Design Pattern

(OOProg chapter 3)

Peter Levinsky, IT Roskilde

14.03.2024

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

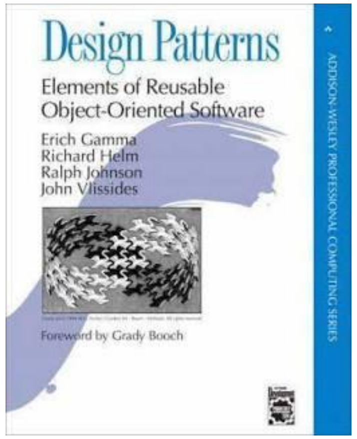
Design Pattern – other patterns from 1st year

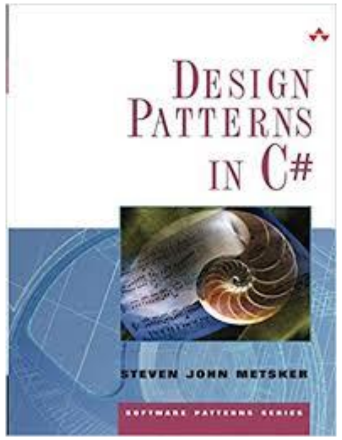
• Singleton	- only one object
• Controller	- PageModel

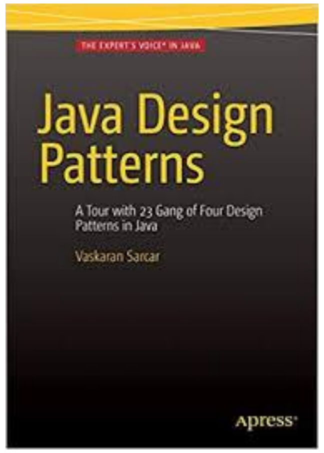
Patterns from this course

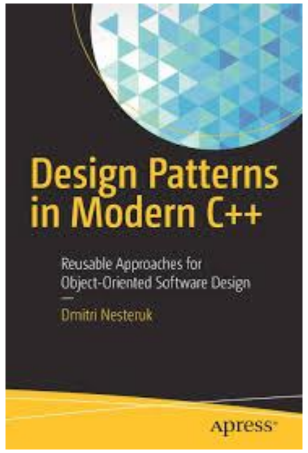
• Template	- reuse og code
• State	- different behaviour depending on states

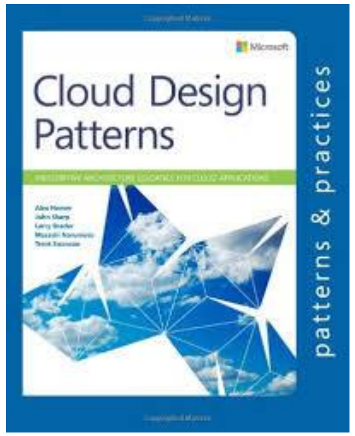
Books of Design Patterns


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

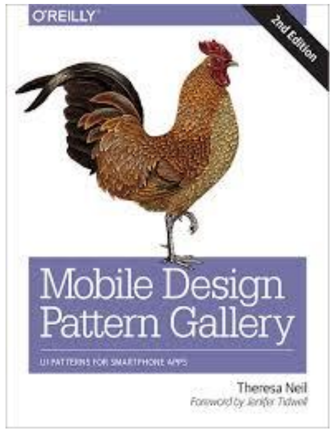


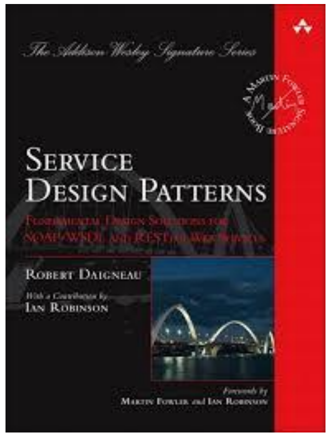

THE EXPERT'S VOICE IN JAVA
Java Design Patterns
A Tour with 23 Gang of Four Design Patterns in Java
Vaskaran Sarcar
Apress

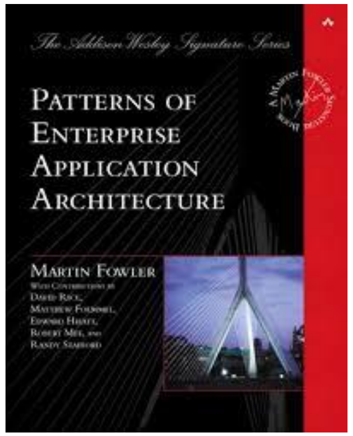

Design Patterns in Modern C++
Reusable Approaches for Object-Oriented Software Design
Dmitri Nesteruk
Apress



Praseed Pai, Shine Xavier
Foreword by Arun Narayanan
.NET Design Patterns
Explore the world of .NET design patterns and bring the benefits that the right patterns can offer to your toolkit today
Packt


O'REILLY
2nd Edition
Mobile Design Pattern Gallery
UI PATTERNS FOR SMARTPHONE APPS
Theresa Neil
Foreword by Jenifer Tidwell


The Addison-Wesley Signature Series
SERVICE DESIGN PATTERNS
ROBERT DAIGNEAU
IAN ROBINSON
MARTIN FOWLER and IAN ROBINSON


The Addison-Wesley Signature Series
PATTERNS OF ENTERPRISE APPLICATION ARCHITECTURE
MARTIN FOWLER

