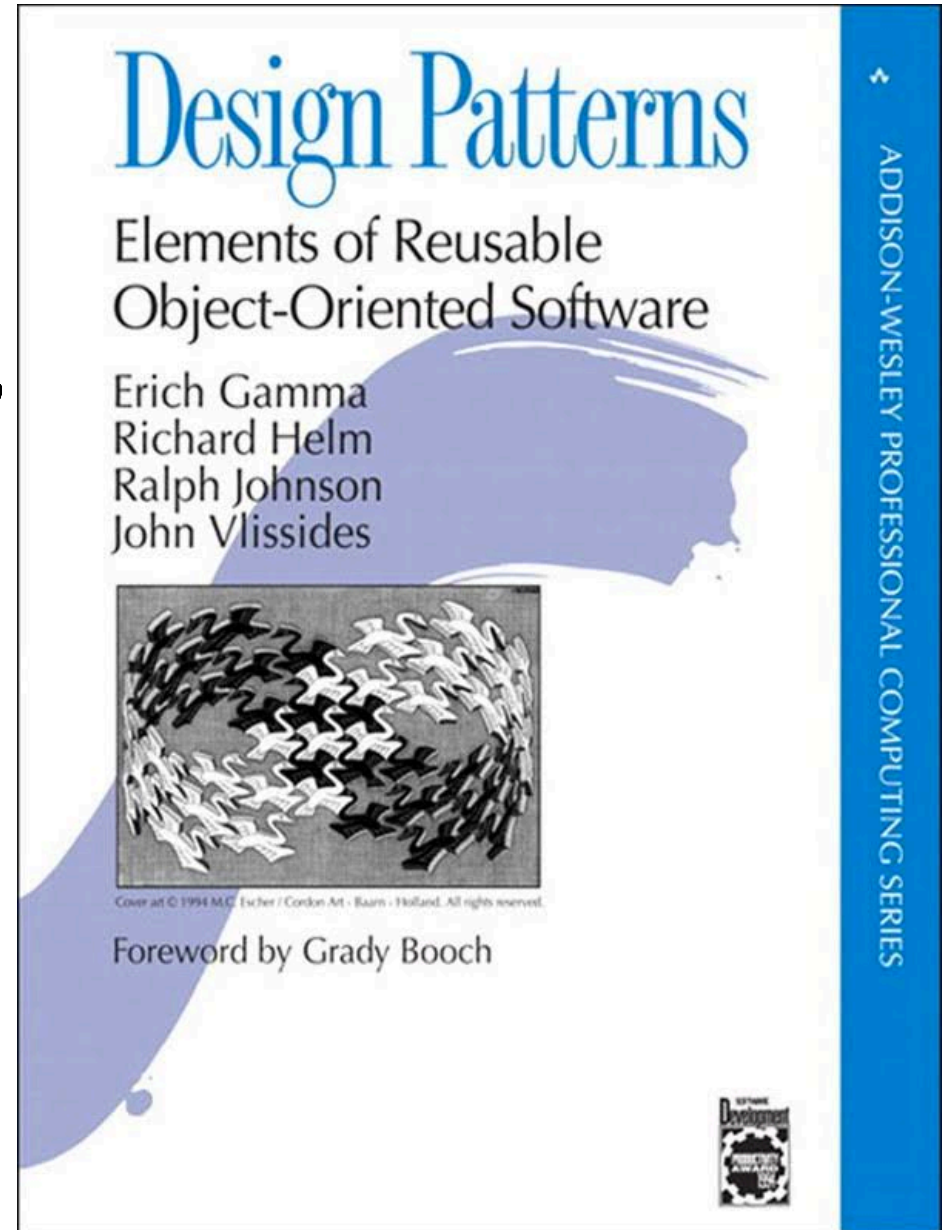
Design Patterns, cont.



Problem: Social media updates

- You have your InstaTwitInYouFaceTrest app open and a friend makes a post / updates their status. How do you get the info before the next time you (manually) refresh your app?

