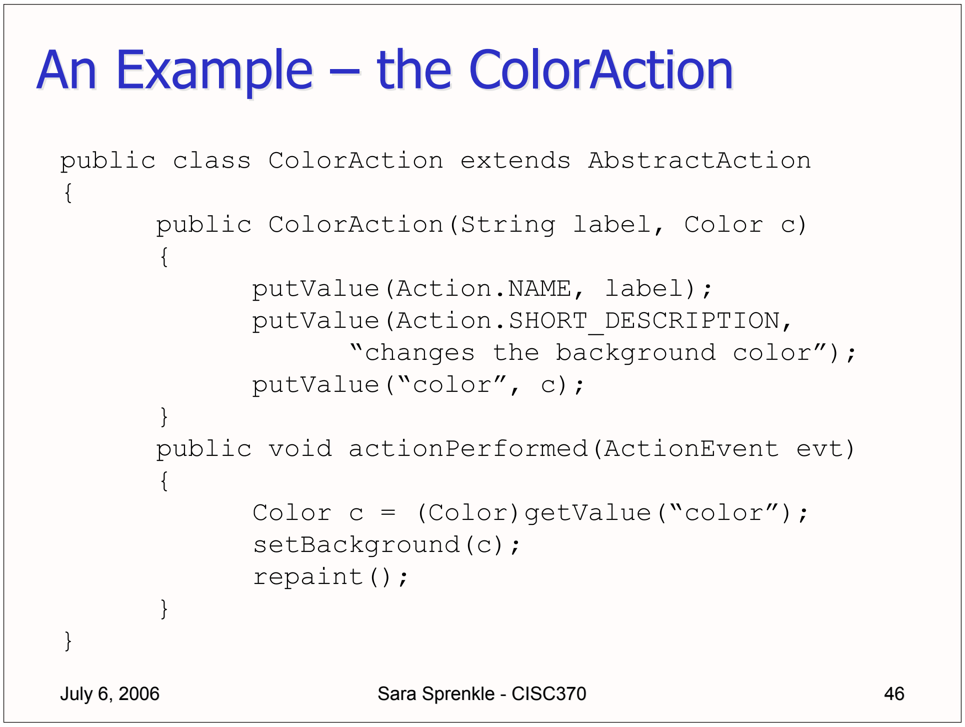
Action Interface Methods

- The actionPerformed() method is like we’ve seen
- setEnabled() and isEnabled()
  - control if the action is permitted right now
  - a disabled action appears grayed out on a menu
- putValue() and setValue()
  - let you store arbitrary name/values pairs in the object
  - basically, the object has its own hash table (HashMap)
- Important Action values to set:
  - **NAME** – name of the action
    - displayed on buttons and menu items
  - **SHORT_DESCRIPTION** – short description of the action
    - appears as a ToolTip



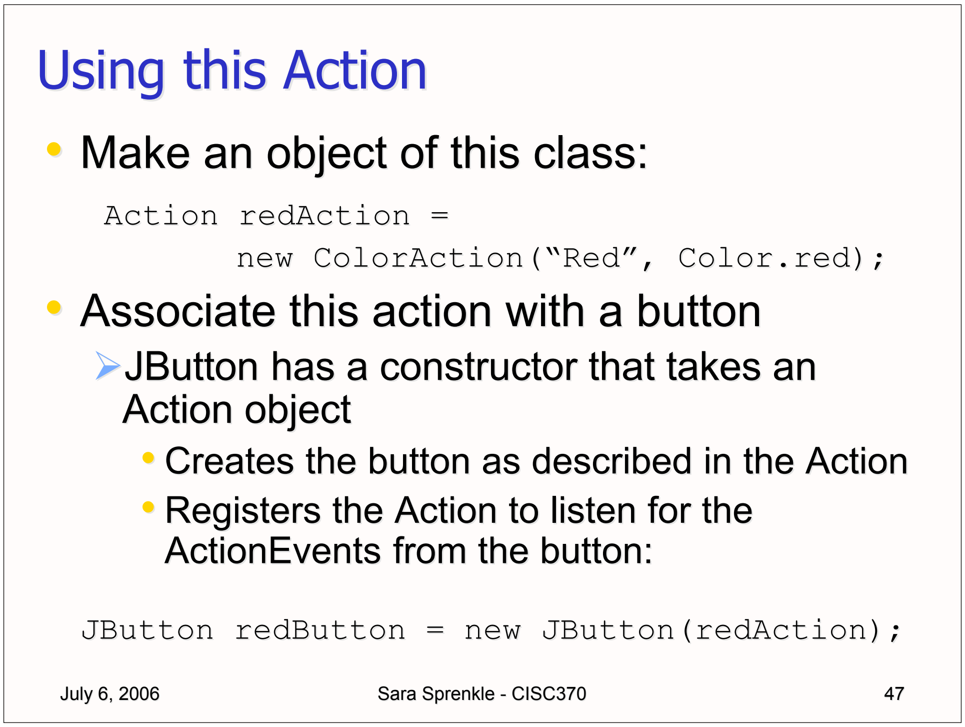
AbstractAction class

- To use this Action interface, we need to define all these methods, including a hash table
- **AbstractAction** class implements all of the methods
- Creating new actions:
  - extend the AbstractAction class
  - provide an actionPerformed() method
- Now you can use your action in making UI components, such as buttons…



An Example – the ColorAction

```
public class ColorAction extends AbstractAction
{
      public ColorAction(String label, Color c)
      {
            putValue(Action.NAME, label);
            putValue(Action.SHORT_DESCRIPTION,
                  “changes the background color”);
            putValue(“color”, c);
      }
      public void actionPerformed(ActionEvent evt)
      {
            Color c = (Color)getValue(“color”);
            setBackground(c);
            repaint();
      }
}
```



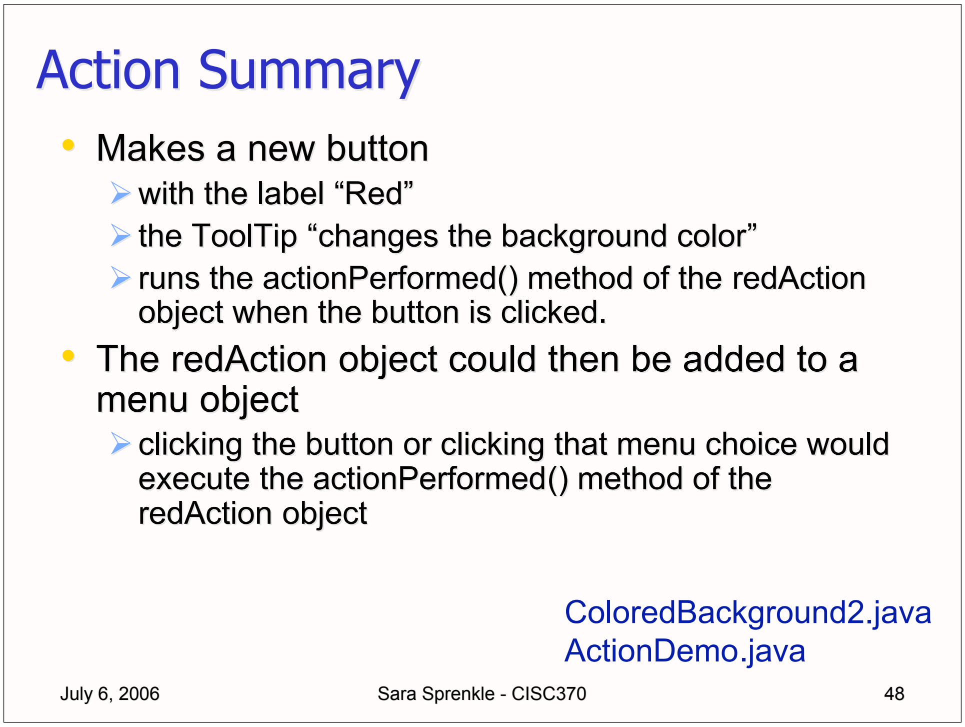
Using this Action

- Make an object of this class:

```
Action redAction =
        new ColorAction("Red", Color.red);
```

- Associate this action with a button
  - JButton has a constructor that takes an Action object
    - Creates the button as described in the Action
    - Registers the Action to listen for the ActionEvents from the button:

```
JButton redButton = new JButton(redAction);
```



Action Summary

- Makes a new button
  - with the label “Red”
  - the ToolTip “changes the background color”
  - runs the actionPerformed() method of the redAction object when the button is clicked.
- The redAction object could then be added to a menu object
  - clicking the button or clicking that menu choice would execute the actionPerformed() method of the redAction object

ColoredBackground2.java
ActionDemo.java

Swing

Text Input UI Components

- Some UI components let the user input and edit text:
  - **JTextField**: text fields
  - **JTextArea**: text areas
- A text field can only accept one line of text
- A text area can accept multiple lines of text

Text Input Component Classes

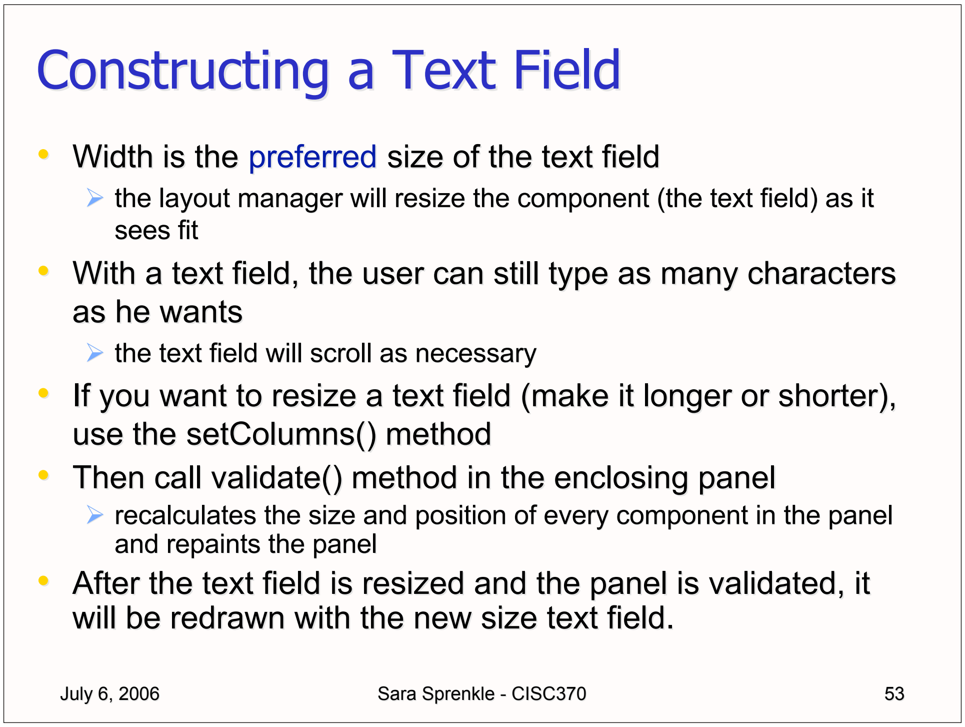
- Both JTextField and JTextArea derive from a class named **JTextComponent**.
    - an abstract class
    - don’t try to make components of this type
    - Common methods to JTextField and JTextArea are defined in JTextComponent

Constructing a Text Field

- Constructing a text field

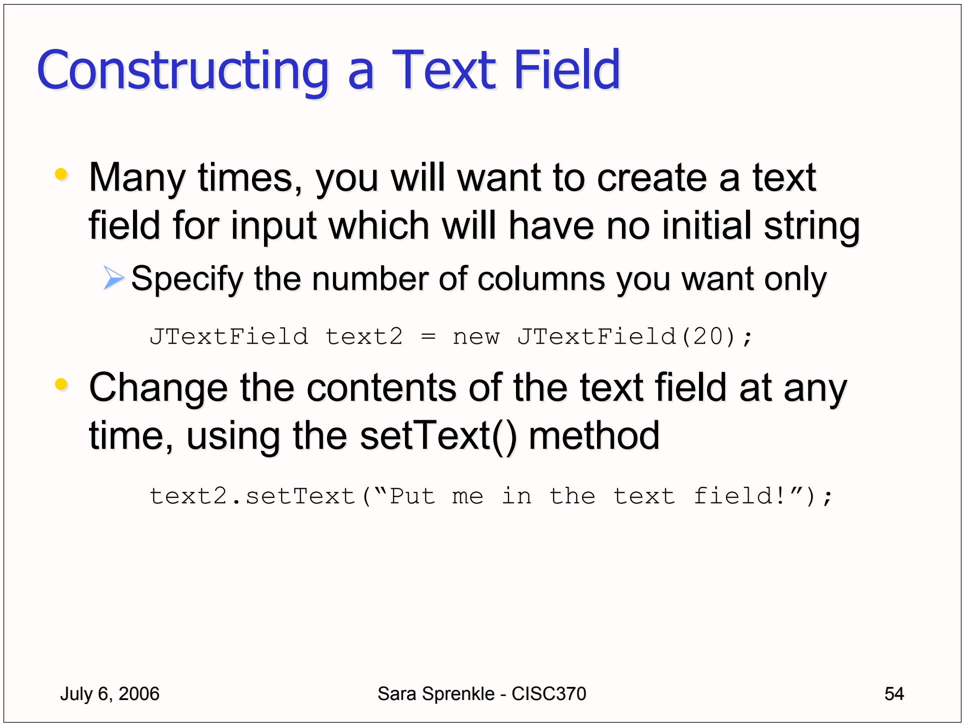
```
JPanel panel1 = new JPanel();
JTextField text1 = new JTextField(“Hi there.”, 20);
panel1.add(text1);
```

- Adds a text field to the panel and place the string “Hi there.” inside of it.
- Second parameter of the constructor sets the width
    - Measured in columns
    - One column is the “expected width” of one character
        - very imprecise measurement
        - add 3 or 4 to the maximum number of characters you expect



Constructing a Text Field

- Width is the preferred size of the text field
  - the layout manager will resize the component (the text field) as it sees fit
- With a text field, the user can still type as many characters as he wants
  - the text field will scroll as necessary
- If you want to resize a text field (make it longer or shorter), use the setColumns() method
- Then call validate() method in the enclosing panel
  - recalculates the size and position of every component in the panel and repaints the panel
- After the text field is resized and the panel is validated, it will be redrawn with the new size text field.