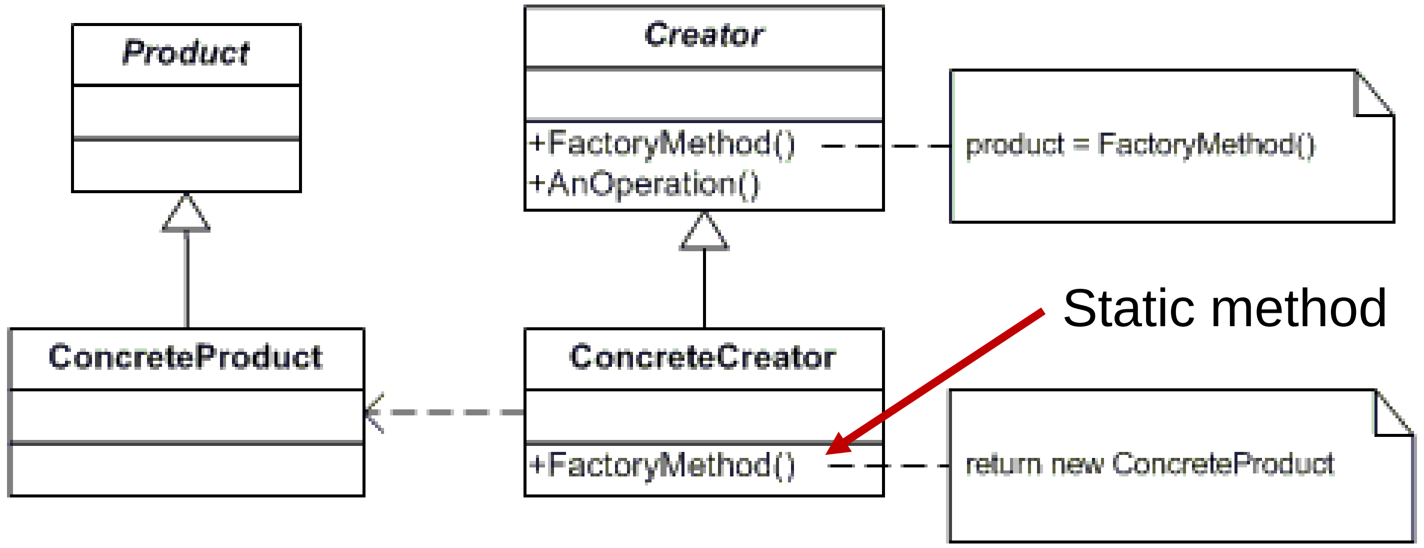
...

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

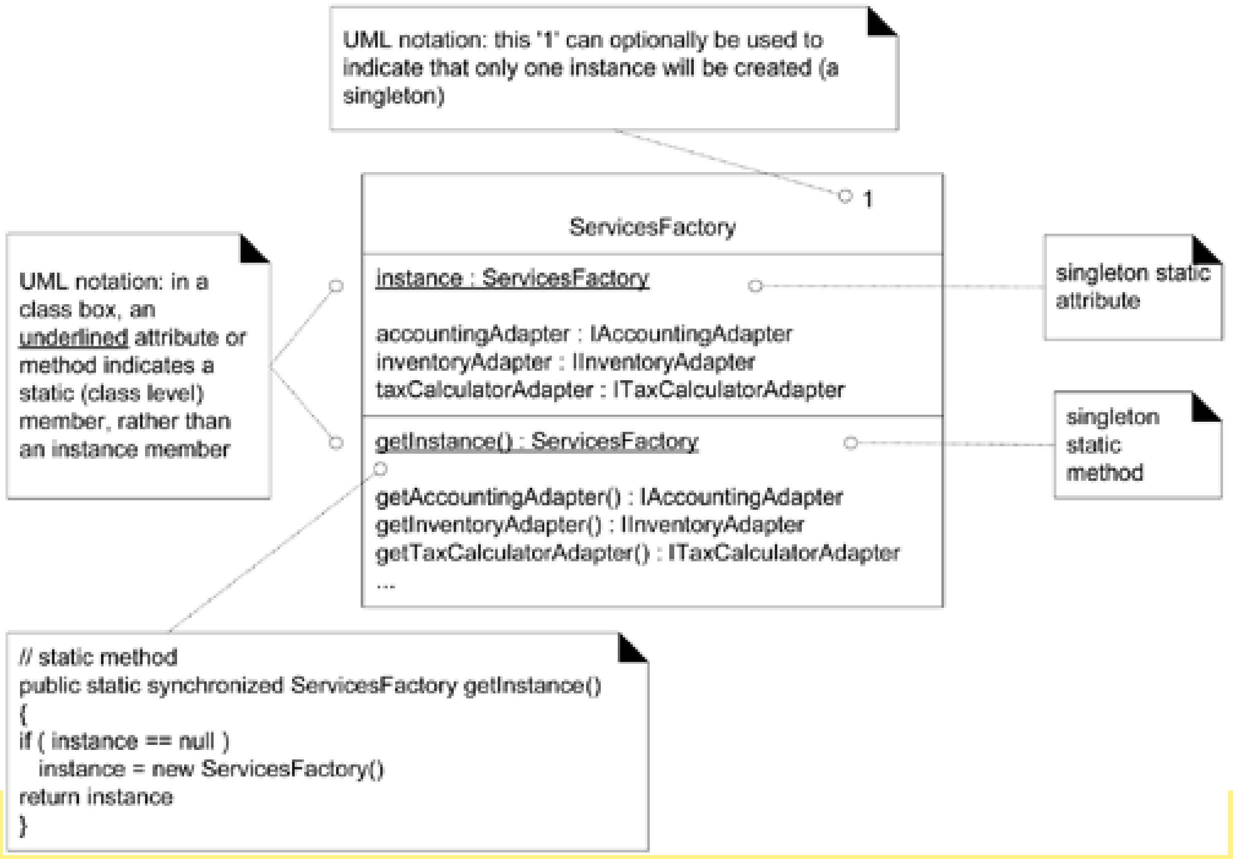
Design Pattern – Creational Patterns

- **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:



Design Pattern – Creational Patterns

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



1
ServicesFactory
instance : ServicesFactory
accountingAdapter : IAccountingAdapter
inventoryAdapter : IInventoryAdapter
taxCalculatorAdapter : ITaxCalculatorAdapter
getInstance() : ServicesFactory
getAccountingAdapter() : IAccountingAdapter
getInventoryAdapter() : IInventoryAdapter
getTaxCalculatorAdapter() : ITaxCalculatorAdapter
...
singleton static attribute
singleton static method

