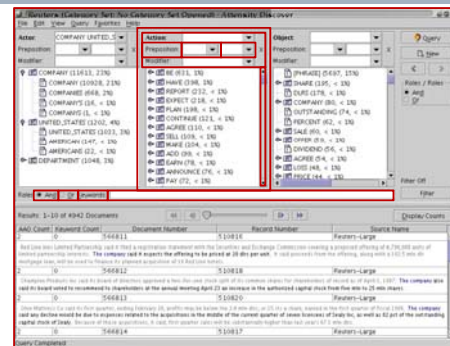


Swing: Components for Graphical User Interfaces

Lecture 23

GUI: A Hierarchy of Nested Widgets



Visual (Containment) Hierarchy

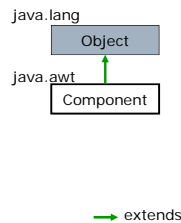
- Top-level widgets: outermost window (a container)
 - Frame, applet, dialog
- Intermediate widgets: allow nesting (a container)
 - General purpose
 - Panel, scroll pane, tabbed pane, tool bar
 - Special purpose
 - Layered pane
- Atomic widgets: nothing nested inside
 - Basic controls
 - Button, list, slider, text field
 - Uneditable information displays
 - Label, progress bar, tool tip
 - Interactive displays of highly formatted information
 - Color chooser, file chooser, tree
- For a visual ("look & feel") of widgets see:
 - <http://java.sun.com/docs/books/tutorial/uiswing/components>
- Vocabulary: Widgets usually referred to as "GUI components" or simply "components"

History

- Java 1.0: AWT (Abstract Window Toolkit)
 - Platform-dependent implementations of widgets
- Java 1.2: Swing
 - Most widgets written entirely in Java
 - More portable
- Main Swing package: javax.swing
 - Defines various GUI widgets
 - Extensions of classes in AWT
 - Many class names start with "J"
 - Includes 16 nested subpackages
 - javax.swing.event, javax.swing.table, javax.swing.text...
- Basic GUI widgets include
 - JFrame, JDialog
 - JPanel, JScrollPane, JTabbedPane, JToolBar
 - JButton, JRadioButton, JCheckBox, JPasswordField, JSlider
 - JLabel, JToolTip
 - JColorChooser, JFileChooser

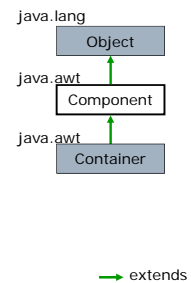
Class Hierarchy: Component

- A *component* is an object having a graphical representation that can be displayed on the screen and that can interact with the user.
- Operations common to nonmenu-related GUI widgets
 - More than 60 (public) methods!
- Drawing the widget
 - paint(): draw the whole widget
 - repaint(): schedule the widget to be redrawn, will result in framework calling...
 - update(): modifies part of widget, or just calls paint() for full refresh
- Appearance of widget
 - setVisible(): determine whether widget will be visible on screen
 - setLocation()
- Dealing with user events



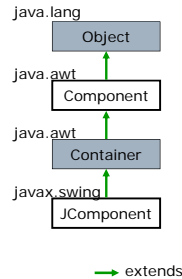
Class Hierarchy: Container

- A widget that can include other widgets
 - Visual nesting
- Contained widgets are called "children"
 - But not children as in behavioral subtypes
- Methods for managing contained widgets
 - add: adds widgets to container
 - setLayout: specifies the layout manager that helps container position and size contained widgets

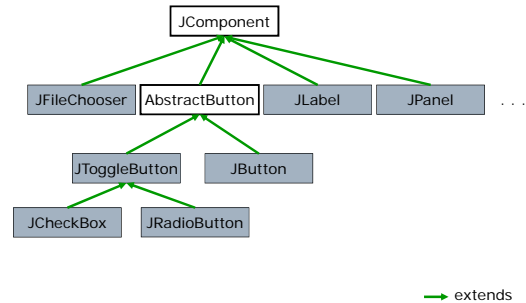


Basic Hierarchy: JComponent

- Base class for all Swing widgets, except top-level containers (ie applet, frame, dialog)



Part of JComponent Hierarchy



JLabel

- A JLabel object provides text instructions or information on a GUI
 - Displays a single line of *read-only* text, an image, or both
- See
 - Example code
 - Output
- One thing to be emphasized:
 - If you do not *explicitly add* a widget to a container, the widget will not be displayed when the container appears on the screen

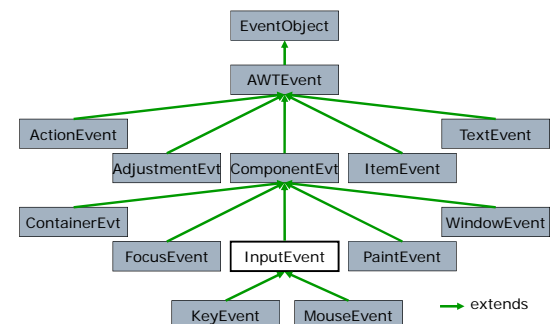
An Interactive GUI Component

- To make an interactive GUI program, you need:
 - Widgets (ie GUI components)
 - Buttons, windows, menus, etc.
 - Events
 - Mouse clicked, window closed, button clicked, etc.
 - Event listeners (implement an interface) and event handlers (methods)
 - Listen for events to be triggered, and then perform actions to handle them

