Design Pattern

(OOProg chapter 3)

Peter Levinsky, IT Roskilde

02.04.2025

S O L I D

- **S** Single Responsibility -> High cohesion for classes
- **O** Open / Closed -> open for extensions
- **L** Liskov Substitution
  -> Subclasses ‘same’ behaviour e.g. pre- and post conditions
- **I** Interface Segregation -> Separate interfaces (minimize)
- **D** Dependency Injection/Inversion -> parameter, methods, objects

Design Pattern - Description

Name – common term – a technical term/concepts among programmers

Problem – description of the problem

Solution – Only! A Design solution (UML diagrams)

Design Pattern – GRASP (General Responsibility Assignment Software Patterns)

- Information Expert
- Creator Pattern
- Controller
- Low Coupling
- High Cohesion

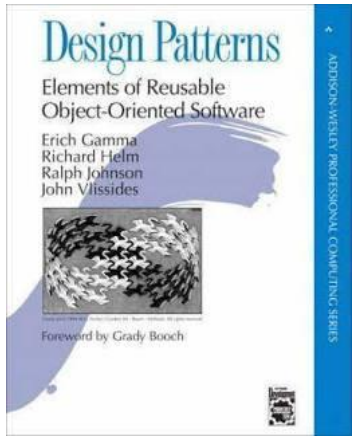
Design Pattern – other patterns from 1st year

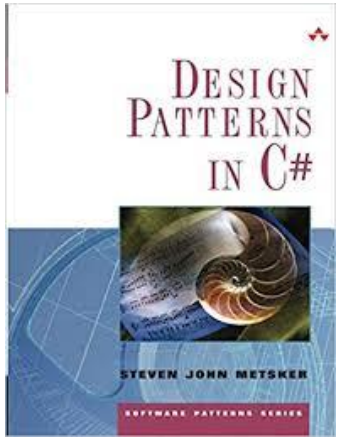
- Singleton - only one object
- Controller - PageModel

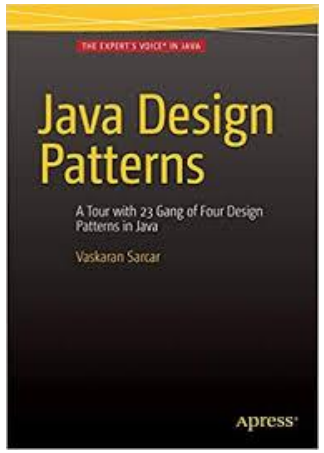
Patterns from this course

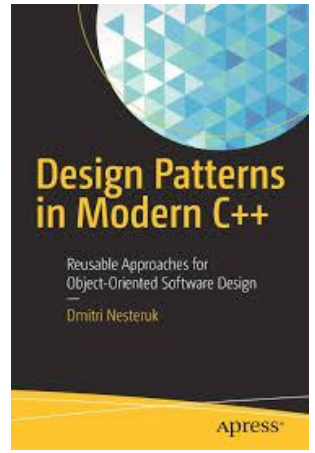
- Template - reuse of code
- State - different behaviour depending on states

