Object-oriented Software Design and Development
CCP114N

Week 8

Building Graphical User Interfaces in Java

- Steps to create a GUI
- Swing GUI components
- Event handling

Software Design Pattern

- Observer, Composite, Strategy, Decorator

An example of Swing application

- You have used some Swing components in previous exercises
  - firstNumber = JOptionPane.showInputDialog("Enter first integer:");
  - JOptionPane.showMessageDialog(null, result, "Comparison Results", JOptionPane.INFORMATION_MESSAGE);
- Swing components examples
  - Run the applications and take a closer look at what are the GUI components in there.

Event-driven programming

Event-driven programming vs. imperative programming

- An ‘imperative’ program interacts with the data (primitives and objects) through control structures such as IF...THEN, WHILE..., SWITCH..., FOR..., etc. The entire program's flow is defined by these structures before hand.
- In even-driven programming: instead of tightly controlling the run of a program, the software just sits idle until the user does something. In other words, the flow of the program is controlled by user-generated **events**.
  - The meat of event-driven programming is in the event-handlers. These are code in the software that are called in response to certain events
  - The most important conceptual feature of even-driven programming is that the user is in control of your code.

Four basic steps to create a GUI (1/2)

- *Check one of the examples to identify the steps below.*
- (1) Create and configure the components
  - using their constructors as they are all objects in Java
  - setting their properties
- (2) Add the components to a container
  - Components must be placed in a container
    - Top-level containers are: JApplet, JFrame and JDialog
    - Important one: JPanel can be placed in other container e.g. JFrame

Four basic steps to create a GUI (2/2)

- (3) Arrange, or layout, the components
  - to set positions and sizes
  - layout manager objects (again, they are objects in Java)
  - A layout manager object is then used to set a layout of a container e.g.
    - northPanel.setLayout(new BorderLayout());
  - there exist several layout manager objects in Java
- (4) handle the events generated by the components
  - event object contains info about user interaction
  - event listener object is notified and respond in a way that was coded.

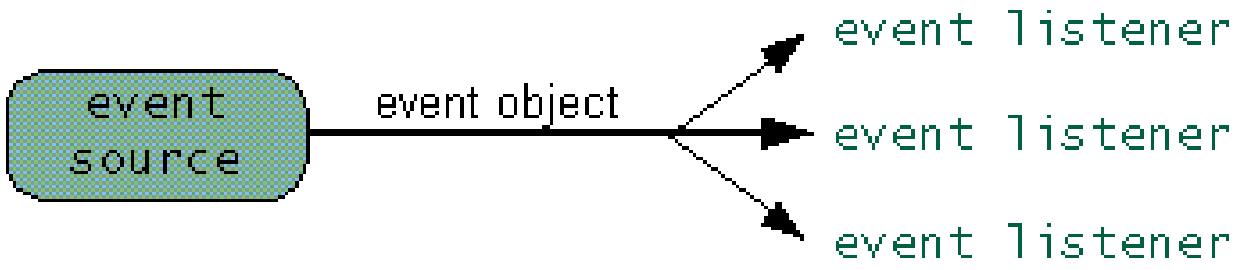
An overview of Swing components

Source: The Sun Java Tutorial at
http://java.sun.com/docs/books/tutorial/uiswing/components/components.html

- Top-Level Containers
  - The components at the top of any Swing containment hierarchy.
- General-Purpose Containers
  - Intermediate containers that can be used under many different circumstances.
- Special-Purpose Containers
  - Intermediate containers that play specific roles in the UI.
- Basic Controls
  - Atomic components that exist primarily to get input from the user; they generally also show simple state.
- Uneditable Information Displays
  - Atomic components that exist solely to give the user information.
- Editable Displays of Formatted Information
  - Atomic components that display highly formatted information that (if you choose) can be edited by the user.
- NB: the containment hierarchy of your GUI

Event handling in Java (1/3)

- Every time the user interacts with a GUI, an event occurs
- Any object can be notified of the event
- Response to the event is coded in an event handler.



Event handling in Java (2/3)

- *Event source*: any GUI component such as buttons, Textfield, Label, listbox, combobox (drop-down box) etc.
- *Event type*: they are classes in Java
  - KeyEvent, MouseEvent, ActionEvent, WindowEvent, FocusEvent, MenuEvent, CareEvent
- *Event listener*: those objects want to capture an event and respond to it
  - They must be registered with the event's source e.g. someComponent.addActionListener(instanceOfMyClass);
- *Event handler*: methods of a class / an object that implements *an event listener's interface* or override methods of *an event adapter's class*
  - E.g. WindowListener is an interface that specifies the methods of the WindowEvent Listener, while WindowAdapter is a class that implements dummy methods of the WindowListener interface

Event handling in Java (3/3)

- Methods to implement an event handler
  - To implement a listener interface, all methods must be implemented
    - public class myClass implements ActionListener
    - BTW, ActionListener interface has only 1 method actionPerformed
  - To extend an event adapter class: overriding only required methods

```
public class myClass extends WindowAdapter {
        public void WindowClosing(WindowEvent e) {
        System.exit(0); } }
```