UML notation: this '1' can optionally be used to indicate that only one instance will be created (a singleton)

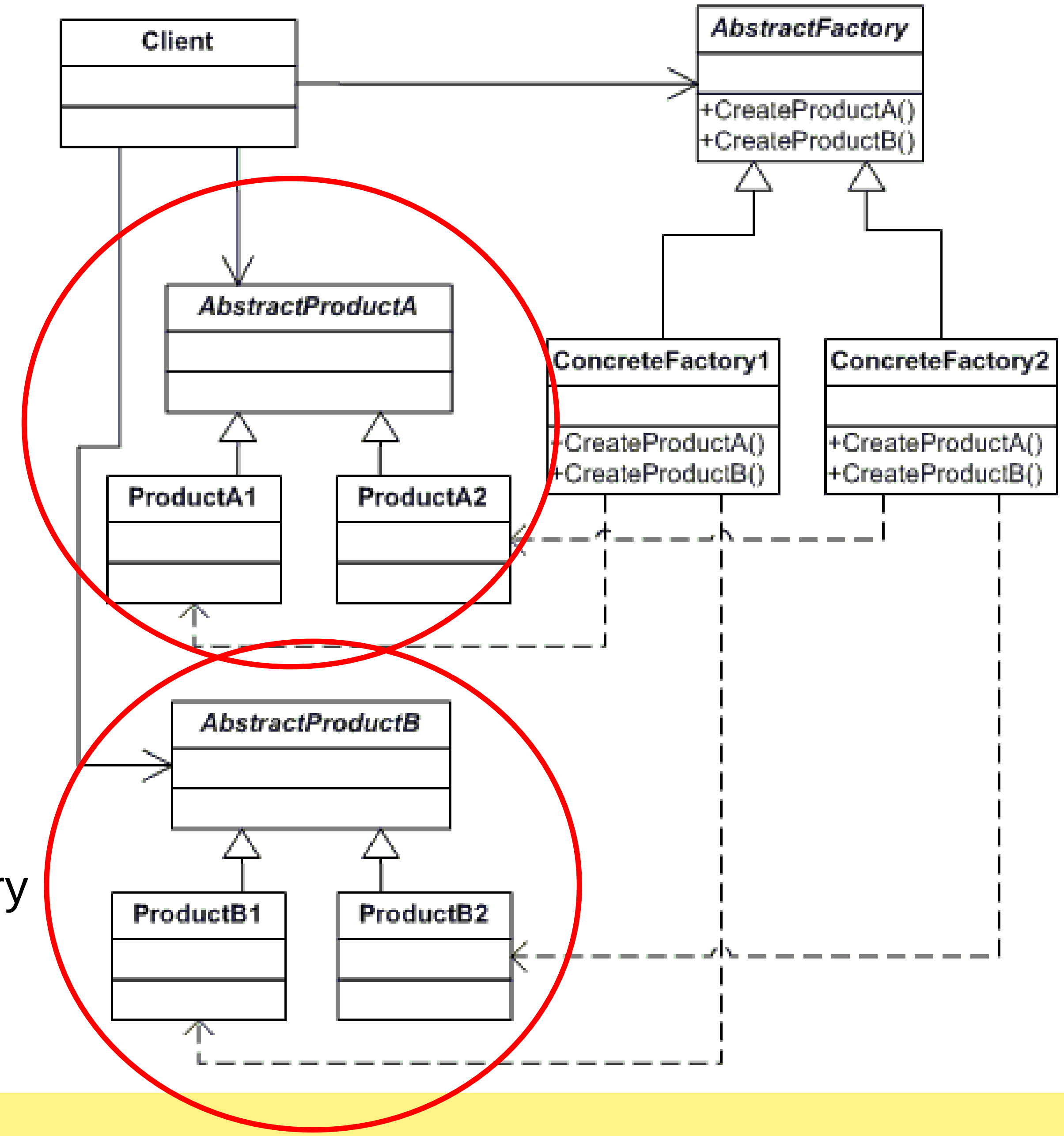
UML notation: in a class box, an underlined attribute or method indicates a static (class level) member, rather than an instance member

```
// static method
public static synchronized ServicesFactory getInstance()
{
if ( instance == null )
   instance = new ServicesFactory()
return instance
}
```

