Design Patterns

GoF

- Erich Gammer
- Richard Helm
- Ralph Johnson
- John Vlissides

Musterarten

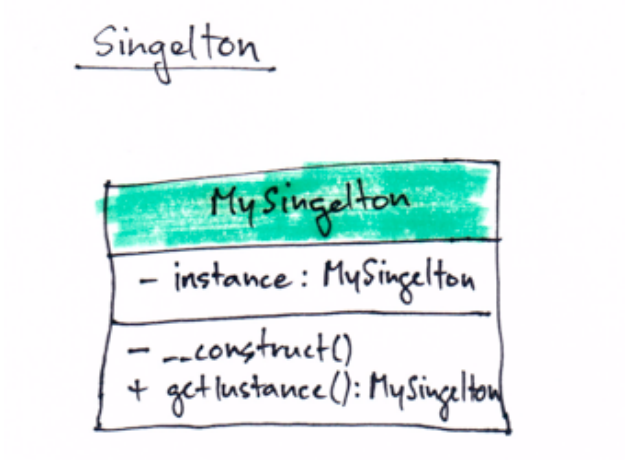
- Erzeugungsmuster
- Strukturmuster
- Verhaltensmuster

Erzeugungsmuster

- Singleton
- Factory-Method
- Abstract-Factory
- Builder
- Prototype

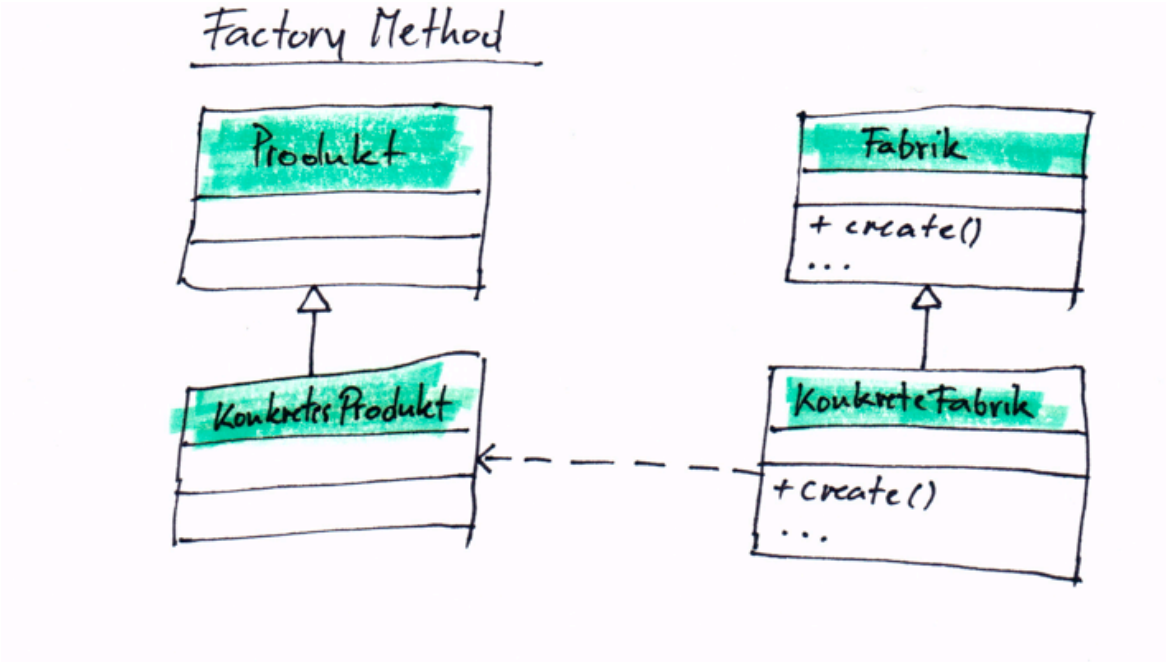
Singelton

"Sichere ab, dass eine Klasse genau ein Exemplar besitzt, und stelle einen globalen Zugriffspunkt darauf bereit."


Singelton
MySingelton
- instance : MySingelton
- __construct()
+ getInstance(): MySingelton