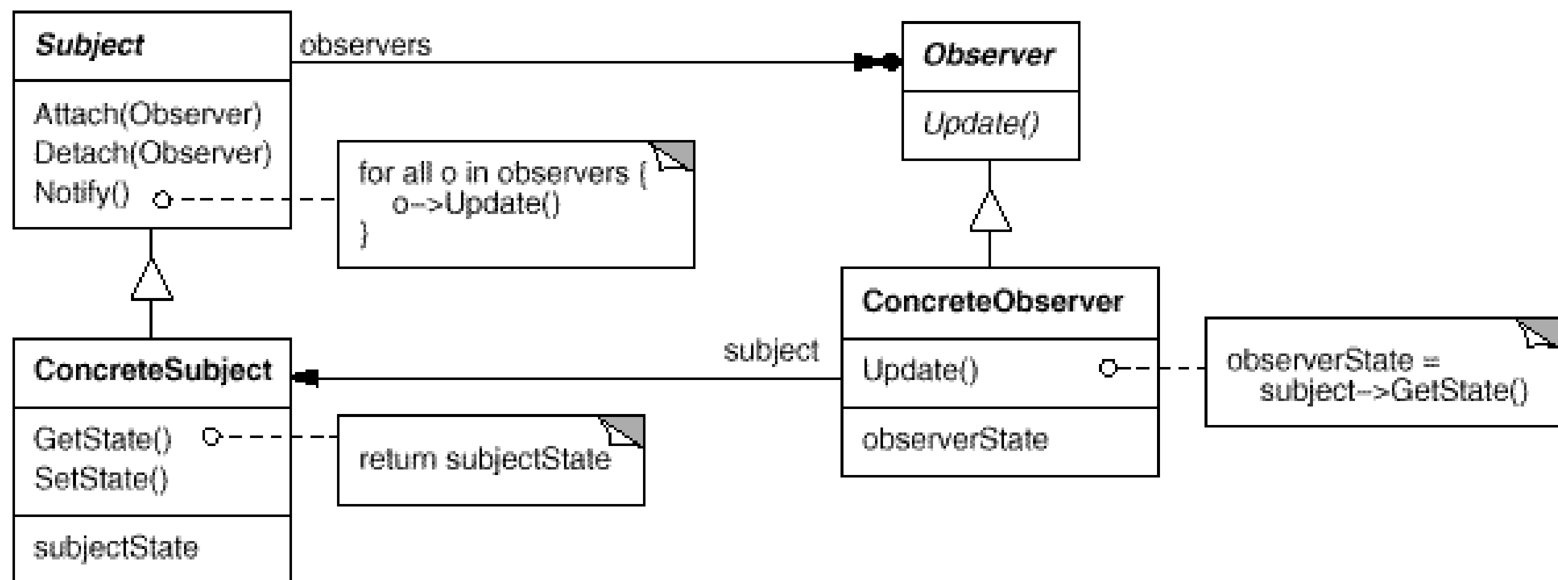
The Observer Pattern (a.k.a. Publish/Subscribe)

- **Problem: Keep a group of objects "in sync" in the presence of asynchronous updates, while minimizing the amount of coupling.**
- **Intent: Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.**
- **Use the Observer pattern when:**
  - When changes to one object requires changes to other and you don't know which and/or how many.
  - When an object should be able to notify other objects without making assumptions about who these other objects are (*i.e.,* you don't want these objects tightly coupled).