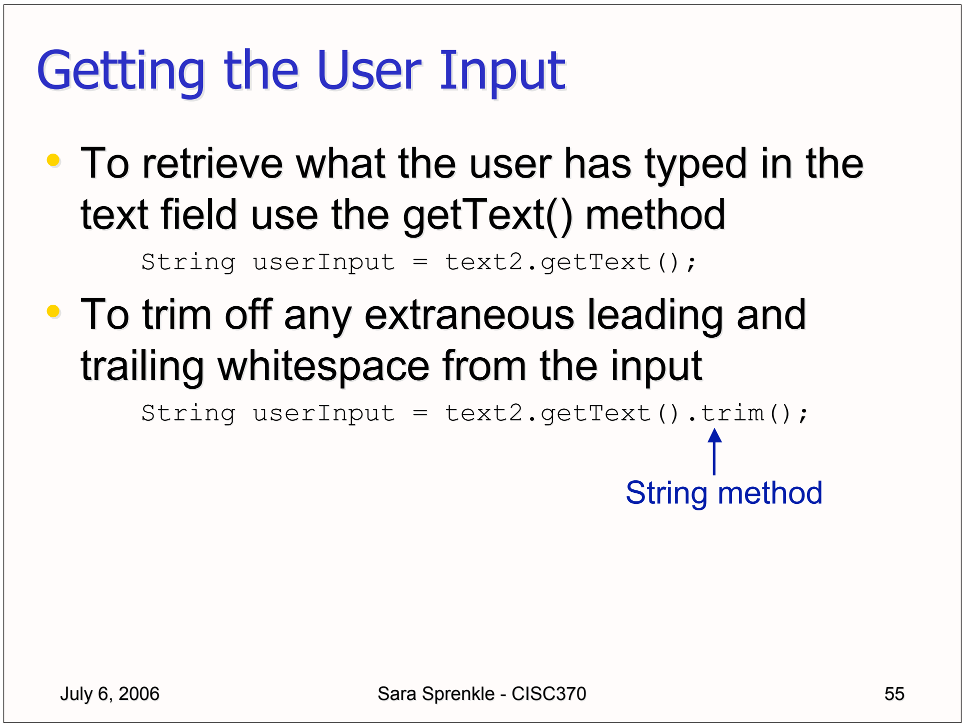
Constructing a Text Field

- Many times, you will want to create a text field for input which will have no initial string
  - Specify the number of columns you want only

```
JTextField text2 = new JTextField(20);
```

- Change the contents of the text field at any time, using the setText() method

```
text2.setText(“Put me in the text field!”);
```



Getting the User Input

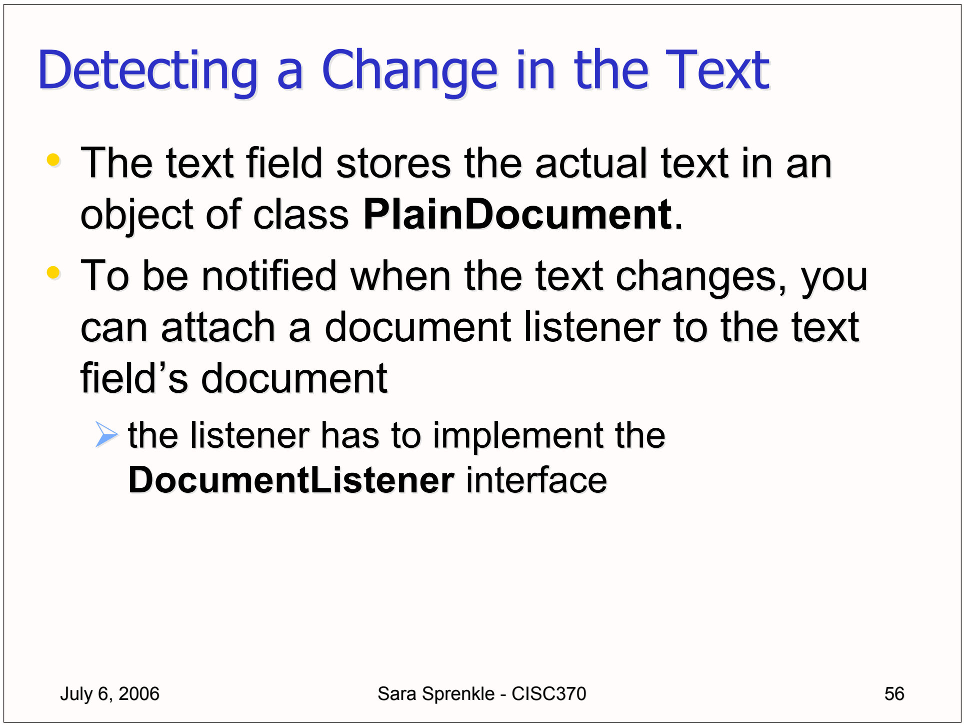
- To retrieve what the user has typed in the text field use the getText() method

```
String userInput = text2.getText();
```

- To trim off any extraneous leading and trailing whitespace from the input

```
String userInput = text2.getText().trim();
```

String method



Detecting a Change in the Text

- The text field stores the actual text in an object of class **PlainDocument**.
- To be notified when the text changes, you can attach a document listener to the text field's document
  - the listener has to implement the **DocumentListener** interface

The DocumentListener Interface

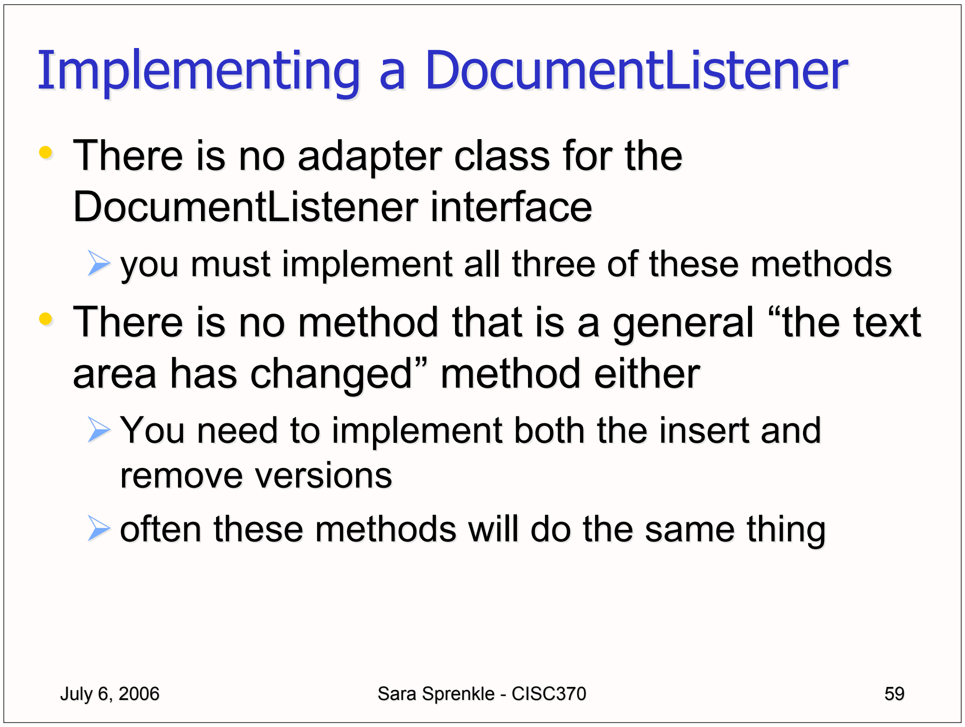
```
public interface DocumentListener
{
      void insertUpdate(DocumentEvent evt1);
      void removeUpdate(DocumentEvent evt2);
      void changedUpdate(DocumentEvent evt3);
}
```

- insertUpdate gets called whenever a character is inserted into the text field
- removeUpdate gets called whenever a character is deleted from the text field
- changedUpdate is never called

Implementing a DocumentListener

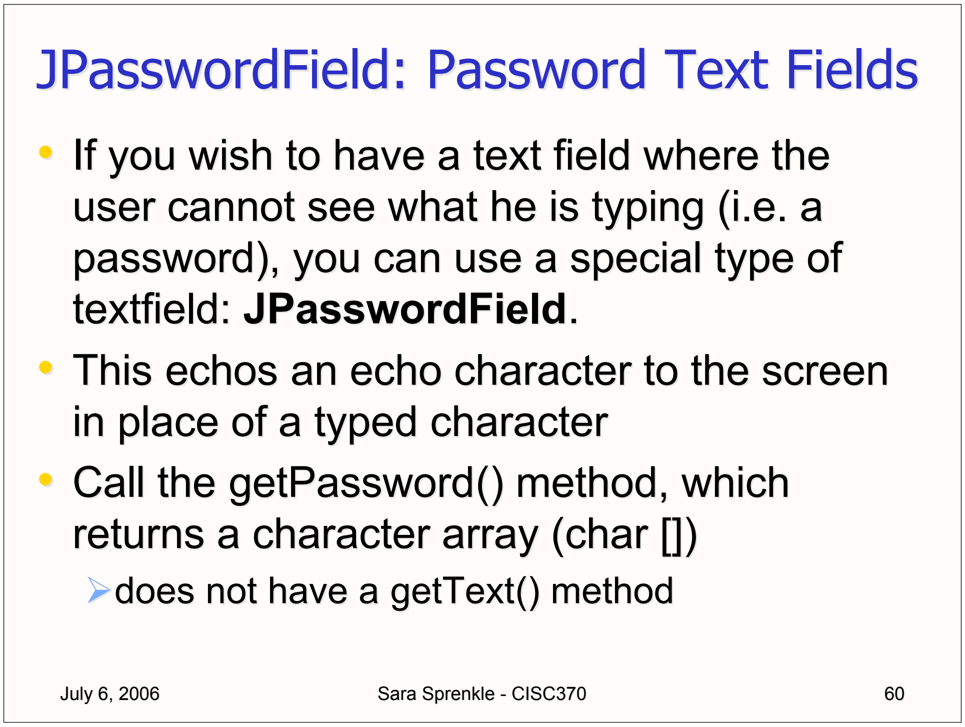
- To implement a class to listen to a textfield

```
private class MyFieldListener implements DocumentListener
{
      public void insertUpdate(DocumentEvent evt)
      {
            // get the text from the field and process it
      }
      public void removeUpdate(DocumentEvent evt)
      {
            // get the text from the field and process it
      }
      public void changedUpdate(DocumentEvent evt)
      { }
}
```



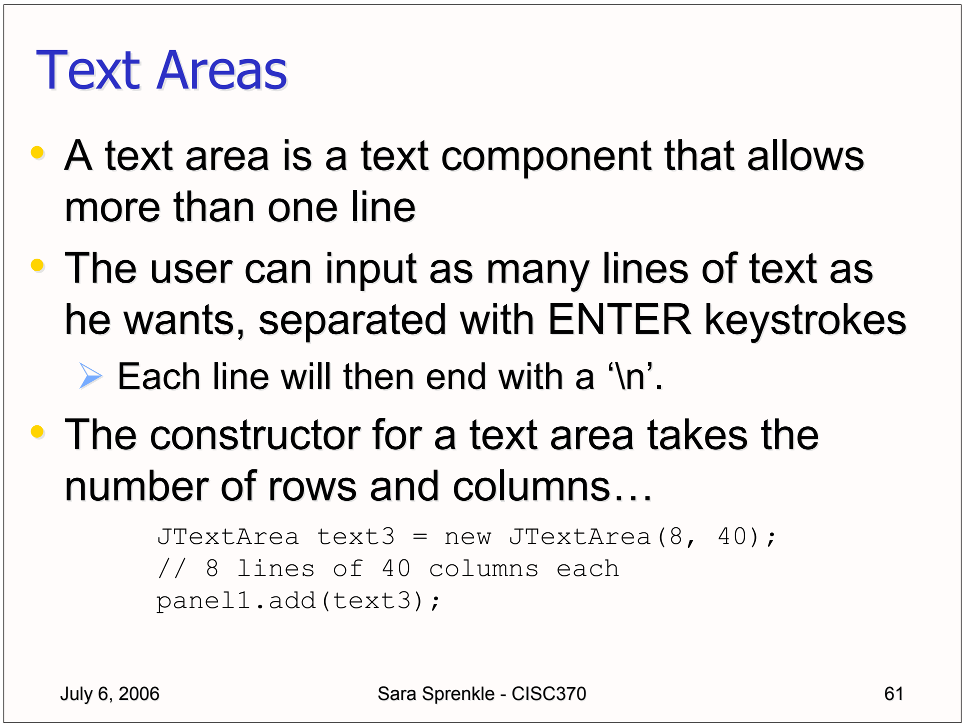
Implementing a DocumentListener

- There is no adapter class for the DocumentListener interface
  - you must implement all three of these methods
- There is no method that is a general “the text area has changed” method either
  - You need to implement both the insert and remove versions
  - often these methods will do the same thing



JPasswordField: Password Text Fields

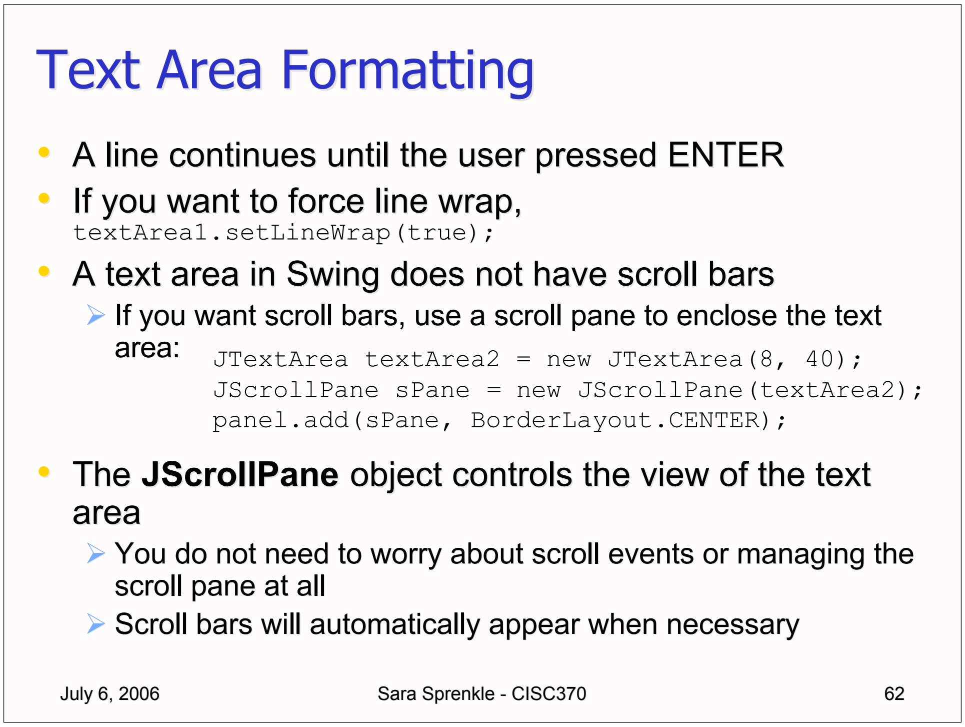
- If you wish to have a text field where the user cannot see what he is typing (i.e. a password), you can use a special type of textfield: **JPasswordField**.
- This echos an echo character to the screen in place of a typed character
- Call the getPassword() method, which returns a character array (char [])
  - does not have a getText() method



