Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering
InIT Institut für angewandte
Informationstechnologie

Bachelor of Science (BSc) in Informatik

Modul Software-Entwicklung 1 (SWEN1)

LE 08 – Entwurf mit Design Pattern I
Zusammenfassung

SWEN1/PM3 Team:
R. Ferri (feit), D. Liebhart (lieh), K. Bleisch (bles), G. Wyder (wydg)

Ausgabe: HS24

Lernziele LE 08 – Entwurf mit Design Patterns I

- Sie sind in der Lage:
  - Den allgemeinen Aufbau und Zweck von Design Patterns (Entwurfsmuster) zu erklären.
  - Den Aufbau und Einsatz der folgenden Design Patterns zu erklären und anzuwenden:
    - Adaptor
    - Factory
    - Singleton
    - Dependency Injection
    - Proxy
    - Chain of Responsibility

Agenda

1. **Einführung in Design Patterns**
2. Repetition GRASP
3. Design Pattern
4. Wrap-up und Ausblick

Design Pattern: Was ist das? Warum?

Bewährte Lösungsschablonen
für wiederkehrende Entwurfsprobleme

- Rad nicht neu erfinden
- Gemeinsame Sprache/Verständnis
- Best-practices lernen

Aufbau Design Patterns

- Beschreibungsschema
  - Name
  - Beschreibung Problem
  - Beschreibung Lösung
  - Hinweise für Anwendung
  - Beispiele

GRASP und GoF

- GRASP
  - Grundlegende Prinzipien der Zuweisung von Verantwortlichkeiten, formuliert als Design Patterns
- GoF
  - Buch «Design Patterns», herausgekommen 1995
  - 23 Patterns, 15 sind gebräuchlich
  - Erste systematische Publikation von Design Patterns
  - Können als Spezialisierungen von GRASP interpretiert werden

Agenda

1. Einführung in Design Patterns
2. **Repetition GRASP**
3. Design Pattern
4. Wrap-up und Ausblick

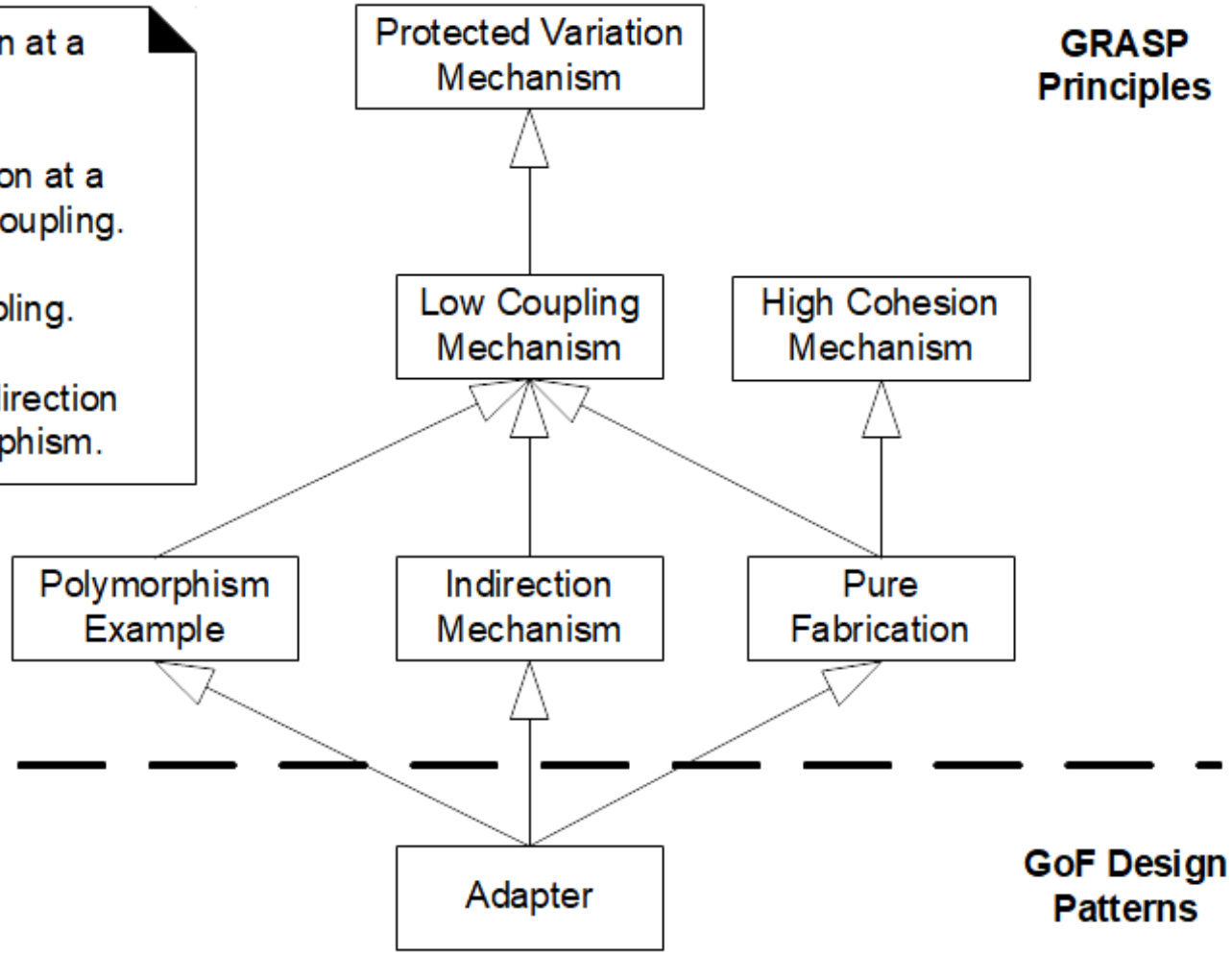
GRASP Prinzipien und GoF Design Patterns

