Design Pattern

(OOProg chapter 3)

Peter Levinsky, IT Roskilde

22.03.2021

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 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

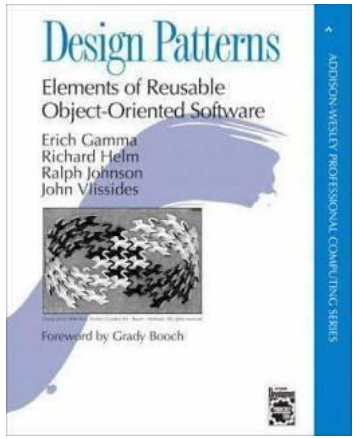
Design Pattern – other patterns from 1st year

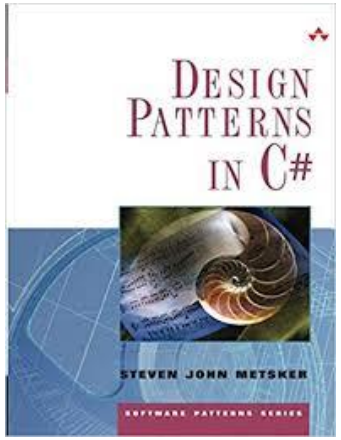
- Singleton - only one object
- Observer - ensure low coupling – especially between view and viewmodel
- Controller - ViewModel

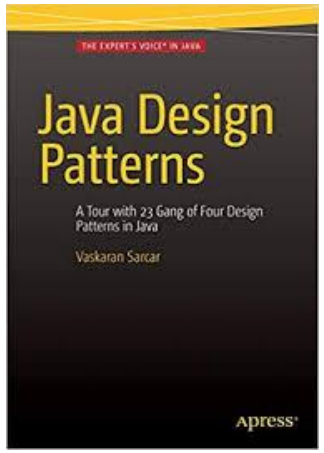
Patterns from this course

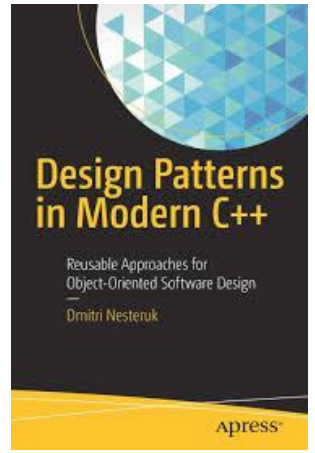
- Template - reuse og code
- State - different behaviour depending on states

