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

2) 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

3) Draw and explain the basic MVC diagram from pg. 530 in Head First Design Patterns
    a. 1 – Controller queries Model for data
    b. 2 – Controller tells View what data to display
    c. 3 – View draws data on screen
    d. 4 – View passes user input to Controller
    e. 5 – Controller
        i. Queries state of View (if needed)
        ii. Tells Model to change its state
        iii. Tells View to change its state (if needed) (e.g., enable/disable, error messages, sort, select/unselect)
    f. 6 – Model notifies Controller that the model state has changed
    g. 7 – Controller queries Model for new state
    h. 8 – Controller tells View what data to display
    i. 9 – View draws new data on screen

4) Model notifies Controller of changes using the Observer Pattern

5) VARIATION: View observes Model directly

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

7) EXAMPLE: Book Club Example
    a. Code Structure (BookClubStructure.vsd)
    b. IMemberView and IMemberViewController interfaces
    c. MemberView class code (<u>Implements</u> interface, <u>Creates/calls</u> controller, <u>Swing</u> code)
    d. MemberViewController class code (<u>Implements</u> interface, <u>Ref</u> to view, <u>Attaches</u> observer, <u>Loads</u> data, <u>Processes</u> user input)