Low coupling is a way to achieve protection at a variation point.

Polymorphism is a way to achieve protection at a variation point, and a way to achieve low coupling.

An indirection is a way to achieve low coupling.

The Adapter design pattern is a kind of Indirection and a Pure Fabrication, that uses Polymorphism.



GoF Patterns sind Spezialfälle der GRASP Prinzipien

Agenda

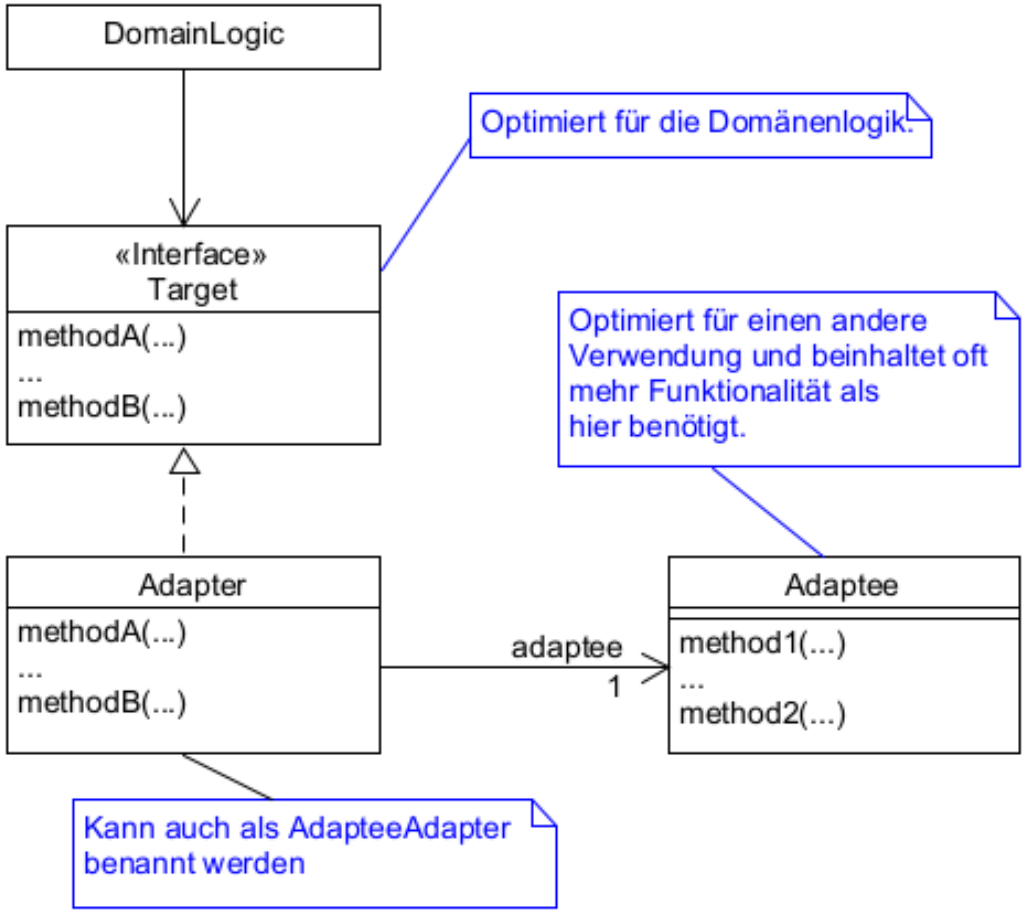
1. Einführung in Design Patterns
2. Repetition GRASP
3. **Design Patterns**
4. Wrap-up und Ausblick