Handling Events

- GUI is *event driven*
- Event handling occurs as a loop:
 - GUI program is idle
 - User performs an action, for example:
 - Moving the mouse, clicking a button, closing a window, typing in a text box, selecting an item from a menu, ...
 - Such an action generates an event
 - The event is sent to the program, which responds
 - Code executes, GUI updates
 - GUI program returns to being idle
- Many event types defined in `java.awt.event` and `javax.swing.event`

Part of AWTEvent Hierarchy



Handling Events Mechanism

- Three parts of the event-handling mechanism
 - *Event source*: the widget with which the user interacts
 - *Event object*: encapsulated information about the occurred event
 - *Event listener*: an object that is notified by the event source when an event occurs, and responds to the event



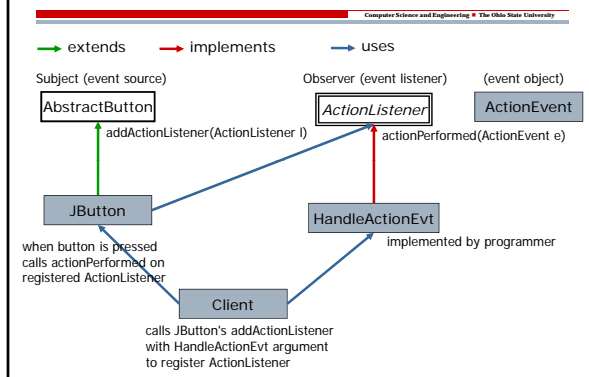
Programmer Tasks

- Implement an event listener
 - A class X that implements one (or more) of the event listener interfaces


```
interface ActionListener {
    void actionPerformed (ActionEvent e);
}
interface FocusListener {
    void focusGained (FocusEvent e);
    void focusLost (FocusEvent e);
}
```
- Register an instance of X with widget
 - java.awt's Component, Container, etc. have methods for adding listeners


```
void addFocusListener (FocusListener f)
```

Observer Pattern



JTextField and JPasswordField

- Single-line areas for text
 - Can be editable (user enters text from keyboard) or not
 - Password field does not show individual characters
- When the user types data into them and presses the Enter key:
 - An event occurs (ActionEvent)
 - All registered listeners (ActionListeners) receive the event
 - Argument to method actionPerformed includes text from field
- See:
 - Example code
 - Output

Buttons

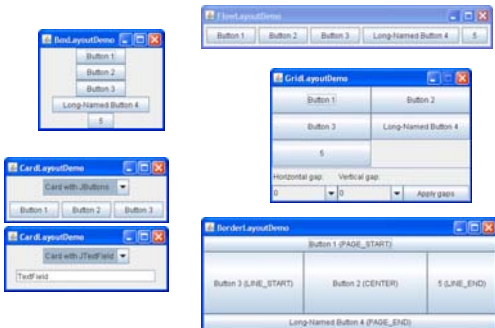
- A button is a clickable widget
- There are several types of buttons, all are subclasses of AbstractButton
 - Command button:
 - Class JButton, generates ActionEvent
 - Toggle button:
 - Has on/off or true/false values
 - Check boxes:
 - A group of buttons in which more than one can be selected, generates ItemEvent
 - Radio buttons:
 - A group of buttons in which only one can be selected, generates ItemEvent
- See:
 - Example code
 - Output

More Widgets...

- JComboBox:
 - A drop-down list from which the user can make a selection
 - Generates an ItemEvent
- JList:
 - A list supporting both single and multiple selection
 - Generates a ListSelectionEvent

Layout Management

Computer Science and Engineering @ The Ohio State University



Layout Management

Computer Science and Engineering @ The Ohio State University

- Layout refers to how components are arranged in the container



- This positioning is determined by a layout manager
 - Buttons in the above example are managed by the flow layout manager, which is the default layout manager for a panel
 - The default manager lines the components horizontally until there is no more room and then start a new row of components
 - After resizing the container, the layout manager reflows the components automatically
 - The default is to center the components in each row, but this can be overridden with left or right alignment
- Other managers: for a visual ("look & feel") index see <http://java.sun.com/docs/books/tutorial/uiswing/layout/visual.html>

Layout Management with Panels

Computer Science and Engineering @ The Ohio State University

- Problem with BorderLayout:
 - The button is stretched to fill the entire southern region of the frame
 - If you add another button to the southern region, it just displaces the first button
- Solution: use additional panels
 - Act as containers for interface elements and can themselves be arranged inside a larger panel
 - Use flow layout by default
- To fix the BorderLayout problem
 - Create a new panel
 - Add each element to the panel
 - Add the panel to the larger container



```

JPanel p = new JPanel();
p.add(button1);
p.add(button2);
p.add(button3);
frame.add(panel,
    BorderLayout.PAGE_END);
    
```

Supplemental Reading

Computer Science and Engineering @ The Ohio State University

- A visual index to the Swing Components
 - <http://download.oracle.com/javase/tutorial/uiswing/components/>
- Creating a GUI with JFC/Swing
 - <http://download.oracle.com/javase/tutorial/uiswing/index.html>
- Building a User Interface
 - <http://java.sun.com/new2java/divelog>
 - <http://www.oracle.com/technetwork/articles/javase/index-142890.html>

Summary

Computer Science and Engineering @ The Ohio State University

- Containment hierarchy
 - Containers (frame, applet, dialog)
 - Components (panel, scroll pane, tabbed pane,...)
 - Controls (button, text field, label,...)
- Event-driven programming
 - Register handlers with components
 - Events are passed from components to handlers
- Layout
- Look and feel?