Observer Pattern

A) Classes

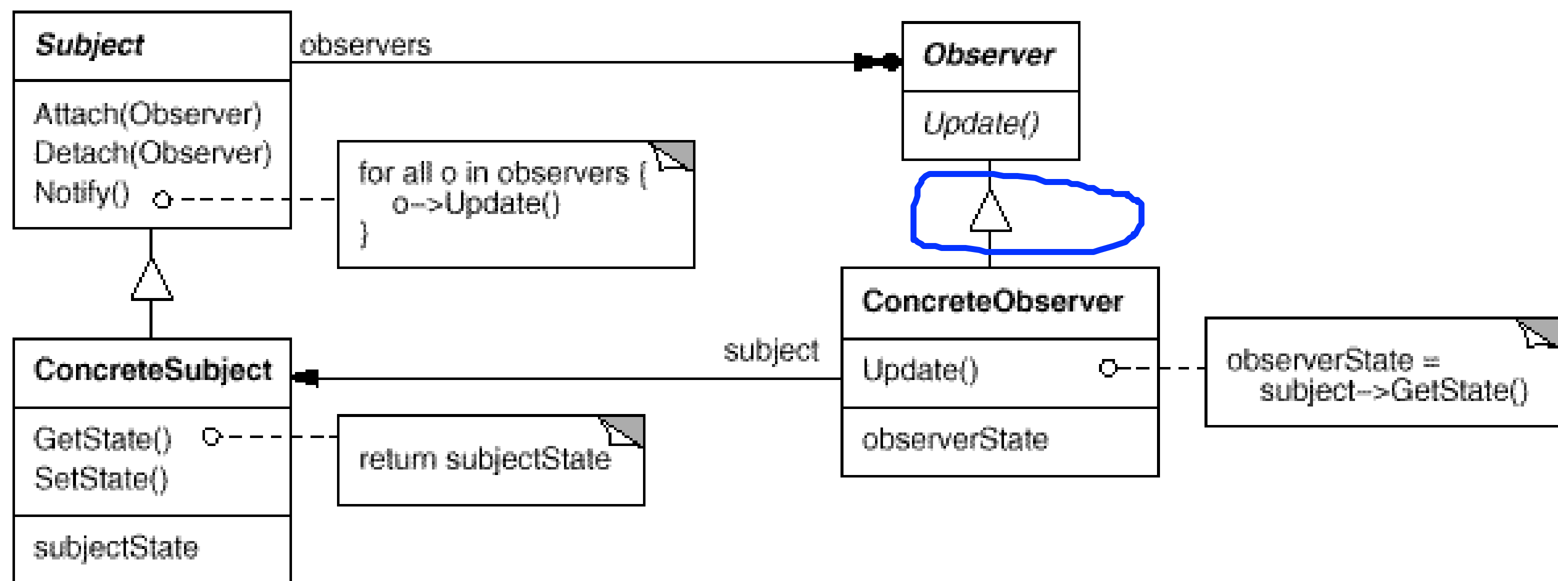
B) Objects



Observer Pattern