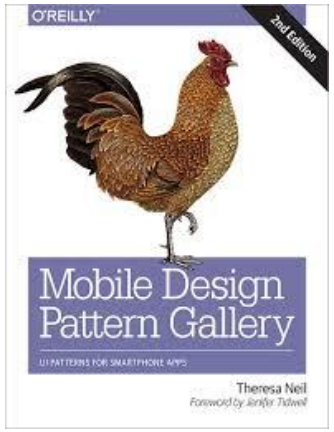
Books of Design Patterns

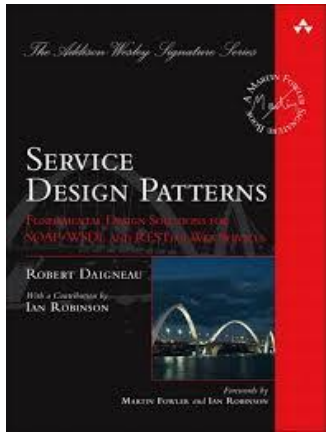


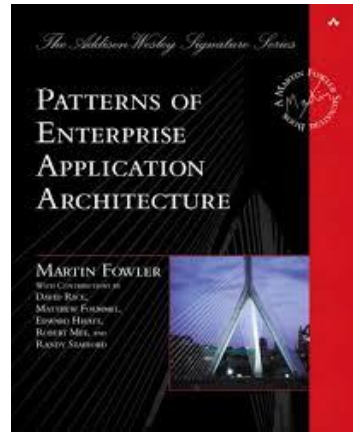










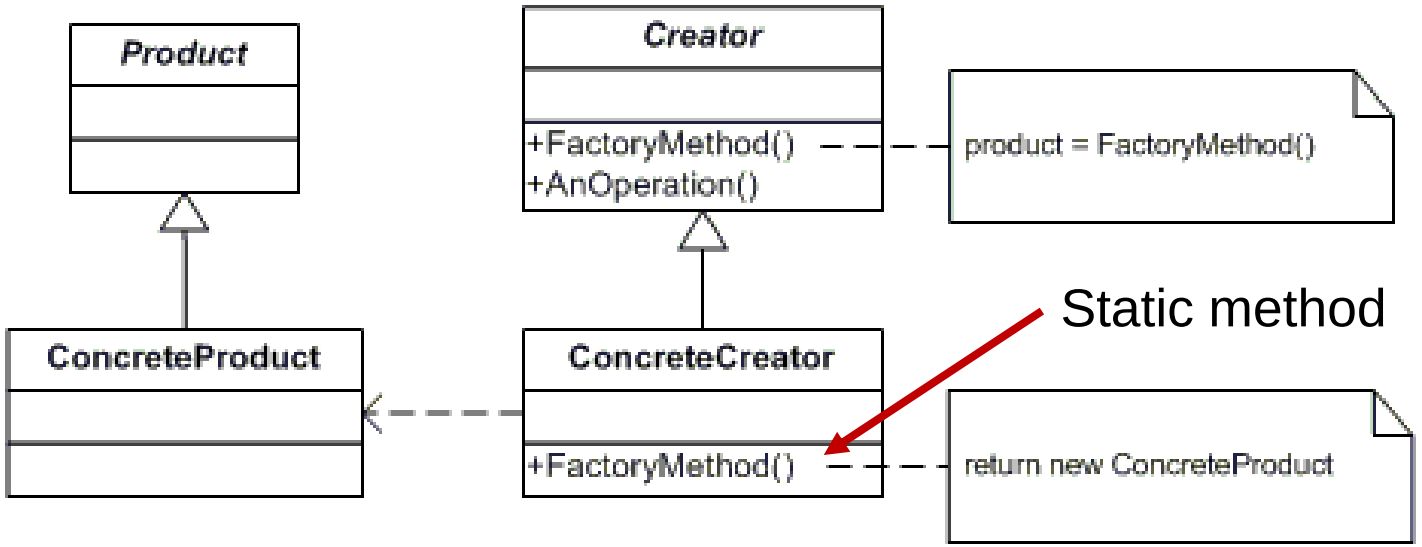


Design Pattern – Categories

- **Creational Patterns**
  - Factory, Abstract Factory, Singleton …
- **Structural Patterns**
  - Adaptor, Proxy, Facade, Decorator …
- **Behavioral Patterns**
  - Observer, Template, Strategy, State …
- **Concurrency patterns**
  - Monitor, Lock, Thread Pool