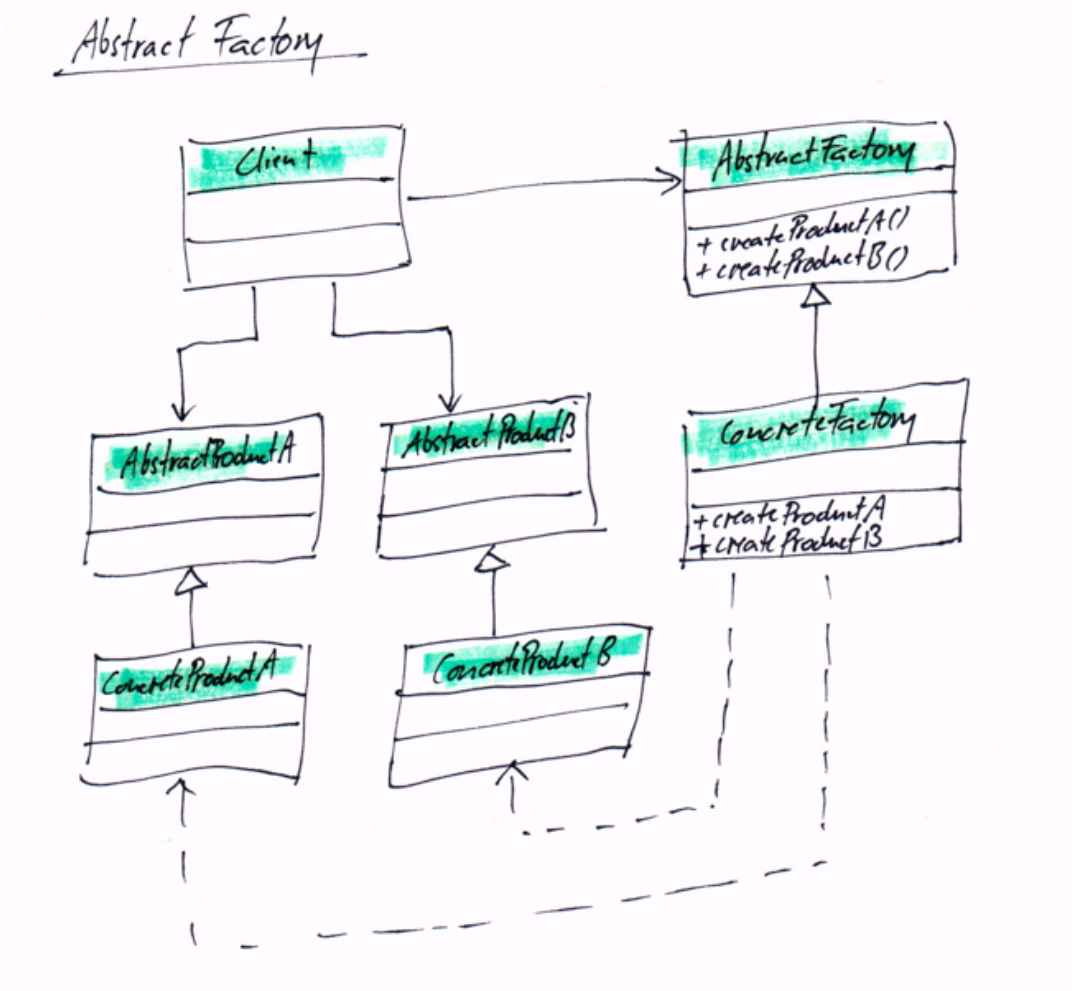
Factory-Method

"Definiere eine Klassenschnittstelle mit Operationen zum Erzeugen eines Objekts, aber lasse Unterklassen entscheiden, von welcher Klasse das zu erzeugende Objekt ist. Fabrikmethoden ermöglichen es einer Klasse, die Erzeugung von Objekten an Unterklassen zu delegieren."


Factory Method
Produkt
Fabrik
+ create()
...
KonkretesProdukt
KonkreteFabrik
+ create()
...