Abstract Factory

One set of factory

Another set of factory

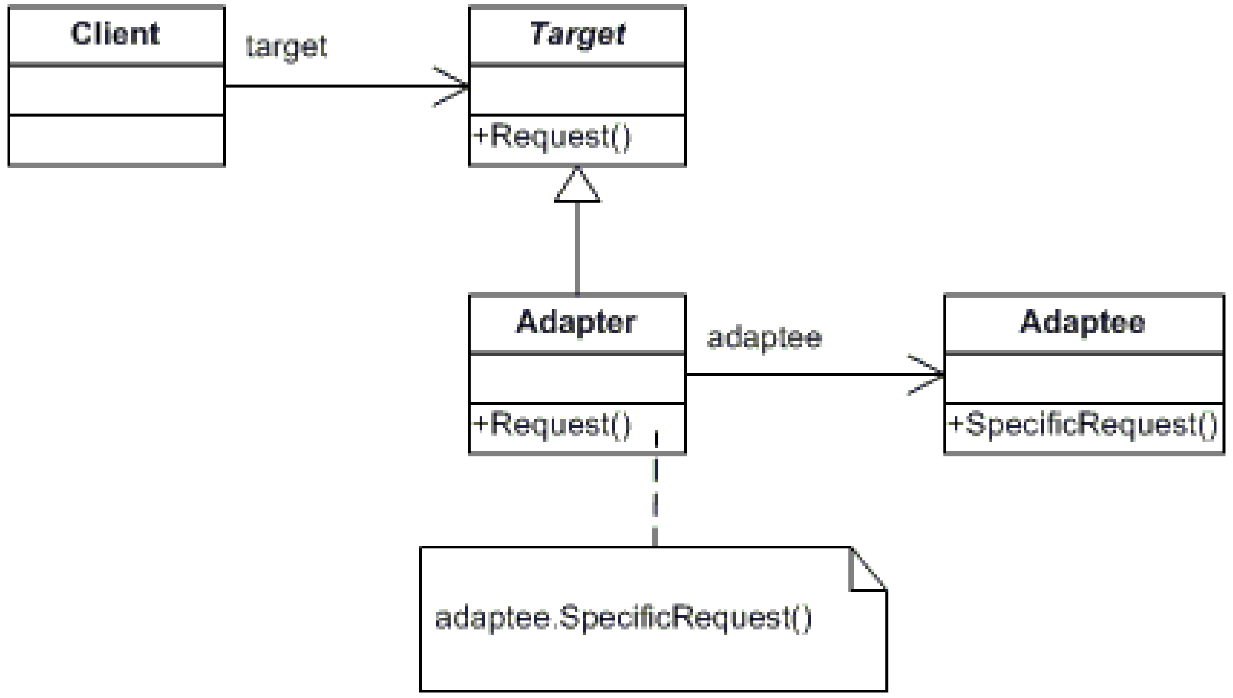


Demo

- Demo af Factory, Singleton og Abstract Factory

Design Pattern – Structural Patterns

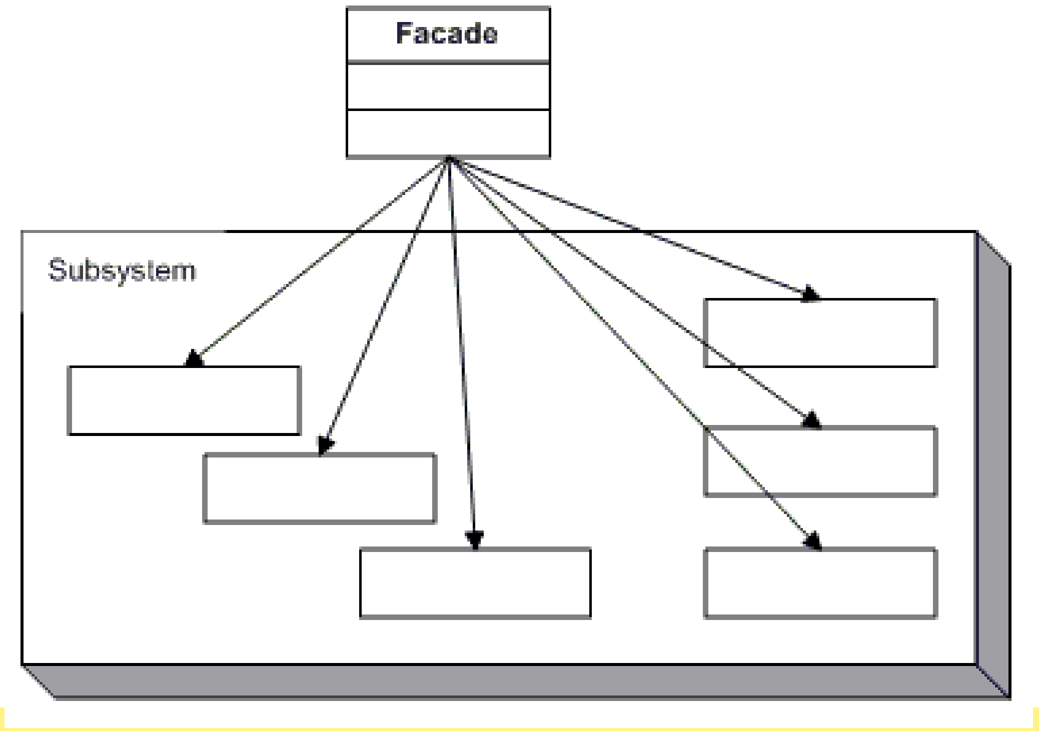
- **Adaptor**
  - Problem: How to resolve incompatible interfaces, or provide a stable interface to similar components with different interfaces?
- Solution:



Design Pattern – Structural Patterns

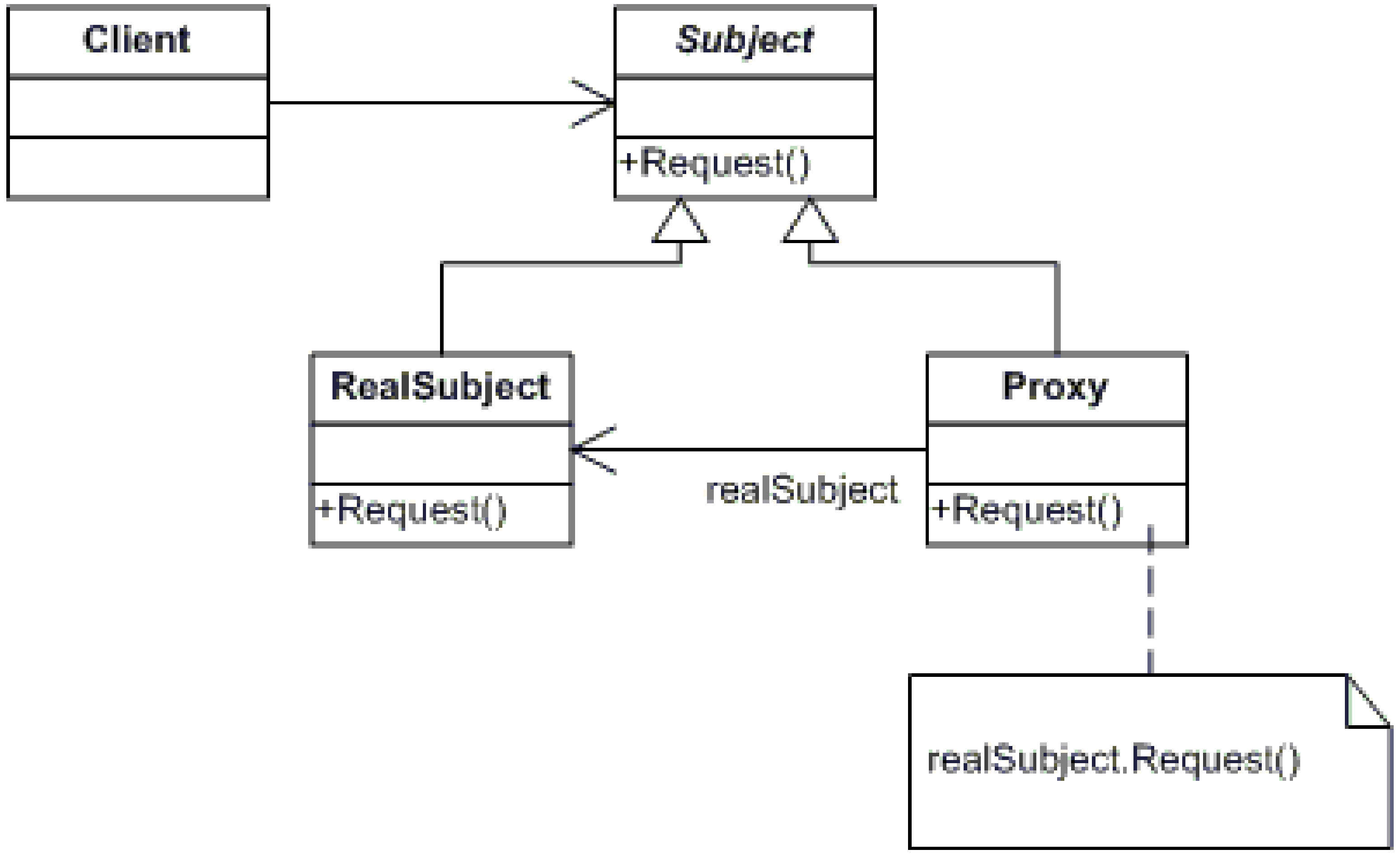
- **Facade**
  - Problem: A common, unified interface to a disparate set of implementations or Interfaces such as within a subsystem is required.

  - Solution:



Design Pattern – Structural Patterns

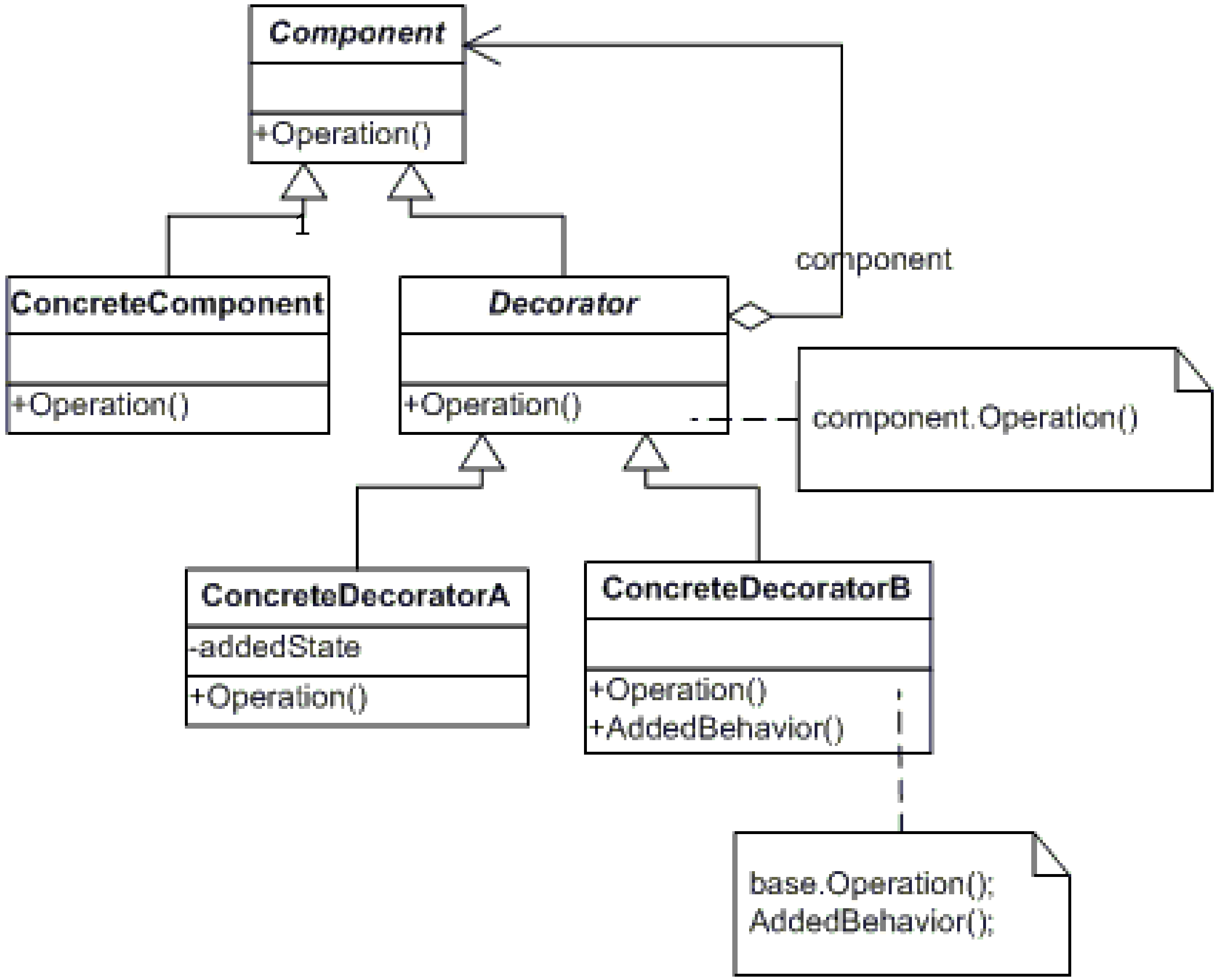
- **Proxy**
  - Problem: How to provide a placeholder for another object to control access to it.
  - Solution:



Design Pattern – Structural Patterns

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

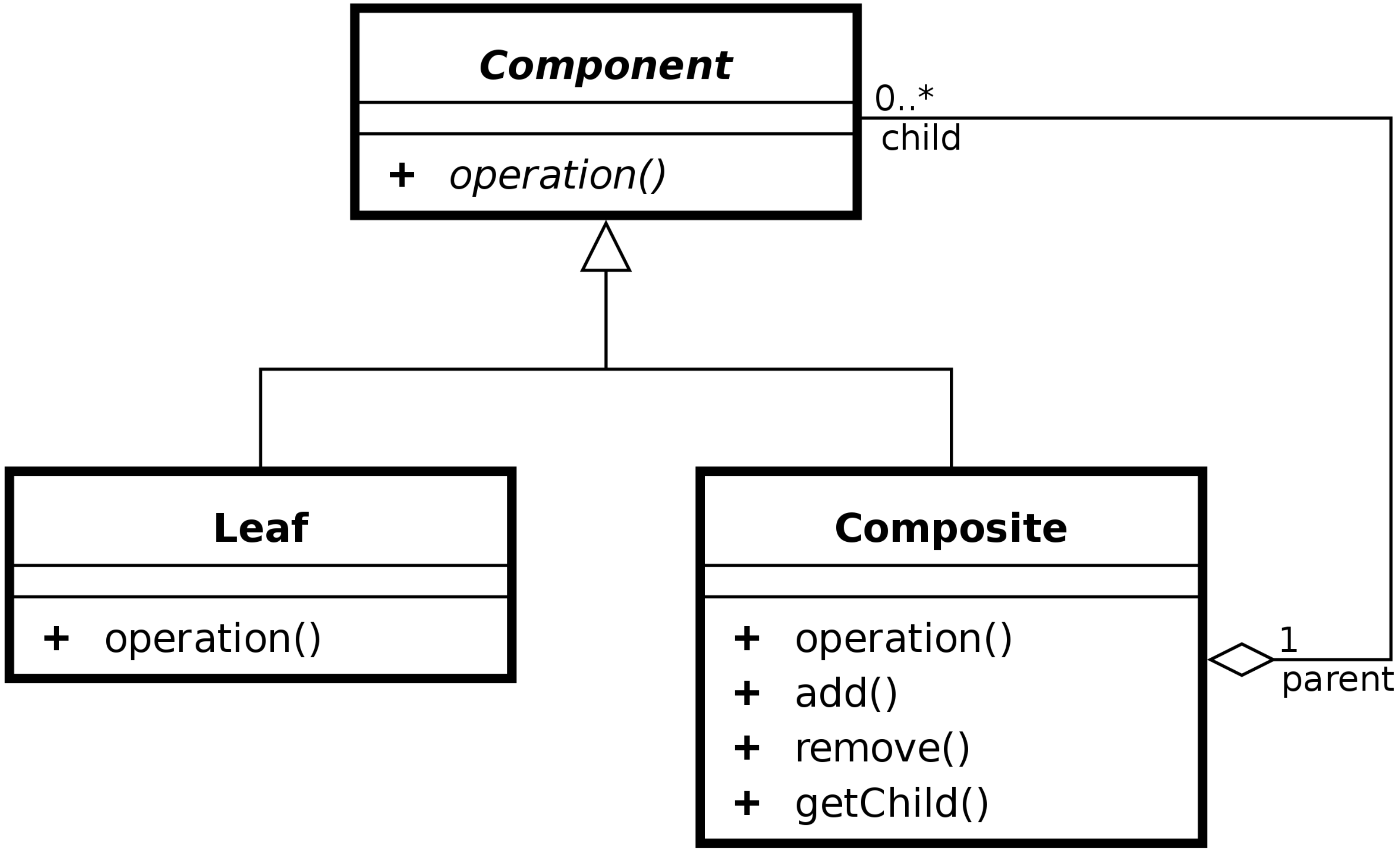
  - Solution:



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

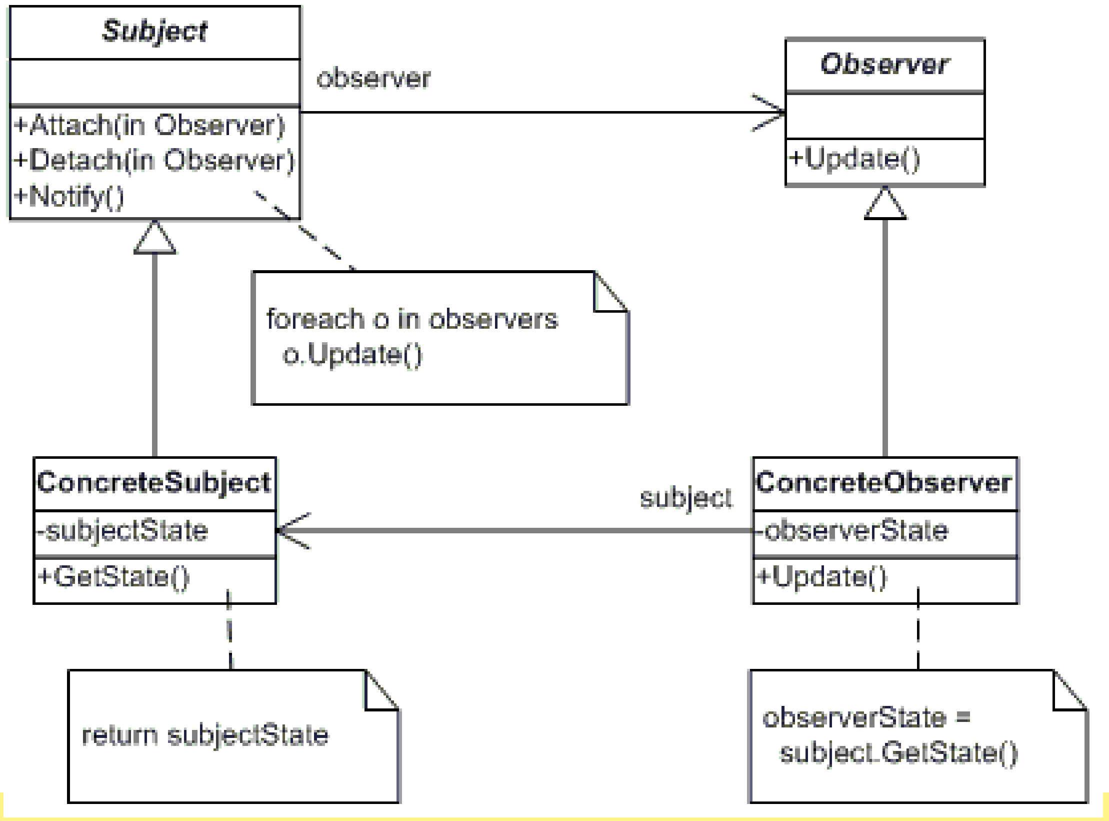
- Training: Exercises 3.1 (Factory), 3.2(Abstract Factory), 3.3 (Adaptor)
- Mandatory Assignment