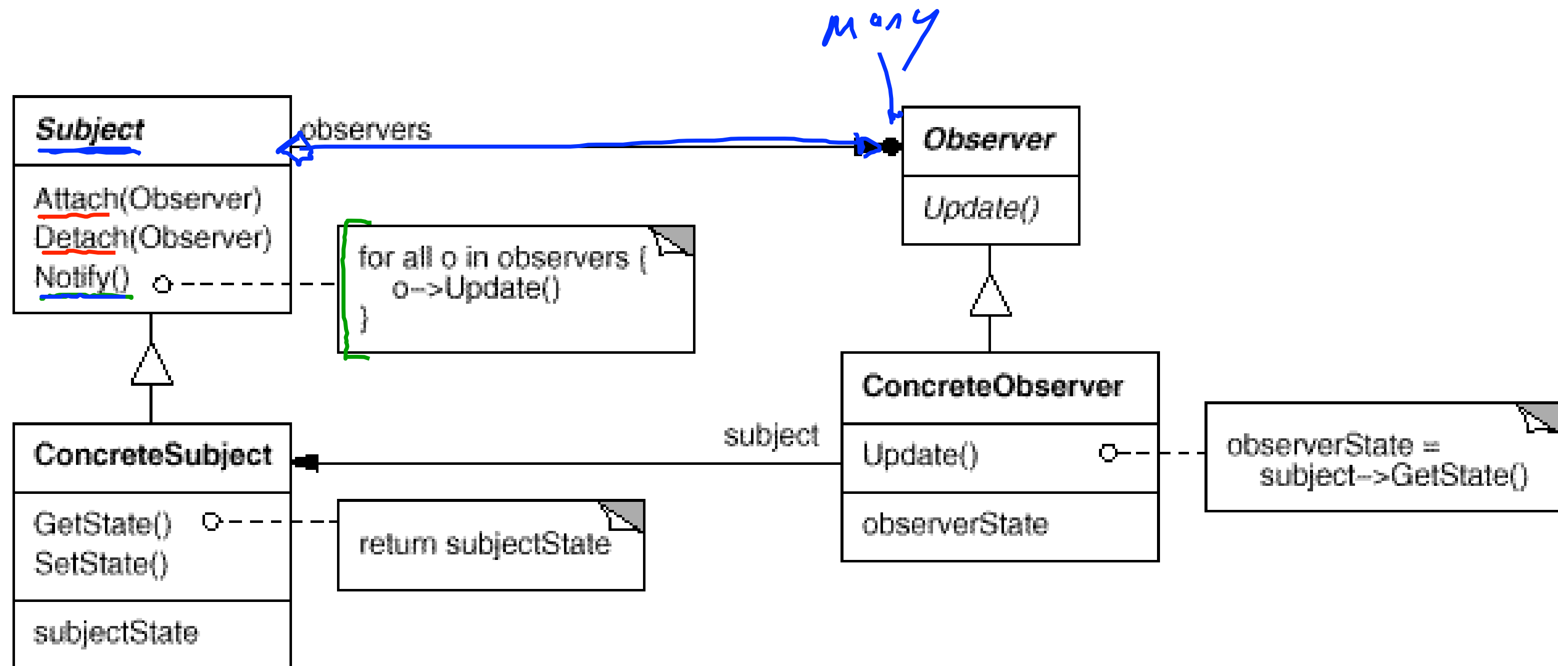
A) HasA (containment)

B) IsA (inheritance)