Liste der Design Patterns für die Lerneinheit

- Adapter
- Simple Factory
- Singleton
- Dependency Injection
- Proxy
- Chain of Responsibility

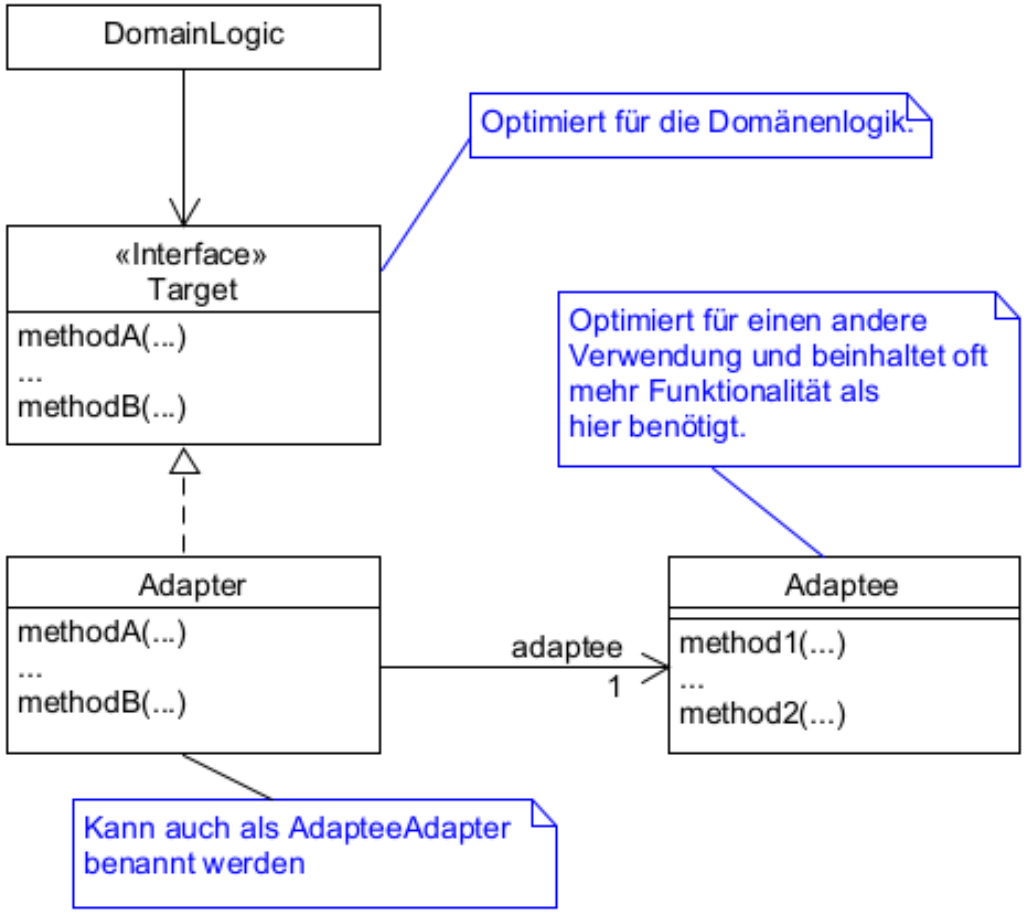
Adapter: Problem und Lösung

- Problem
  - Eine Klasse soll eingesetzt werden, die aber inkompatibel mit einem bereits definierten domänen-spezifischem Interface ist.
- Lösung
  - Eine eigene Adapter Klasse wird dazwischengeschaltet.