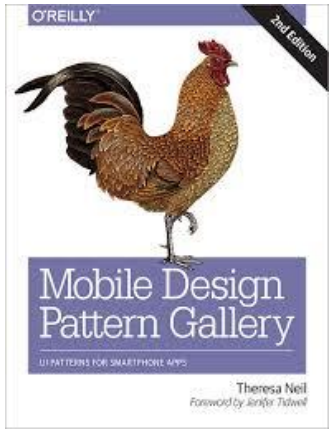
Books of Design Patterns

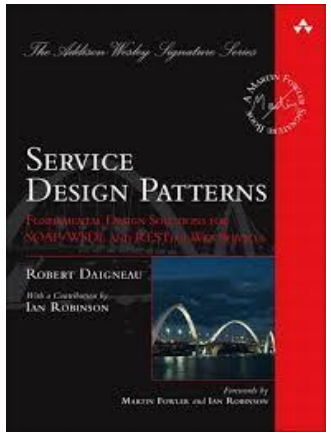


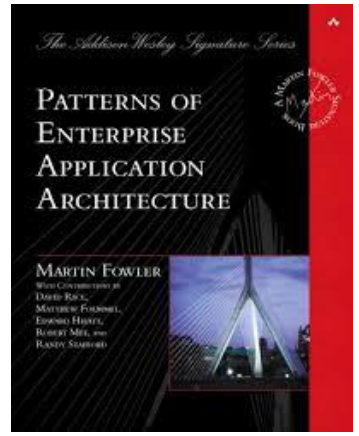










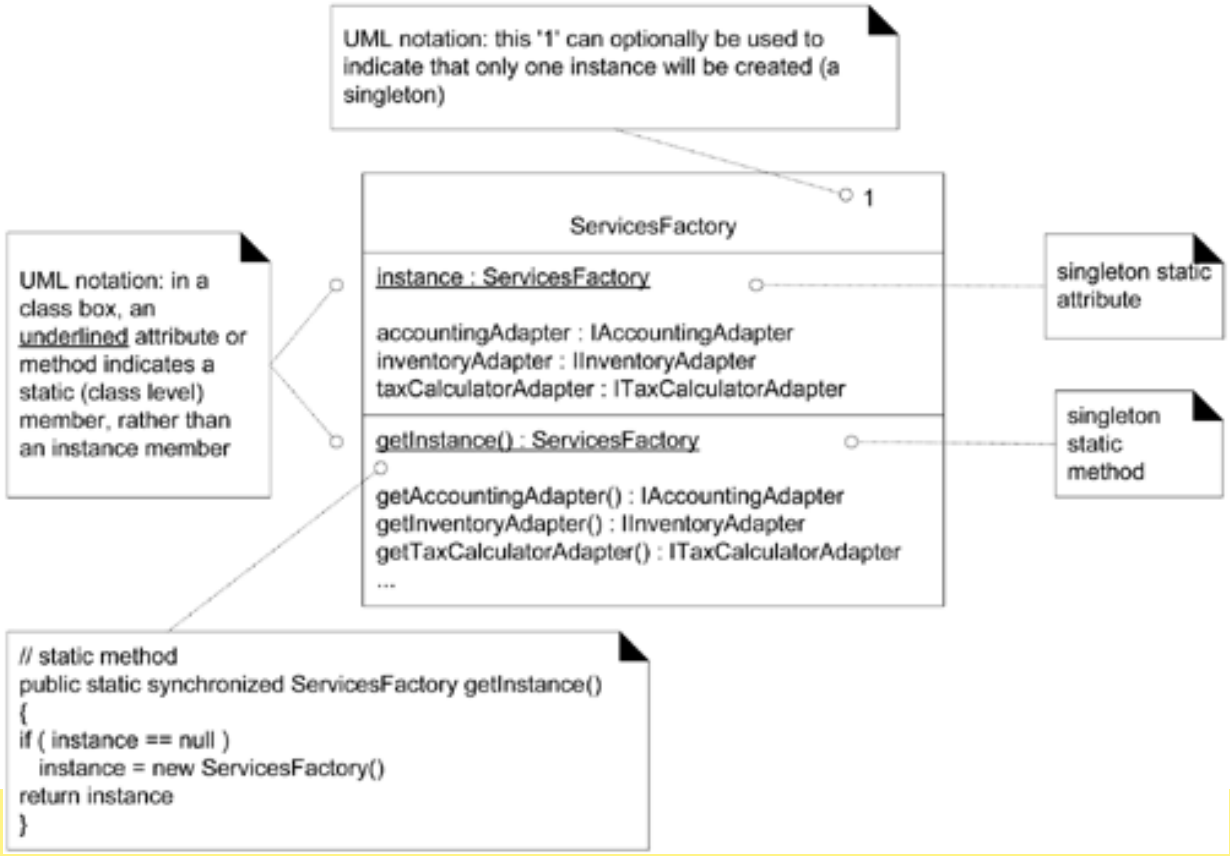


Design Pattern – Categories

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