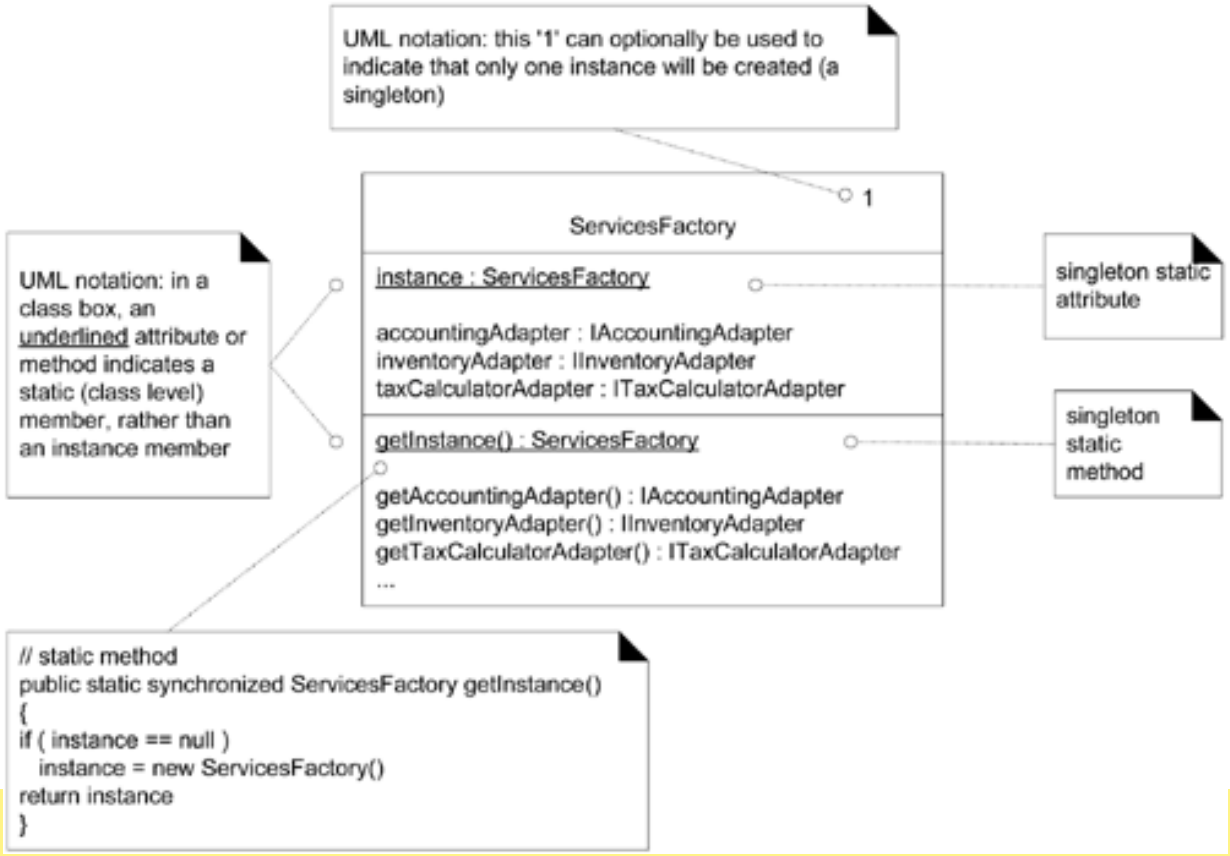
Creational Patterns - Factory

- **Factory**
  - Problem: Who should be responsible for creating objects when there are special considerations, such as complex creation logic, a desire to separate the creation responsibilities for better cohesion, and so forth?
  - Solution:



Creational Patterns - Singleton

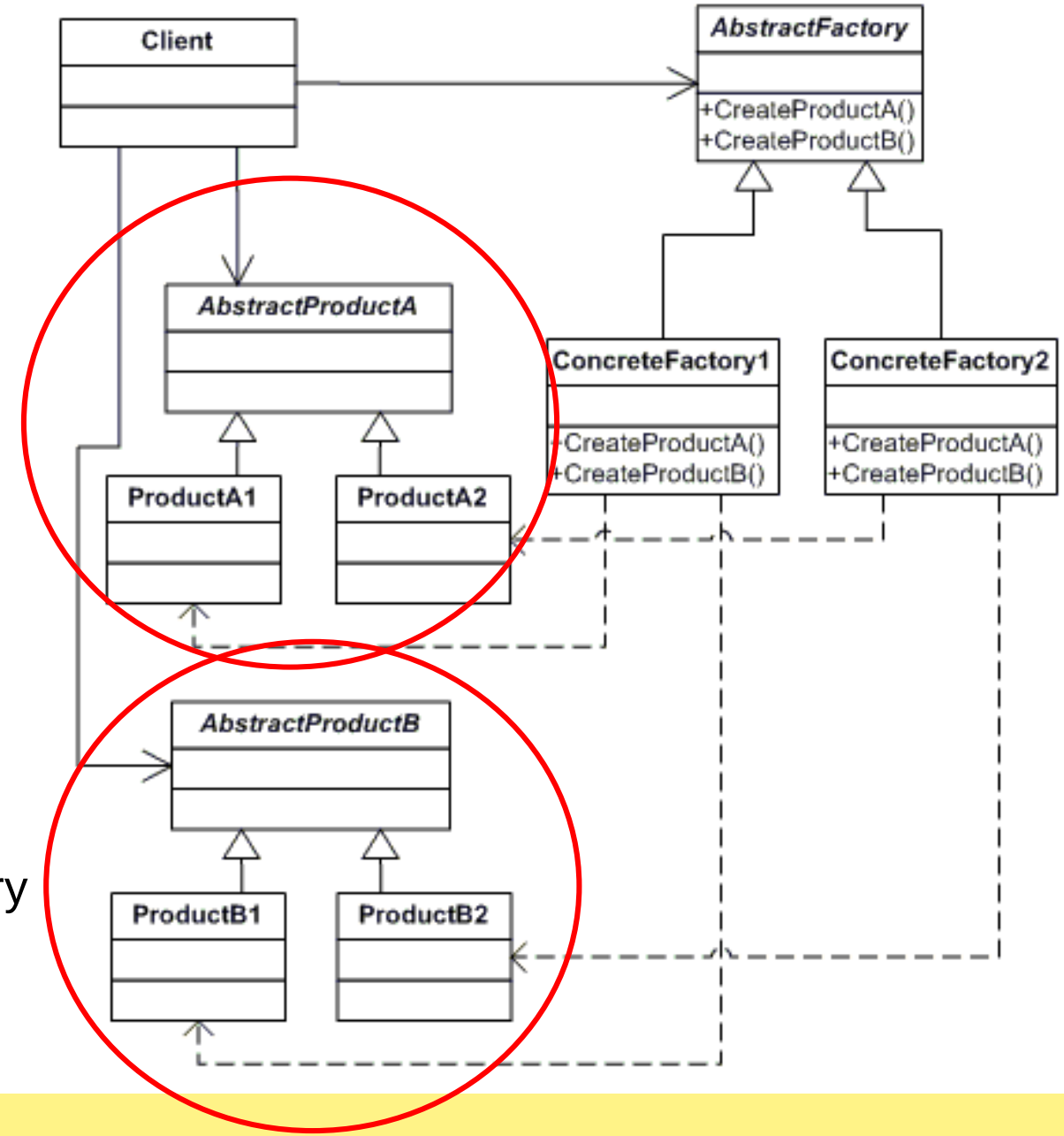
- **Singleton**
  - Problem: Exactly one instance of a class is allowed.
  - Solution:



Abstract Factory

One set of factory

Another set of factory



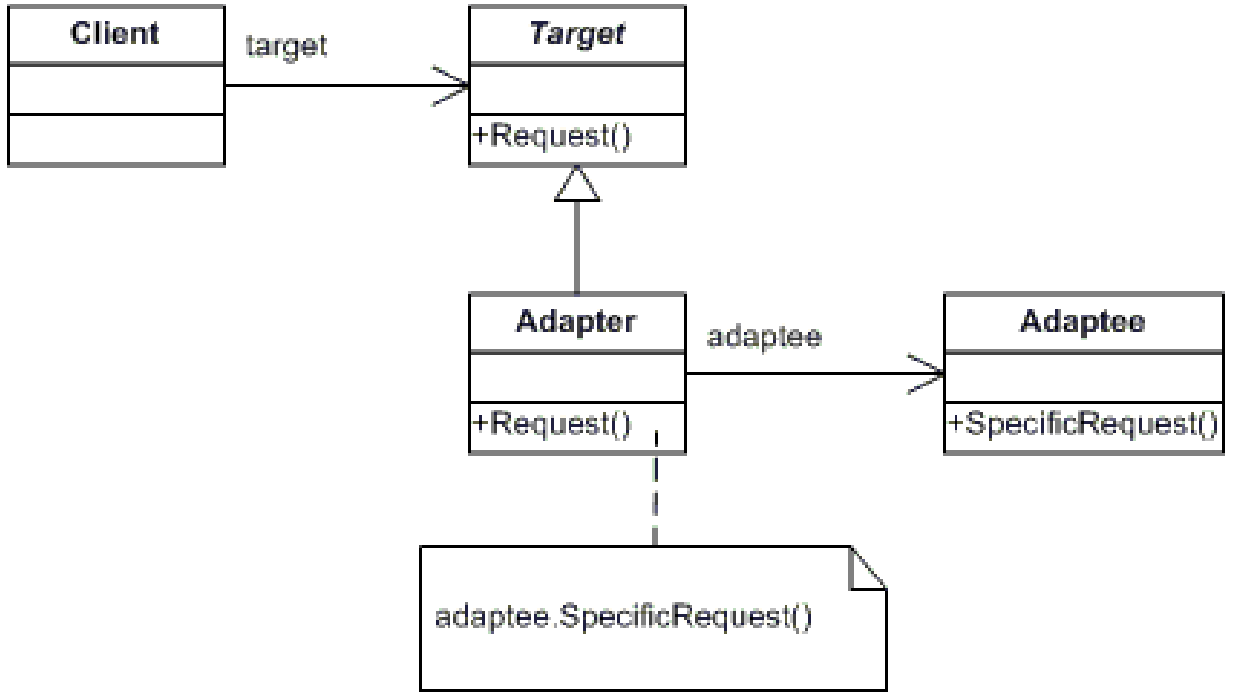
Demo

- Demo af Factory, Singleton og Abstract Factory

Structural Patterns - Adaptor

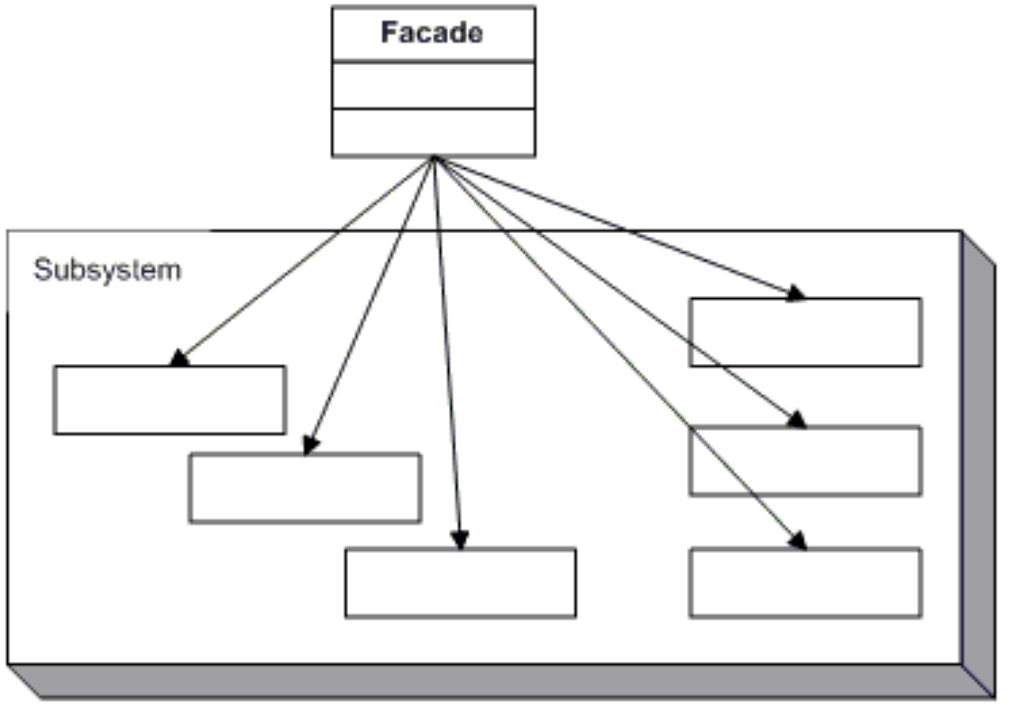
- **Adaptor**
  - Problem: How to resolve incompatible interfaces, or provide a stable interface to similar components with different interfaces?