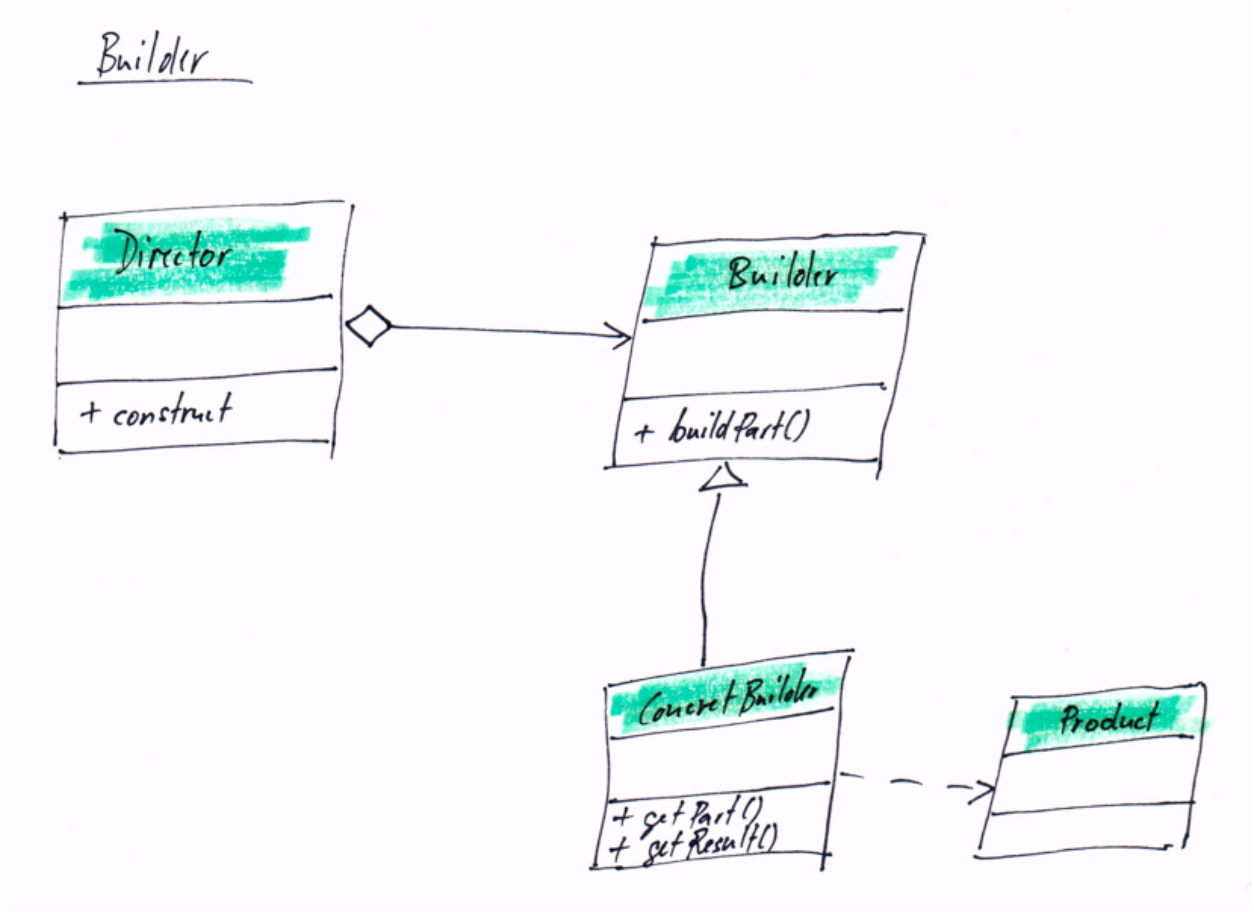
Abstract-Factory

"Biete eine Schnittstelle zum Erzeugen von Familien verwandter oder voneinander abhängiger Objekte, ohne ihre konkreten Klassen zu benennen."


Abstract Factory
Client
AbstractFactory
+ createProductA()
+ createProductB()
AbstractProductA
AbstractProductB
ConcreteFactory
+ createProductA
+ createProductB
ConcreteProductA
ConcreteProductB

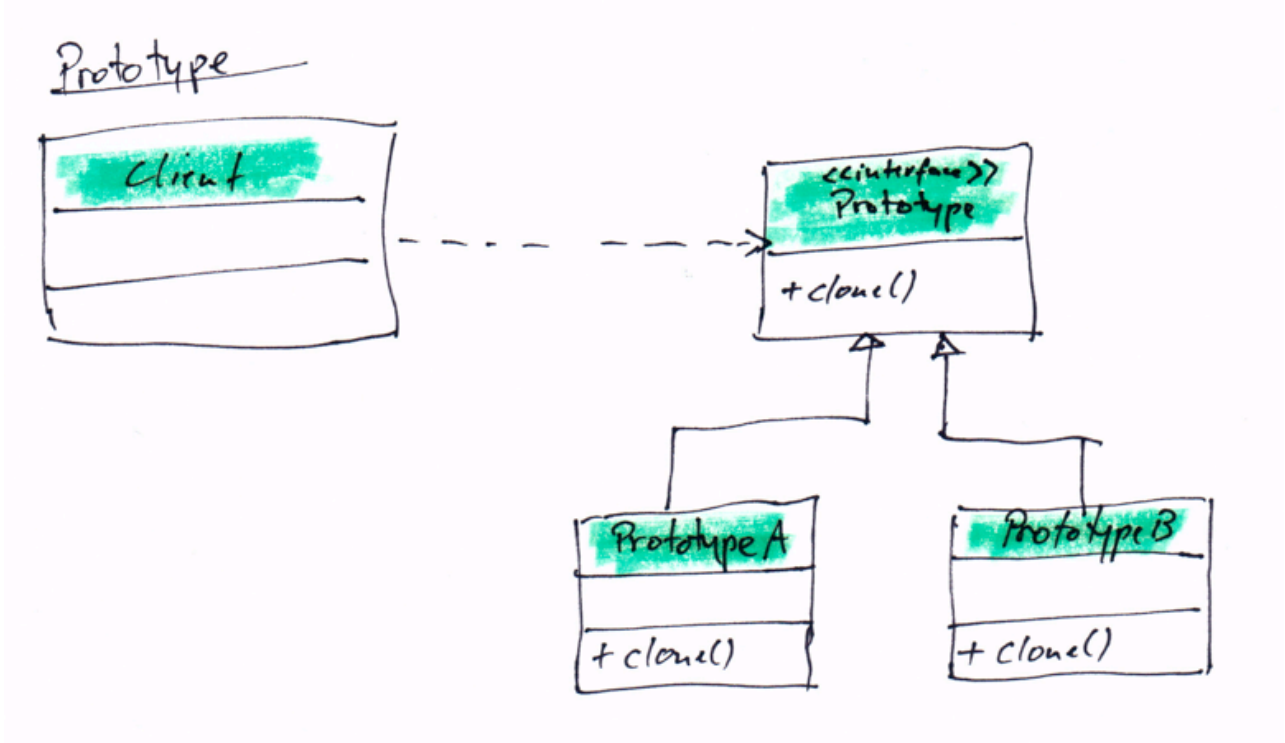
Builder

"Trenne die Konstruktion eines komplexen Objekts von seiner Repräsentation, so dass derselbe Knostruktionsprozess unterschiedliche Repräsentationen erzeugen kann."


Builder
Director
+ construct
Builder
+ buildPart()
ConcreteBuilder
+ getPart()
+ getResult()
Product

Prototype

"Bestimme die Arten zu erzeugender Objekte durch die Verwendung eines prototypischen Exemplars und erzeuge neue Objekte durch Kopieren dieses Prototypen."


