Dependency Inversion

Layering helps us to structure dependencies between subsystems.

One problem with layering is that many situations require up-calls from lower to higher layers. How do we allow up-calls without creating dependencies between lower and higher layers? **"I want to call you, but I don't want to depend on you."**

This problem is solved using a technique called "dependency inversion".

Examples: (examples.vsd)

1) URLFetcher progress notifications
2) Chess pawn promotion

General dependency inversion class diagram.

Observer

Slides

Model-View-Controller

Having studied Layering, Dependency Inversion, and Observer, we are prepared to discuss another important principle of software design: **Separation of the user interface from the core object model**.

Core model represents the central functionality of the system (data + algorithms).

The user interface's job is to let the user browse and edit the underlying information, and to invoke underlying functionality.

It is very important to keep the details of the user interface out of the core model, for several reasons:

1) Separation of concerns
2) There may be a need to support multiple different user interfaces (desktop GUI, web browser, PDA, cell phone, etc. Core functionality should be reusable across all different interfaces.
3) Interface details change frequently. Changing UI details should not affect core functionality.

The Model-View-Controller design pattern describes this separation of user interface from core object model.

(Whiteboard discussion)

1) Review Java Swing's Listener architecture
    a. Listeners are just Observers

2) Draw Model and Presentation on the board
    a. Separation of Model and Presentation

3) UI's job is to:
    a. Display information on the screen
    b. Let user edit the information
    c. 2 copies of the application's state that must be kept in synch
        i. view updated in response to changes in underlying model
        ii. model updated in response to user editing operations

4) Draw and explain the basic MVC diagram from pg. 530 in Head First Design Patterns
    a. 1 – View asks Model for state
    b. 2 – View draws data on screen
    c. 3 – View tells Controller when user does something
    d. 4 – Controller tells Model to change its state
    e. 5 – Controller tells View to change its state (e.g., enable/disable, error messages, sort, select/unselect)
    f. 6 – Model notifies View that the model state has changed
    g. 7 – View asks model for updated state
    h. 8 – View draws new data on screen

5) Model notifies View of changes using the Observer Pattern

6) How does this relate to Class Stereotypes: Boundary, Control, and Entity classes?

    a. Control and Entity classes are part of the Model
    b. View and Controller classes are Boundary classes

7) Multiple Views
    a. Multiple presentations of the model data on the screen at once
    b. Each View has its own Controller
    c. Each View observes the Model
    d. Observer mechanism keeps all Views in synch with the model and each other

8) Draw picture depicting the following:
    a. Views contain many UI controls
    b. Each UI control has event listeners observing it
    c. Event listeners notify Controllers through the Controller interface

d. Controller functionality can be directly embedded in event listeners instead of in a separate Controller object

9) EXAMPLE: Temperature Editor (with integrated controllers)
    a. Show diagram in mvc.vsd
    b. The core model is something that we have to write.
        i. Show code for the Temperature model class.
    c. The views are composed of Swing GUI components used to display application data: Text Fields, Check Boxes, Sliders, Spinners, Tables, etc. The views in this example are extremely simple, consisting of one UI control each. In general, a view is typically much larger and more complex, possibly consisting of an entire window, tab, or panel of controls.
        i. Show code for the View classes, including the code that instantiates GUI components
        ii. Show code for observer notifications
    d. The controllers are the Listener objects that are attached to GUI components for event handling.
        i. Show code that creates event listeners
    e. A View can be any part of the GUI that you want (usually a window or part of a window). The views in the Temperature Editor example atypically small.

10) In the Temperature Editor, the Controller objects were the Java event listeners. This is the most common way of implementing controllers, but this results in a relatively tight coupling between a View and its Controller. (This might be what you want).

11) Another approach is to keep the View and Controller less tightly coupled, thus allowing you to swap in different Controllers for the same View.

12) EXAMPLE: Temperature Editor (with separate controllers)

13) OLD EXAMPLE: Head First DJView
    a. BeatModelInterface
        i. BeatObserver
        ii. BPMObserver
    b. BeatModel
    c. DJView
        i. Accepts Controller and Model in constructor
        ii. When user performs actions, DJView notifies the Controller
    d. ControllerInterface
    e. BeatController
    f. View and Controller interact through a well-defined interface (as opposed to making Controllers the event listeners). This makes is easier to attach a

different Controller to the same View without having to change the View code.
    i. Example: Use a different Controller when a user is logged in than you use when they are not

14) Application of MVC to Web Applications
    a. MVC is a common architecture for building web applications

15) Draw and explain the basic MVC diagram from pg. 549 in Head First Design Patterns
    a. 1 – Browser sends HTTP request to Front Controller
    b. 2 – Front Controller inspects request, and delegates processing of the request to the appropriate Page Controller object
    c. 3 – Page Controller updates Model in response to request
        i. Processes the “input”
    d. 4 – Page Controller passes control to the appropriate Page View object, which generates the HTML that is sent back to the browser
        i. Processes the “output”