- Solution:



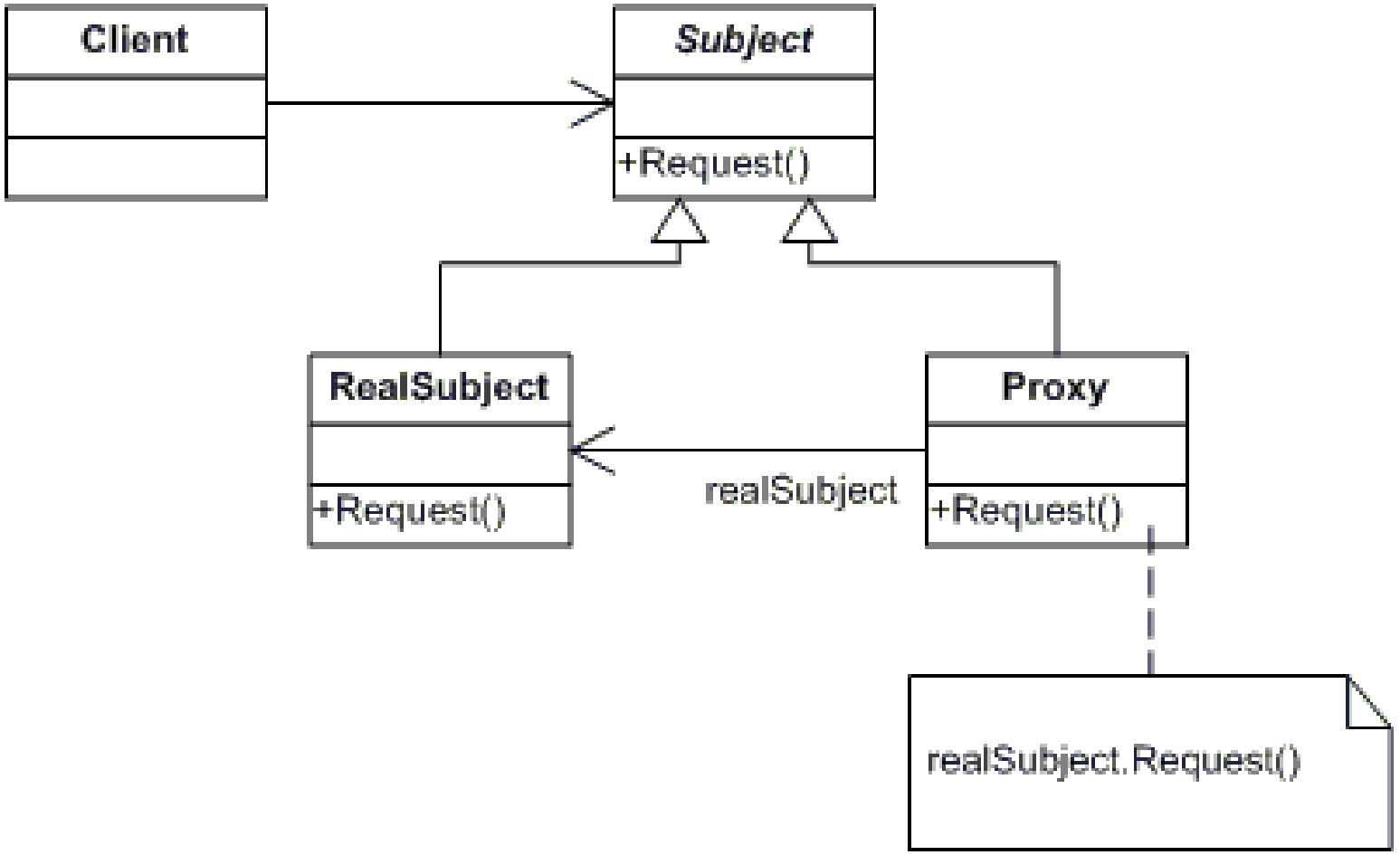
Structural Patterns - Facade

- **Facade**
  - Problem: A common, unified interface to a disparate set of implementations or Interfaces such as within a subsystem is required.
  - Solution:



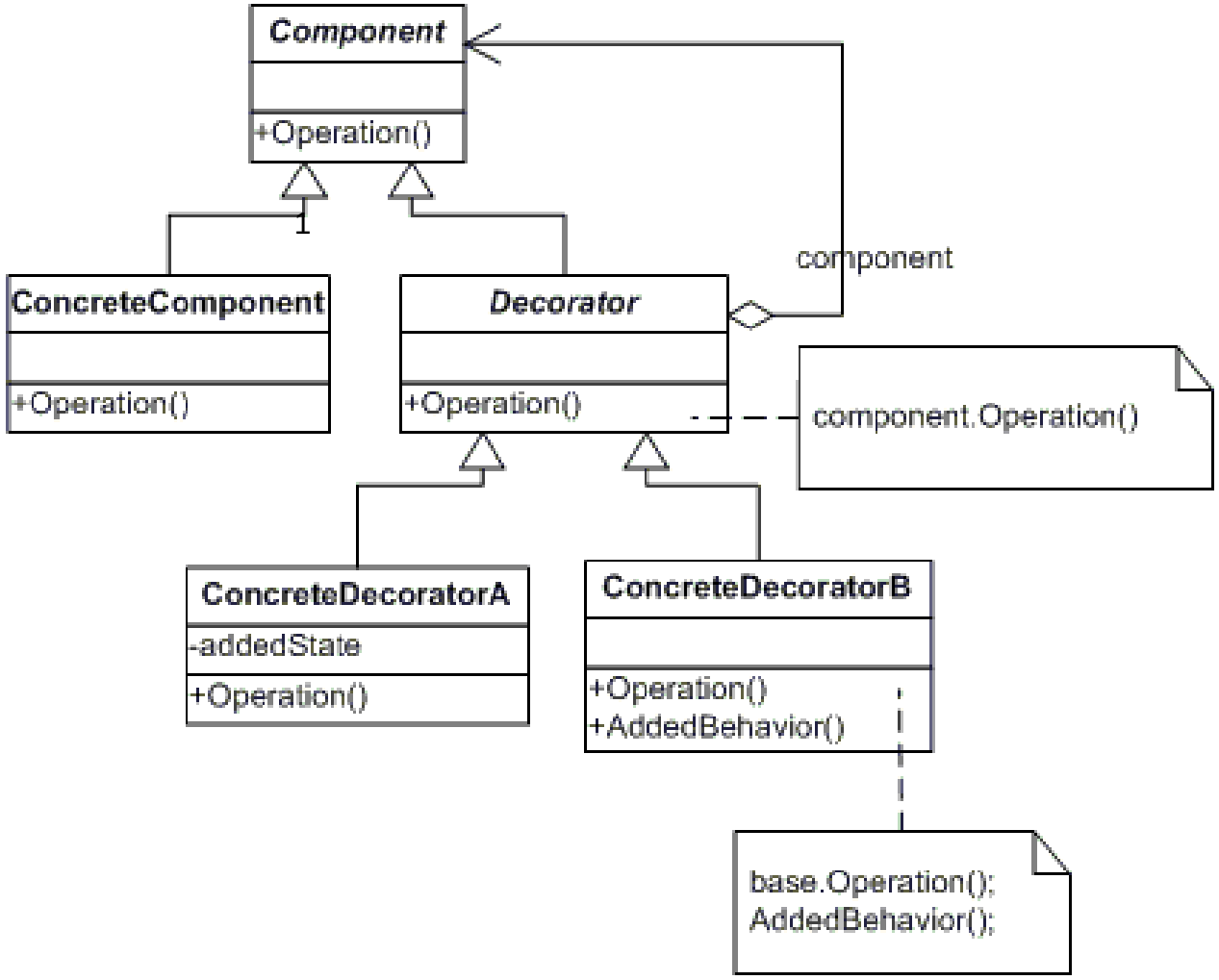
Structural Patterns - Proxy

- **Proxy**
  - Problem: How to provide a placeholder for another object to control access to it.
  - Solution:



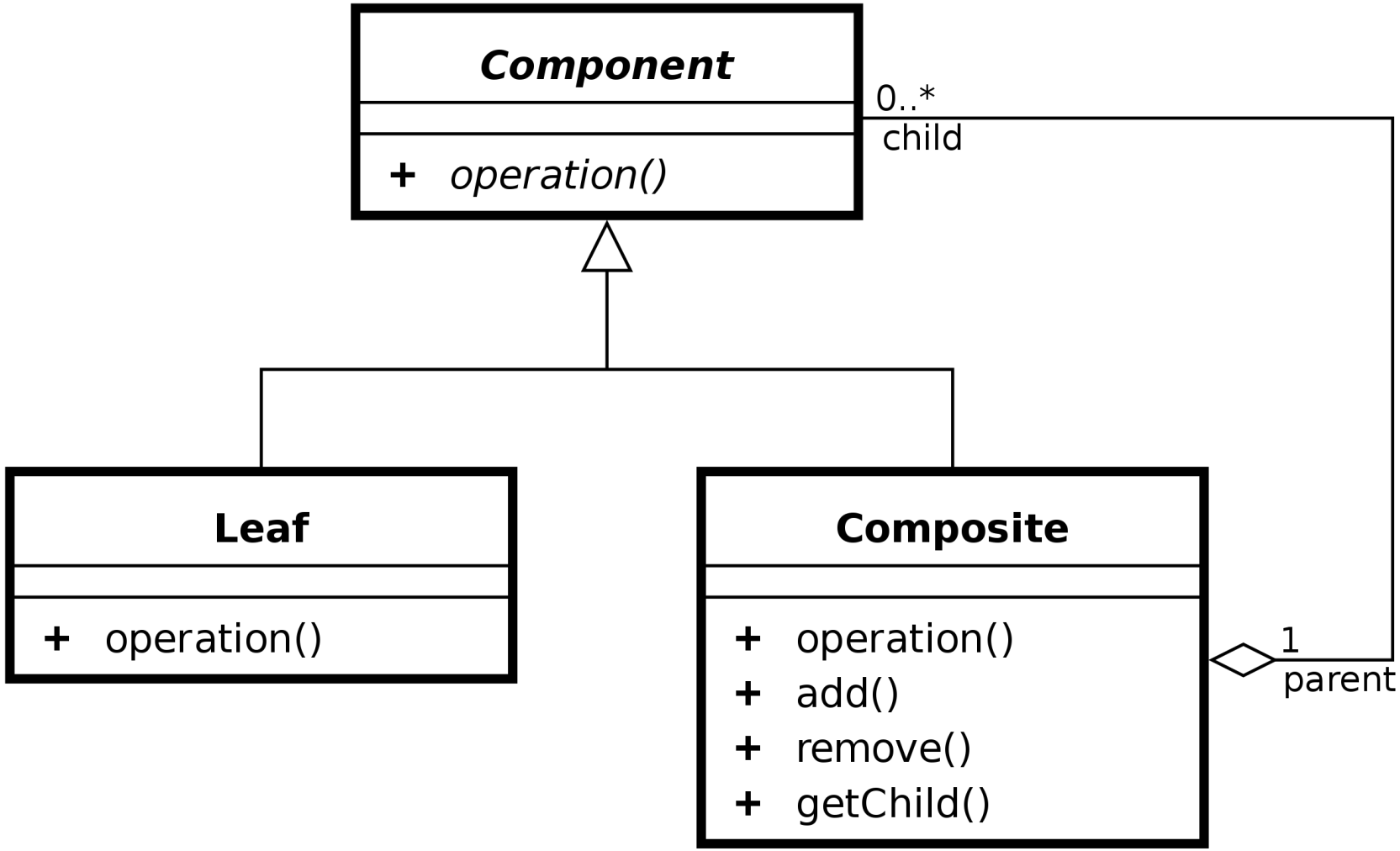
Structural Patterns - Decorator

- **Decorator**
  - Problem: How to Attach additional responsibilities to an object dynamically
  - Solution:



Structural Patterns - Composite

- **Composite**
  - Problem: How to represented a part-whole hierarchy so that clients can treat part and whole objects uniformly.
  - Solution:



Demo

- Adaptor, Proxy, Facade, Decorator, Composite

- Training: Exercises 3.1 (Factory), 3.2(Abstract Factory), 3.3 (Adaptor)
- Mandatory Assignment