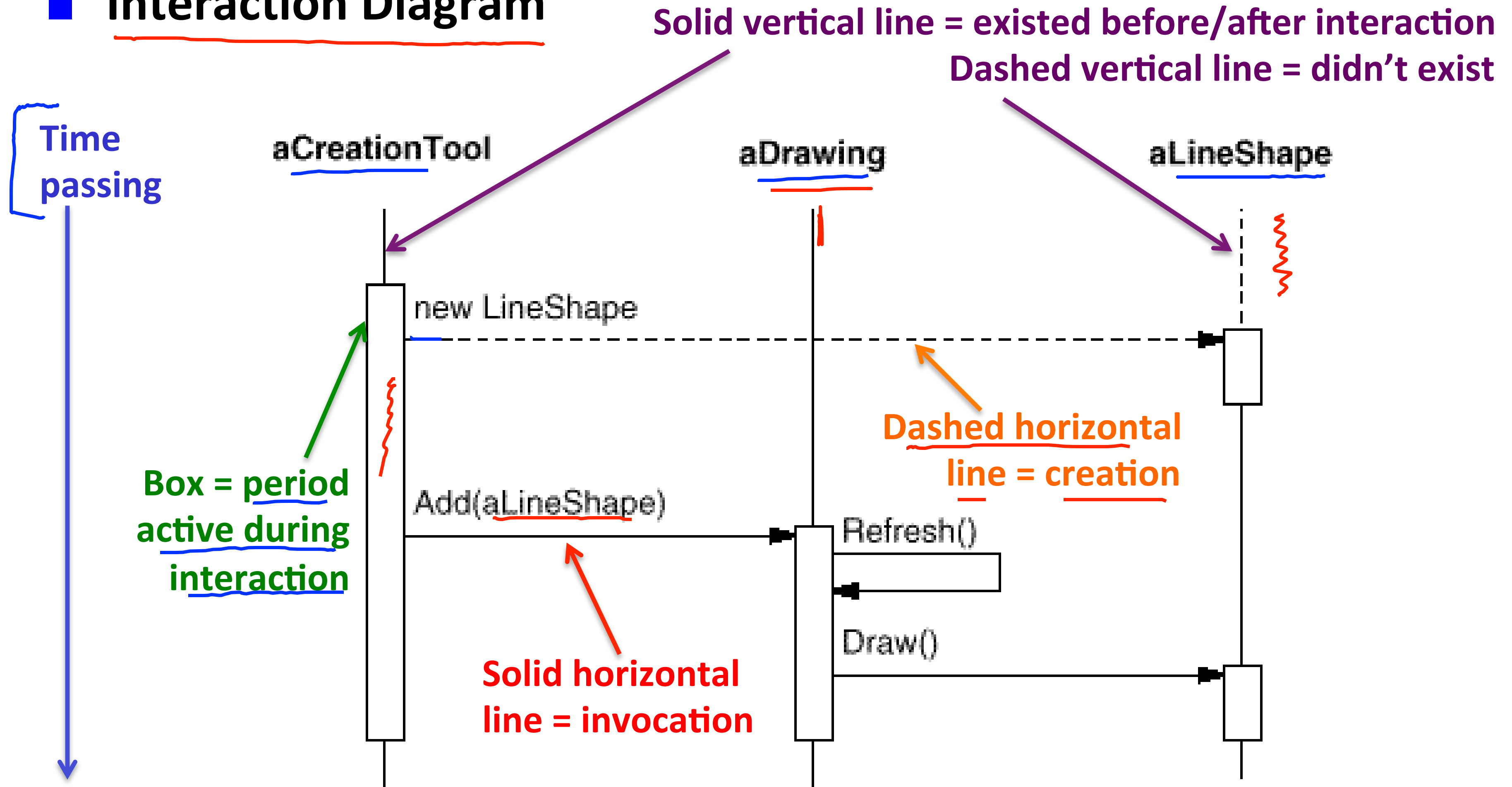
Observer Pattern

- **Solution:**
  - Observers can “attach” to a Subject.
  - When Subject is updated, it calls Update() on all Observers
  - Observers can query Subject for updated state.



