Separation of concerns. Model-View-Controller design pattern

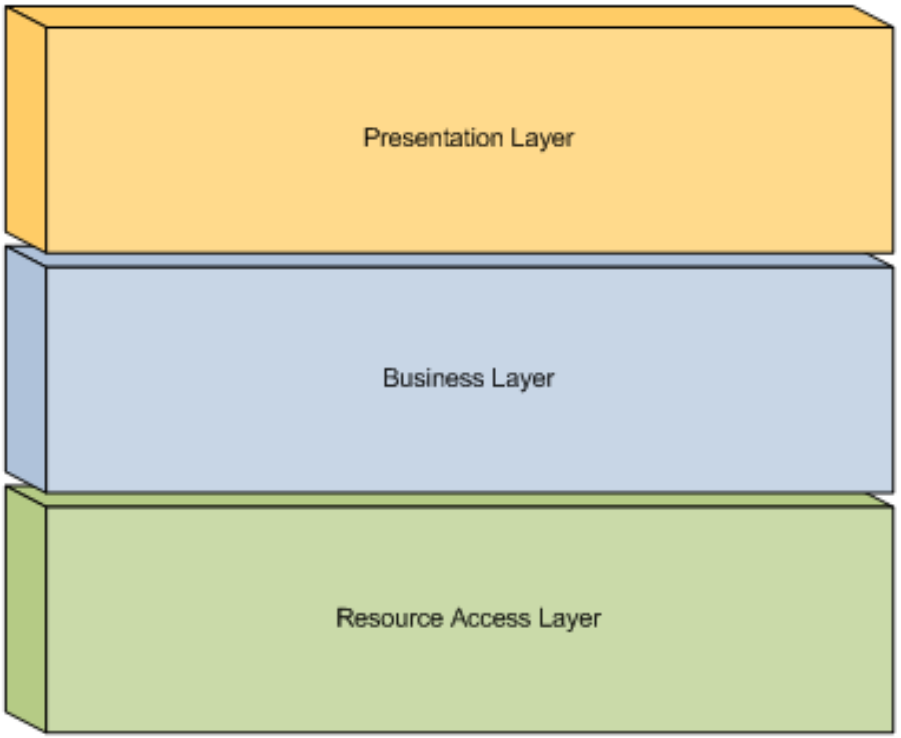
Lecture 25

Separation of concerns

- *"**Separation of concerns** … even if not perfectly possible is yet the only available technique for effective ordering of one's thoughts" (*Dijkstra, "On the role of scientific thought", 1974**).**
- To be able to manage the complexity of a software system it must be decomposed into parts that overlap in functionality as little as possible.

Separate your application into distinct layers

This allows different layers within the application to vary independently



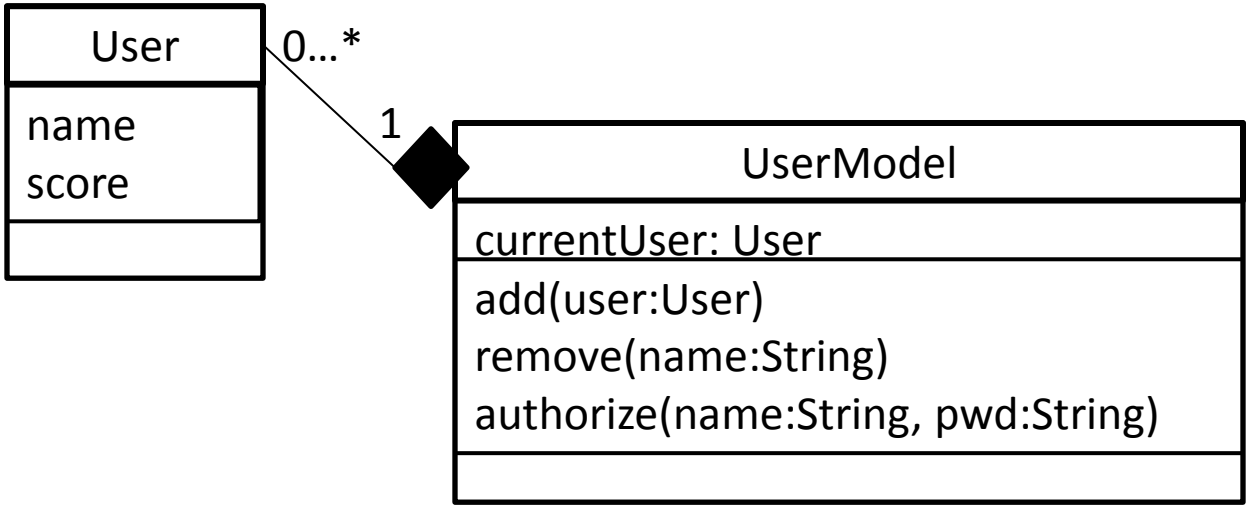
Separation of concerns with objects

- **Coupling:** the degree of dependency between two objects. We always want **low coupling**.
- **Cohesion:** the measure of how strongly-related is the set of functions performed by an object. We always want **high cohesion**.

