Section 3.3
Design Patterns

1. Overview
2. Façade
3. Observer
4. Factory
5. Anti-patterns

3.3.1 Overview

- What is a pattern?
  - “the regular and repeated way in which something is done”
    Merriam-Webster
- What is a *design pattern*?
  - solution to a commonly occurring programming problem
  - an established way of organizing classes to solve the problem
  - each pattern dictates the precise usage of:
    - inheritance
    - delegation, through composition
    - some specific operations to be implemented

Overview (cont.)

- Authoritative textbook on design patterns

  Design Patterns: Elements of Reusable Object-Oriented Software
  by Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides
  (also known as the “Gang of Four”) in 1994

- What is the *client class* in a design pattern?
  - it’s the class that is **using** the classes in the design pattern

- Types of design patterns
  - creational
  - structural
  - behavioural
  - architectural (not really design patterns)

Types of Design Patterns

- Creational
  - they specify how objects are created
    - which objects create other objects
  - examples: Factory, Abstract Factory, Singleton, etc.
- Structural
  - they specify how objects are associated with each other
    - through inheritance and composition
  - examples: Façade, Bridge, Decorator, Proxy, etc.

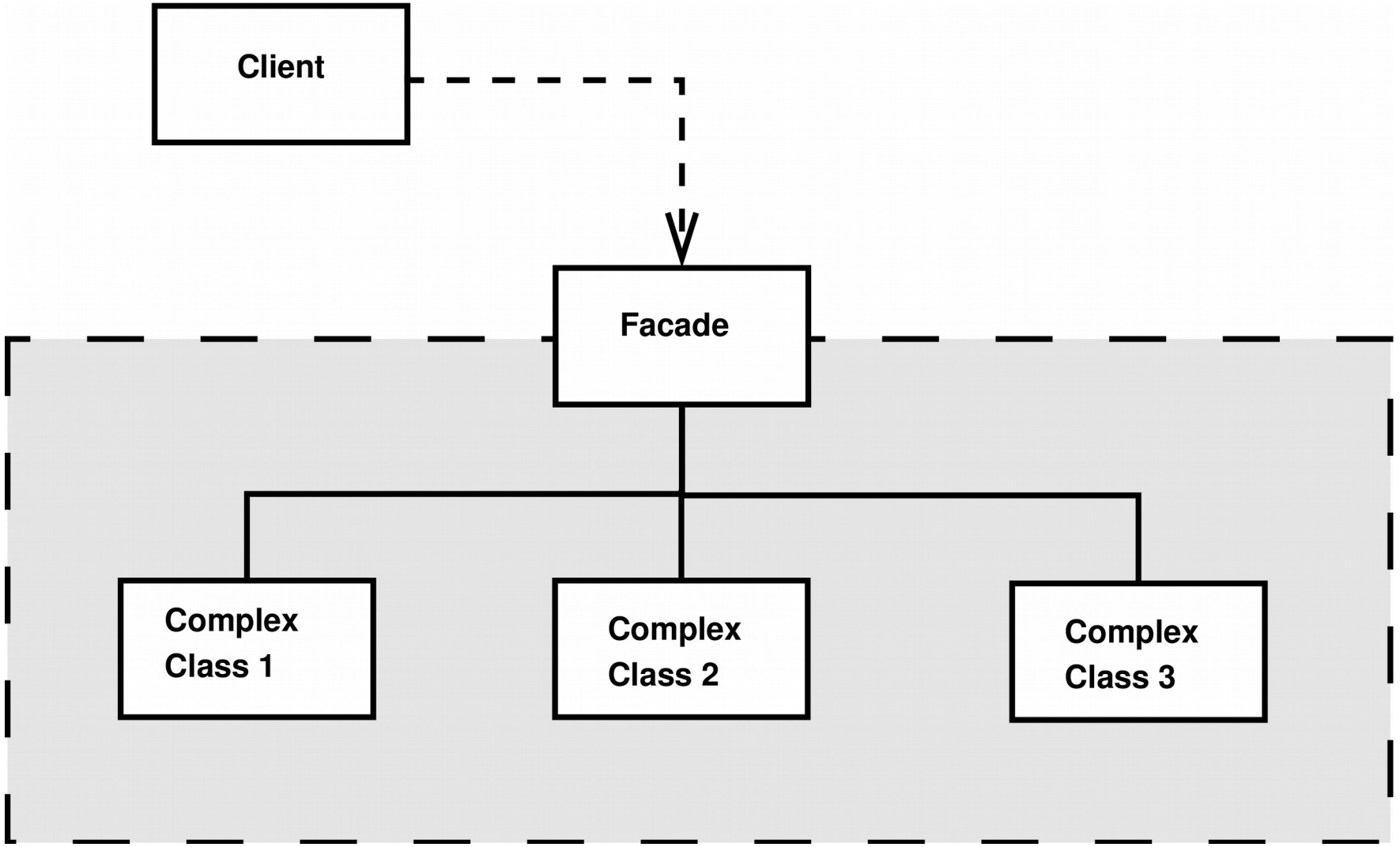
Types of Design Patterns (cont.)