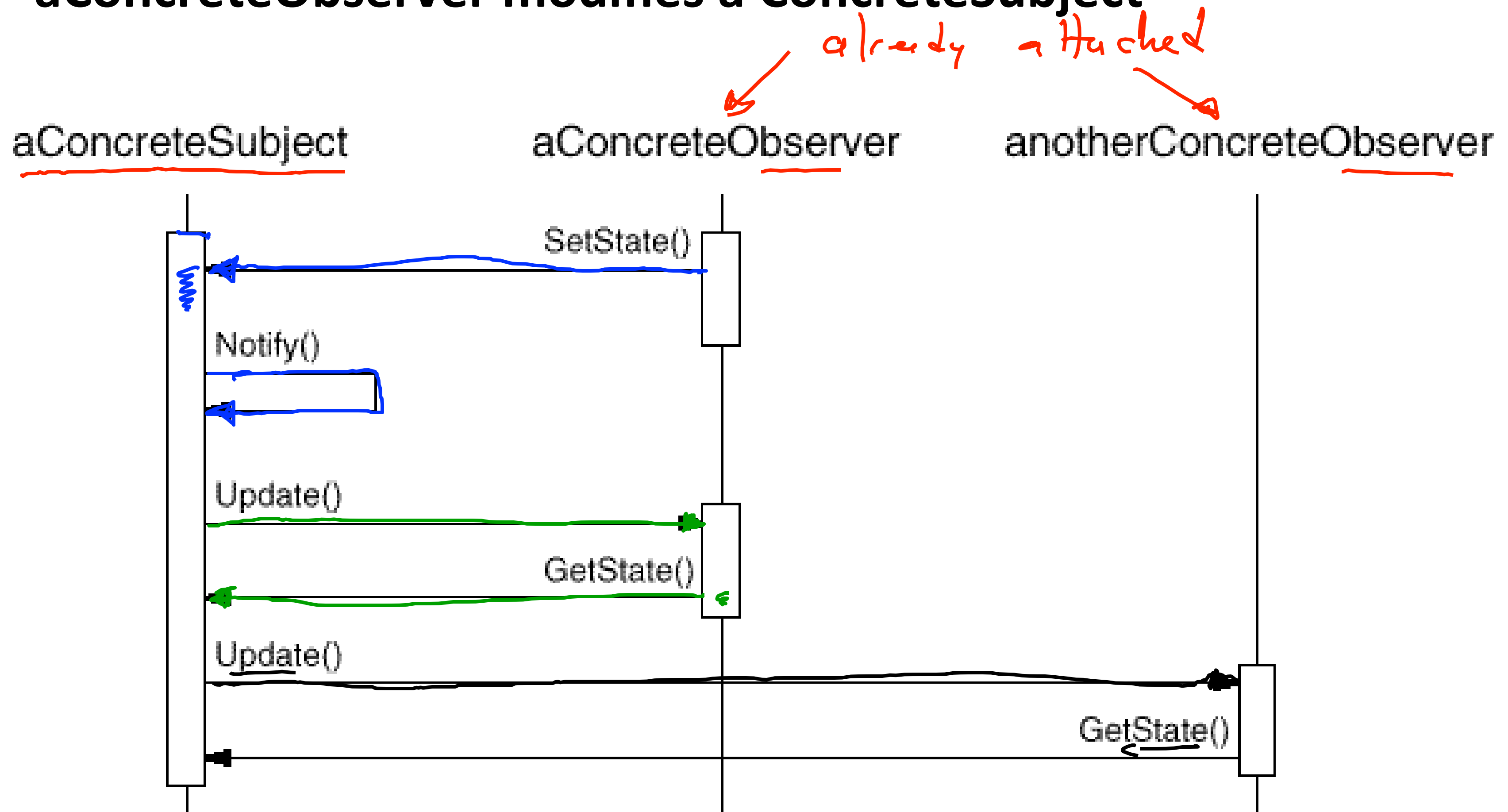
Class/Object Notation (cont.)

- Interaction Diagram



Observer Pattern Interaction Example

- **aConcreteObserver modifies a ConcreteSubject**



Problem: Playing AKQ in different system

- acekingqueen.com opens an online contest for best robot players for that game. You want to submit your entry, but the expect a different interface to the robot players. Do you have to re-write your code to implement their interface?