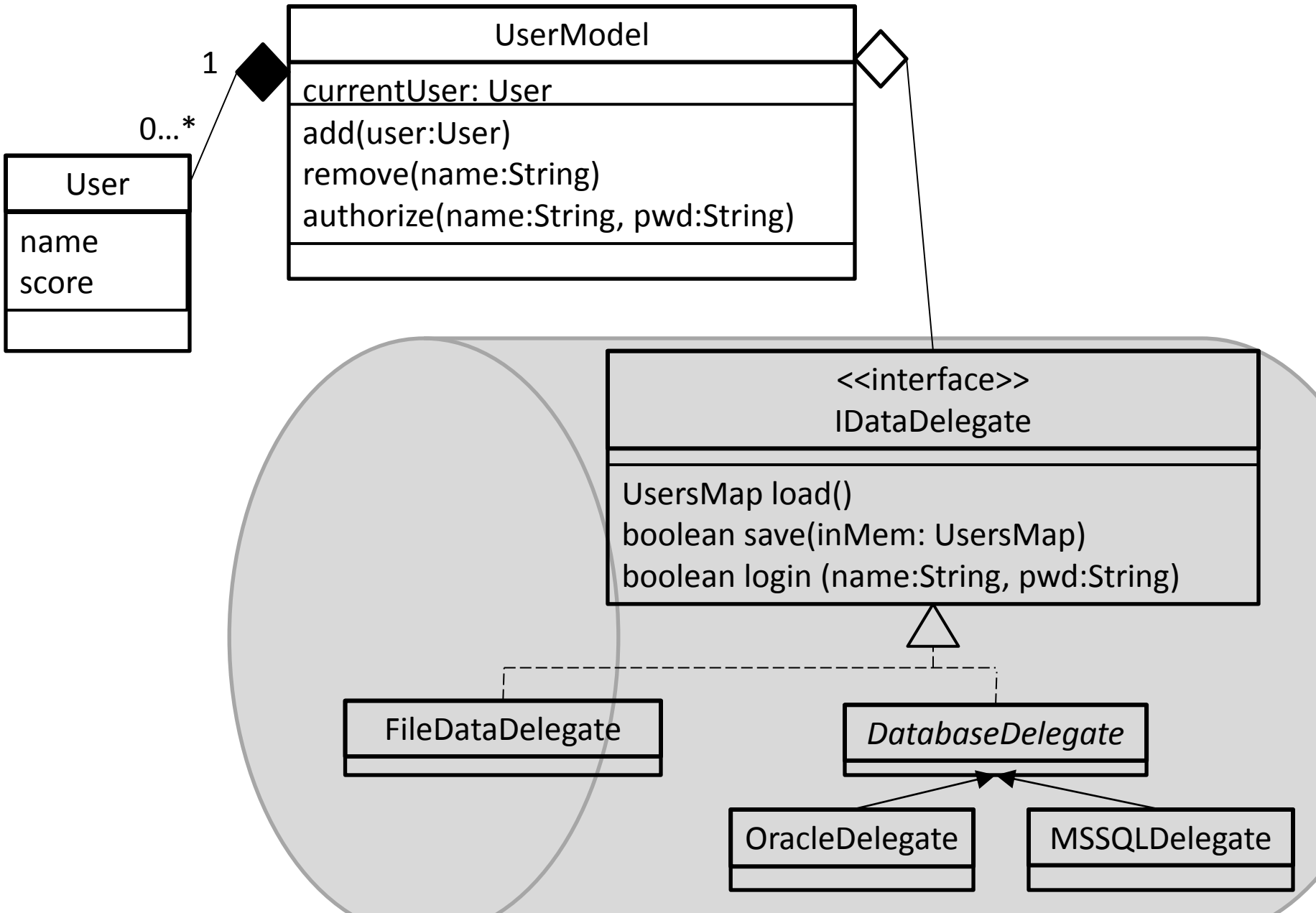
Model-View-Controller design pattern

- **Model–View–Controller** (MVC) is an architecture that separates the representation of information from the user's interaction with it.
- The *model* consists of application data and business rules, and the *controller* mediates input, converting it to commands for the model or view.
- Many of the most popular WEB application frameworks use the MVC architecture, including ASP.NET, CodeIgniter, Zend, Django, and Ruby on Rails.
- The central idea behind MVC is *code reusability* and *separation of concerns*