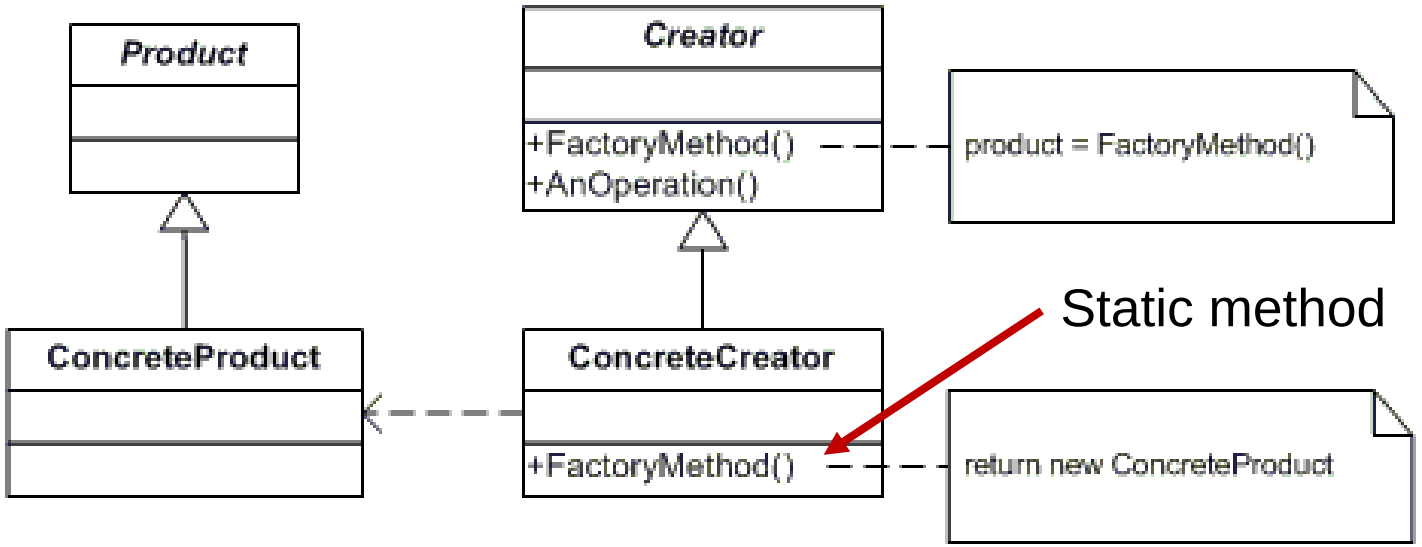
Design Pattern – Creational Patterns

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

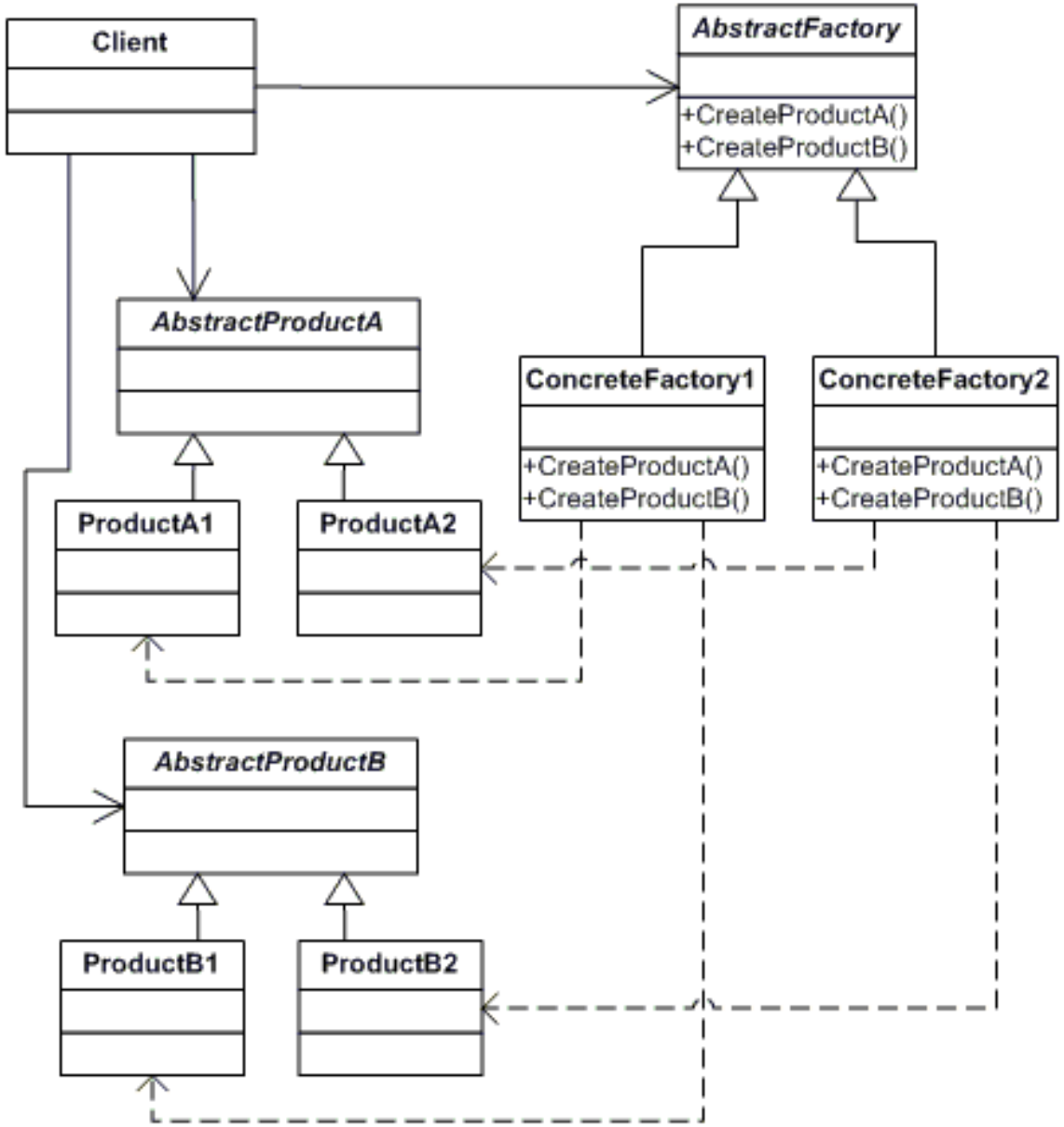


Design Pattern – Creational Patterns

- **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:



Abstract Factory



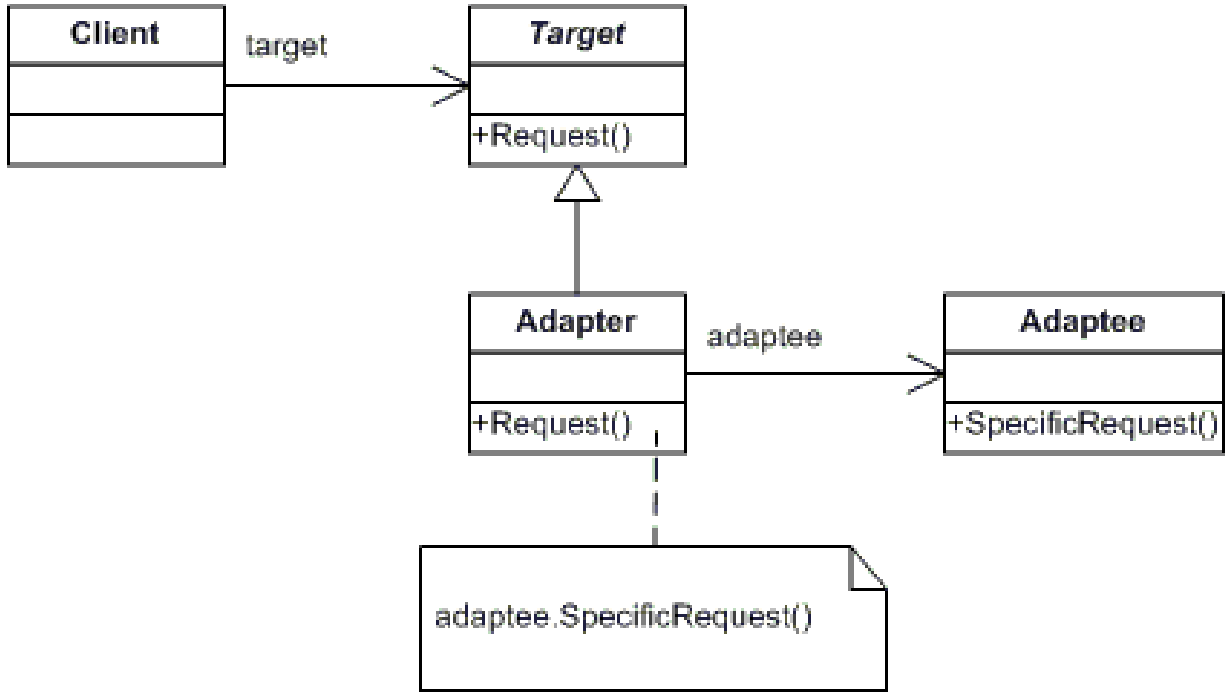
Demo

- Demo af Factory, Singleton og Abstract Factory

Design Pattern – Structural Patterns

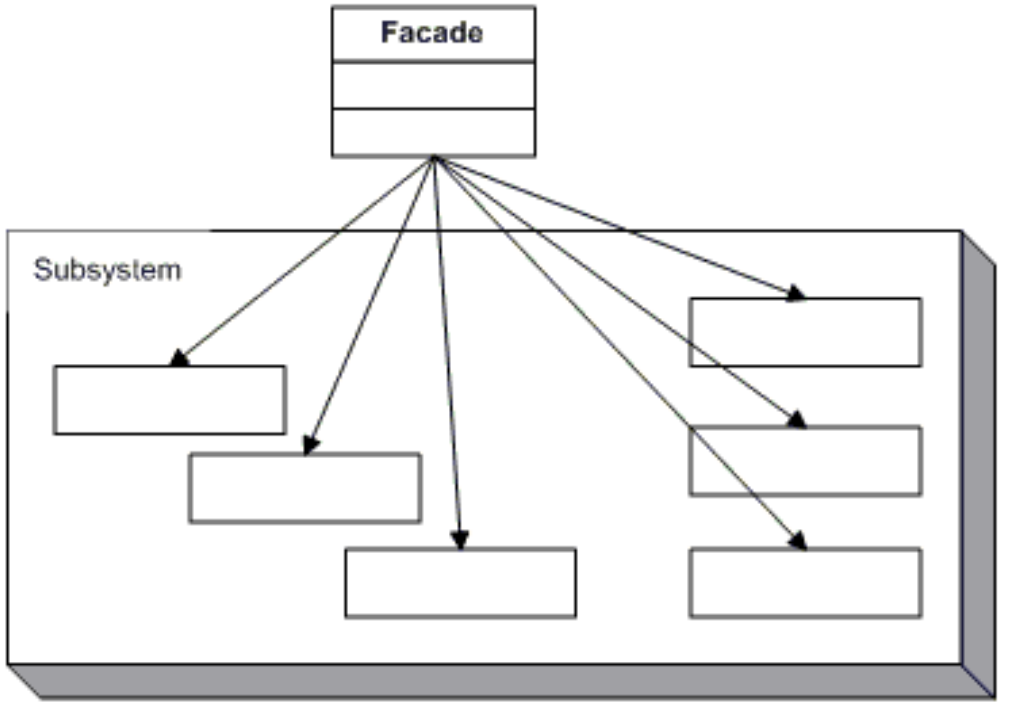
- **Adaptor**
  - Problem: How to resolve incompatible interfaces, or provide a stable interface to similar components with different interfaces?

- Solution:



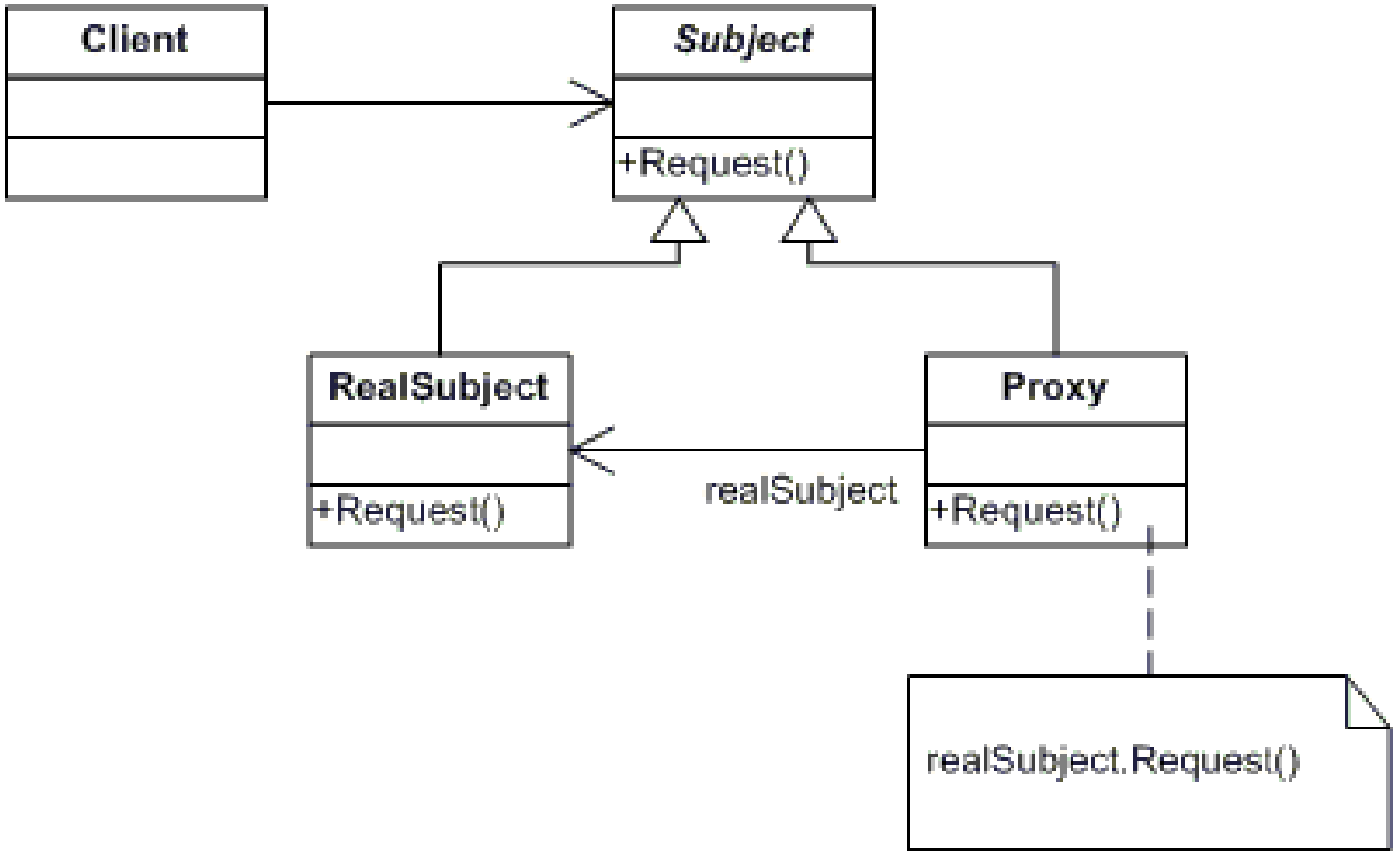
Design Pattern – Structural Patterns

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



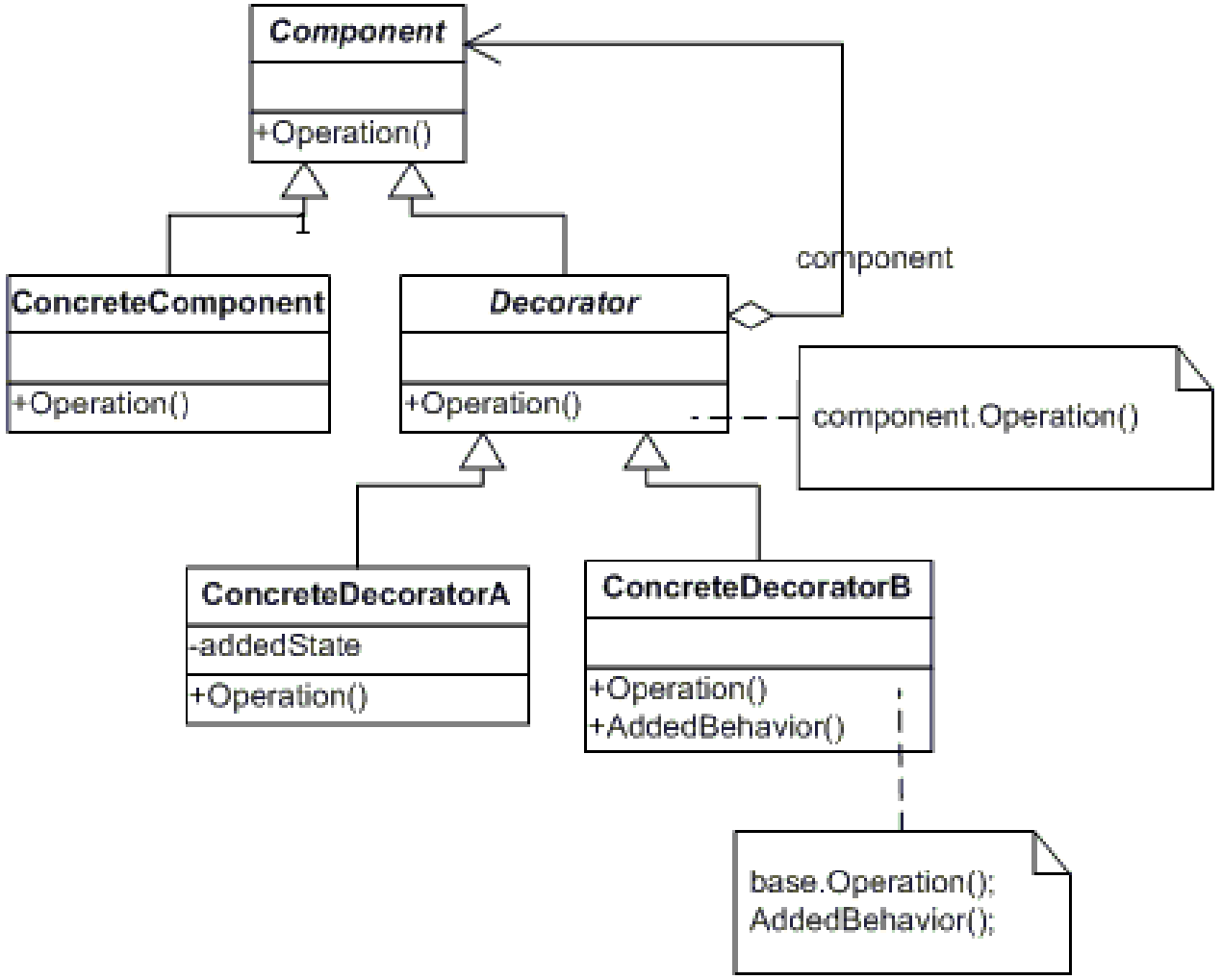
Design Pattern – Structural Patterns

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



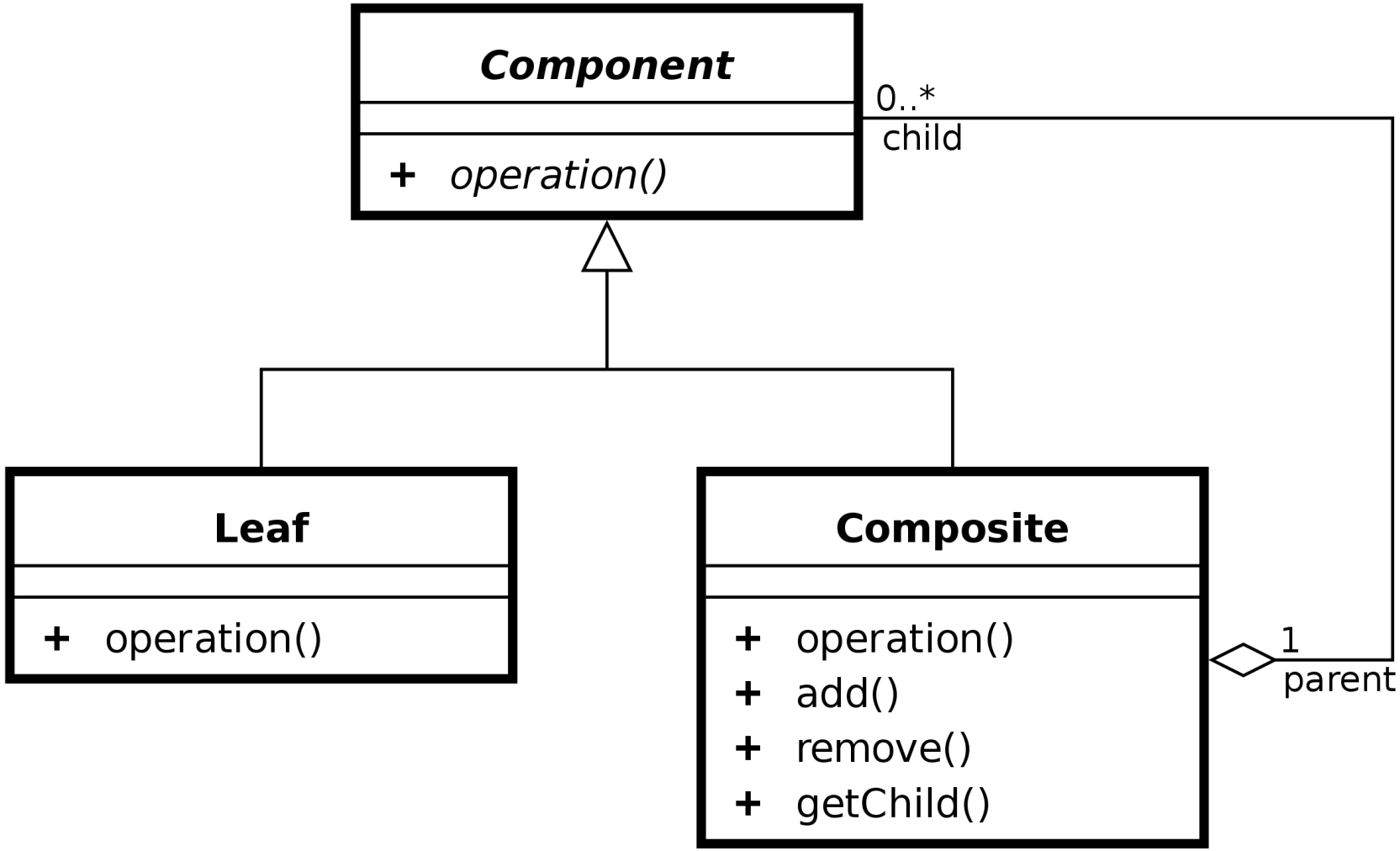