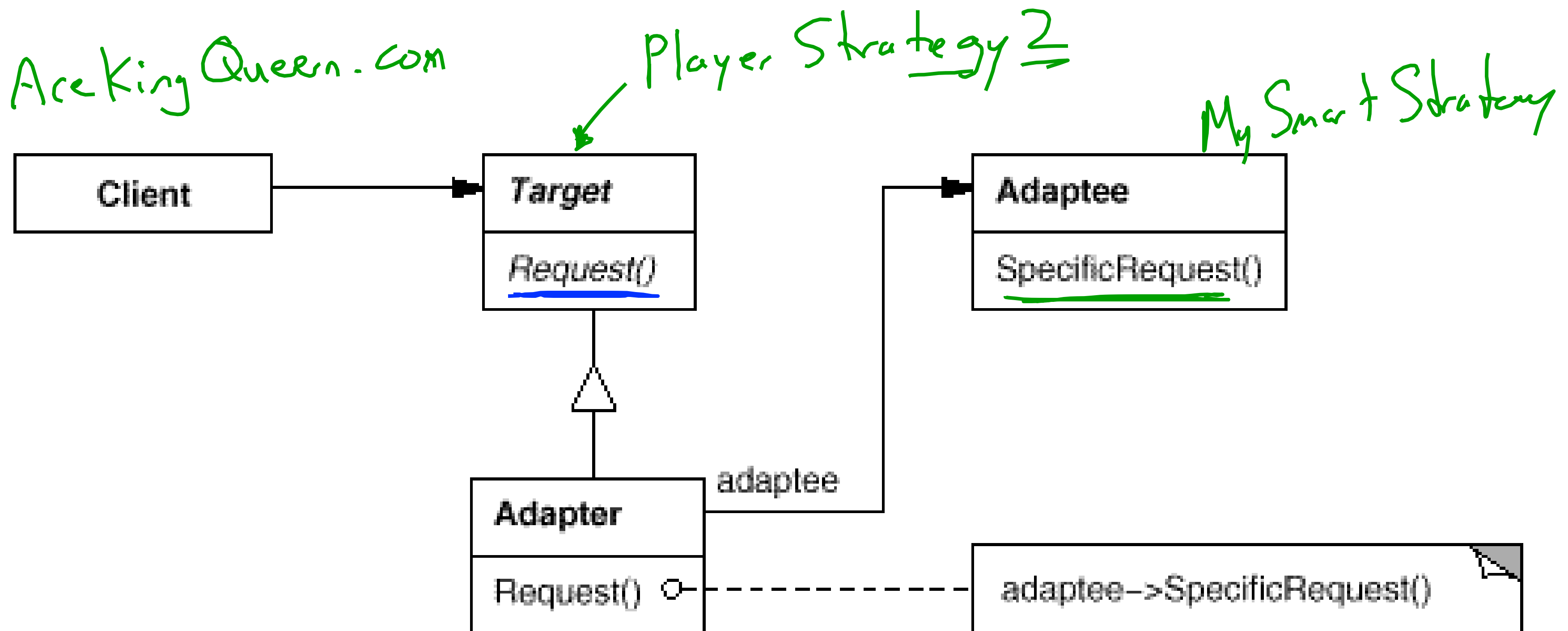
Adapter Pattern

- **Intent: Convert the interface of a class into another interface that a client expects. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.**

- **Use the Adapter pattern when:**
  - You want to use an existing class and interface doesn't match the one that you need
  - You want to create a reusable class that cooperates with unrelated and unforeseen classes (non-compatible interfaces)
  - You need to use several existing subclasses, but it's impractical to adapt their interface by subclassing every one.