Text Areas

- A text area is a text component that allows more than one line
- The user can input as many lines of text as he wants, separated with ENTER keystrokes
  - Each line will then end with a ‘\n’.
- The constructor for a text area takes the number of rows and columns…

```
JTextArea text3 = new JTextArea(8, 40);
// 8 lines of 40 columns each
panel1.add(text3);
```

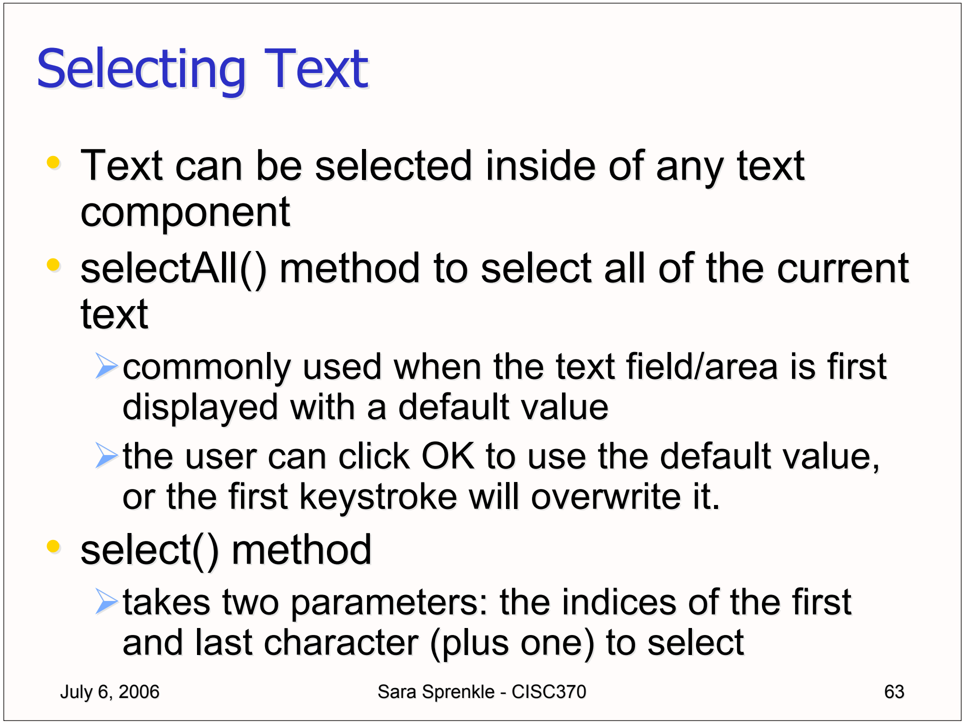


Text Area Formatting

- A line continues until the user pressed ENTER
- If you want to force line wrap, `textArea1.setLineWrap(true);`
- A text area in Swing does not have scroll bars
  - If you want scroll bars, use a scroll pane to enclose the text area:

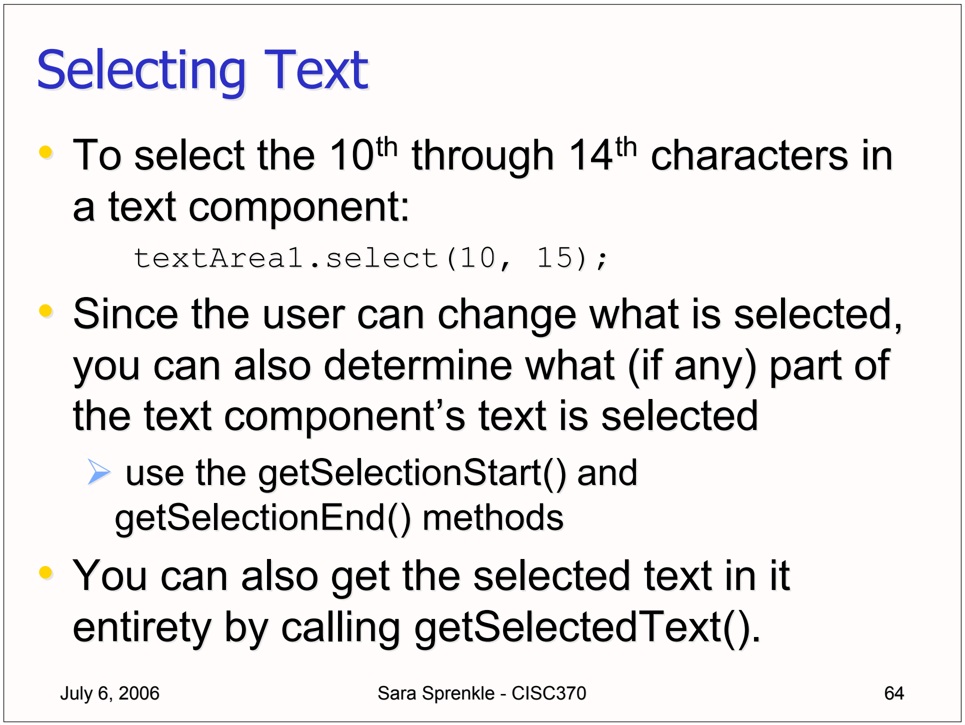
```
JTextArea textArea2 = new JTextArea(8, 40);
JScrollPane sPane = new JScrollPane(textArea2);
panel.add(sPane, BorderLayout.CENTER);
```

- The **JScrollPane** object controls the view of the text area
  - You do not need to worry about scroll events or managing the scroll pane at all
  - Scroll bars will automatically appear when necessary



Selecting Text

- Text can be selected inside of any text component
- selectAll() method to select all of the current text
  - commonly used when the text field/area is first displayed with a default value
  - the user can click OK to use the default value, or the first keystroke will overwrite it.
- select() method
  - takes two parameters: the indices of the first and last character (plus one) to select

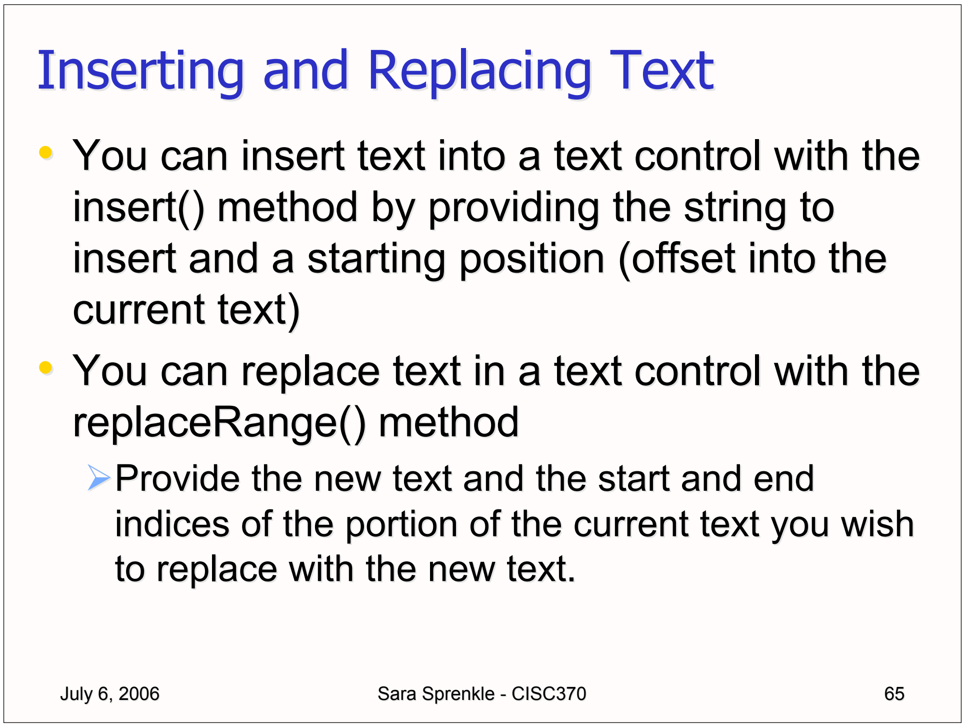


Selecting Text

- To select the 10th through 14th characters in a text component:

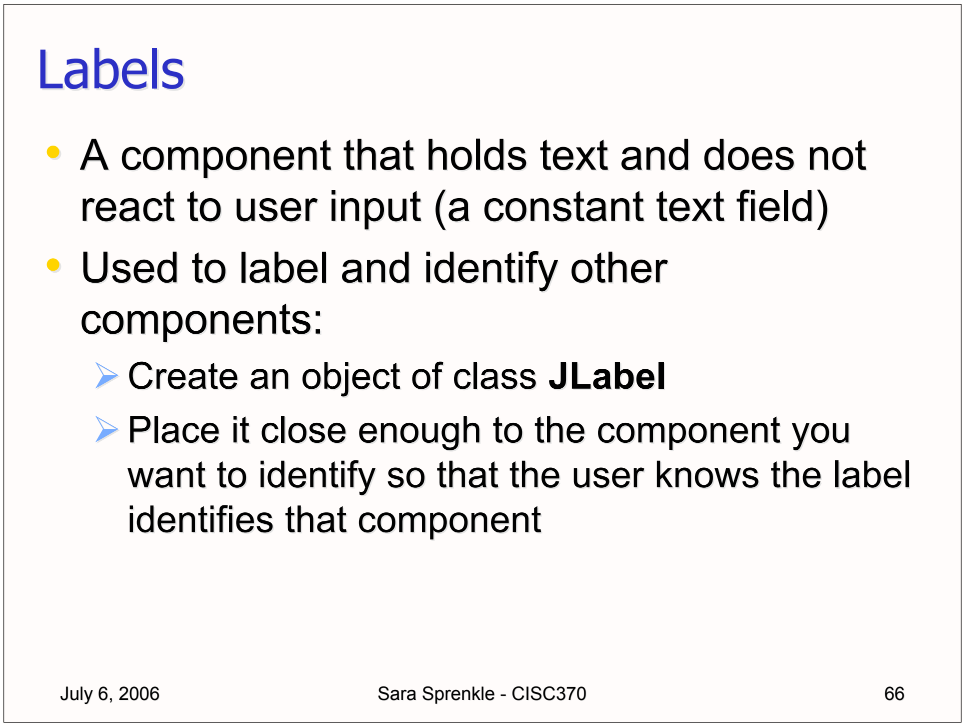
```
textArea1.select(10, 15);
```

- Since the user can change what is selected, you can also determine what (if any) part of the text component’s text is selected
  - use the getSelectionStart() and getSelectionEnd() methods
- You can also get the selected text in it entirety by calling getSelectedText().



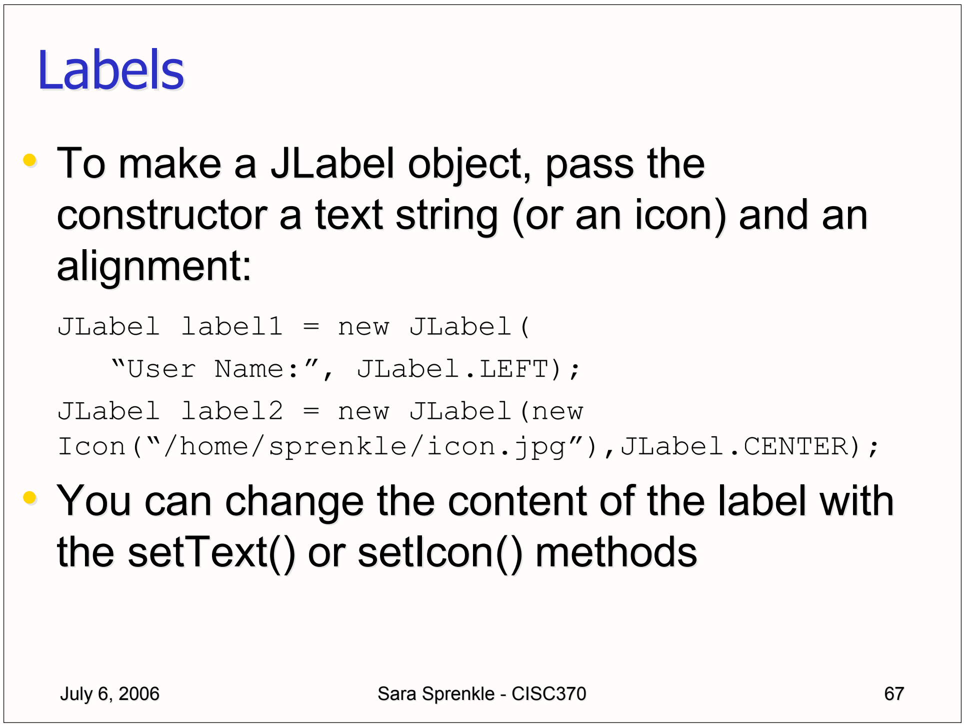
Inserting and Replacing Text

- You can insert text into a text control with the insert() method by providing the string to insert and a starting position (offset into the current text)
- You can replace text in a text control with the replaceRange() method
  - Provide the new text and the start and end indices of the portion of the current text you wish to replace with the new text.