Design Pattern – Structural Patterns

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



Design Pattern – Structural Patterns

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



Demo

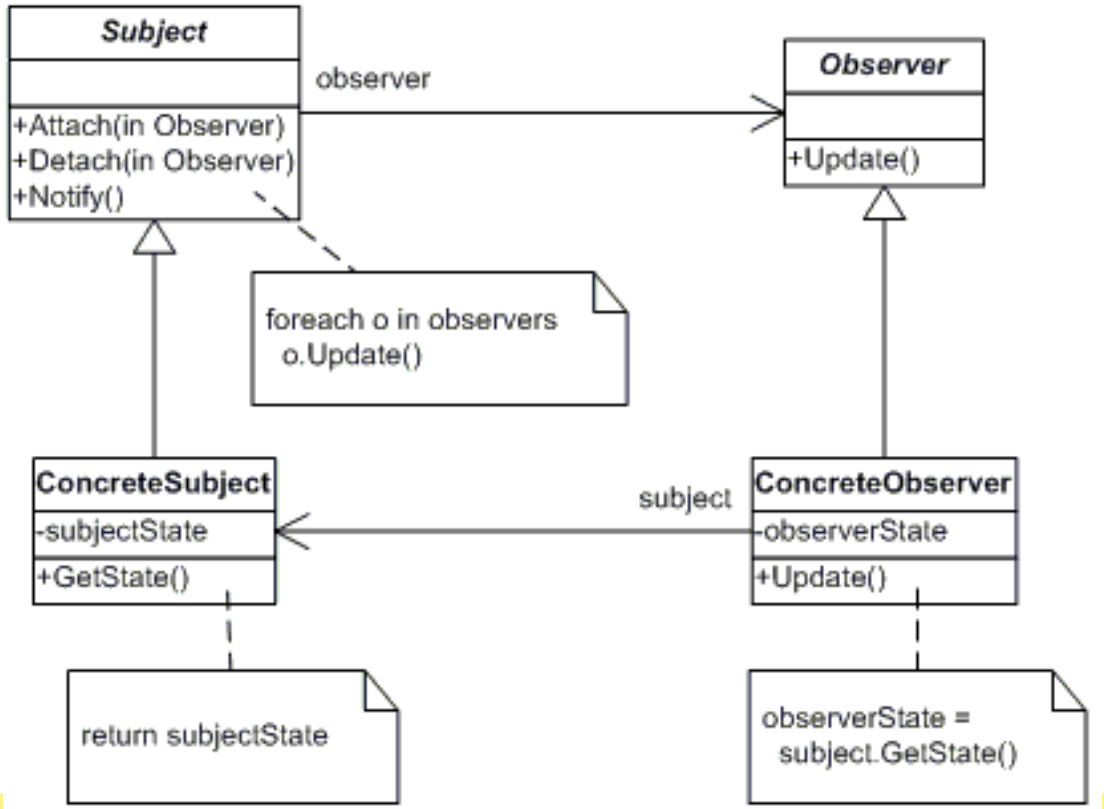
- Adaptor, Proxy, Facade, Decorator, Composite