Prototype
Client
<<interface>>
Prototype
+ clone()
Prototype A
+ clone()
Prototype B
+ clone()

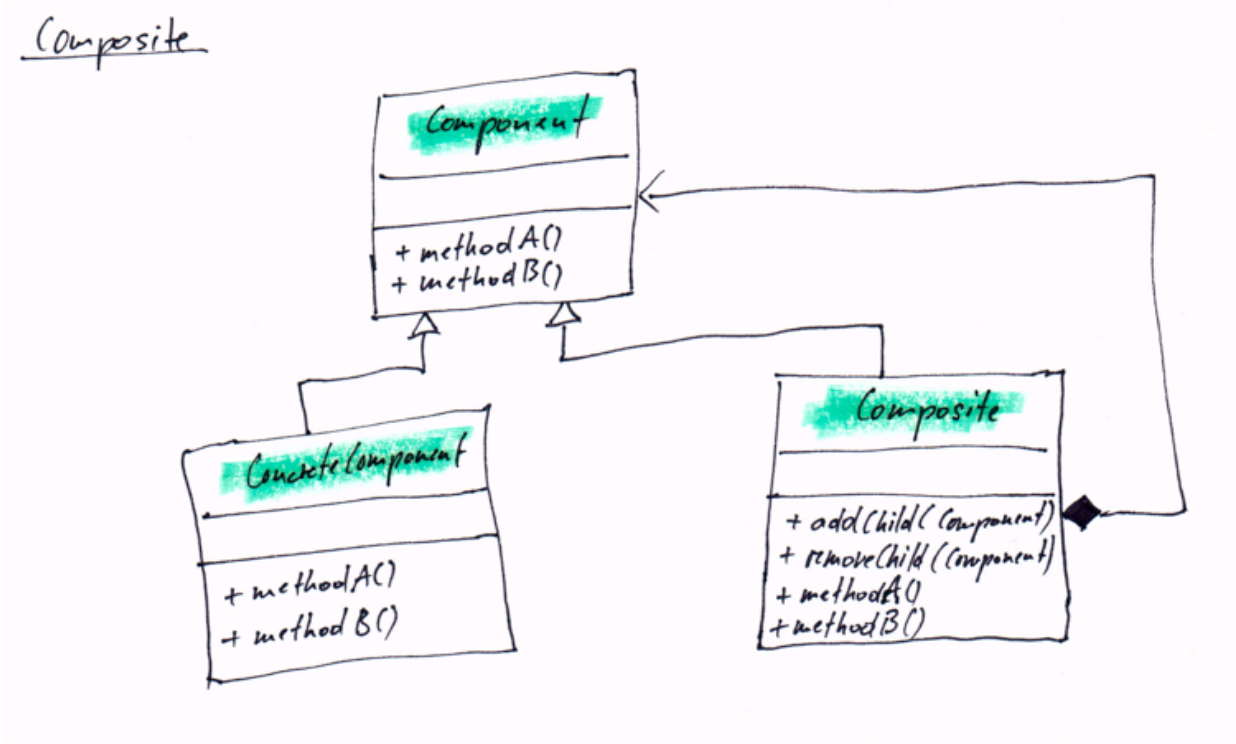
Strukturmuster

- Composite
- Decorator
- Facade
- Adapter

Composite

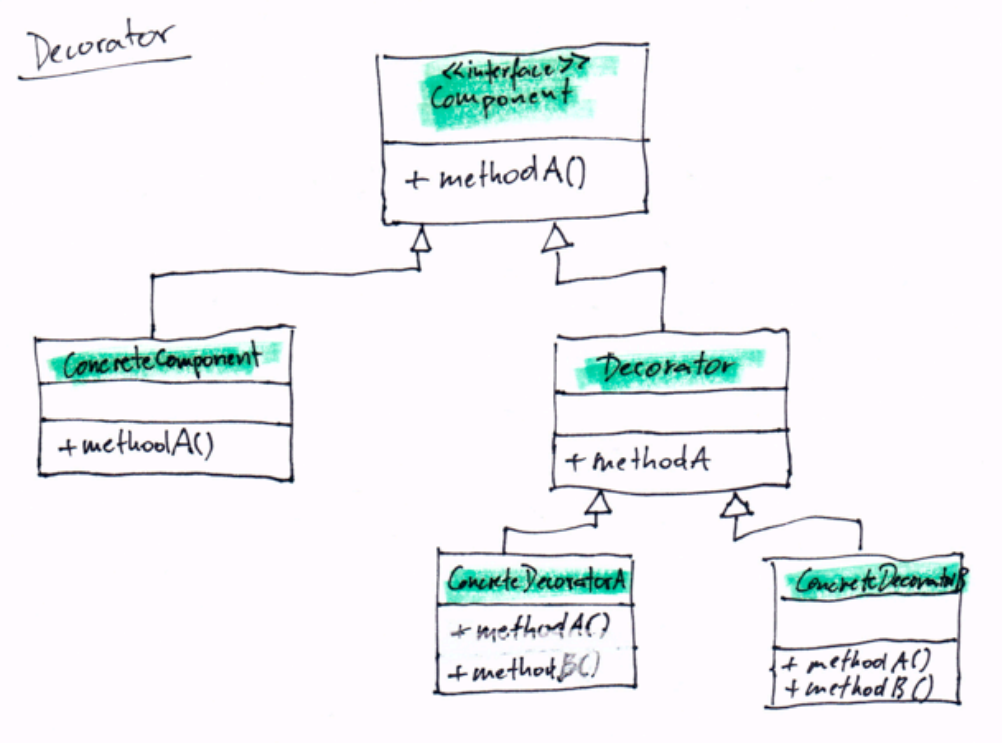
"Fügt mehrere Objekte zu einer Baumstruktur zusammen und ermöglicht es, diese von außen wie ein einzelnes zu verwenden."