Labels

- A component that holds text and does not react to user input (a constant text field)
- Used to label and identify other components:
  - Create an object of class **JLabel**
  - Place it close enough to the component you want to identify so that the user knows the label identifies that component

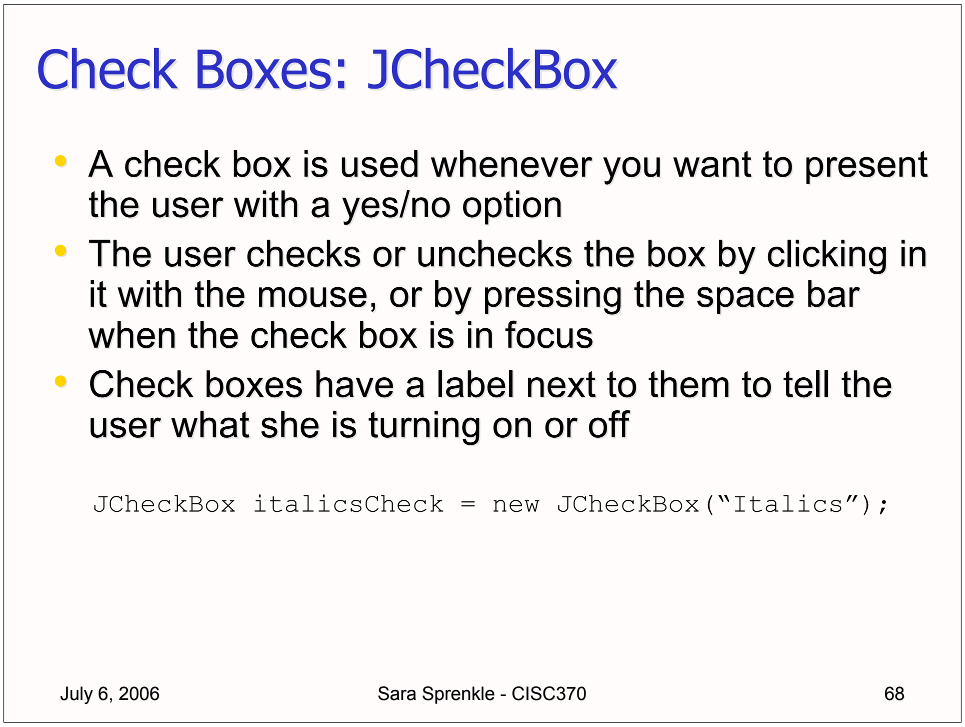


Labels

- To make a JLabel object, pass the constructor a text string (or an icon) and an alignment:

```
JLabel label1 = new JLabel(
   “User Name:”, JLabel.LEFT);
JLabel label2 = new JLabel(new
Icon(“/home/sprenkle/icon.jpg”),JLabel.CENTER);
```

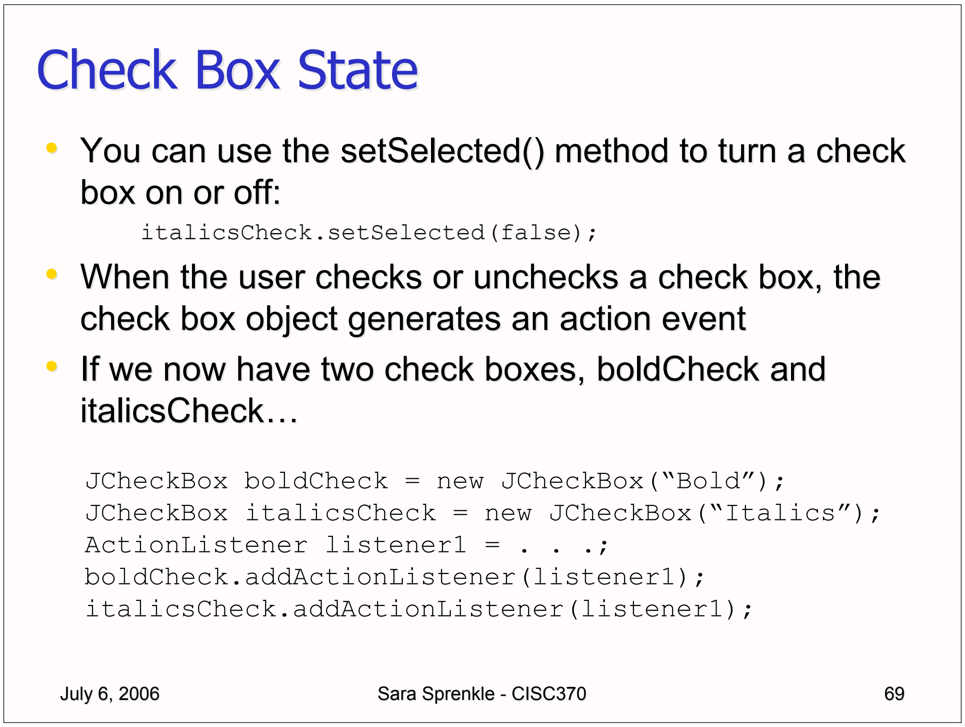
- You can change the content of the label with the setText() or setIcon() methods



Check Boxes: JCheckBox

- A check box is used whenever you want to present the user with a yes/no option
- The user checks or unchecks the box by clicking in it with the mouse, or by pressing the space bar when the check box is in focus
- Check boxes have a label next to them to tell the user what she is turning on or off

```
JCheckBox italicsCheck = new JCheckBox(“Italics”);
```



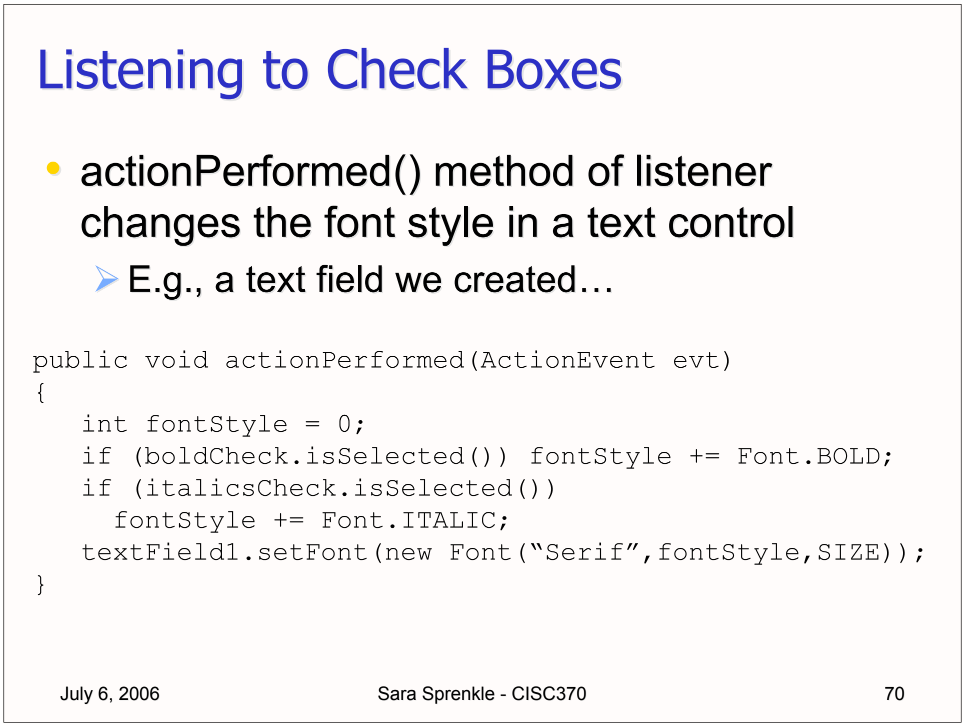
Check Box State

- You can use the setSelected() method to turn a check box on or off:

```
    italicsCheck.setSelected(false);
```

- When the user checks or unchecks a check box, the check box object generates an action event
- If we now have two check boxes, boldCheck and italicsCheck…

```
JCheckBox boldCheck = new JCheckBox(“Bold”);
JCheckBox italicsCheck = new JCheckBox(“Italics”);
ActionListener listener1 = . . .;
boldCheck.addActionListener(listener1);
italicsCheck.addActionListener(listener1);
```



Listening to Check Boxes

- actionPerformed() method of listener changes the font style in a text control
    - E.g., a text field we created…

```
public void actionPerformed(ActionEvent evt)
{
   int fontStyle = 0;
   if (boldCheck.isSelected()) fontStyle += Font.BOLD;
   if (italicsCheck.isSelected())
     fontStyle += Font.ITALIC;
   textField1.setFont(new Font(“Serif”,fontStyle,SIZE));
}
```

JRadioButton: Radio Buttons

- Sometimes, you may wish to present the user with a couple of choices
- Radio button: only one of a group of radio buttons can be selected at a time.
- A group of radio buttons is defined as a **ButtonGroup** class object
  - Add the actual radio buttons **JRadioButton**

Radio Buttons Example

- For example…

```
ButtonGroup group1 = new ButtonGroup();

// make the buttons, true for the default button
JRadioButton noneBu = new JRadioButton(“Normal”, true);
JRadioButton boldBu = new JRadioButton(“Bold”, false);
JRadioButton italBu = new JRadioButton(“Italics”, false);
JRadioButton bandiBu= new JRadioButton(“BoldItal”,false);

group1.add(noneBu);
group1.add(boldBu);
group1.add(italBu);
group1.add(bandiBu);
```

Radio Button Listeners

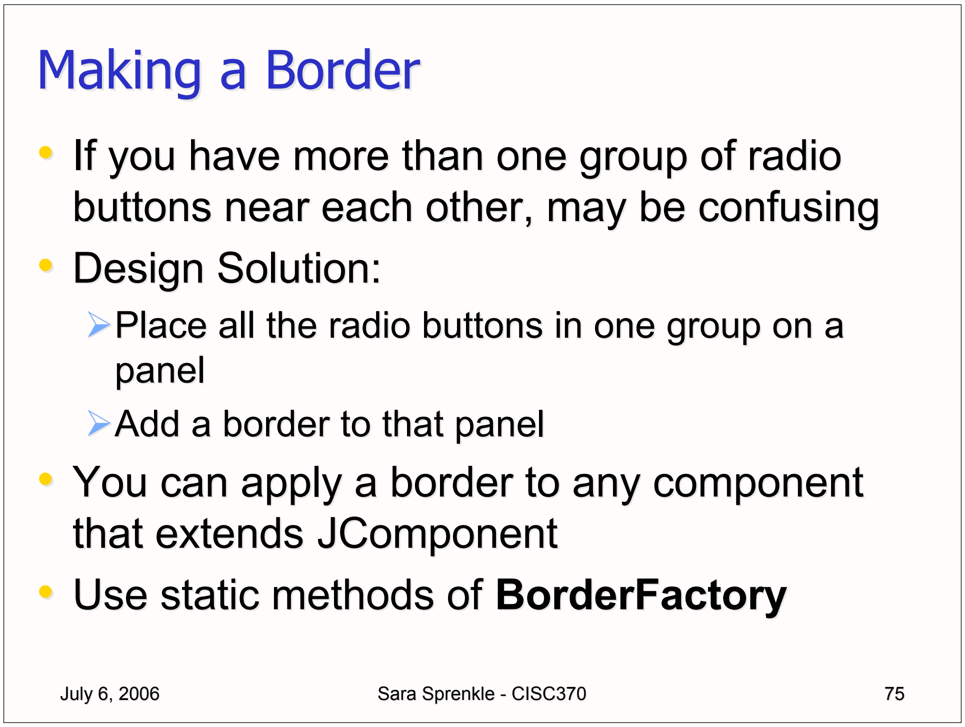
- We add listeners to radio buttons in much the same way as other “action” components
  - Example: one listener for all buttons

```
ActionListener listener1 = . . .;

noneButton.addActionListener(listener1);
boldButton.addActionListener(listener1);
italButton.addActionListener(listener1);
bandiButton.addActionListener(listener1);
```

Radio Button Listeners

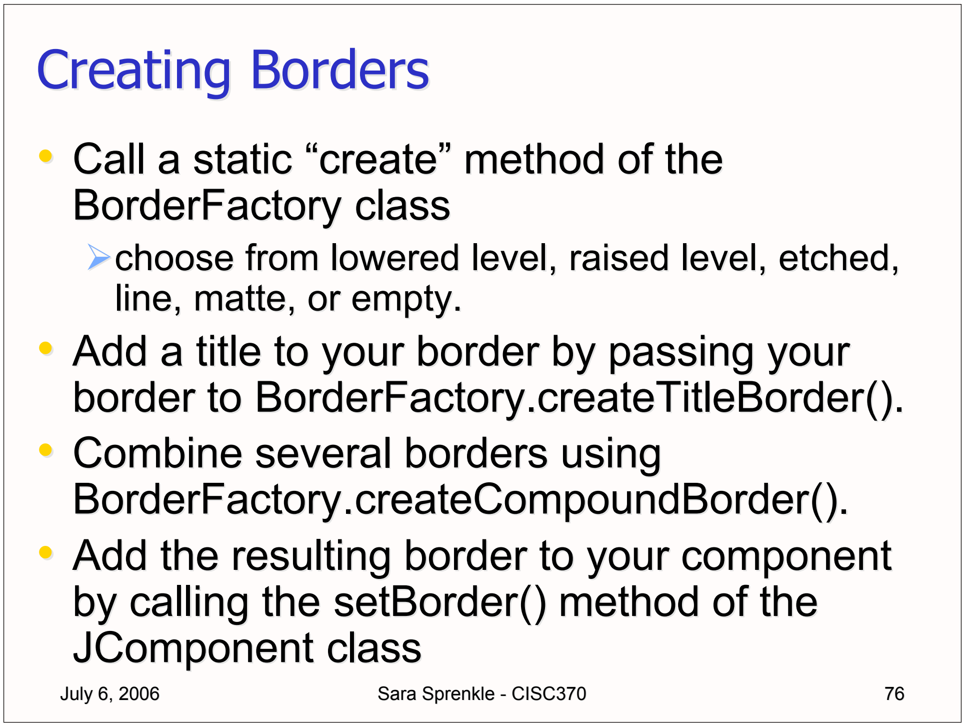
- Example: one listener for all the radio buttons
  - determine the source of the event in the listener’s actionPerformed() method…

```
public void actionPerformed(ActionEvent evt)
{
      Object source = evt.getSource();
      int style;
      if (source == noneButton)  style = 0;
      if (source == boldButton)  style = Font.BOLD;
      if (source == italButton)  style = Font.ITALIC;
      if (source == bandiButton) style = Font.BOLD +
            Font.ITALIC;
      textField1.setFont(new Font(“Serif”,fontStyle,SIZE);
}
```



Making a Border

- If you have more than one group of radio buttons near each other, may be confusing
- Design Solution:
  - ➢Place all the radio buttons in one group on a panel
  - ➢Add a border to that panel
- You can apply a border to any component that extends JComponent
- Use static methods of **BorderFactory**



Creating Borders

- Call a static “create” method of the BorderFactory class
  - ➢choose from lowered level, raised level, etched, line, matte, or empty.
- Add a title to your border by passing your border to BorderFactory.createTitleBorder().
- Combine several borders using BorderFactory.createCompoundBorder().
- Add the resulting border to your component by calling the setBorder() method of the JComponent class

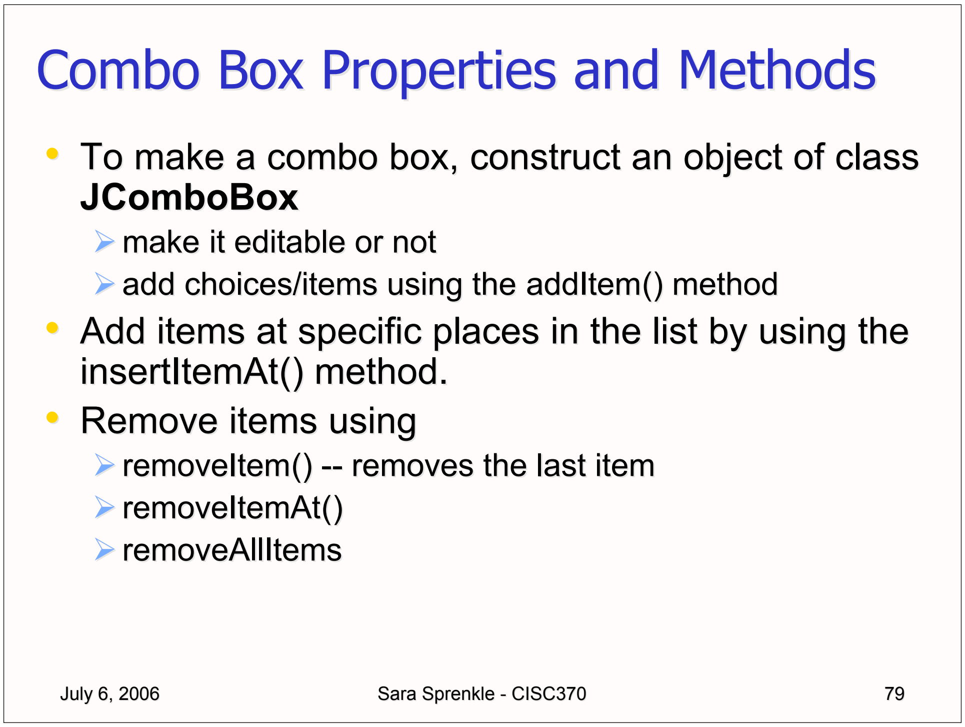
Creating Borders

- To create a compound etched and matte border with a title and place that border around a panel

```
Border etched = BorderFactory.createEtchedBorder();
Border matte  = BorderFactory.createMatteBorder();
Border combo  = BorderFactory.createCompoundBorder(
      etched, matte);
Border titled = BorderFactory.createTitledBorder(
      combo, “The Panel’s Title”);
panel1.setBorder(titled);
```