- Exercises 3.1, 3.2, 3.3
- Extra 3.4

Design Pattern – Behavioural Patterns

- **Observer** *(known fom 1st year)*
  Problem: How to handle different kinds of subscriber objects are interested in the state changes or events of a publisher object

- Solution:



Design Pattern – Behavioural Patterns

- **Observer** - the C# way of doing it

The one that Observe

```
…
XX x = new XX();

// Register as observer
x.PropertyChanged += Update;




….
protected void Update(object sender,
                      PropertyChangedEventArgs arg)
{
. . .
}
```

To be Observered

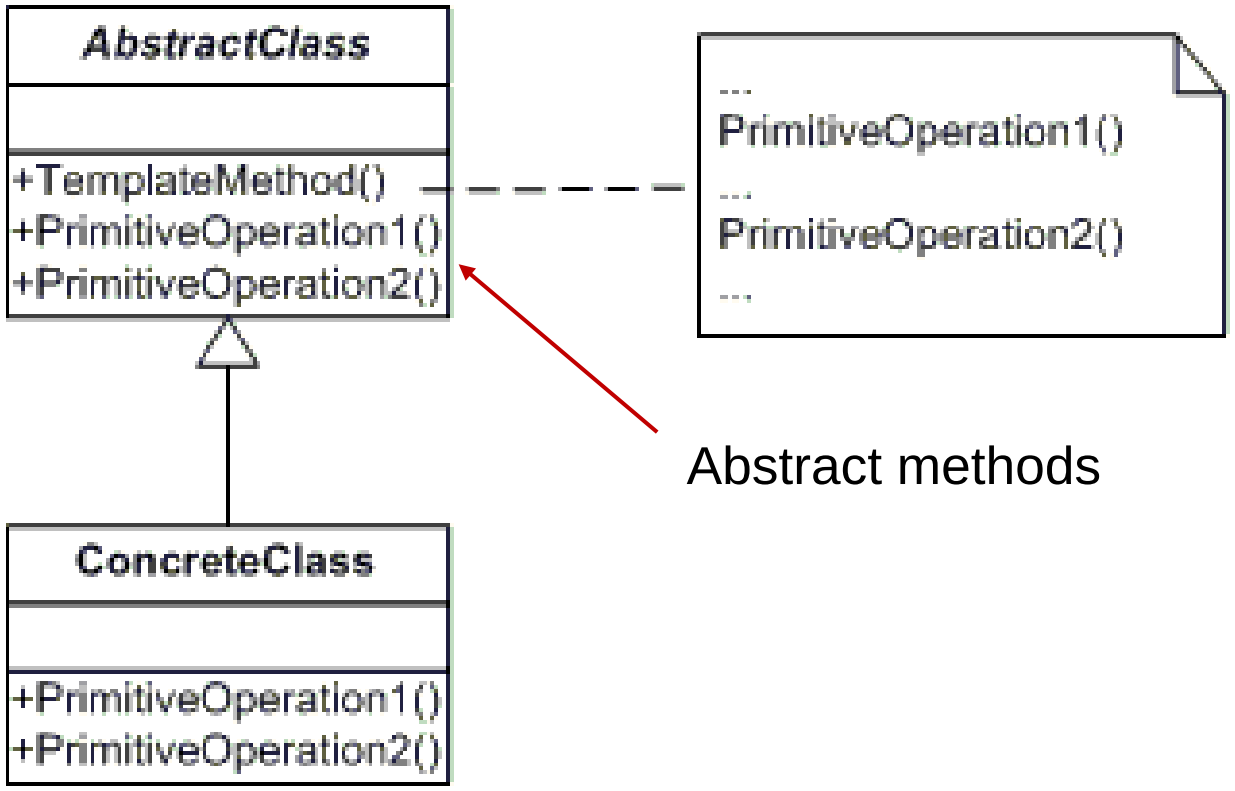
```
Class XX : INotifyPropertyChanged
{
. . .
// Attach, Deattach
public event PropertyChangedEventHandler PropertyChanged;

// notify
protected virtual void OnPropertyChanged(string propertyName)
{
    PropertyChanged?.Invoke(this,
                            new PropertyChangedEventArgs(propertyName));
}
}
```

Design Pattern – Behavioural Patterns

- **Template** **(seen at the TCP server generalisation)**
  Problem: How to reuse a skeleton of an algorithm in an operation