Composite


Component
+ methodA()
+ methodB()
ConcreteComponent
+ methodA()
+ methodB()
Composite
+ addChild(Component)
+ removeChild(Component)
+ methodA()
+ methodB()

Decorator

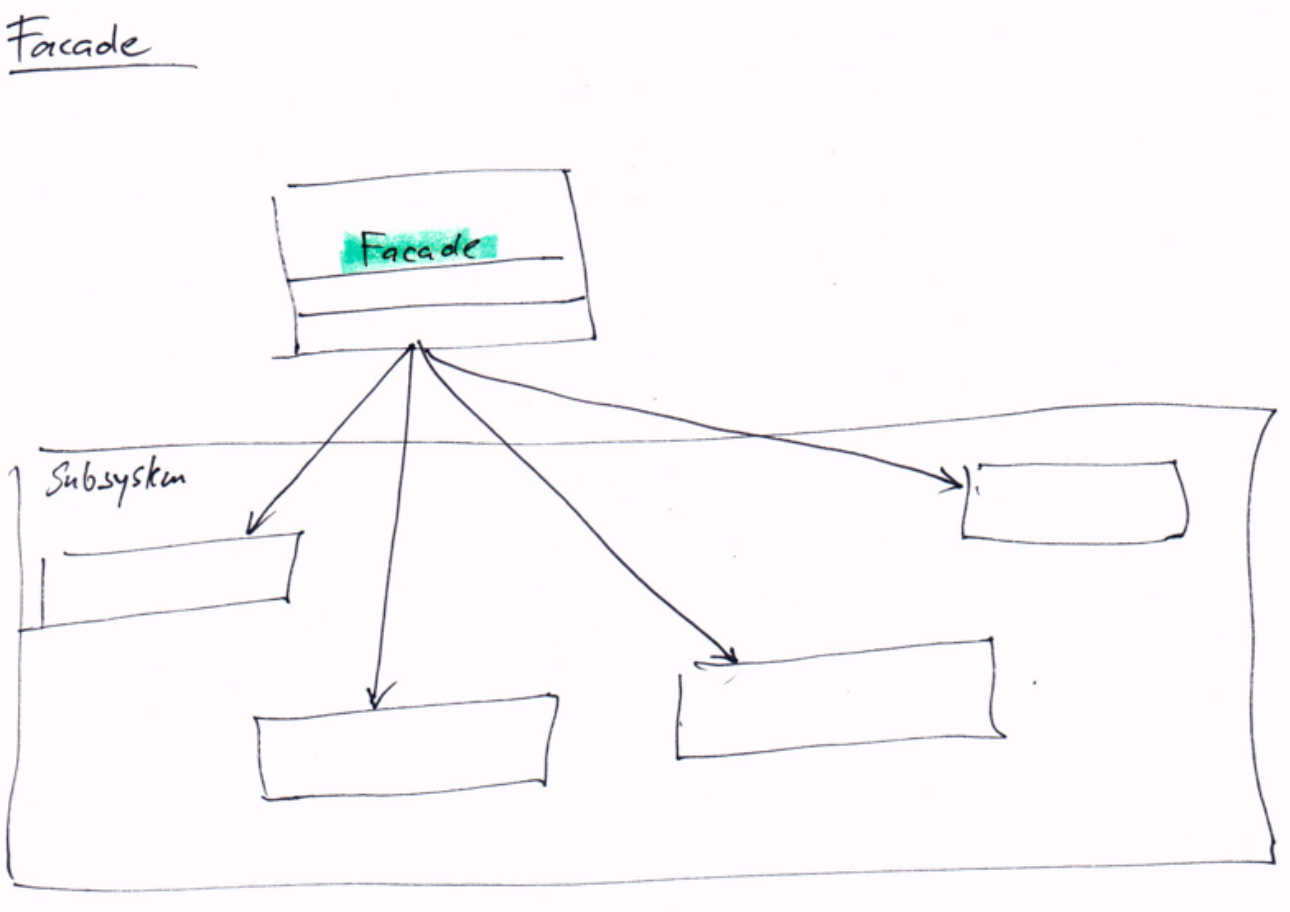
"Erweitere ein Objekt dynamisch um Zuständigkeiten. Dekorierer bieten eine flexible Alternative zur Unterklassenbildung, um die Funktionalität einer Klasse zu erweitern."


Decorator
<<interface>>
Component
+ methodA()
ConcreteComponent
+ methodA()
Decorator
+ methodA
ConcreteDecoratorA
+ methodA()
+ methodB()
ConcreteDecoratorB
+ methodA()
+ methodB()

Facade

"Biete eine einheitliche Schnittstelle zu einer Menge von Schnittstellen eines Subsystems. Die Fassadenklasse definiert eine abstrakte Schnittstelle, welche die Verwendung des Subsystems vereinfacht."