Very simple application: model



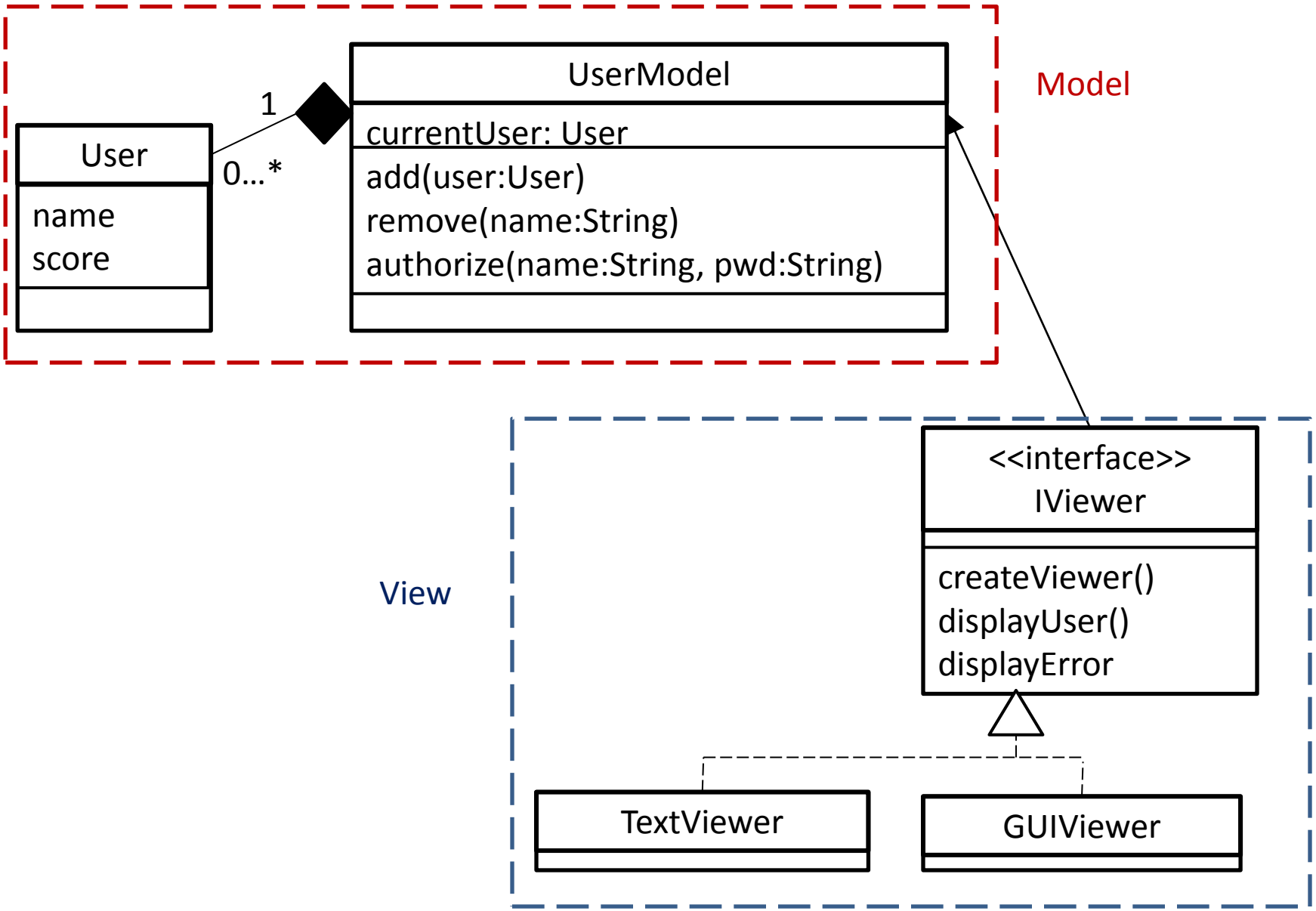
Data layer separation



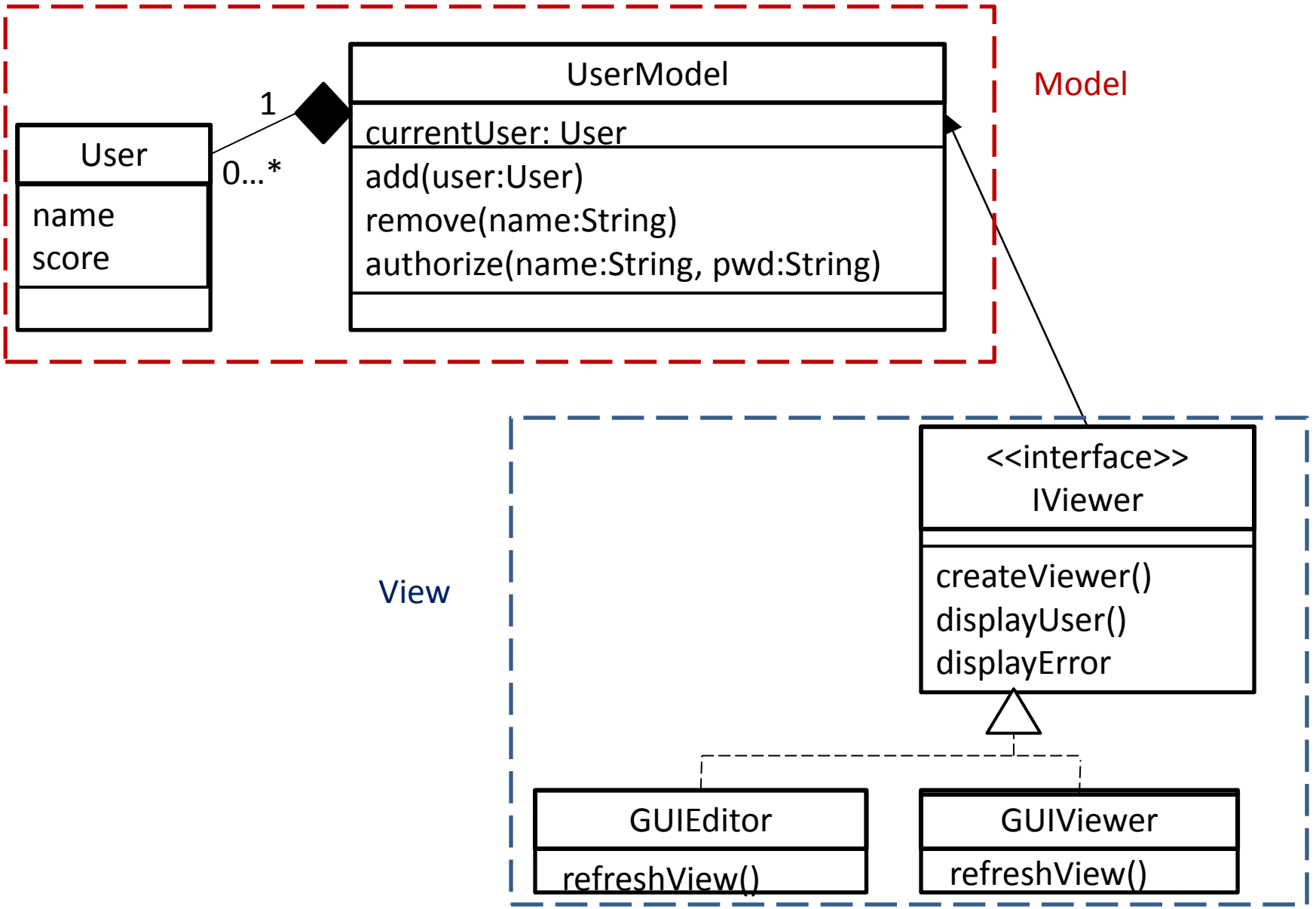
Step 1. Model Implementation with data layer

