Java Swing, Events

Readings:
Just Java 2: Chap 19 & 21, or
Eckel’s Thinking in Java: Chap 14

Slide credits to CMPUT 301, Department of
Computing Science
University of Alberta

Java Foundation Classes

• JFC:
  – Abstract Window Toolkit (AWT)
    – original user interface toolkit
    – don’t go there!
  – Swing
    – package javax.swing.*, introduced in Java 1.2

Swing

• Portable API:
  – The appearance and behavior (look-and-feel) of the user interface components are implemented in Java …
    – might work slightly differently from any host platform
    – pluggable look-and-feels
e.g., Motif, windows,…

Containment Hierarchy

• Top-level container:
  – place for other Swing components to paint themselves
  – e.g., JFrame, JDialog, Japplet
• Intermediate container:
  – simplify positioning of atomic components
  – e.g., JPanel, JSplitPane, JTabbedPane

• Atomic components:
  – self-sufficient components that present information to and get input from the user
  – e.g., JButton, JLabel, JComboBox, JTextField, JTable

Swing

• Components and containers:
  – superclasses and interfaces
  – extends and implements

© O’Reilly 1999
Swing

- Java Documentation:
  - http://java.sun.com/java1.4
- SwingSet:
- Quick tutorial:
  - http://java.sun.com/docs/books/tutorial/uiswing/start/swingTour.html

Containers

- Notes:
  - Container objects group components, arranging them for display with a layout manager.

Top-Level Containers

- JFrame example:
  - contains a single component JRootPane, which has a JMenuBar (optional) and a content pane
  - theFrame.setJMenuBar(theMenuBar)
  - theFrame.setContentPane(thePanel)
  - add non-menu components to this content pane
  - theFrame.getContentPane().add(aButton)

Events

- Two approaches to event handling
  - read-evaluation loop (client-written loop)
  - notification-based (callbacks)
- Swing uses the 2nd approach

Events

- Swing:
  - objects communicate by "firing" and "handling" events (event objects)
  - (conventional method call)
  - events are sent from a single source object to one or more registered listener objects

Events

- Swing:
  - different event sources produce different kinds of events
  - e.g., a JButton object, when clicked, generates an ActionEvent object, which is handled by an ActionListener (an object whose class implements this interface)
Events

- Handling:
  - create a component
    - e.g., a JButton
  - add it to the GUI
    - e.g., to a JPanel
  - register a listener to be notified when the component generates an event
    - e.g., interface ActionListener
  - define the callback method
    - e.g., actionPerformed()

\[
\text{class MyListener implements ActionListener}{
  \text{public void actionPerformed( ActionEvent event ) {}}
  \text{// react to event}
  \text{}}
\]

Event Handling

- Options for implementing listeners:
  - listener class
  - anonymous inner classes
  - named inner classes

UML Sequence Diagram

Event Handling

- Listener class:
**Event Handling**

- Anonymous inner listener class:

  ```java
  public class MyButton extends JButton implements ActionListener {
  ...
  public MyButton() {
    addActionListener( this );
  }
  public void actionPerformed( ActionEvent event ) {
  ...
  }
  }
  ...
  JButton button = new MyButton(); ...
  ```

- Named inner listener class:

  ```java
  public class MyButton extends JButton implements ActionListener {
  ...
  public MyButton() {
    addActionListener( this );
  }
  public void actionPerformed( ActionEvent event ) {
  ...
  }
  }
  ```

- Note:
  - A class could potentially be both an event source and event listener.
  - Good or bad idea? …

**Dependencies**

- Problems:
  - need to maintain consistency in the views (or observers)
  - need to update multiple views of the common data model (or subject)
  - need clear, separate responsibilities for presentation (look), interaction (feel), computation, persistence
Model/View/Controller

- MVC roles:
  - model
    - complete, self-contained representation of object managed by the application e.g., spreadsheet document
    - provides a number of services to manipulate the data e.g., recalculate, save
    - computation and persistence issues – ...  

Model/View/Controller

- MVC roles:
  - view
    - tracks what is needed for a particular perspective of the data e.g., bar chart view
    - presentation issues
  - controller
    - gets input from the user, and uses appropriate information from the view to modify the model e.g., get slider value, trigger chart modify
    - interaction issues  

Model/View/Controller

- Separation:
  - you can modify or create views without affecting the underlying model
  - the model should not need to know about all the kinds of views and interaction styles available for it
  - separate threads?  

Model/View/Controller

- In Swing:
  - in practice, views and controllers are implemented with Swing components and listeners
  - both views and controllers will be dependent on Swing APIs

Model/View/Controller

- In Swing:
  - still, try to separate the model and its services so that it is Swing-free
  - model is like a "virtual machine" or "kernel" specific to the application
Model/View/Controller

- Smalltalk:
  - originated the MVC concept
  - integral support in interactive applications with MVC classes

Model/View/Controller

- Java and Swing:
  - concept is still valid to help structure interactive applications
    e.g., use a framework that supports MVC
  - Swing internally uses a variant of MVC for its pluggable look-and-feel capability ...

Pluggable Look-and-Feel

- Swing:
  - the look-and-feel is implemented in Java, but could mimic Windows, Motif, Classic, Aqua, etc.
    - UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
    - UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");

- Idea:
  - similar to skins, themes, schemes, etc., but must include “feel” as well as “look”

Pluggable Look-and-Feel

- Swing internals:
  - each component uses a user interface delegate object (responsible for view and controller roles)
Pluggable Look-and-Feel

- Swing internals:
  - each component specifies a model interface that an associated model class must implement

Model/View/Controller

- CRC cards for MVC:
  - discuss what models, views, and controllers there are in the system
  - be a design critic