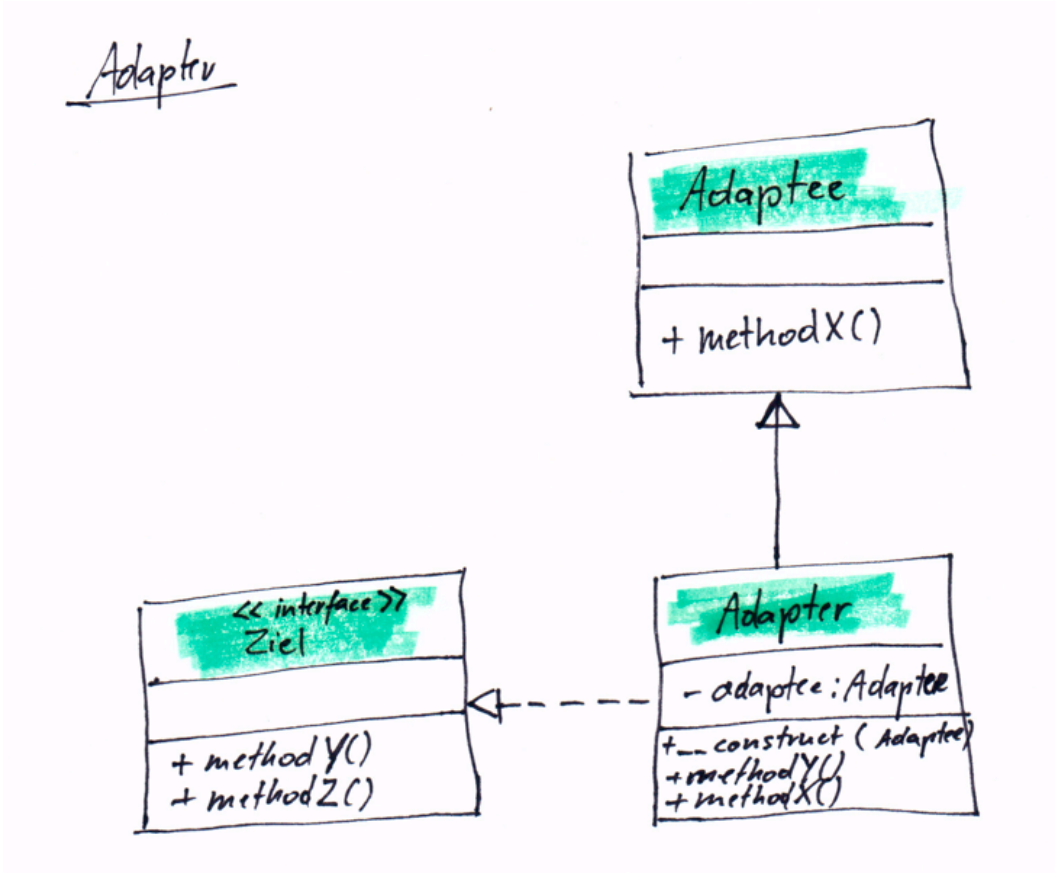
Facade



Adapter

"Passe die Schnittstelle einer Klasse an eine andere von ihren Klienten erwartete Schnittstelle an. Das Adaptermuster lässt Klassen zusammenarbeiten, die wegen inkompatibler Schnittstellen ansonsten dazu nicht in der Lage wären."