1. Implement all model classes
2. Implement data layer – at least one concrete class
3. Test all important scenarios using simple text viewer

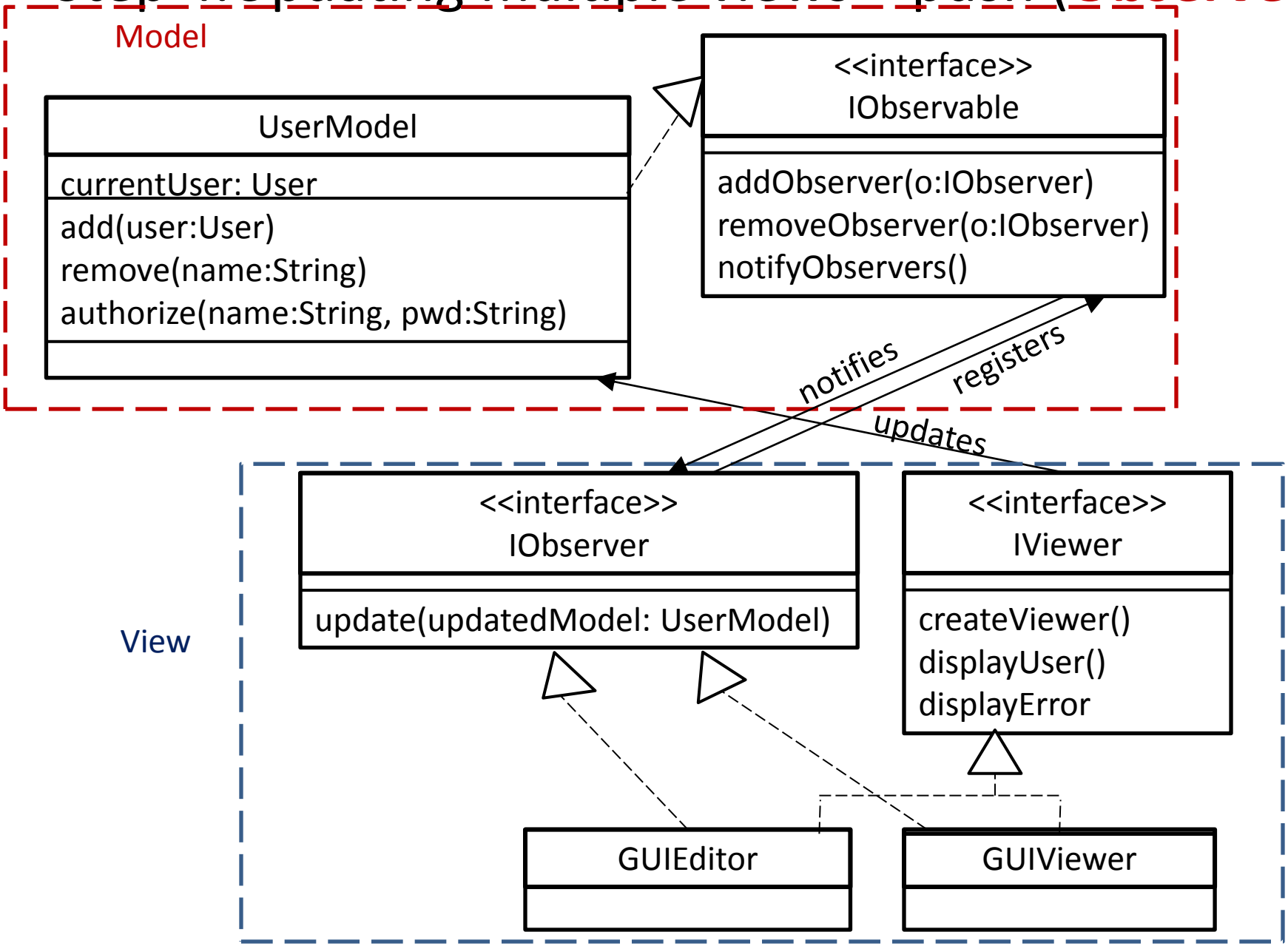
Step 2. Different Views can be plugged in - for the same model