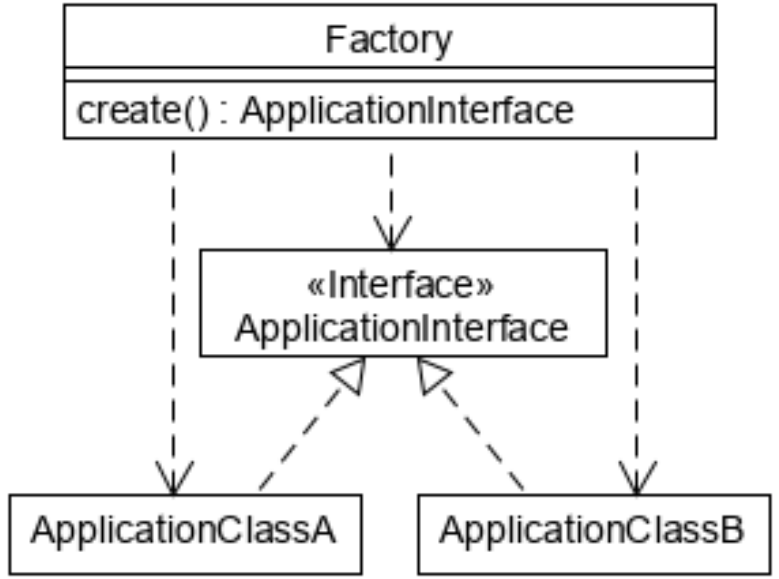
Adapter: Hinweise

- Hinweise
  - Oft wird mit einem Adapter ein externer Dienst in die eigene Anwendung integriert, insbesondere wenn der Dienst austauschbar sein soll.
  - Das Target Interface ist bewusst für die Domänenlogik optimiert, während der Adaptee oft von extern bezogen wird.
  - Falls es sich beim Adaptee um einen externen Dienst handelt, kann im Adapter allenfalls die Kommunikation integriert werden.



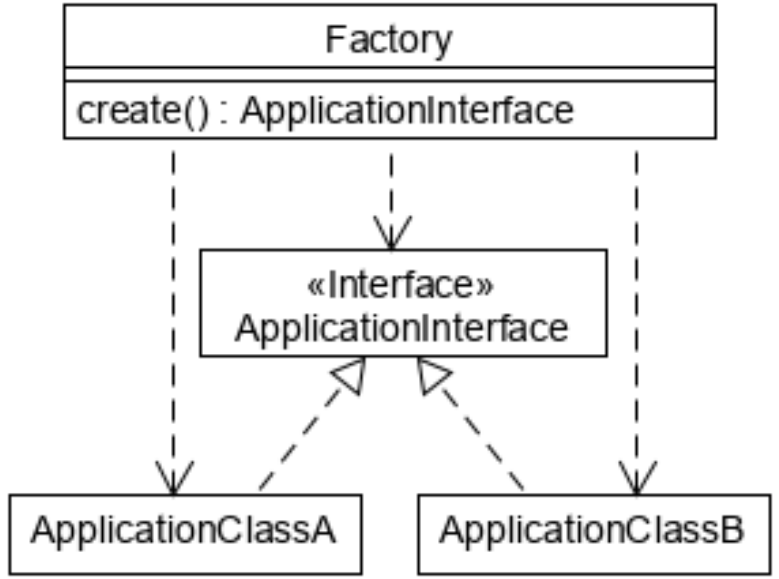
Simple Factory: Problem und Lösung

- Problem
  - Das Erzeugen eines neuen Objekts ist aufwändig.
- Lösung
  - Eine eigene Klasse für das Erzeugen eines neuen Objekts wird geschrieben.



Simple Factory: Hinweise

- Hinweise
  - Oft ist die Erzeugung des neuen Objekts von irgendeiner Art von Konfiguration abhängig.
  - Es ist auch möglich, die create() Methode mit Parametern zu ergänzen.
  - Die Factory kann allenfalls die erzeugten Objekte zwischenspeichern und später wiederverwenden.