Adapter

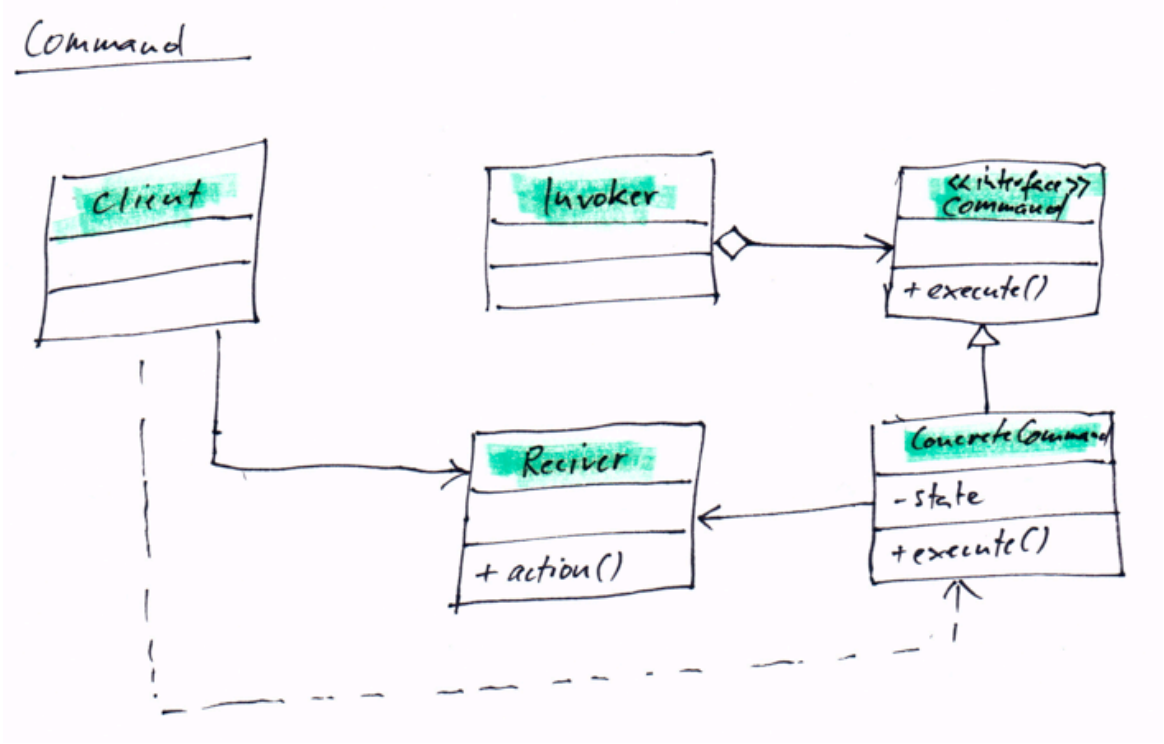

Adaptee
+ methodX()
<< interface >>
Ziel
+ methodY()
+ methodZ()
Adapter
- adaptee : Adaptee
+ construct (Adaptee)
+ methodY()
+ methodX()

Verhaltensmuster

- Command
- Observer
- Iterator
- Memento
- Template Method
- Strategy
- Mediator
- State
- Chain of Resposibility
- Visitor
- Interpreter

Command

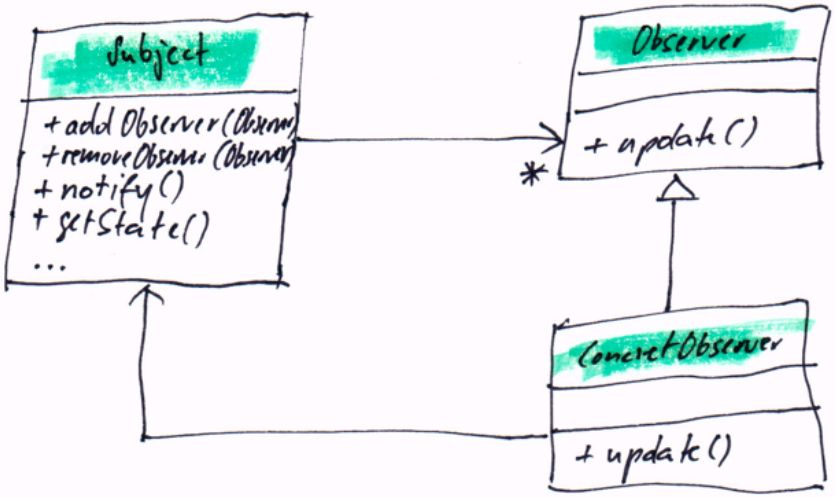
"Kapsle einen Befehl als ein Objekt. Dies ermöglicht es, Klienten mit verschiedenen Anfragen zu parametrisieren, Operationen in eine Schlange zu stellen, ein Logbuch zu führen und Operationen rückgängig zu machen."


Command
Client
Invoker
<<interface>>
Command
+ execute()
Receiver
+ action()
ConcreteCommand
- state
+ execute()

Observer

"Definiere eine 1-zu-n-Abhängigkeit zwischen Objekten, so dass die Änderung des Zustands eines Objekts dazu führt, das alle abhängigen Objekte benachrichtigt und automatisch aktualisiert werden."

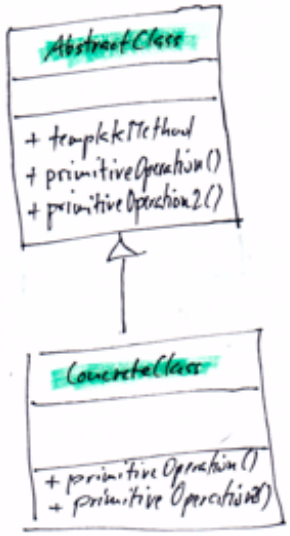
Observer


Subject
+ addObserver(Observer)
+ removeObserver(Observer)
+ notify()
+ setState()
...
Observer
+ update()
*
ConcreteObserver
+ update()

Template Method

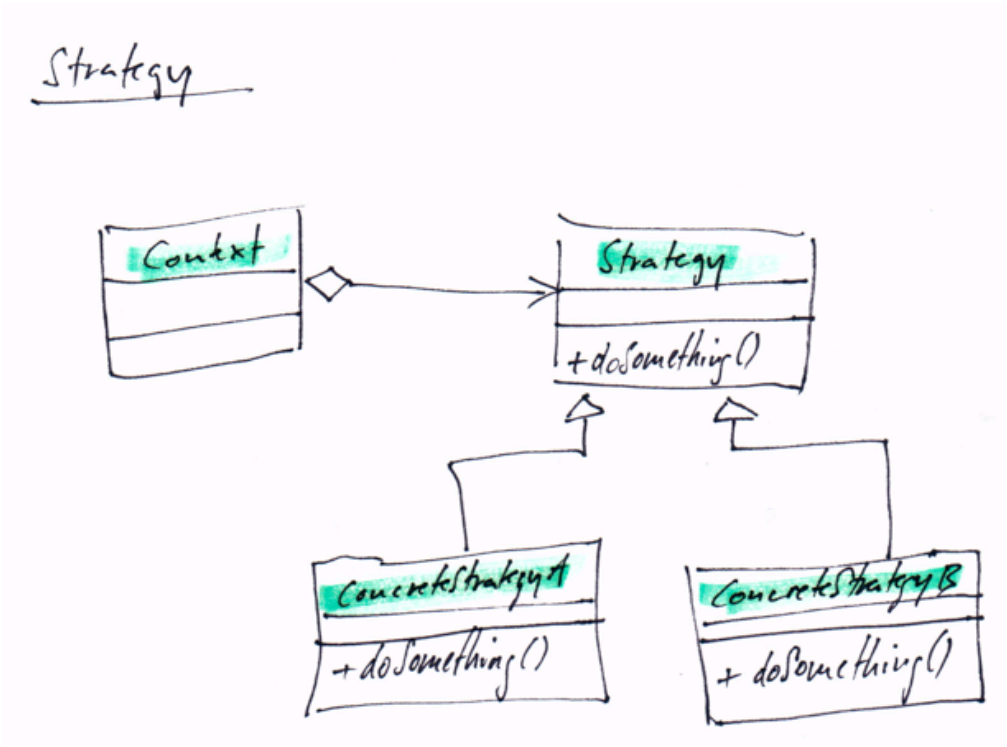
"Definiert die Schritte eines Algorithmus in einer Methode und überlässt die Implementierung der einzelnen Schritte den Unterklassen. Diese können Teile des Algorithmus modifizieren, ohne dessen Struktur zu verändern."

