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 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 data model**.

Core data 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 data model, for several reasons:

1) 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.
2) Interface details change frequently. Changing UI details should not affect core functionality.
3) It should be easy to add a new user interface.

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

(Whiteboard discussion)

1) Draw and label box representing the core data model.
2) Show how there may be multiple UIs layered atop the same core data model.
3) Erase all UIs but one and explain we will focus on just one of the UIs
4) A UI contains "views" of the underlying data. A view presents a graphical display of some data (**Temperature example**)
5) The UI also allows the user to modify the underlying data by interacting with the views
6) The view's job is to present the data in some fashion
7) Each view has a controller object that processes user input and modifies the data model
8) Whenever the data model changes, all views must be immediately notified so they can update their graphical presentations
    a. What would be a good way to notify views of changes to the underlying data model? (Observer pattern)

Temperature Example in Detail

(Show diagram in mvc.vsd)

The core data model is something that we have to write.

1) Show code for the Temperature model class.

The views are composed of Swing GUI components used to display application data: Text Fields, Check Boxes, Sliders, Spinners, Tables, etc.

1) Show code for the View classes, including the code that instantiates GUI components
2) Show code for observer notifications

The controllers are the Listener objects that are attached to GUI components for event handling.

1) Show code that creates event listeners