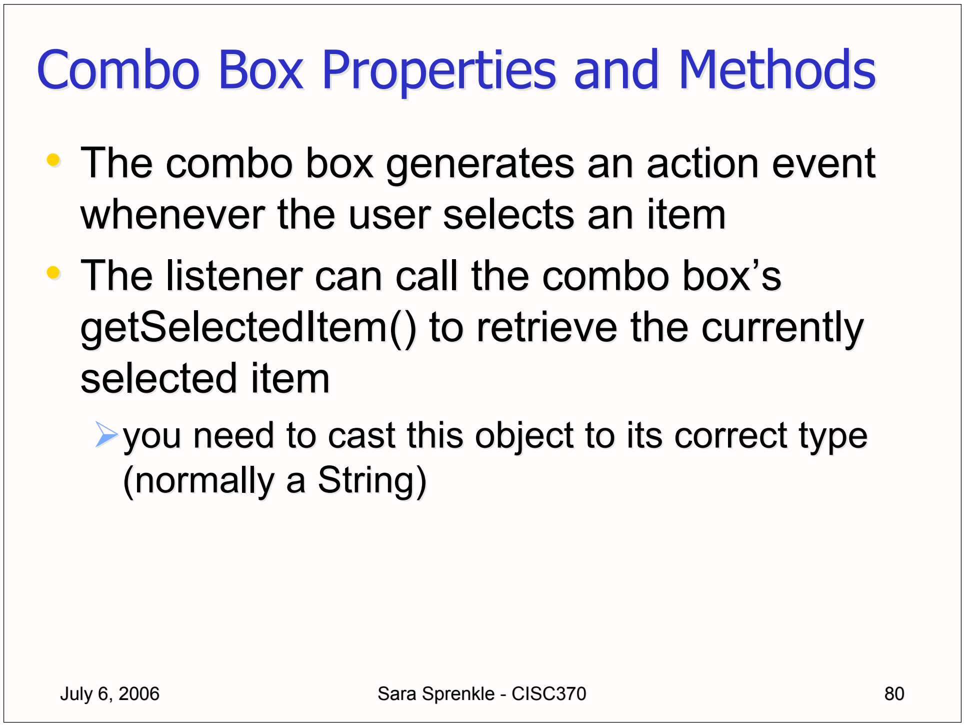
Combo Boxes

- When you have a lot of different options, radio buttons are not practical
  - can take up a lot of space
- The solution is to use a combo box
  - like a list
  - shows the currently selected item
  - when the user clicks on the component, it drops down a list of the possible choices for the user to select
- You can edit the current selection as if it were a text field
  - Turn on/off the editing feature by calling the setEditable() method



Combo Box Properties and Methods

- To make a combo box, construct an object of class **JComboBox**
  - make it editable or not
  - add choices/items using the addItem() method
- Add items at specific places in the list by using the insertItemAt() method.
- Remove items using
  - removeItem() -- removes the last item
  - removeItemAt()
  - removeAllItems



Combo Box Properties and Methods

- The combo box generates an action event whenever the user selects an item
- The listener can call the combo box’s getSelectedItem() to retrieve the currently selected item
  - you need to cast this object to its correct type (normally a String)

A Combo Box Example

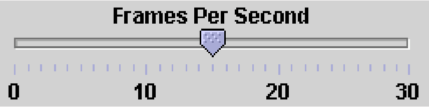
- To construct a combo box:

```
JComboBox faceCombo = new JComboBox();
faceCombo.setEditable(false);
faceCombo.addItem(“Serif”);
faceCombo.addItem(“SansSerif”);
```

- Create the listener

```
public void actionPerformed(ActionEvent evt)
{
      textArea1.setFont( new Font(
            (String)faceCombo.getSelectedItem(),
            Font.PLAIN,
            DEFAULT_SIZE));
}
```

Sliders: JSlider

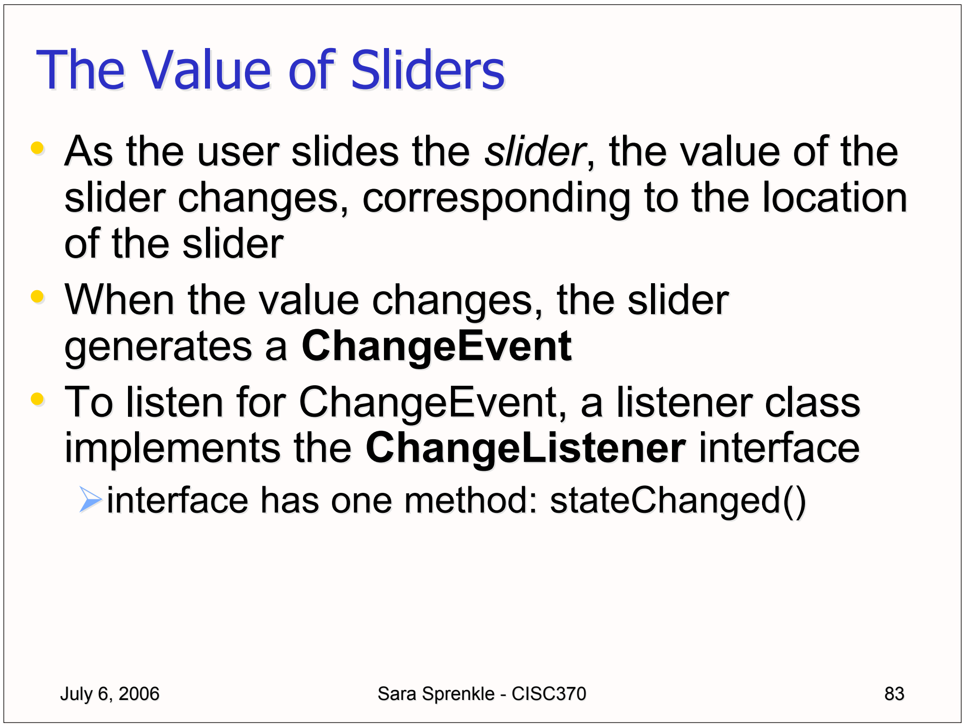


- Combo boxes allow the user to choose from a set of discrete values.
- Sliders let the user choose from a continuum of values
  - e.g., any number from 1 to 100
- The most common way to make a slider:

```
JSlider slider1 = new JSlider(min, max, initValue);
```

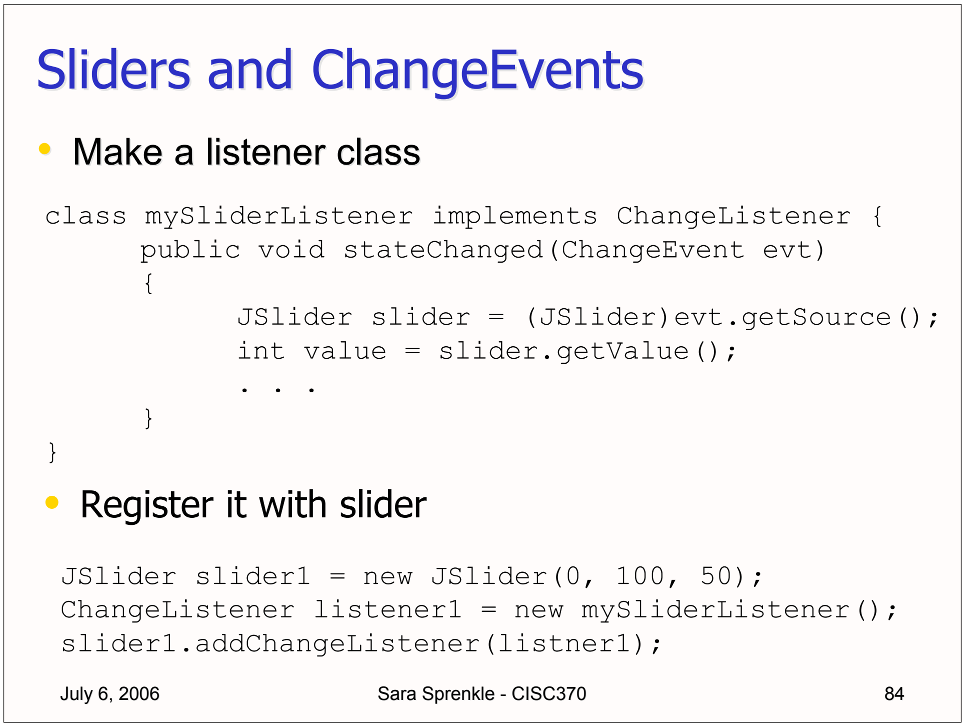
- To make the slider vertical instead of horizontal:

```
JSlider slider2 = new JSlider(
   SwingConstants.VERTICAL, min, max, initValue);
```



The Value of Sliders

- As the user slides the *slider*, the value of the slider changes, corresponding to the location of the slider
- When the value changes, the slider generates a **ChangeEvent**
- To listen for ChangeEvent, a listener class implements the **ChangeListener** interface
  - interface has one method: stateChanged()



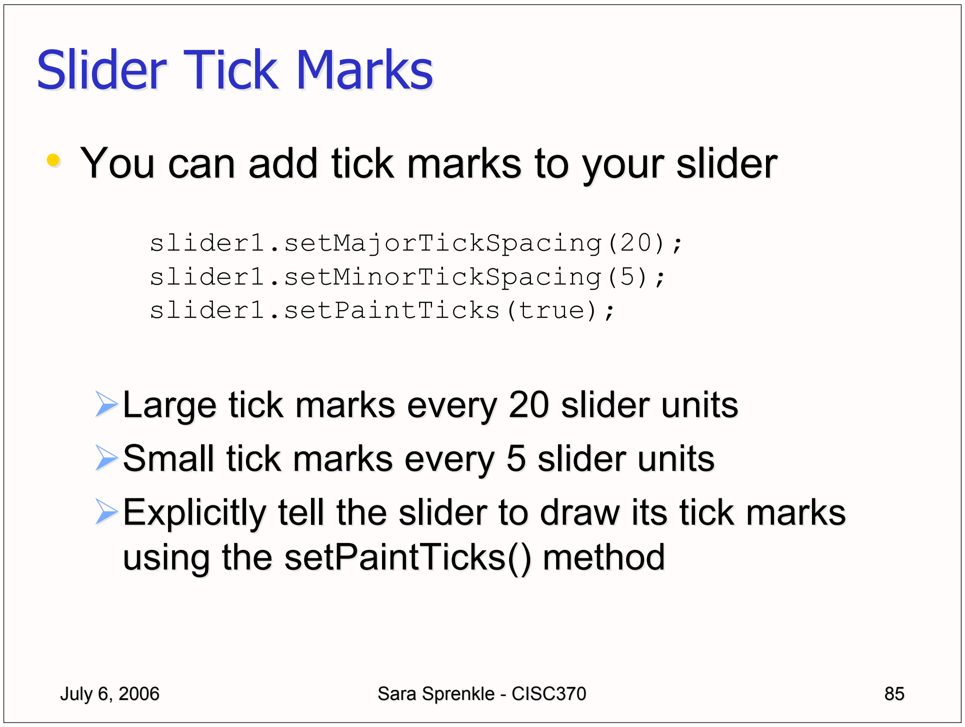
Sliders and ChangeEvents

- Make a listener class

```
class mySliderListener implements ChangeListener {
     public void stateChanged(ChangeEvent evt)
     {
          JSlider slider = (JSlider)evt.getSource();
          int value = slider.getValue();
          . . .
     }
}
```

- Register it with slider

```
JSlider slider1 = new JSlider(0, 100, 50);
ChangeListener listener1 = new mySliderListener();
slider1.addChangeListener(listner1);
```

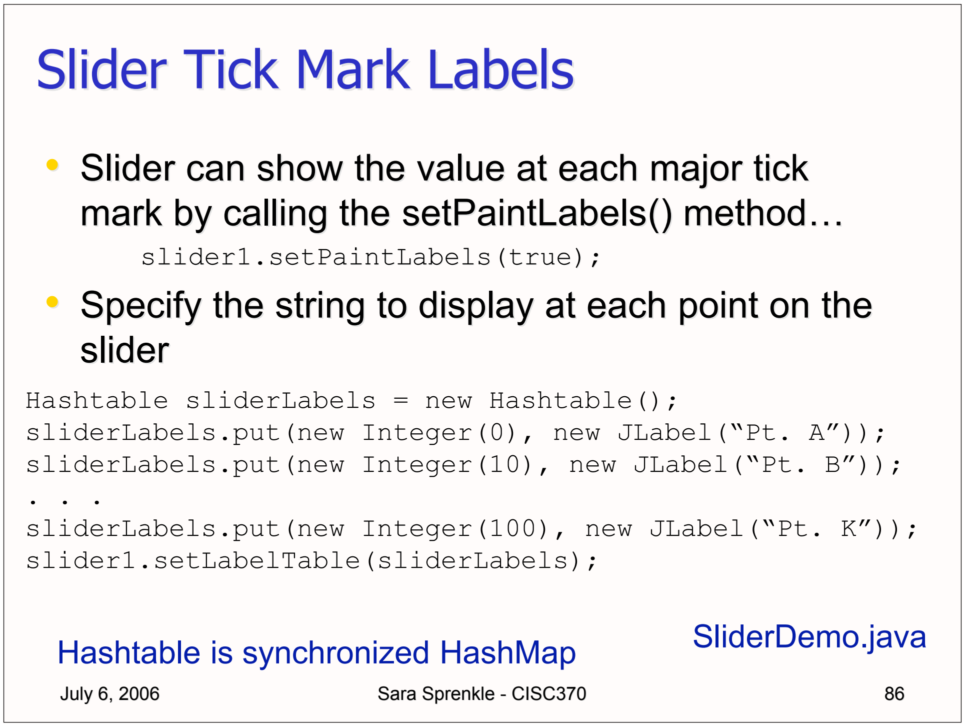


Slider Tick Marks

- You can add tick marks to your slider

```
slider1.setMajorTickSpacing(20);
slider1.setMinorTickSpacing(5);
slider1.setPaintTicks(true);
```