Template - Method



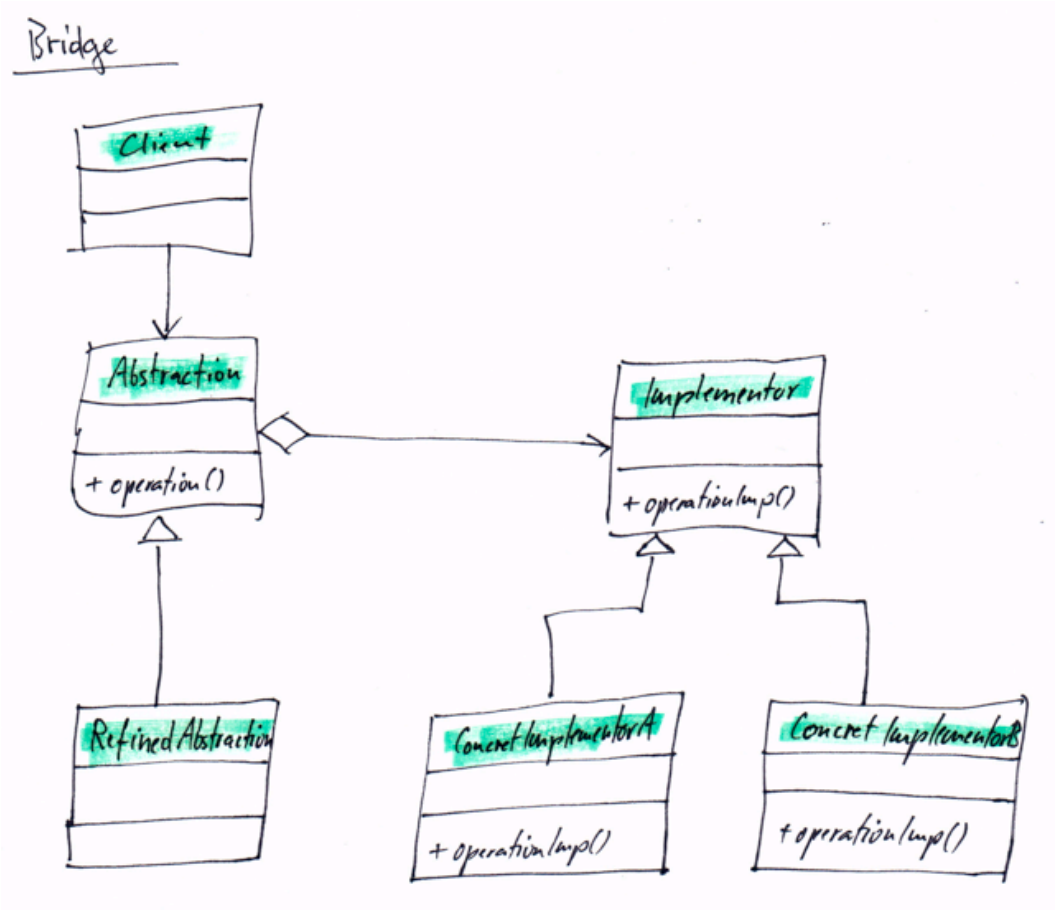
Strategy

"Definiere eine Familie von Algorithmen, kapsele jeden einzelnen und mach sie austauschbar. Das Strategiemuster ermöglicht es, den Algorithmus unabhängig von ihn nutzenden Klienten zu variieren."


Strategy
Context
Strategy
+ doSomething()
ConcreteStrategyA
+ doSomething()
ConcreteStrategyB
+ doSomething()

Bridge

"Entkopple eine Abstraktion von ihrer Implementierung, so dass beide unabhängig voneinander variiert werden können."


Bridge
Client
Abstraction
+ operation()
Implementor
+ operationImp()
RefinedAbstraction
ConcreteImplementorA
+ operationImp()
ConcreteImplementorB
+ operationImp()

Enterprise Patterns

Patterns

- Domain Model
- Table Data Gateway
- Active Record
- Data Mapper
- Front Controller

  Data Transfer Object
- Value Object
- Repository