- Behavioural
  - they specify how objects communicate with each other
    - which objects call what operations on which other objects
  - examples: Observer, Strategy, Visitor, etc.
- Architectural (these are not true design patterns!)
  - they specify how objects are grouped together into subsystems
    - subsystems are groups of classes that belong together functionally
  - examples: client-server, peer-to-peer, MVC, etc.

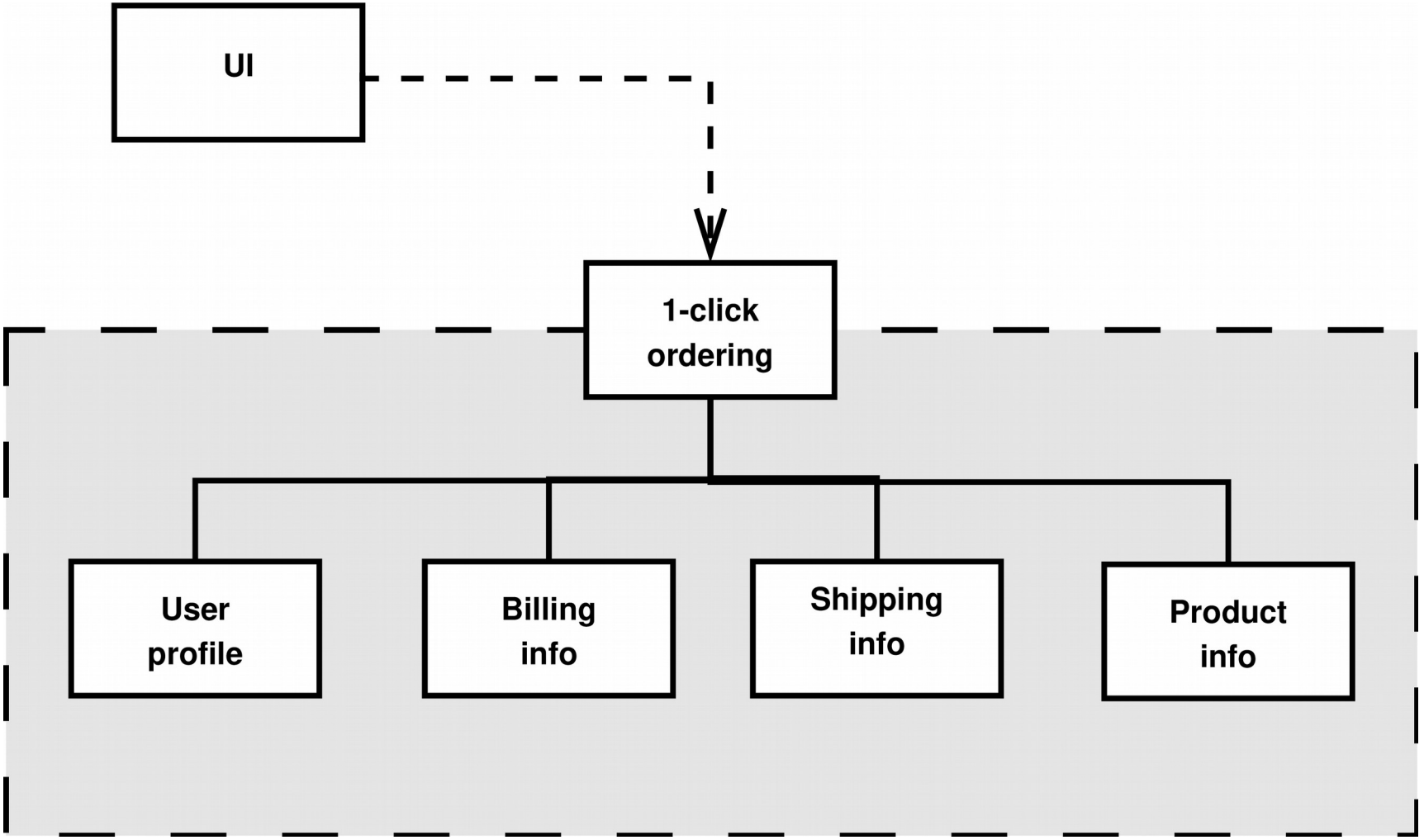
3.3.2 Façade

- Structural design pattern
- Provides a simplified interface to complex classes
  - the client class calls simple the operations on the Façade class
  - the Façade class calls operation(s) on the actual class(es)
    - using delegation
  - operations on the actual classes are still available to client class
    - but it’s simpler to go through Façade
    - Façade serves to *encapsulate* the details of the actual classes

Façade (cont.)