  - Large tick marks every 20 slider units
  - Small tick marks every 5 slider units
  - Explicitly tell the slider to draw its tick marks using the setPaintTicks() method



Slider Tick Mark Labels

- Slider can show the value at each major tick mark by calling the setPaintLabels() method…

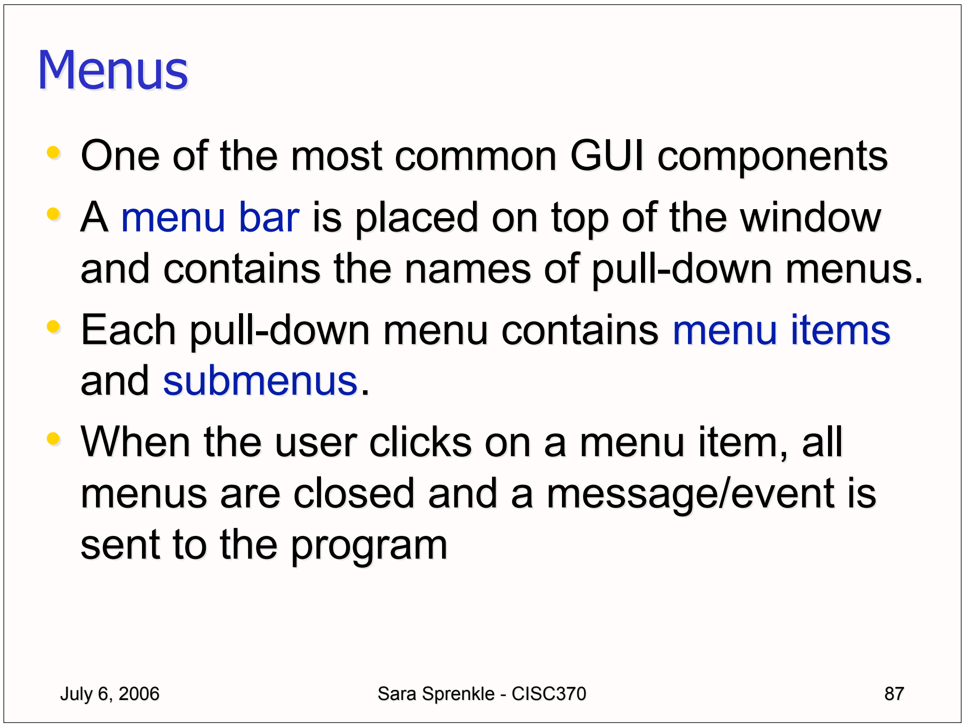
```
slider1.setPaintLabels(true);
```

- Specify the string to display at each point on the slider

```
Hashtable sliderLabels = new Hashtable();
sliderLabels.put(new Integer(0), new JLabel(“Pt. A”));
sliderLabels.put(new Integer(10), new JLabel(“Pt. B”));
. . .
sliderLabels.put(new Integer(100), new JLabel(“Pt. K”));
slider1.setLabelTable(sliderLabels);
```

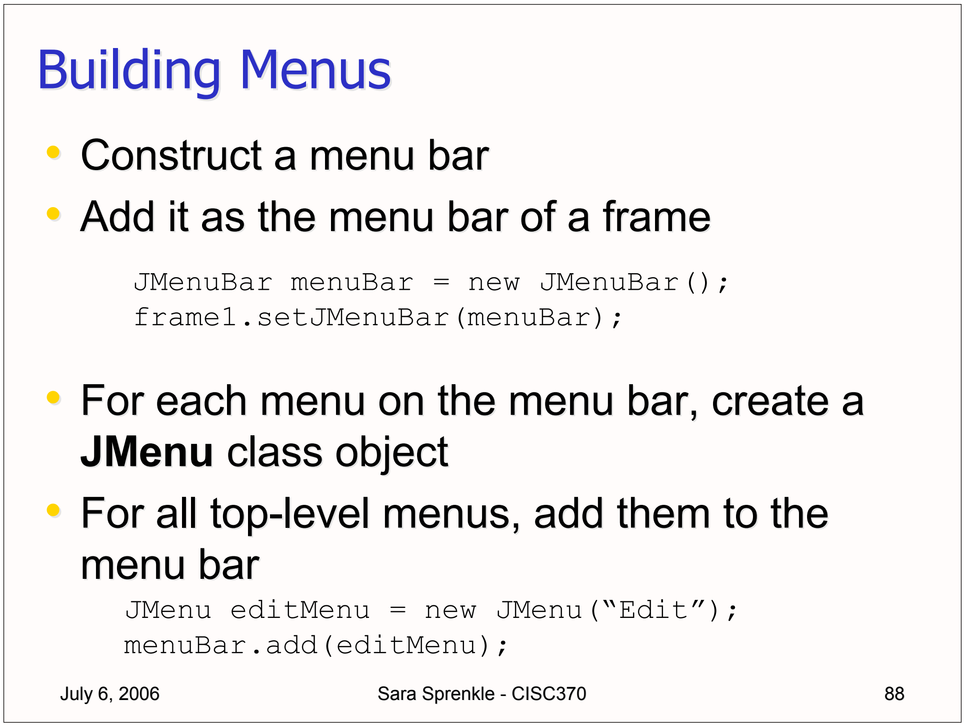
SliderDemo.java

Hashtable is synchronized HashMap



Menus

- One of the most common GUI components
- A menu bar is placed on top of the window and contains the names of pull-down menus.
- Each pull-down menu contains menu items and submenus.
- When the user clicks on a menu item, all menus are closed and a message/event is sent to the program



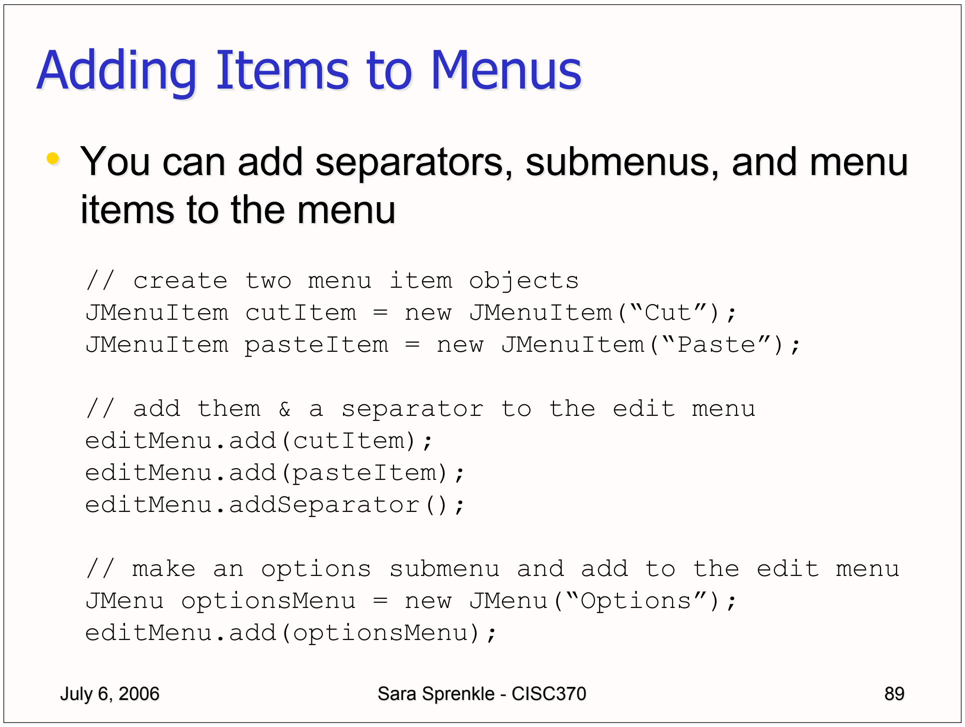
Building Menus

- Construct a menu bar
- Add it as the menu bar of a frame

```
JMenuBar menuBar = new JMenuBar();
frame1.setJMenuBar(menuBar);
```

- For each menu on the menu bar, create a **JMenu** class object
- For all top-level menus, add them to the menu bar

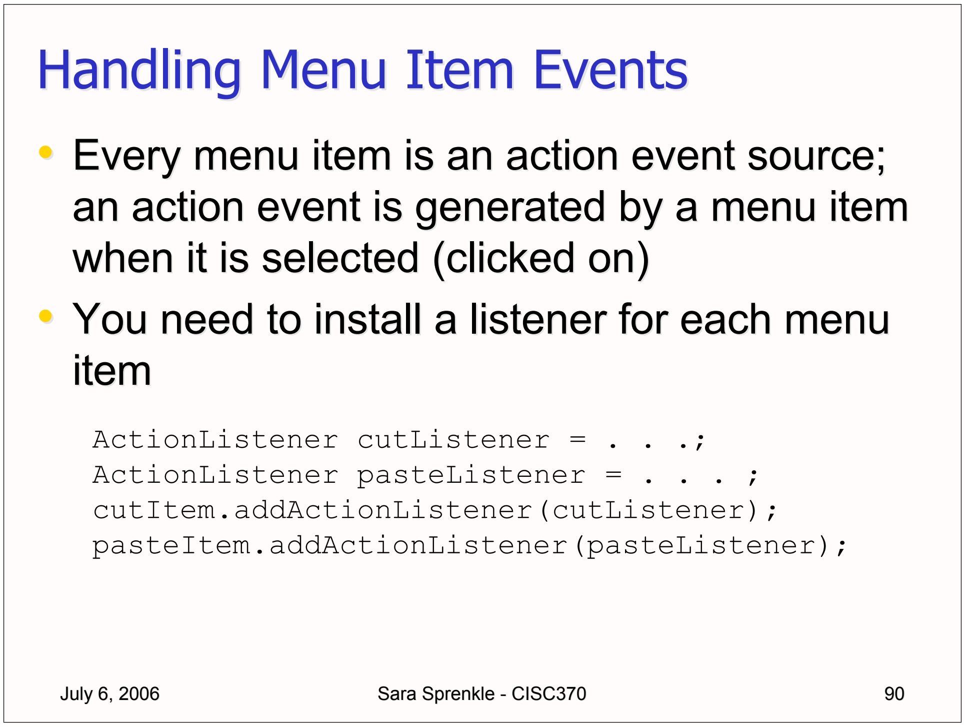
```
JMenu editMenu = new JMenu(“Edit”);
menuBar.add(editMenu);
```



Adding Items to Menus

- You can add separators, submenus, and menu items to the menu

```
// create two menu item objects
JMenuItem cutItem = new JMenuItem(“Cut”);
JMenuItem pasteItem = new JMenuItem(“Paste”);

// add them & a separator to the edit menu
editMenu.add(cutItem);
editMenu.add(pasteItem);
editMenu.addSeparator();

// make an options submenu and add to the edit menu
JMenu optionsMenu = new JMenu(“Options”);
editMenu.add(optionsMenu);
```



Handling Menu Item Events

- Every menu item is an action event source; an action event is generated by a menu item when it is selected (clicked on)
- You need to install a listener for each menu item

```
ActionListener cutListener = . . .;
ActionListener pasteListener = . . . ;
cutItem.addActionListener(cutListener);
pasteItem.addActionListener(pasteListener);
```

Adding Actions to Menus

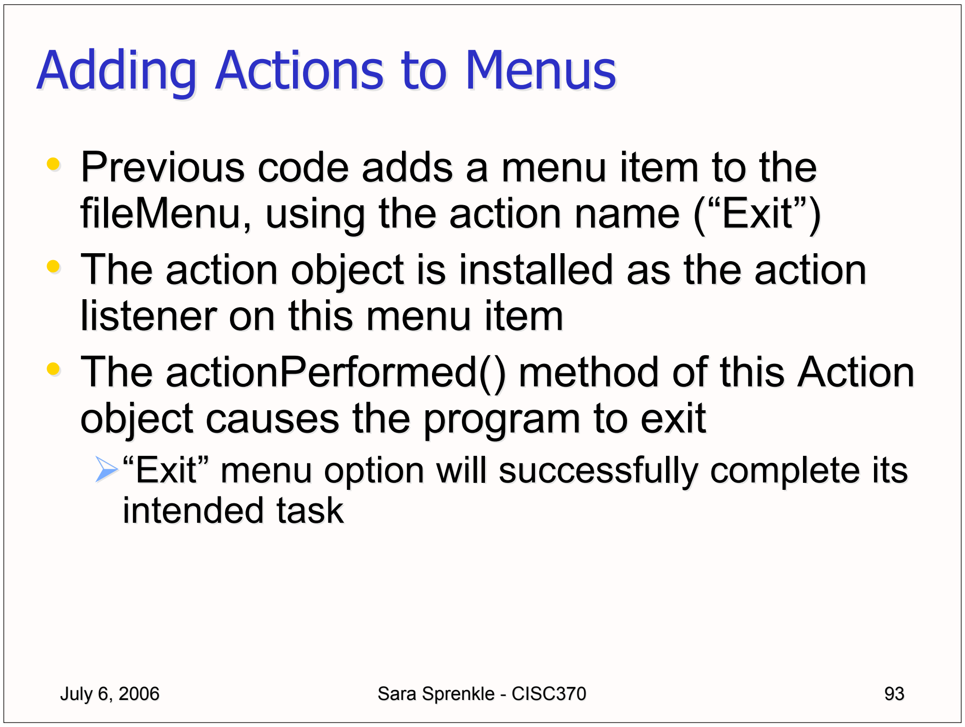
- If you have an Action object
  - can directly add a menu item for that action
  - Instead of
    - making a menu item
    - making a listener
    - installing event handling code in your listener

Adding Actions to Menus

```
Action exitAction = new AbstractAction(“Exit”)
  {
      public void actionPerformed(ActionEvent evt)
      {
             System.exit(0);
      }
  };
```