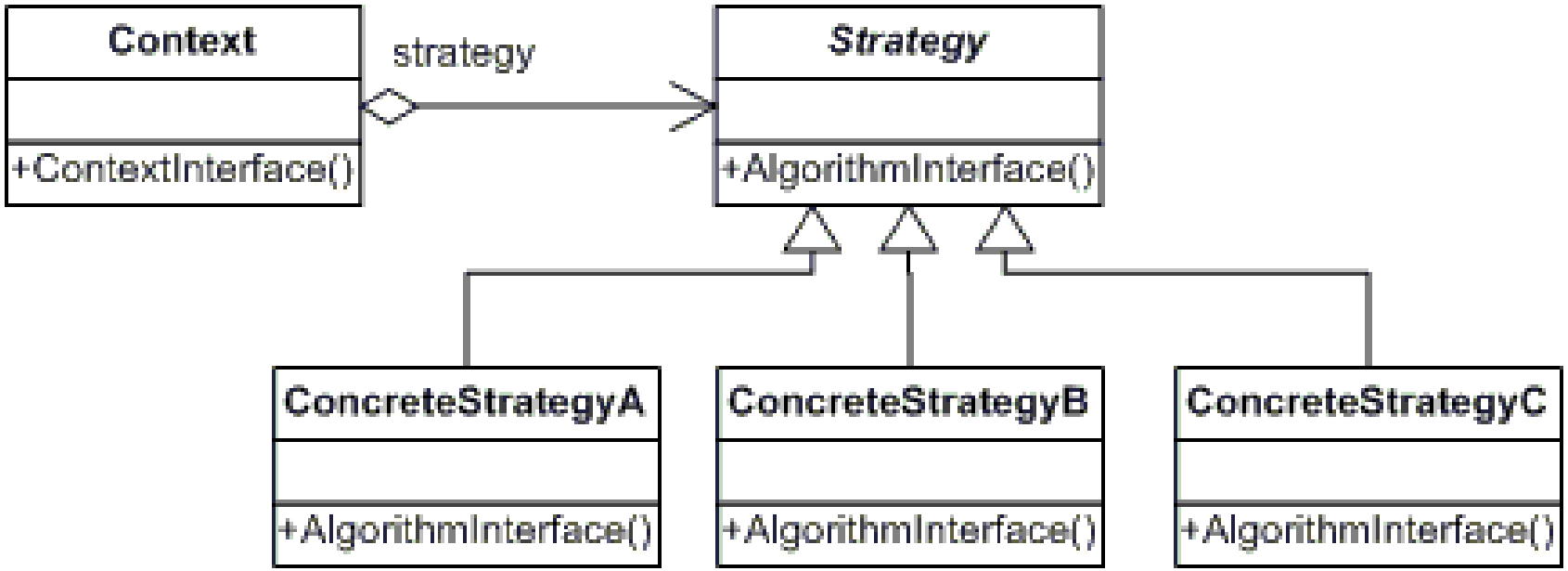
- Solution:



Design Pattern – Behavioural Patterns

- **Strategy**
  Problem: How to interchange part of algorithm dynamically

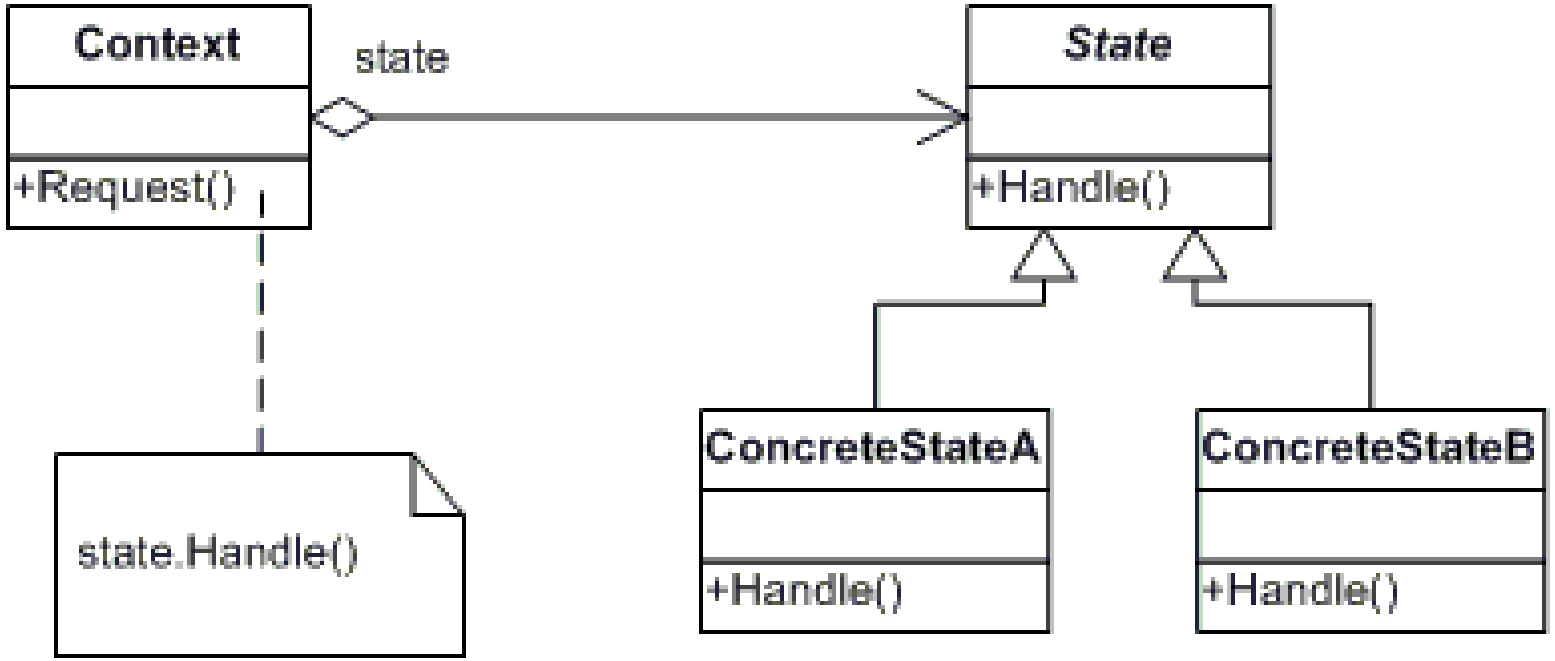
- Solution:



Design Pattern – Behavioural Patterns

- **State**
  Problem: How to Allow an object to alter its behaviour when its internal state changes

- Solution:



Demo

- Demo of Decorator, Observer, Template og Strategy

- Exercises: 3.7, 3.9
- ***And of cause the mandatory assignment***