Adapter Pattern

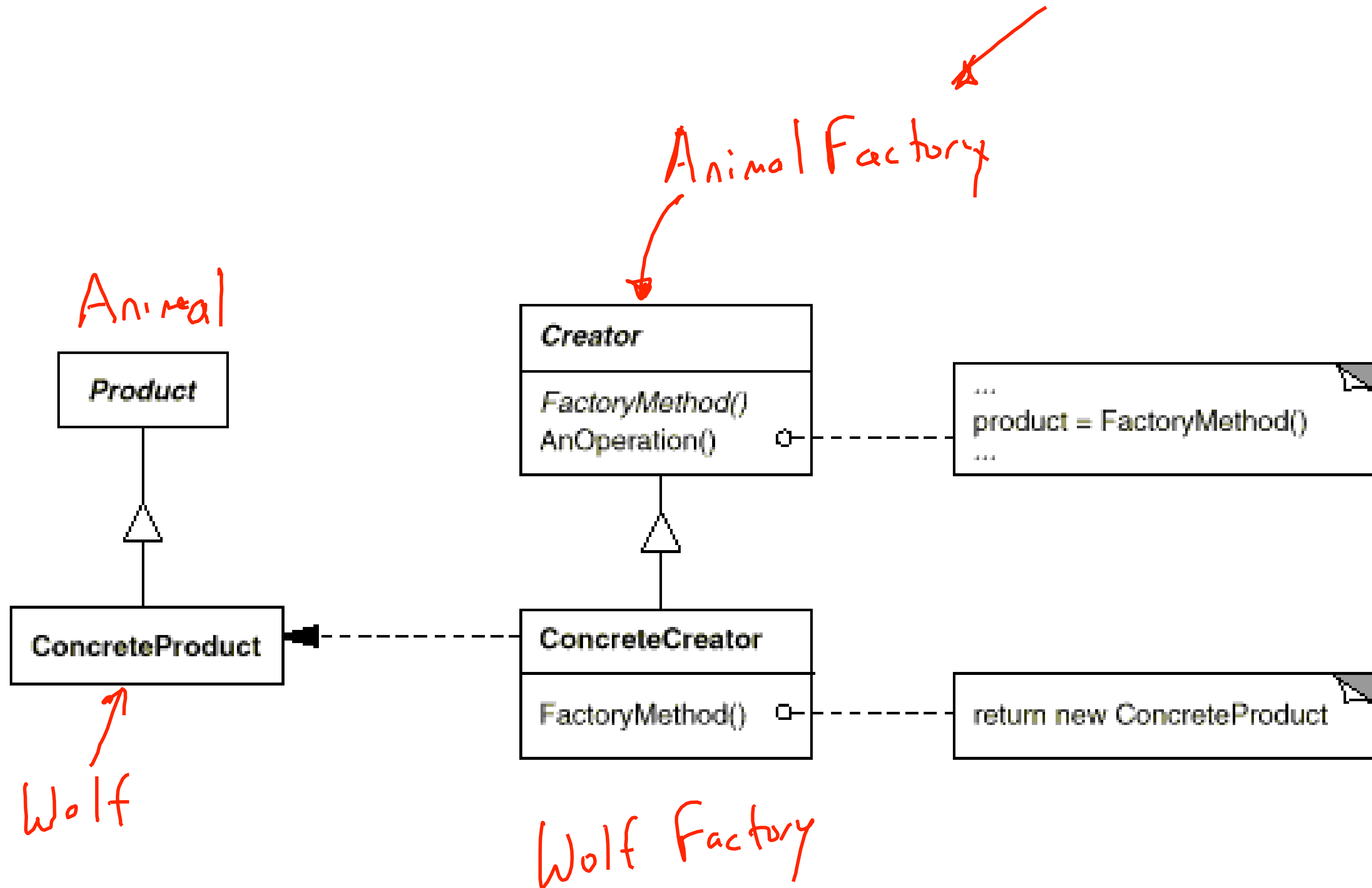
- **Solution:**
  - Adapter class IsA derived class of Target type
  - Adapter class HasA Adaptee class
  - Adapter class delegates requests to Adaptee class



Factory Method

- **Intent: Define an interface for creating an object, but let subclasses decide which class to instantiate. I.e., defer instantiation to subclasses.**
- **Also known as: Virtual Constructor**
- **Use a Factory Method when:**
  - A class can't anticipate the class of objects that it must create
  - A class wants its subclasses to specify the objects it creates
  - Classes delegate responsibility to one of several helper subclasses, and you want to localize the knowledge of which helper subclass is the delegate

Factory Method



Converting Algorithms to Code

- There are lots of good algorithms that exist
- They are generally written in pseudo code
- An important skill for programmers is to be able to convert psuedo-code into code
- This is what we'll do for our next assignment

- So let me introduce to you the idea of dynamic programming

Dynamic Programming

- **Refers to solving a complicated problem by breaking it down into simpler sub-problems in a recursive manner using memoization**

- **Requires:**
- **Optimal Substructure:**
  - a problem that can be solved optimally by breaking it into sub-problems and then recursively finding the optimal solutions to the sub-problems
- **Overlapping Sub-problems**

Memoization

- **Remember the result of solving a sub-problem.**
- **Store in a look-up table where the description of the sub-problem is the key.**
- **Before(re-)solving a sub-problem, check the lookup table and immediately return the stored value if present.**

- **Sometimes we refer to this as *caching* a result**