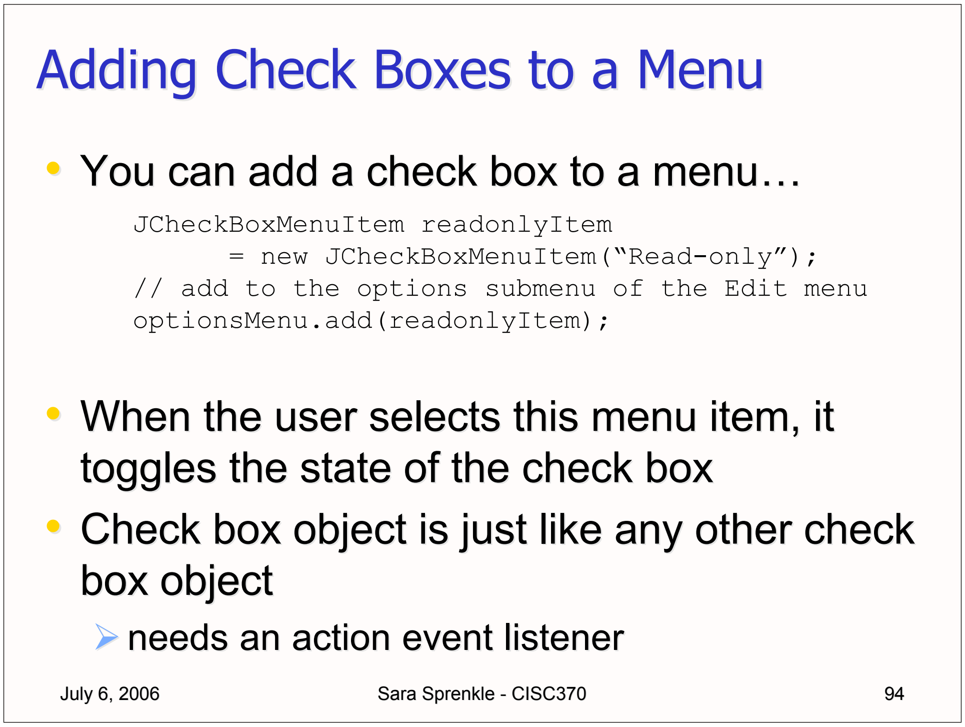
- Then, add the action to the menu…

```
JMenuItem exitItem = new JMenuItem(exitAction);
fileMenu.add(exitItem);

// OR (in a different form . . .)

fileMenu.add( new JMenuItem(exitAction) );
```



Adding Actions to Menus

- Previous code adds a menu item to the fileMenu, using the action name (“Exit”)
- The action object is installed as the action listener on this menu item
- The actionPerformed() method of this Action object causes the program to exit
  - “Exit” menu option will successfully complete its intended task



Adding Check Boxes to a Menu

- You can add a check box to a menu…

```
JCheckBoxMenuItem readonlyItem
      = new JCheckBoxMenuItem(“Read-only”);
// add to the options submenu of the Edit menu
optionsMenu.add(readonlyItem);
```

- When the user selects this menu item, it toggles the state of the check box
- Check box object is just like any other check box object
  - needs an action event listener

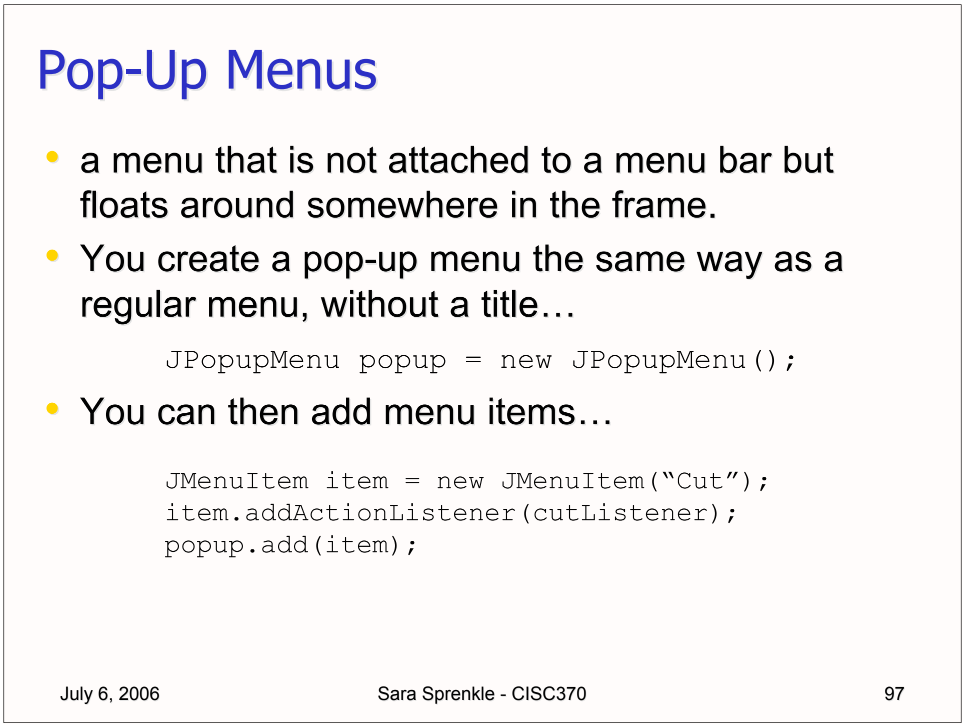
Adding Radio Buttons to a Menu

- You can also add a set of radio buttons to a menu…

```
ButtonGroup insertModeGroup = new ButtonGroup();
JRadioButtonMenuItem insertItem =
      new JRadioButtonMenuItem(“Insert Mode”);
JRadioButtonMenuItem overwriteItem =
      new JRadioButtonMenuItem(“Overwrite Mode”);
insertItem.setSelected(true);
insertModeGroup.add(insertItem);
insertModeGroup.add(overwriteItem);
optionsMenu.add(insertItem);
optionsMenu.add(overwriteItem);
```

Check Boxes and Radio Buttons

- With these types of menu items, you don’t necessarily want to be notified whenever the user changes their state.
- Use the isSelected() method of either of these two classes to see if the check box is checked or the particular radio button is selected
  - whenever your program needs to know these current values.
- Change the state of these menu items using the setSelected() method



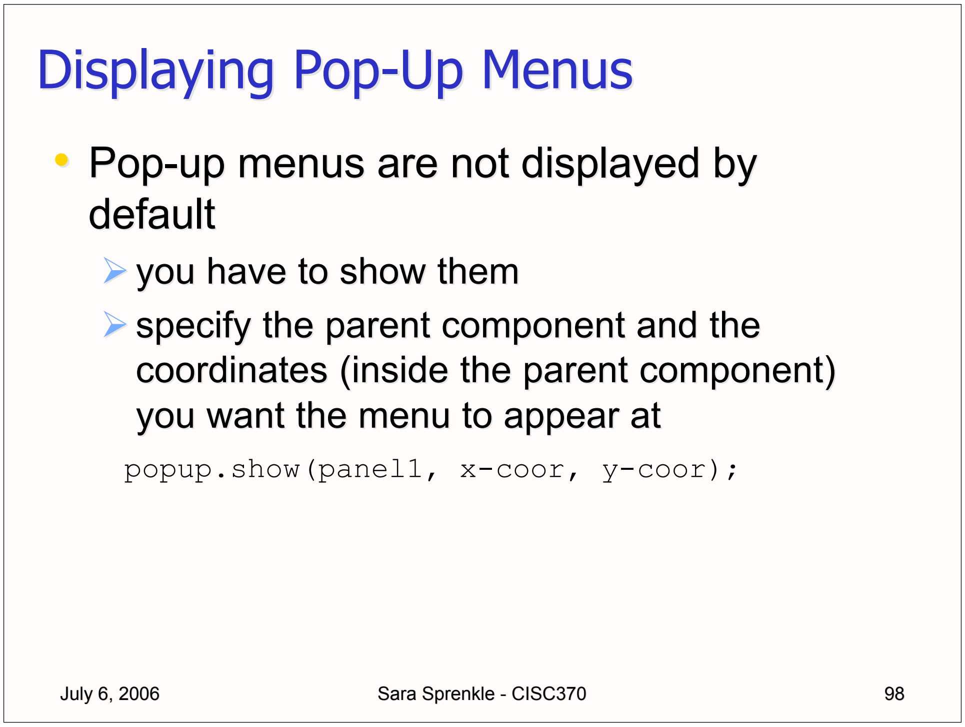
Pop-Up Menus

- a menu that is not attached to a menu bar but floats around somewhere in the frame.
- You create a pop-up menu the same way as a regular menu, without a title…

```
JPopupMenu popup = new JPopupMenu();
```

- You can then add menu items…

```
JMenuItem item = new JMenuItem(“Cut”);
item.addActionListener(cutListener);
popup.add(item);
```



Displaying Pop-Up Menus

- Pop-up menus are not displayed by default
    - you have to show them
    - specify the parent component and the coordinates (inside the parent component) you want the menu to appear at

```
popup.show(panel1, x-coor, y-coor);
```

Pop-Up Triggering

- You can also configure a pop-up menu to appear when the user hits a *pop-up trigger*
  - ➢ Windows: the right mouse button
  - ➢ Mac: Control Key
- Install a MouseListener and add code…

```
public void mouseReleased(MouseEvent evt)
{
        if (evt.isPopupTrigger())
                popup.show(evt.getComponent(),
                        evt.getX(), evt.getY());
}
```

Key Mnemonics

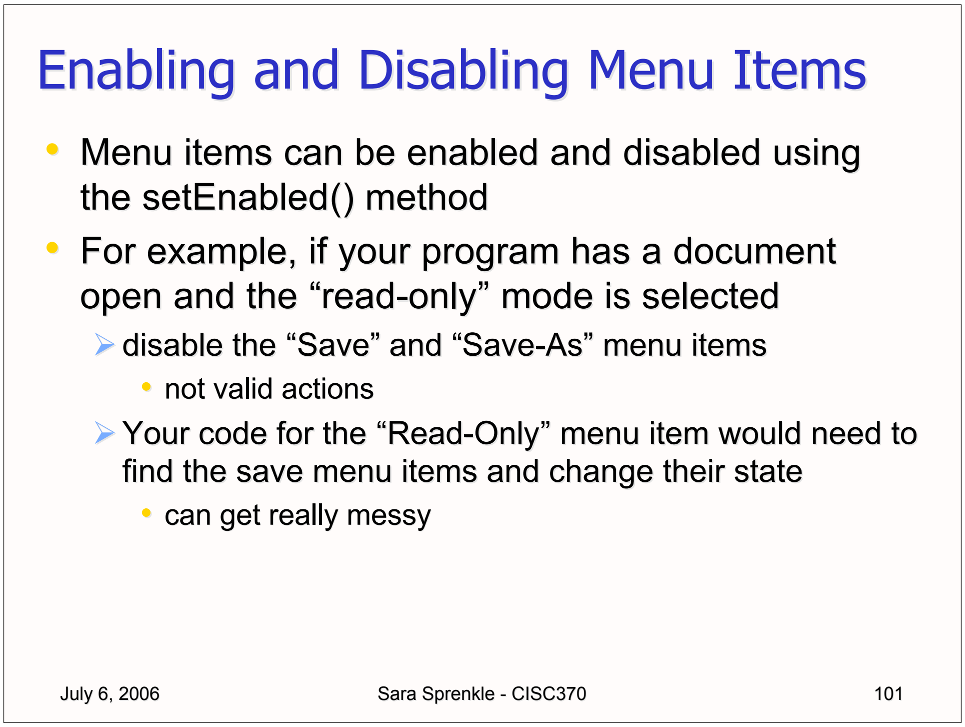
- Each menu item can have a mnemonic
  - ➢ when the menu is selected pressing that mnemonic causes its menu item to be selected…

```
JMenuItem cutItem = new JMenuItem(“Cut”, ‘T’);
```

- Menus can also have mnemonics
  - ➢ can set them in constructor

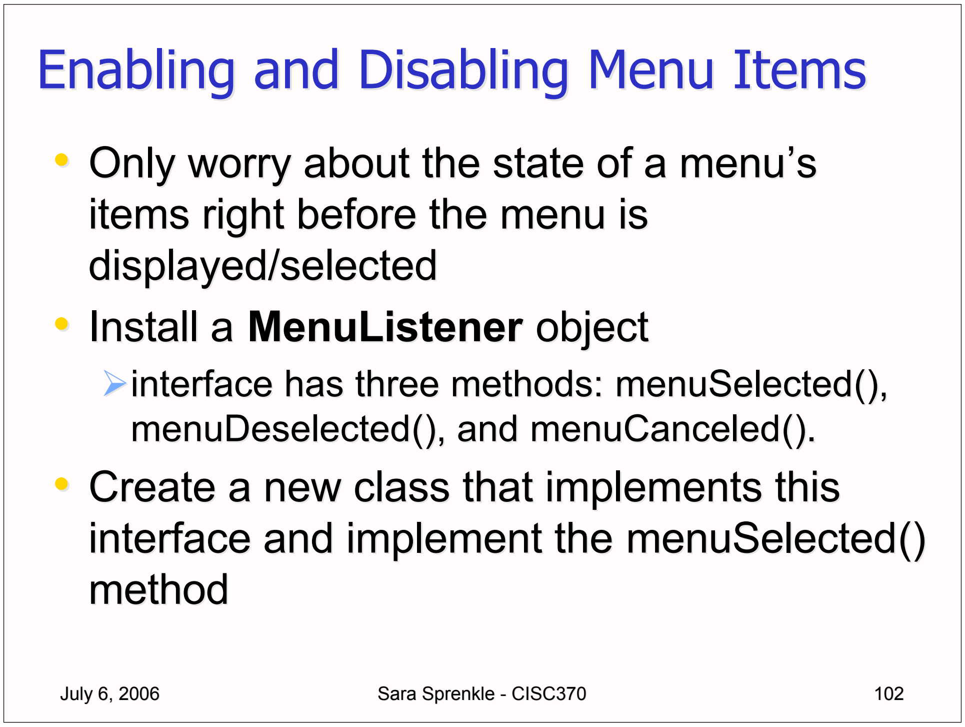
```
JMenu helpMenu = new JMenu(“Help”);
helpMenu.setMnemonic(‘H’);
```

- For top-level menus, pressing <ALT> and the mnemonic will cause that menu to be selected



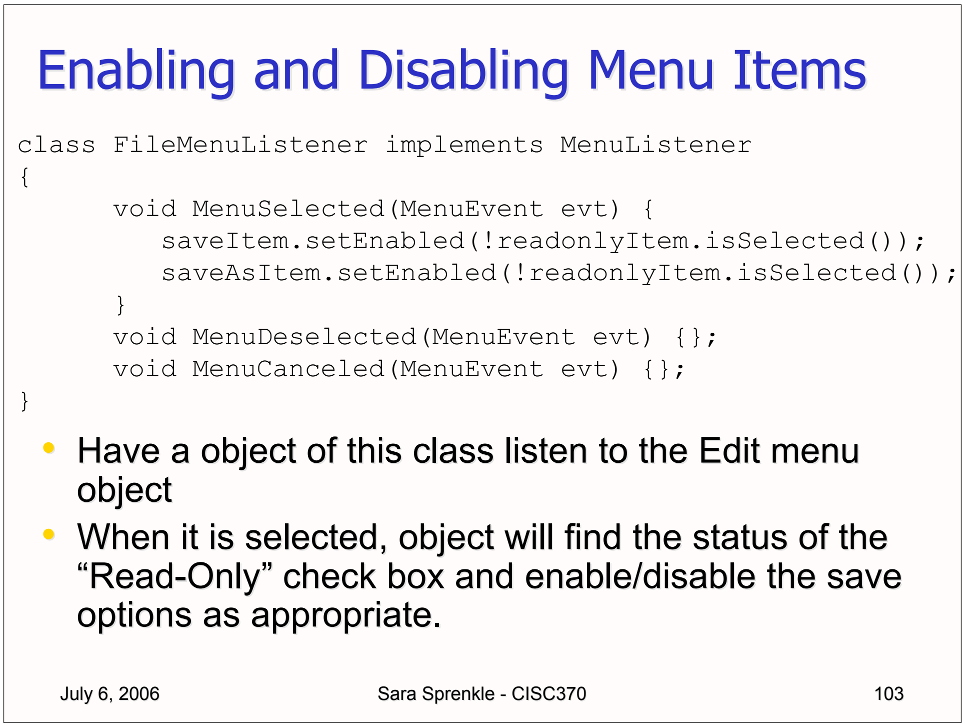
Enabling and Disabling Menu Items

- Menu items can be enabled and disabled using the setEnabled() method
- For example, if your program has a document open and the “read-only” mode is selected
  - disable the “Save” and “Save-As” menu items
    - not valid actions
  - Your code for the “Read-Only” menu item would need to find the save menu items and change their state
    - can get really messy