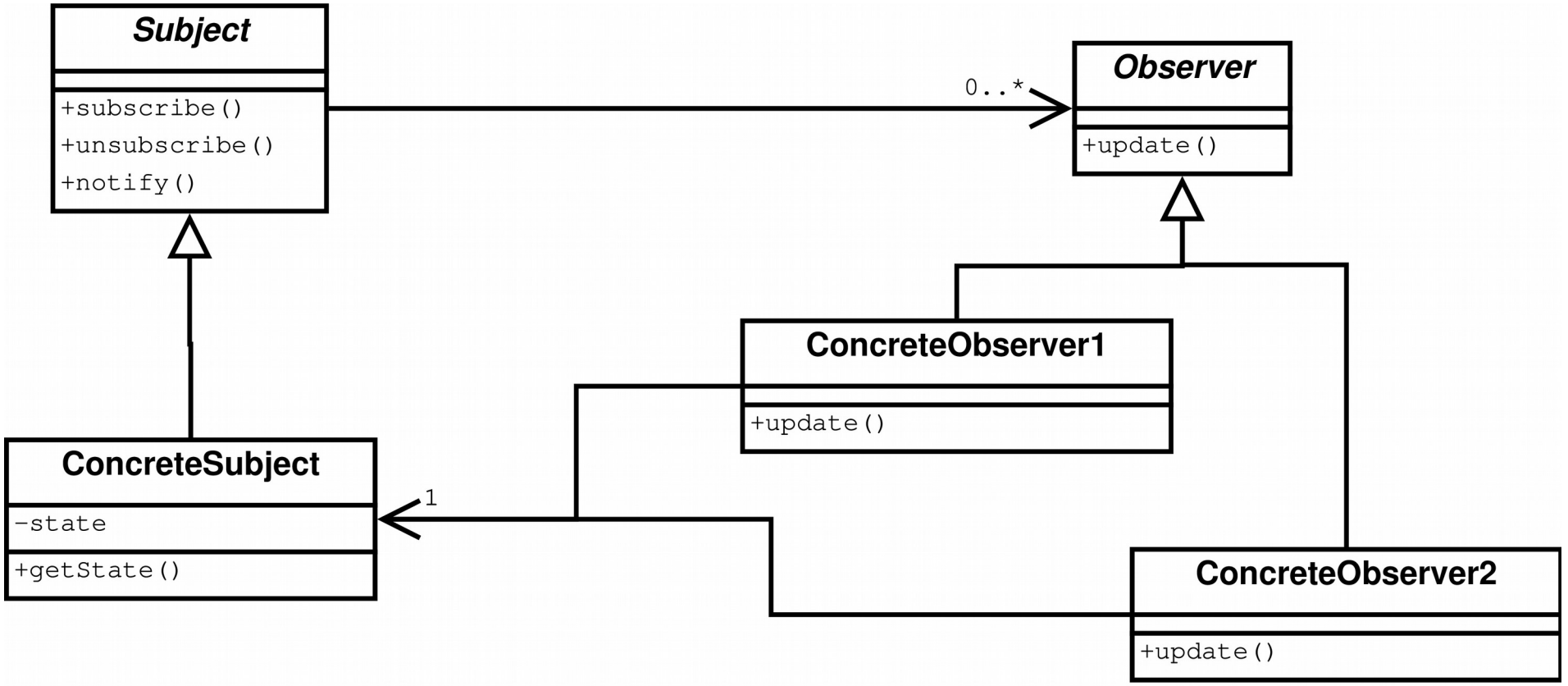
Façade Example



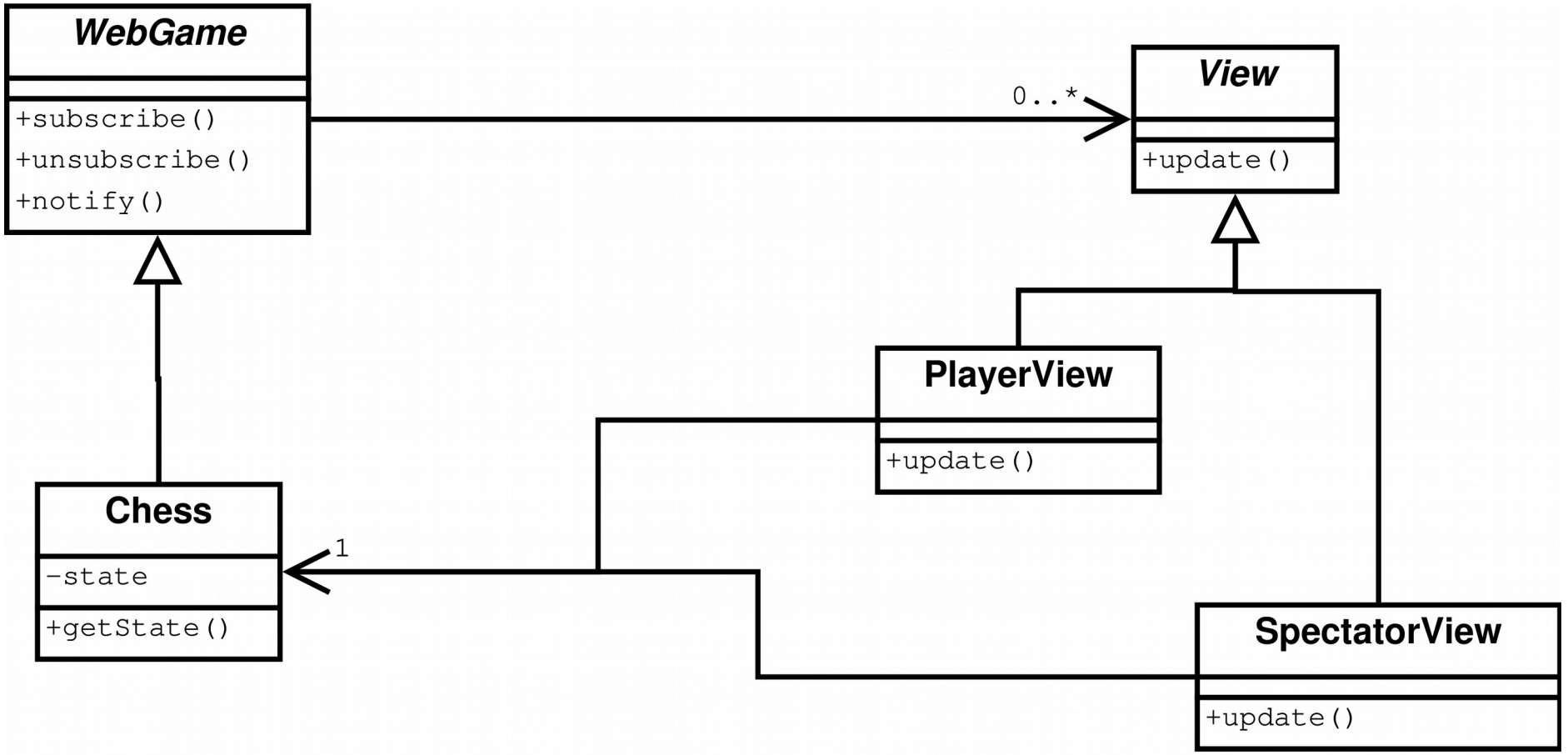
3.3.3 Observer

- Behavioural design pattern
- Allows observer classes to track changes in subject class
  - subject class
    - maintains a collection of observers
    - notifies all observers when its state changes
  - observer class
    - subscribes to notifications from the subject
    - updates itself when notified of a change in the subject's state
  - often used with MVC architectural pattern
  - subject/observer also called publisher/subscriber

Observer (cont.)