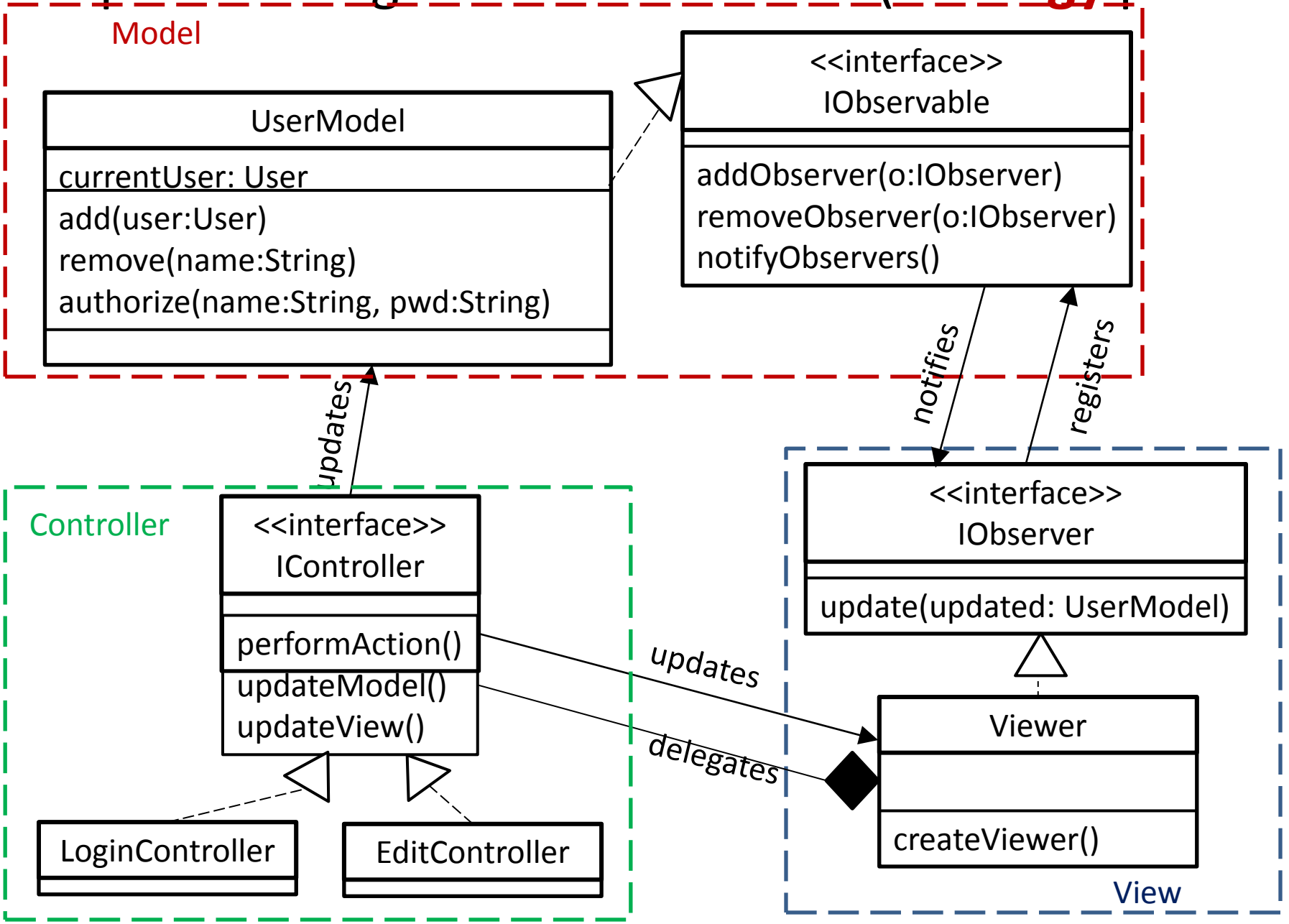
Step 3. Updating multiple views - pull



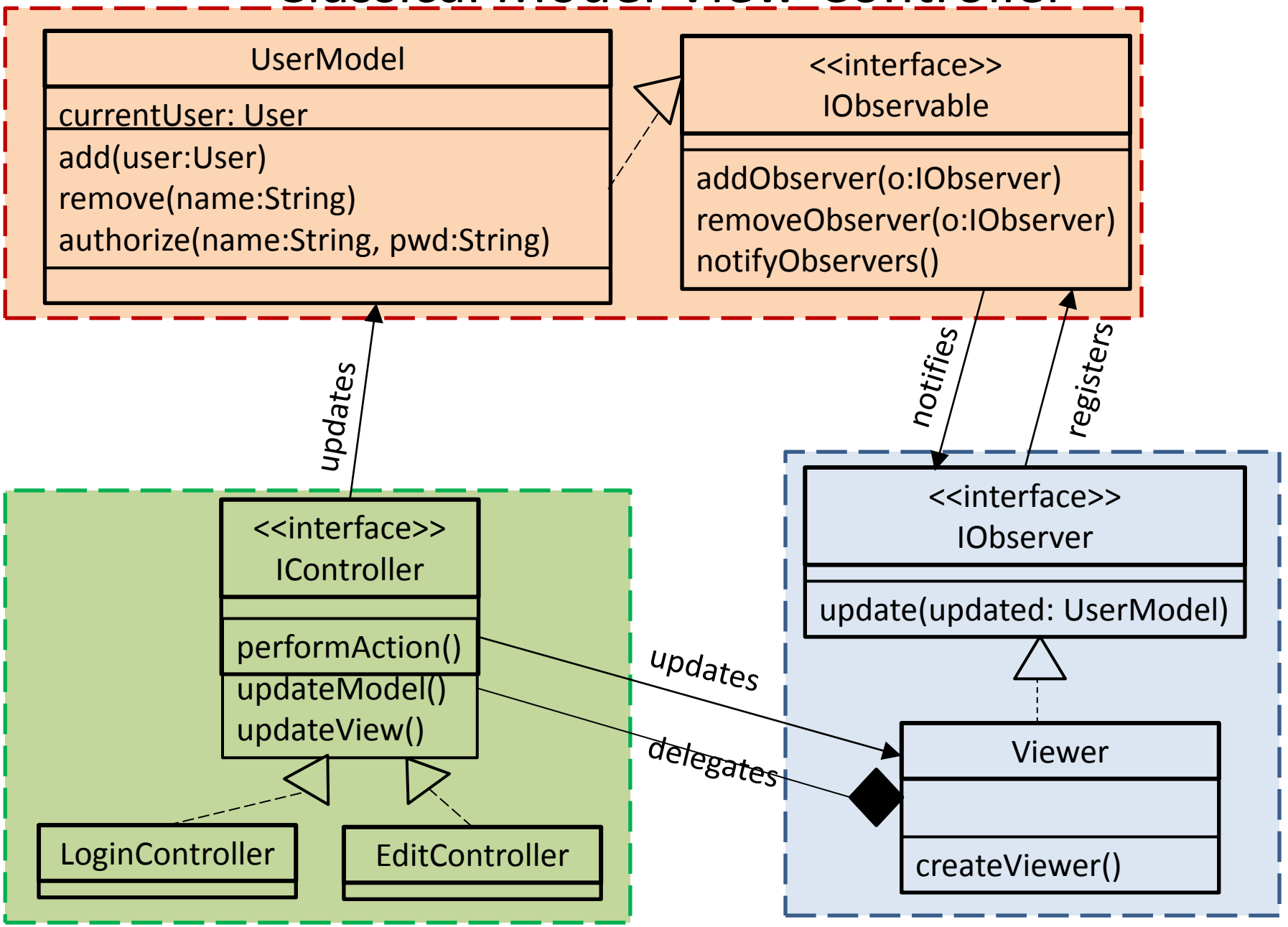