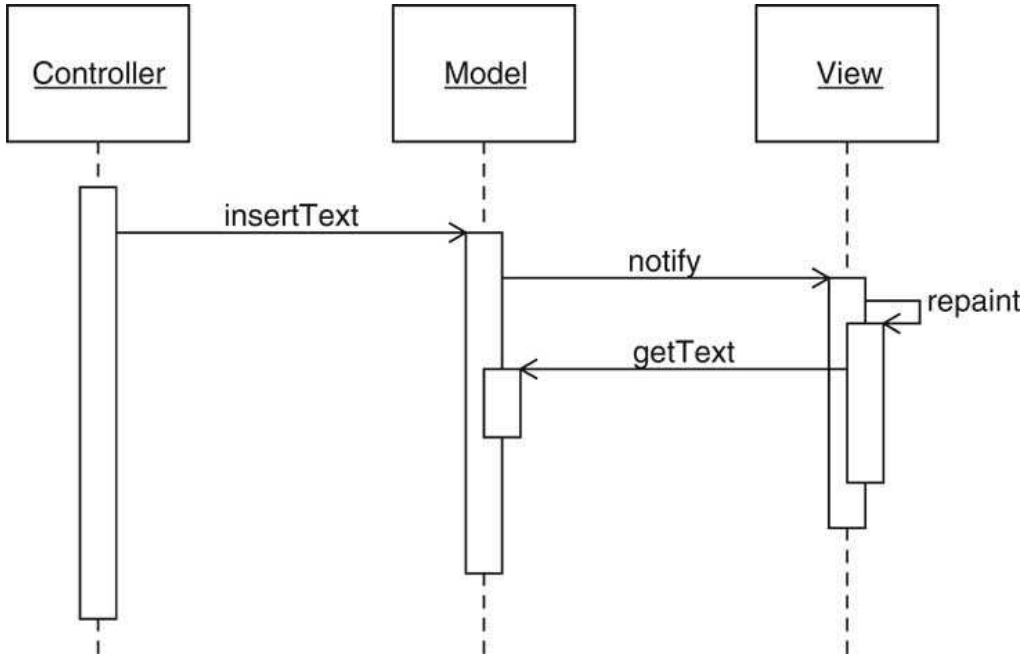
  - To use inner class: a class defined within another class
    - See examples in FiveGUIcompsExamples (downloadable from w8 notes)
- Implementing Listeners for Commonly Handled Events
  - *http://java.sun.com/docs/books/tutorial/uiswing/events/handling.html*

Software pattern

- What is it?
  - A software design pattern is a description of a problem and its solution that may be applied to many programming situations.
  - Benefits
    - Save time and effort
    - Have proven (possiblly optimal or best) solution
    - Avoid possible errors (anti-pattern)
- Originated from practice
  - The book by Gang of Four
    - Design Patterns by Gamma et al.
    - A Google search returns a number of websites

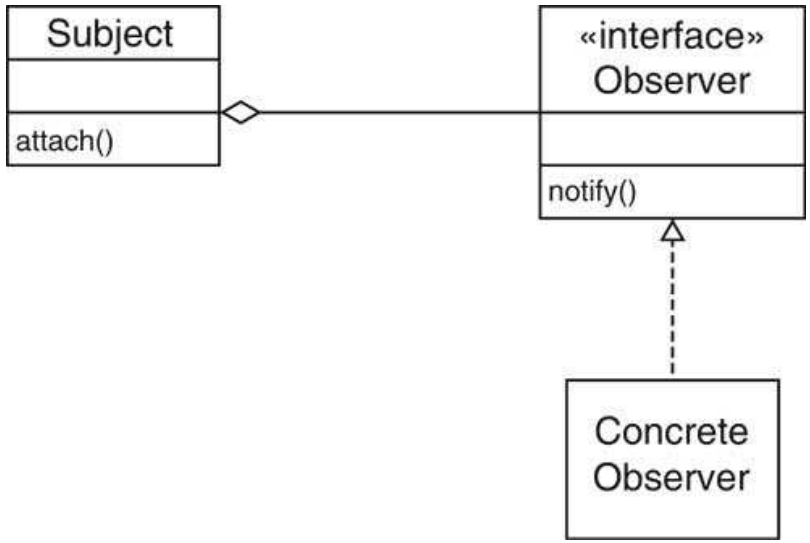
Observer pattern: Intent

- Model/View/Controller
  - Model: data structure, no visual representation
  - Views: visual representations
  - Controllers: user interaction
- Observer Pattern
  - Model notifies views when something interesting happens
  - Button notifies action listeners when something interesting happens
  - Views attach themselves to model in order to be notified
  - Action listeners attach themselves to button in order to be notified
  - Generalize: *Observers* attach themselves to *subject*.



Observer Pattern: Description

- **Context**
  1. An object, called the subject, is source of events
  2. One or more observer objects want to be notified when such an event occurs.
- **Solution**
  1. Define an observer interface type. All concrete observers implement it.
  2. The subject maintains a collection of observers.
  3. The subject supplies methods for attaching and detaching observers.
  4. Whenever an event occurs, the subject notifies all observers.



Names in Observer Pattern: Example

- See the JButton Java example

Name in Design Pattern	**Actual Name (Swing buttons)**
Subject	JButton
Observer	ActionListener
ConcreteObserver	The class that implements the ActionListener interface type
attach()	addActionListener()
notify()	actionPerformed()

Other patterns

Please read Chapter 5, Horstmann book

- Strategy Pattern
  - E.g. Layout managers
- Composite Pattern
  - E.g. AWT Components, Swing JPanel
- Decorator Pattern
  - E.g. Scrollbar
- Iterator
  - The list Iterator in Java

How to Recognize Patterns?

- Look at the *intent* of the pattern
  - E.g. COMPOSITE has different intent than DECORATOR
- Remember common uses (e.g. STRATEGY for layout managers)
- Not everything that is strategic is an example of STRATEGY pattern
- Use context and solution as "litmus test“
  - Test every point of the pattern’s description

Summary

- An overview of building GUIs in Java
  - Steps to create a GUI
    - Components
    - Layout manager
    - Containment hierarchy
  - Event handling in Java
    - The event handling mechanism
    - Methods to implement an event handler
- Design pattern
  - Concept
  - Some examples