

# Swing: Components for Graphical User Interfaces

## Lecture 9

## Overview

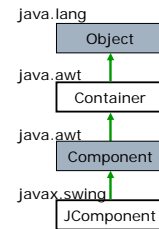
- Classes from package `javax.swing` defines various GUI components — objects with which the user interacts via the mouse, the keyboard or another form of input
  - Total of 17 packages (`javax.swing.event`, `javax.swing.table`,...)
- Basic GUI components include
  - JLabel
  - JTextField
  - JCheckBox
  - JComboBox
  - JList
  - JPanel
- Most of the swing components are written completely in `java`, so they provide a greater portability and flexibility than the original GUI components from package `java.awt`
  - Awt components are platform dependent
  - Some swing components are still platform dependent. E.g, `JFrame`

## Containment Hierarchy

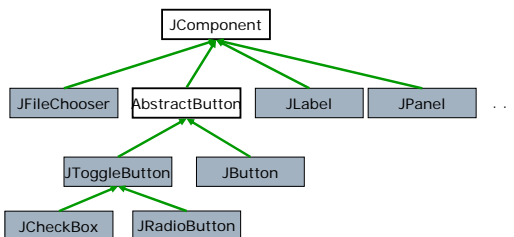
- Top-level containers
  - Applet, dialog, frame
- Intermediate components
  - General purpose
    - Panel, scroll pane, tabbed pane, tool bar
  - Special purpose
    - Layered pane
- Atomic components
  - Basic controls
    - Button, list, slider, text field
  - Uneditable information displays
    - Label, progress bar, tool tip
  - Interactive displays of highly formatted information
    - Color chooser, file chooser, tree
- See: <http://java.sun.com/docs/books/tutorial/uiswing/components> for a visual index

## Basic Hierarchy: JComponent

- Component class
  - Operations common to most GUI components
- Container class
  - `add`: adds components to container
  - `setLayout`: specifies the layout manager that helps container position and size contained components
- JComponent class
  - Base class for all swing components, *except* top-level containers (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 GUI component to a container, the GUI component will not be displayed when the container appears on the screen

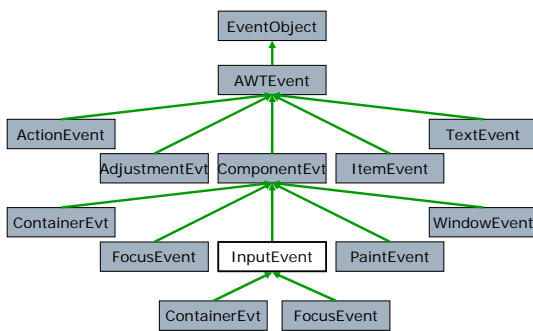
## An Interactive GUI Component

- To make an interactive GUI program, you need:
  - 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 GUI component 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 provides 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 component
  - `java.awt.Container` has methods for adding listeners
  - `void addFocusListener (FocusListener f)`

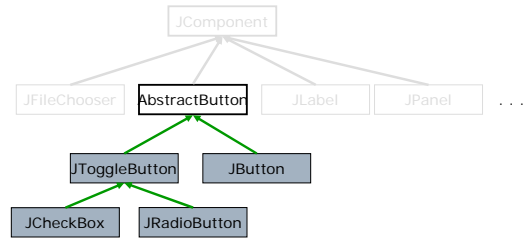
## JTextField and JPasswordField

- 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

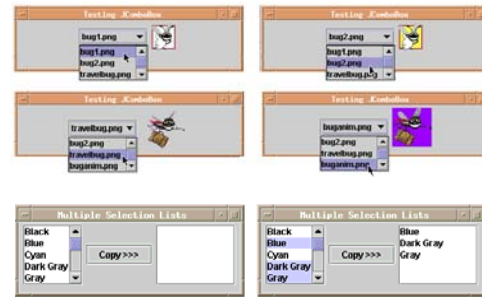
- A button is a component the user clicks to trigger a specific action
- There are several types of buttons, all are subclasses of AbstractButton
  - Command button:
    - class JButton, generates ActionEvent
  - Toggle button:
    - has on/off or true/false values
  - Check boxes:
    - a group of buttons 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

## Part of JComponent Hierarchy



## More Components...

- 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

- 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
    - panel.setLayout(new FlowLayout(FlowLayout.LEFT));
- Other managers: see <http://java.sun.com/docs/books/tutorial/uiswing/layout/visual.html>

## Layout Management with Panels

- 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
  1. Create a new panel
  2. Add each element to the panel
  3. Add the panel to the larger container



```

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

## Supplemental Reading

Computer Science and Engineering @ The Ohio State University

- A visual Index to the Swing Components
  - <http://java.sun.com/docs/books/tutorial/uising/components/components.html>
- Creating a GUI with JFC/Swing
  - <http://java.sun.com/docs/books/tutorial/uising/index.html>
- Building a User Interface
  - <http://java.sun.com/developer/onlineTraining/new2java/divelog/part1/>
  - and part2, part3, ..., part5

## Supplemental Reading

Computer Science and Engineering @ The Ohio State University

- A visual Index to the Swing Components
  - <http://java.sun.com/docs/books/tutorial/uising/components/components.html>
- **Creating a GUI with JFC/Swing**
  - <http://java.sun.com/docs/books/tutorial/uising/index.html>
- **Building a User Interface**
  - <http://java.sun.com/developer/onlineTraining/new2java/divelog/part1/>
  - <http://java.sun.com/developer/onlineTraining/new2java/divelog/part2/index.jsp>
  - <http://java.sun.com/developer/onlineTraining/new2java/divelog/part3/>
  - <http://java.sun.com/developer/onlineTraining/new2java/divelog/part4/>
  - <http://java.sun.com/developer/onlineTraining/new2java/divelog/part5/>