Step 4.Updating multiple views – push (**Observer**)



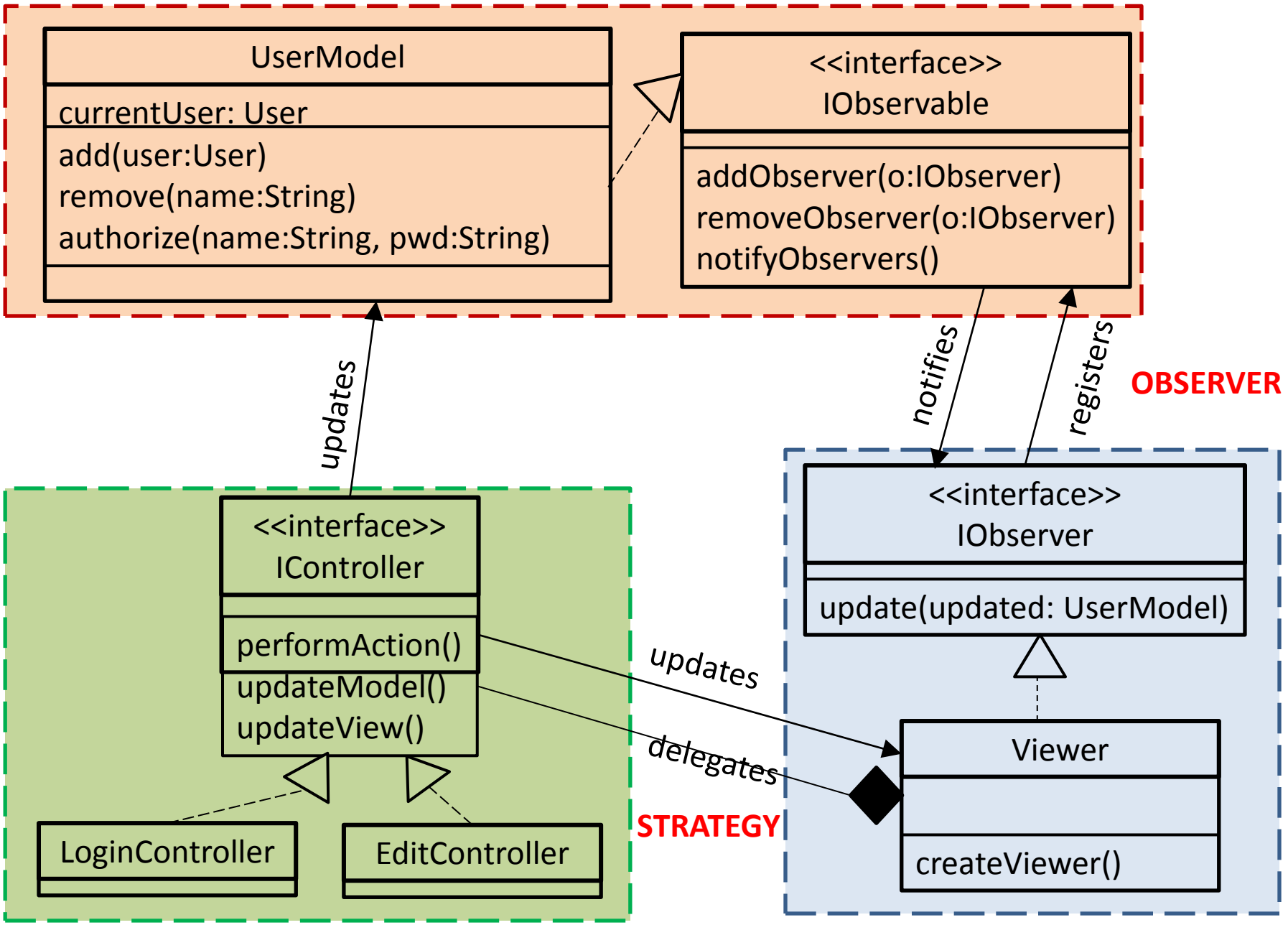
Step 5. Reusing views – Controller (**Strategy** pattern)