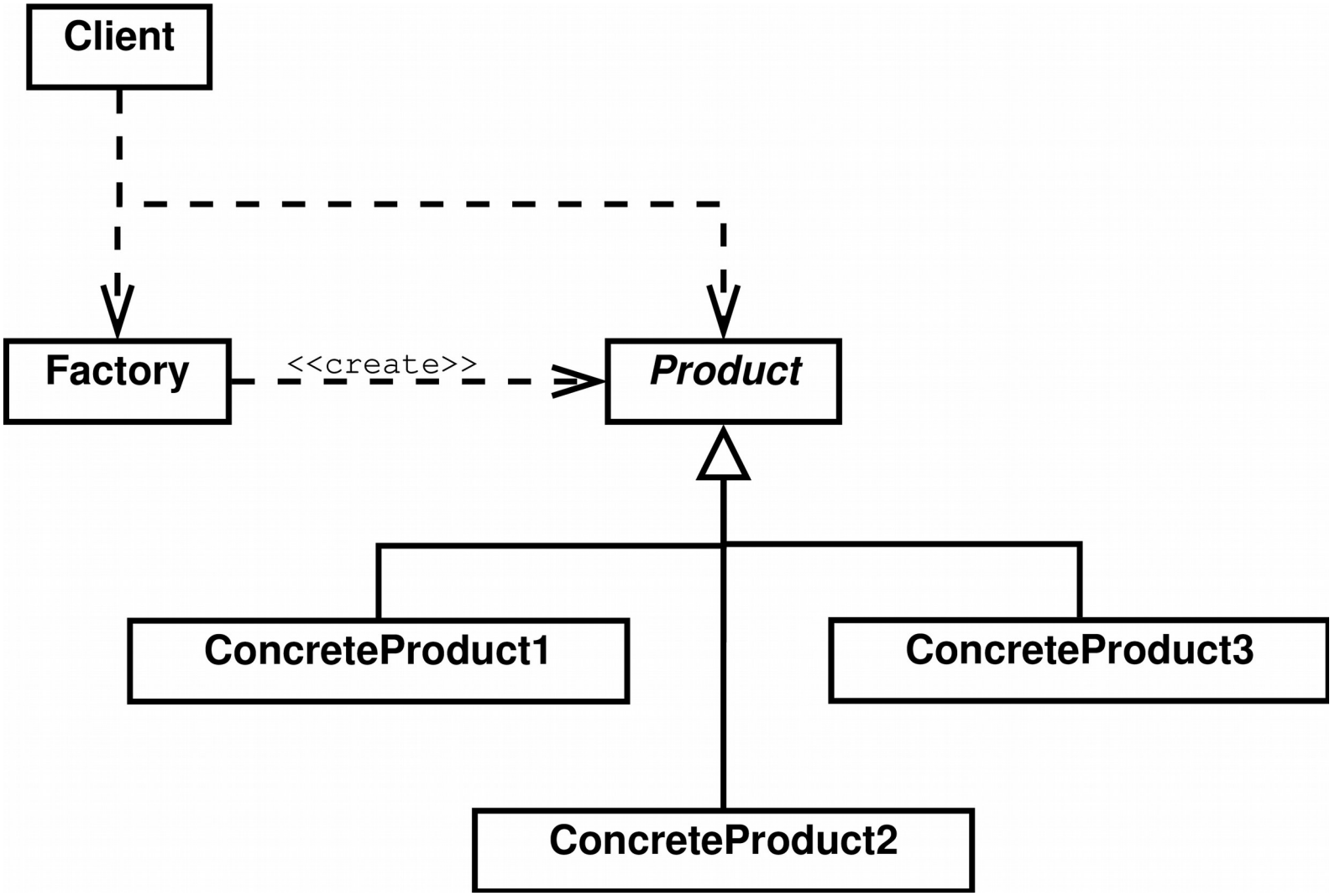
Observer Example



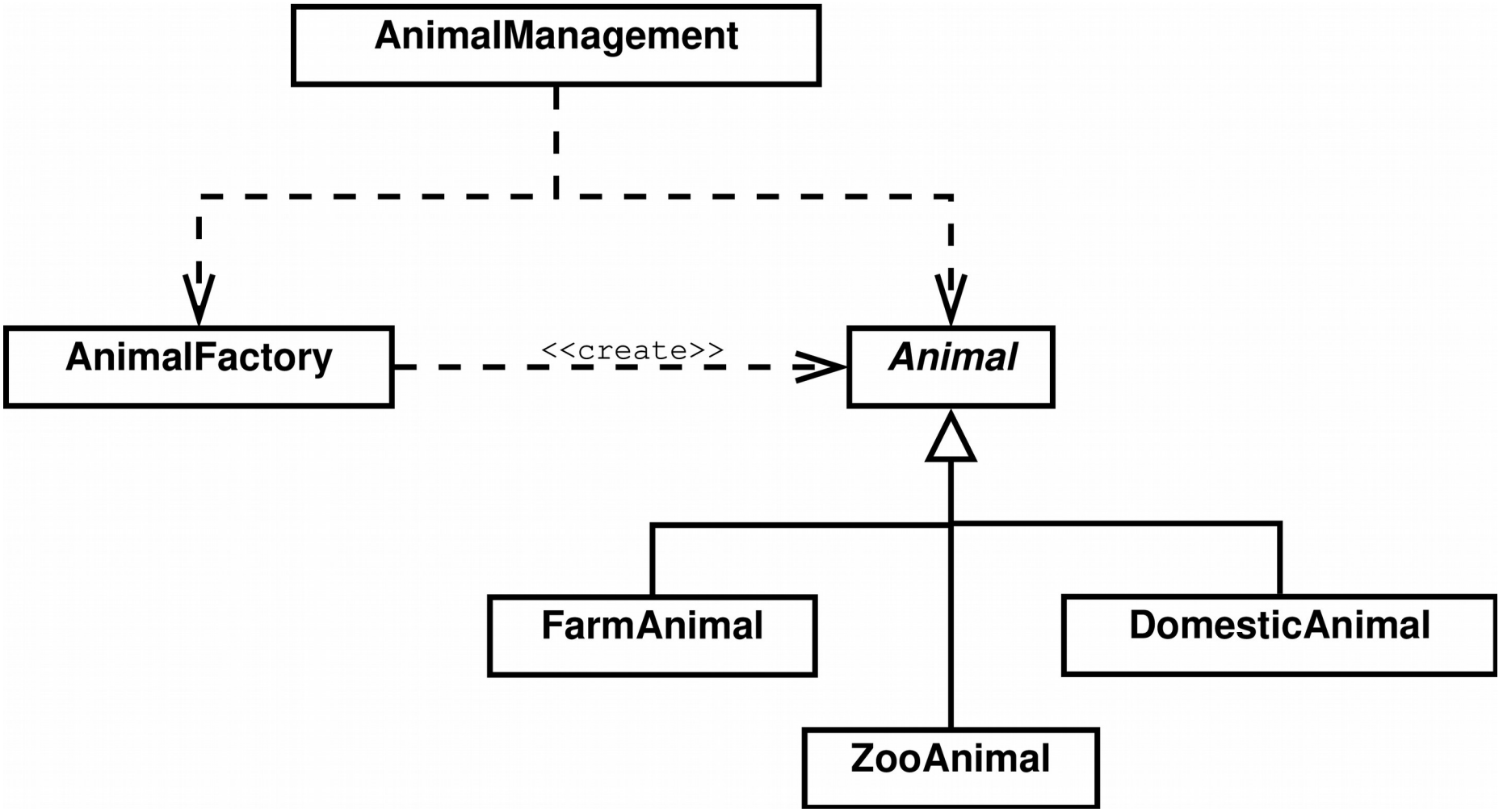
3.3.4 Factory

- Creational design pattern
- Encapsulates the creation of derived objects
  - factory creates derived class object and returns it to client class
  - client class treats derived class object as base class object
  - client class does **not** know the type of derived class object
  - base class is often abstract
    - it only serves to provide a generic interface to the client class

Factory (cont.)



Factory Example



3.3.5 Anti-Patterns

- What is an anti-pattern?
  - a common bad programming habit
    - too many kinds of anti-patterns to count
  - very common: the Blob
    - one class that contains most of the program functionality
      - this can be a danger of using Façade
    - also known as the *God object*