Design Pattern – Behavioural Patterns

- **Observer**

  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

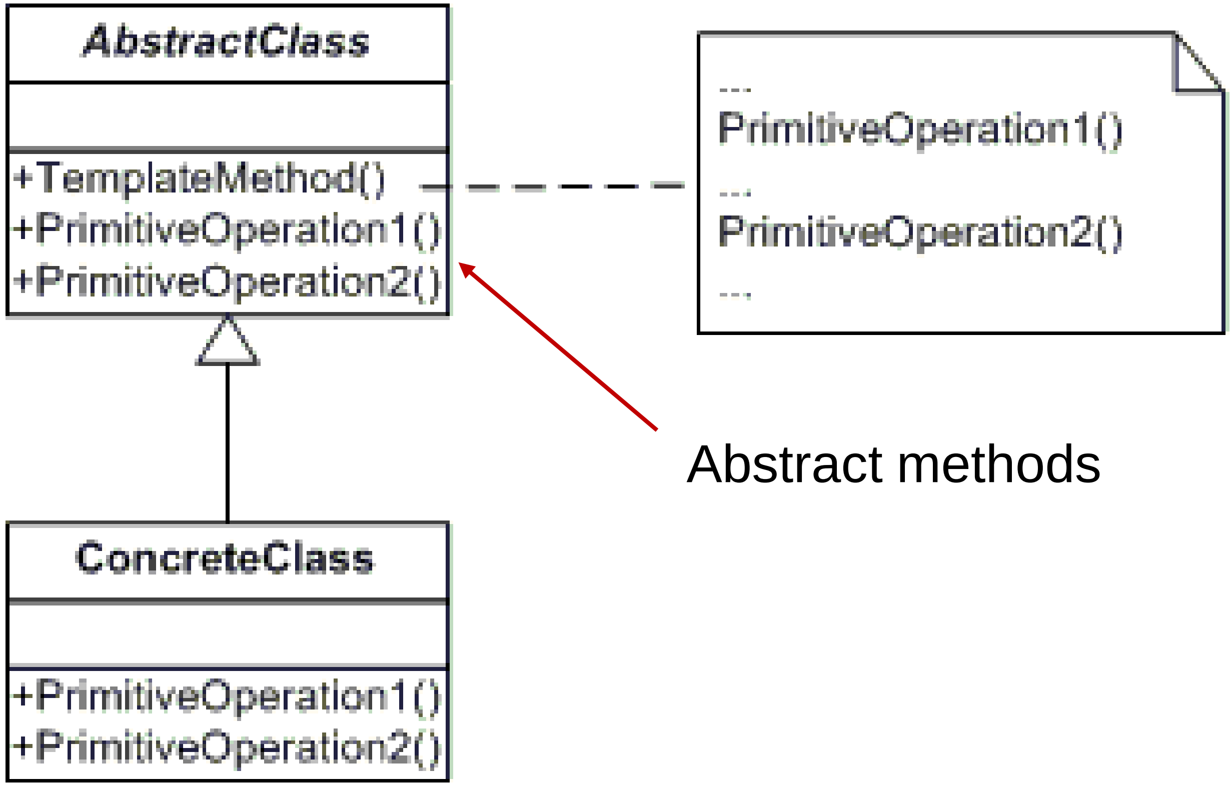
```
Class XX : INotifyPropertyChanged
{
. . .
// Attach, Deattach
public event PropertyChangedEventHandler PropertyChanged;

// 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:

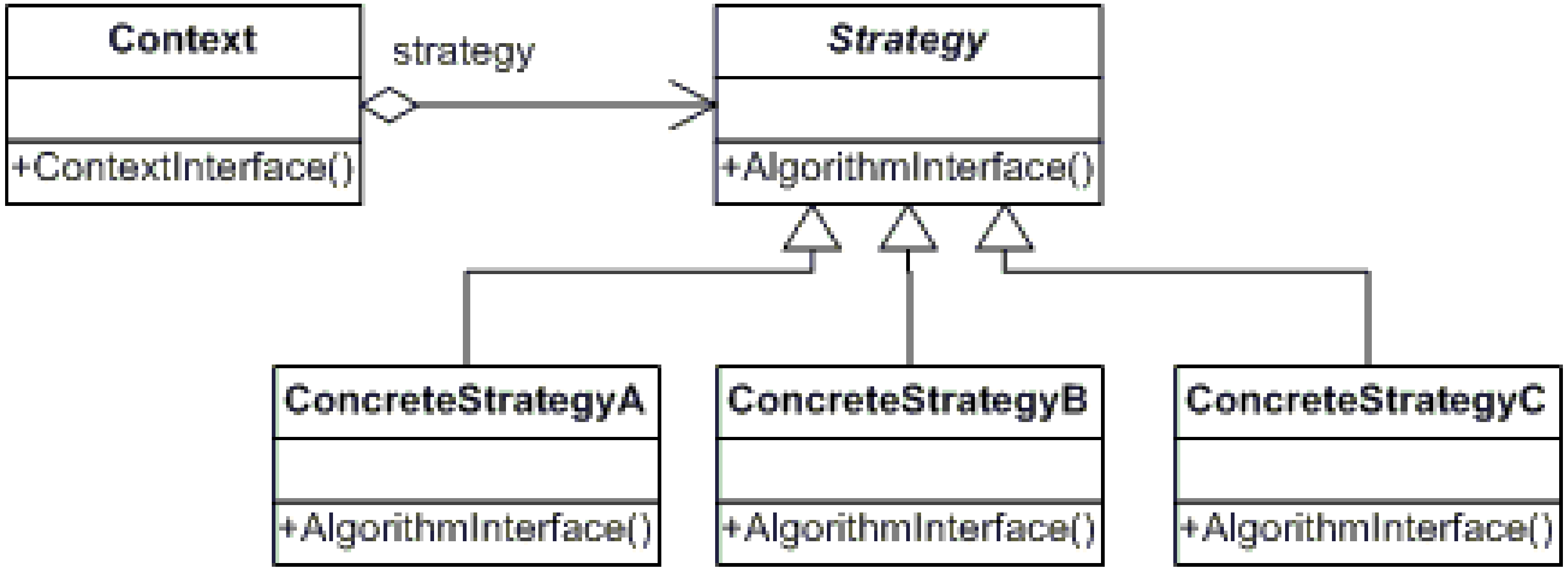


Abstract methods

Design Pattern – Behavioural Patterns

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

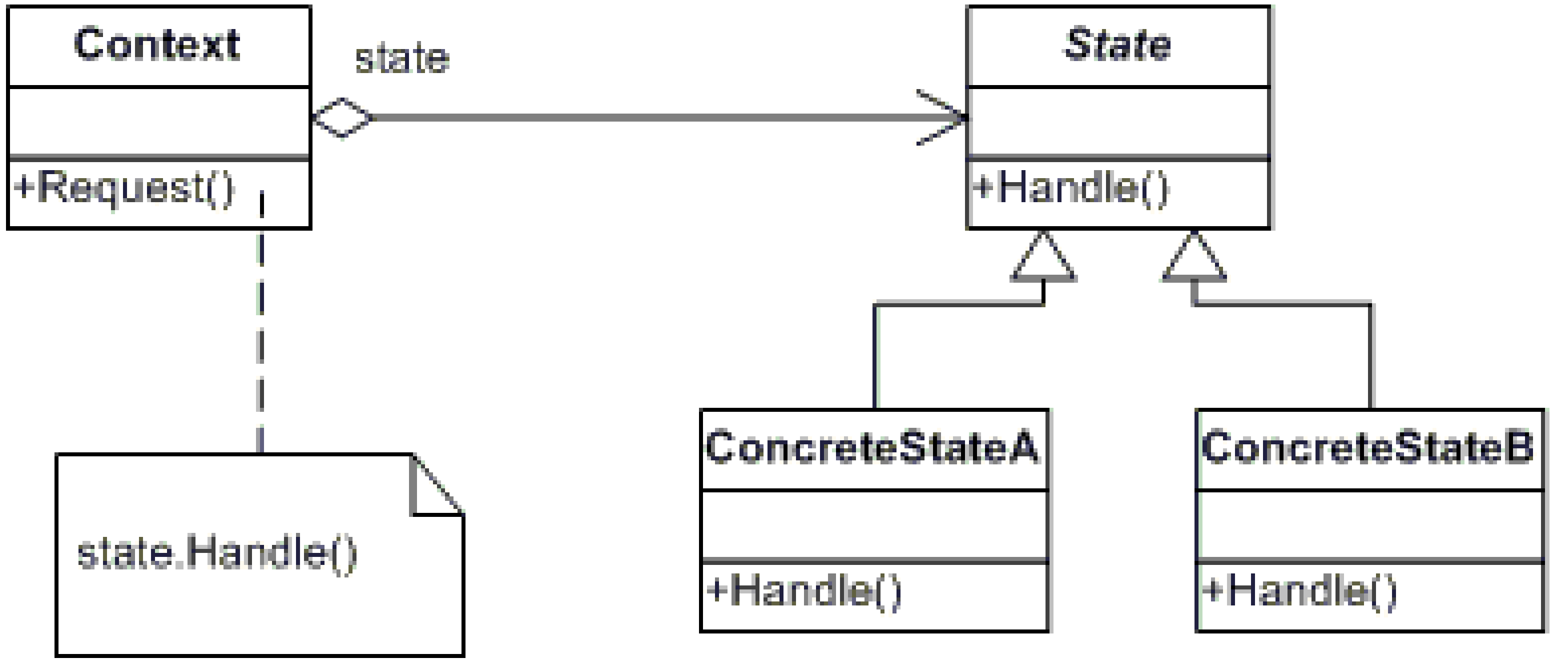
- Solution:



Design Pattern – Behavioural Patterns

- **State (seen at the snake game)**
  Problem: How to Allow an object to alter its behaviour when its internal state changes

- Solution:



Demo

- Demo of Observer, Template og Strategy

- Training: Exercises: 3.7 (Composite), 3.9 (Strategy)

- ***And of course the Mandatory Assignment***