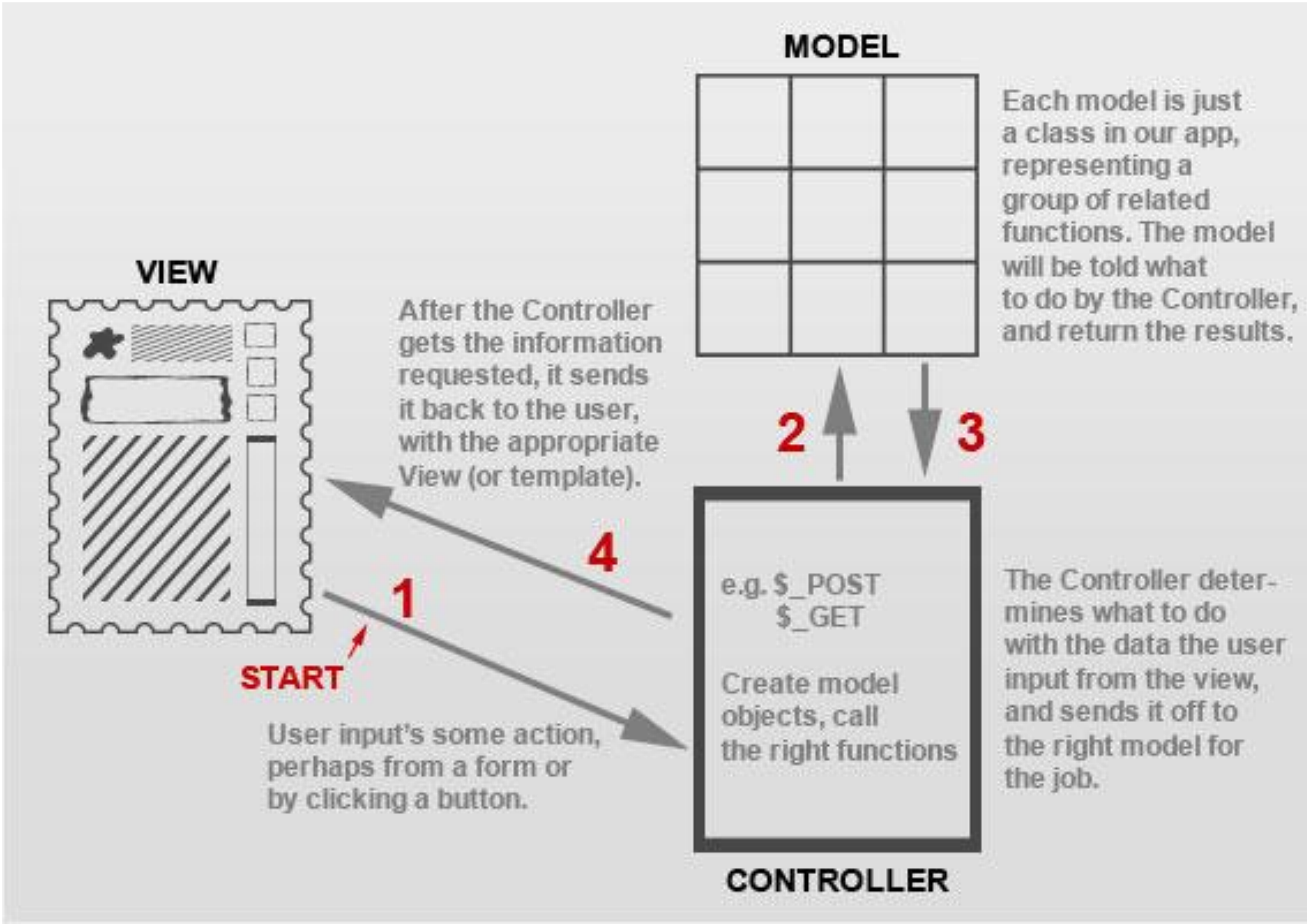
Classical Model-View-Controller



Classical Model-View-Controller



Model 2 (WEB MVC)



MVC: summary

1. MVC decouples views and models by establishing a subscribe/notify protocol between them.

   Whenever the model's data changes, the model notifies views that depend on it. In response, each view gets an opportunity to update itself.

2. MVC also lets you change the way a view responds to user input without changing its visual presentation.

   MVC encapsulates the response mechanism in a Controller object.