Enabling and Disabling Menu Items

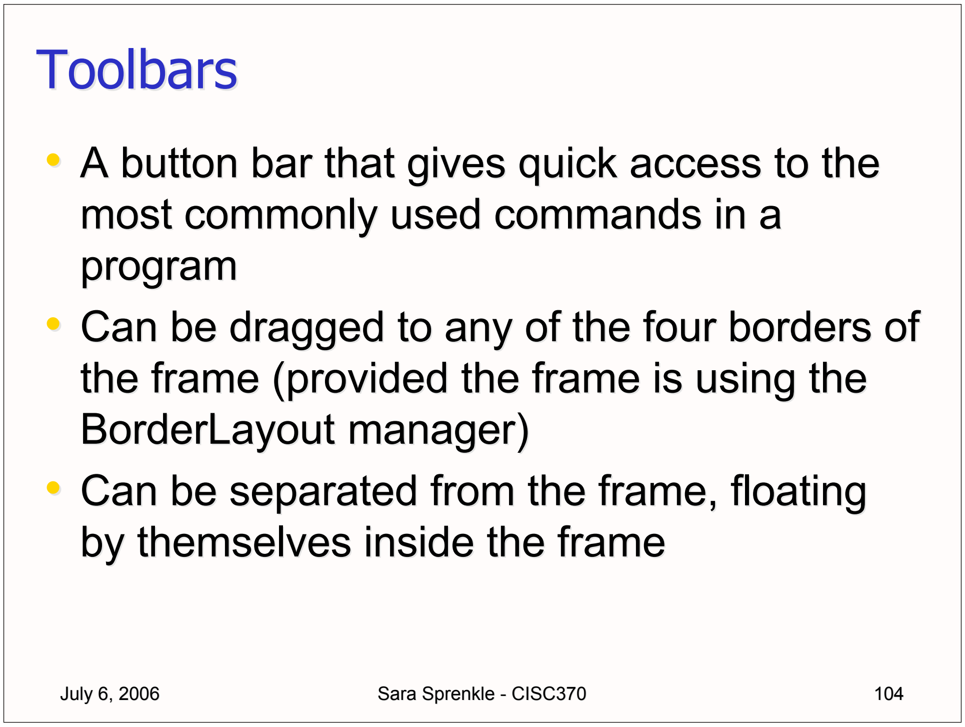
- Only worry about the state of a menu’s items right before the menu is displayed/selected
- Install a **MenuListener** object
  - interface has three methods: menuSelected(), menuDeselected(), and menuCanceled().
- Create a new class that implements this interface and implement the menuSelected() method



Enabling and Disabling Menu Items

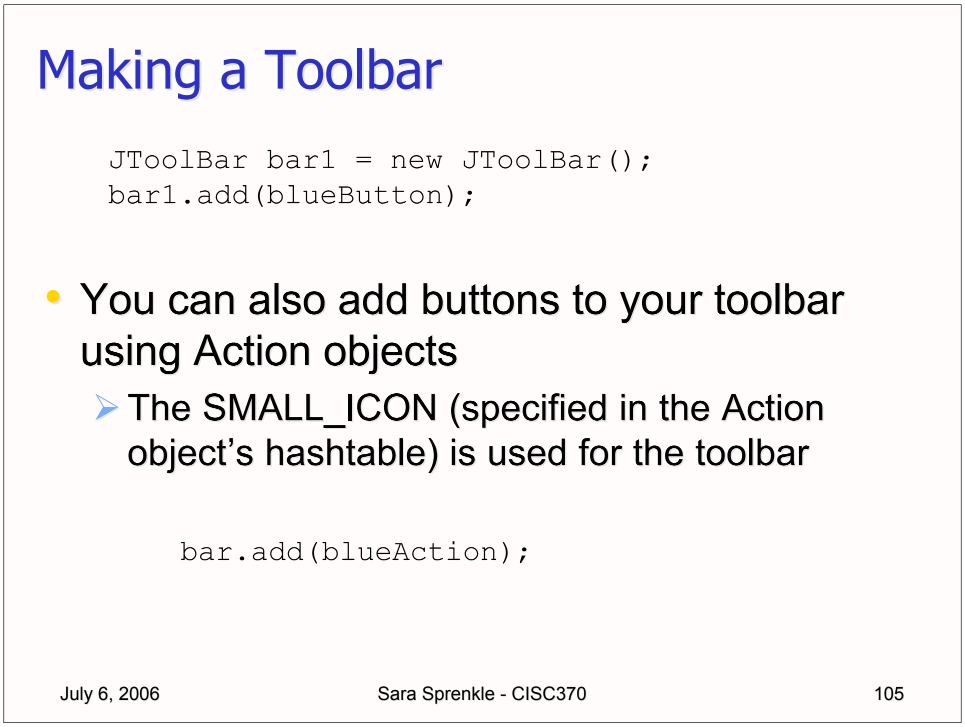
```
class FileMenuListener implements MenuListener
{
      void MenuSelected(MenuEvent evt) {
         saveItem.setEnabled(!readonlyItem.isSelected());
         saveAsItem.setEnabled(!readonlyItem.isSelected());
      }
      void MenuDeselected(MenuEvent evt) {};
      void MenuCanceled(MenuEvent evt) {};
}
```

- Have a object of this class listen to the Edit menu object
- When it is selected, object will find the status of the “Read-Only” check box and enable/disable the save options as appropriate.



Toolbars

- A button bar that gives quick access to the most commonly used commands in a program
- Can be dragged to any of the four borders of the frame (provided the frame is using the BorderLayout manager)
- Can be separated from the frame, floating by themselves inside the frame

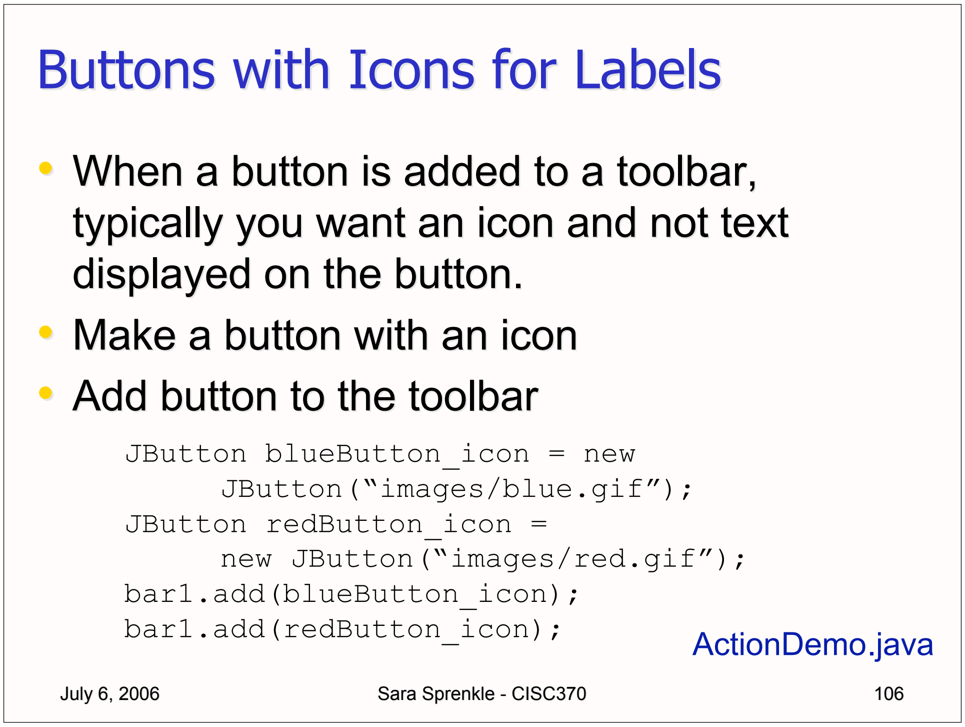


Making a Toolbar

```
JToolBar bar1 = new JToolBar();
bar1.add(blueButton);
```

- You can also add buttons to your toolbar using Action objects
  - The SMALL_ICON (specified in the Action object’s hashtable) is used for the toolbar

```
bar.add(blueAction);
```

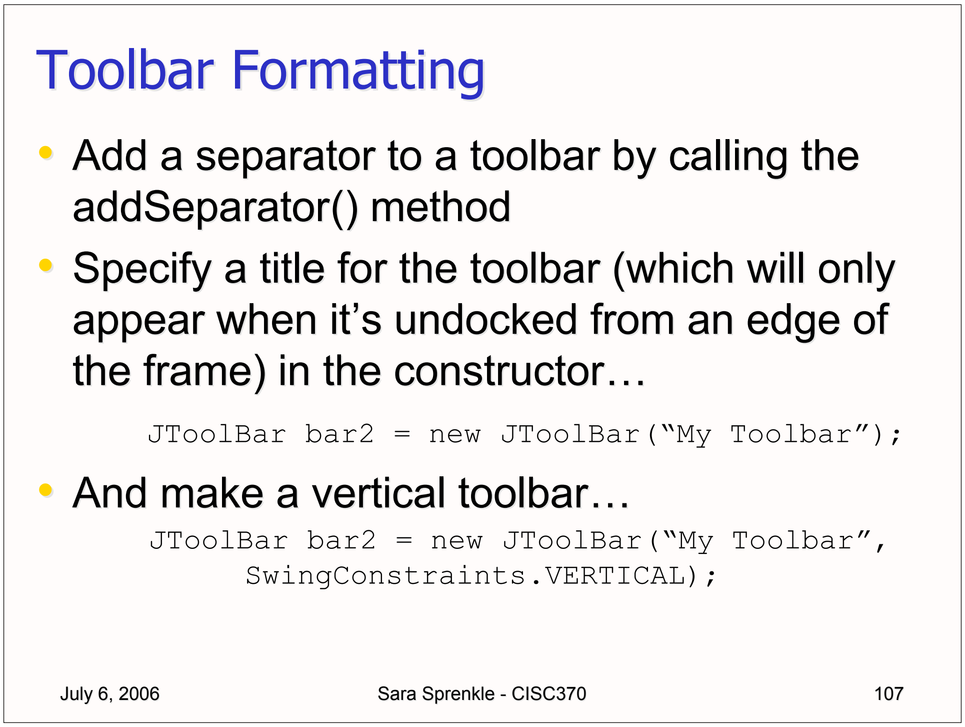


Buttons with Icons for Labels

- When a button is added to a toolbar, typically you want an icon and not text displayed on the button.
- Make a button with an icon
- Add button to the toolbar

```
JButton blueButton_icon = new
      JButton(“images/blue.gif”);
JButton redButton_icon =
      new JButton(“images/red.gif”);
bar1.add(blueButton_icon);
bar1.add(redButton_icon);
```

ActionDemo.java



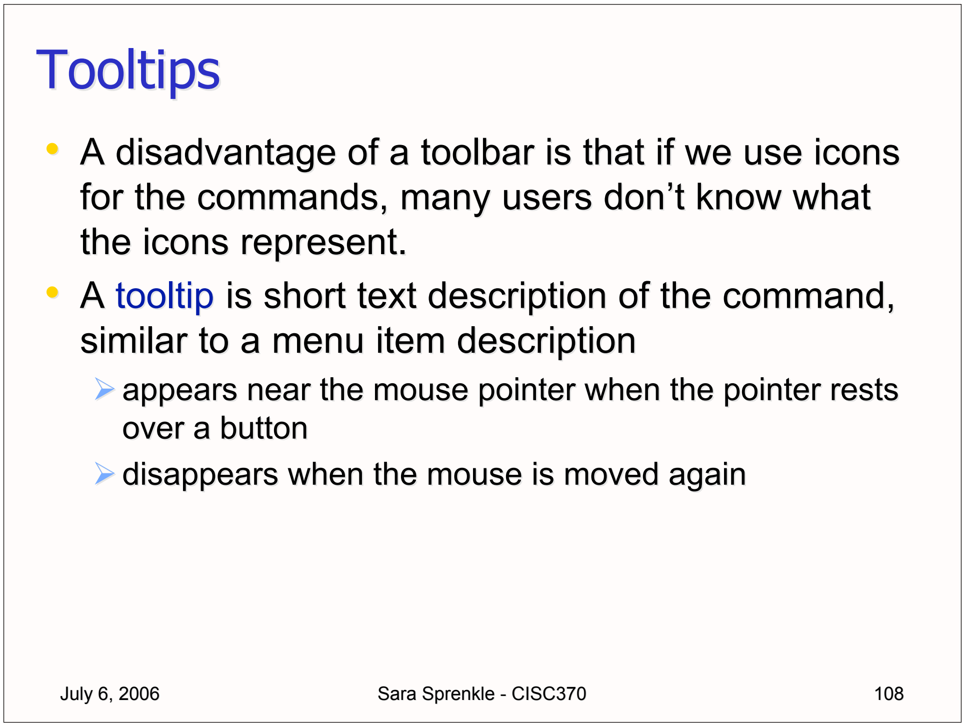
Toolbar Formatting

- Add a separator to a toolbar by calling the addSeparator() method
- Specify a title for the toolbar (which will only appear when it's undocked from an edge of the frame) in the constructor…

```
    JToolBar bar2 = new JToolBar(“My Toolbar”);
```

- And make a vertical toolbar…

```
    JToolBar bar2 = new JToolBar(“My Toolbar”,
         SwingConstraints.VERTICAL);
```



Tooltips

- A disadvantage of a toolbar is that if we use icons for the commands, many users don't know what the icons represent.
- A tooltip is short text description of the command, similar to a menu item description
  - appears near the mouse pointer when the pointer rests over a button
  - disappears when the mouse is moved again

Adding Tooltips

- Any component (not just buttons) can have a tooltip
- Call the JComponent method setTootTipText()

```
    exitButton.setToolTipText(“Exit”);
```

- For an Action object, set the SHORT_DESCRIPTION field in the hashtable
  - used as the tooltip

```
exitAction.putValue(Action.SHORT_DESCRIPTION, “Exit”);
```

A Visual Index to Swing Components

http://java.sun.com/docs/books/tutorial/uiswing/components/components.html