3. A view uses an instance of a Controller subclass to implement a particular response strategy.

   To implement a different strategy, simply replace the instance with a different kind of controller. It's even possible to change a view's controller at run-time to let the view change the way it responds to user input.

Separation anxiety

- Applying the principle of separation of concerns often involves advanced concepts and constructs which bring a certain level of complexity to the application
- These techniques often lead inexperienced or more tactical-minded developers to characterize such designs as “overly-complex” or “over-engineered”
- The real-life obstacles may prevent developing good designs beyond merely solving the technical problems.
- The trade-off is here between **ordered complexity** (good design) and disordered complexity (no design).

Design patterns

- Patterns are general solutions to recurring problems
- Using patterns makes your software flexible, extensible, ready to take change from every direction

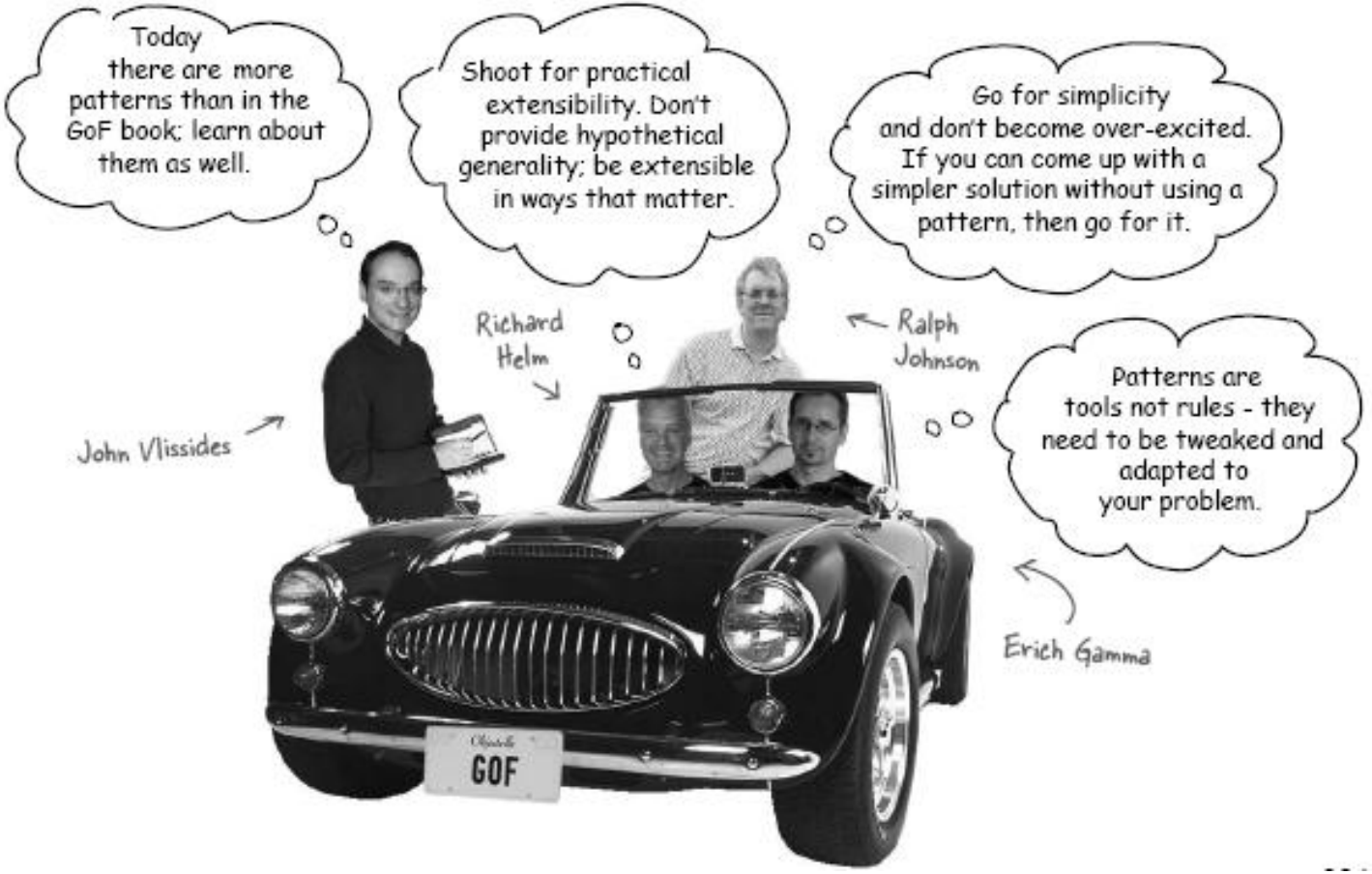
Check out the following links:

http://www.headfirstlabs.com/books/hfdp/

http://today.java.net/pub/a/today/2004/12/23/patterns.html

Founders of the concept

Gang of Four



Design Patterns: Elements of Reusable Object-Oriented Software by
Erich Gamma, Richard Helm,Ralph Johnson, John Vlissides

Patterns we covered

- Strategy: Encapsulates interchangeable behaviors and uses delegation to decide which one to use
- Singleton: ensures one and only object is created
- Observer: allows objects to be notified when state changes
- Decorator: wraps an object to provide new behavior
- Model-View-Controller: decouples model from views and controllers, this allows to plug multiple views for the same model and to reuse views by interchanging their controllers