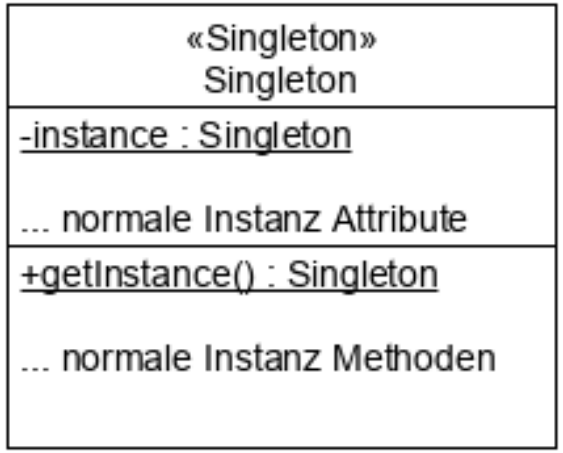
Singleton: Problem und Lösung

- Problem
  - Man benötigt von einer Klasse nur eine einzige Instanz.
  - Diese Instanz muss global sichtbar sein.
- Lösung
  - Klasse mit einer statischen Methode, die immer dasselbe Objekt zurückliefert.
  - Statische Methode wird public deklariert.

```
public class Singleton {

private static Singleton instance = new Singleton();

// privater Konstruktor, verhindert direkte Instanziierung.
private Singleton() {
}

// Instanzvariablen ...

public static Singleton getInstance() {
    return instance;
}

// Instanz-Methoden ...
}
```

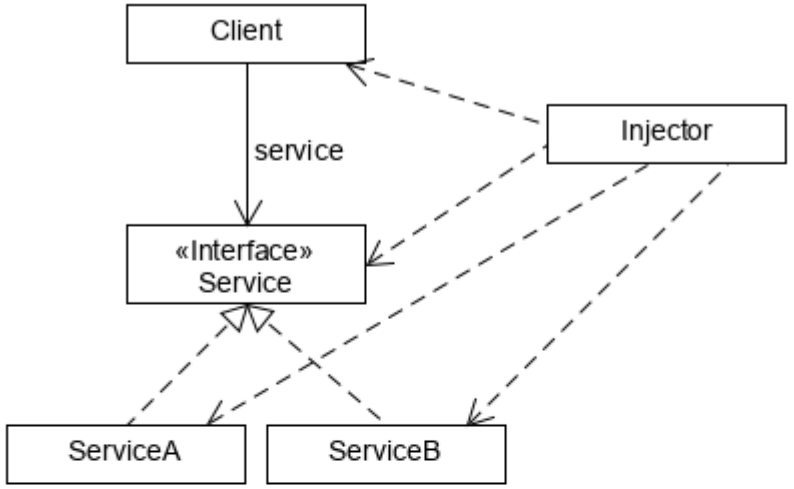
Singleton: Hinweise

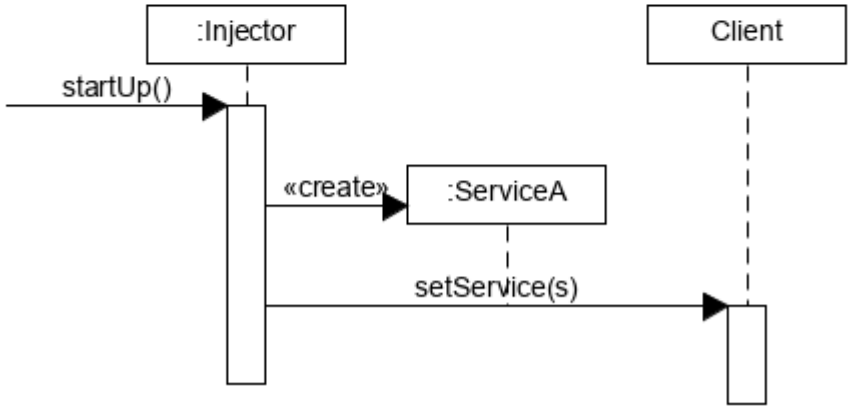
- Hinweise
  - Globale Sichtbarkeit wird heutzutage sehr kritisch betrachtet.
  - Lazy Creation für die Instanz ist möglich, dann sollte aber die getInstance() Methode synchronisiert werden.



Dependency Injection (DI): Problem und Lösung

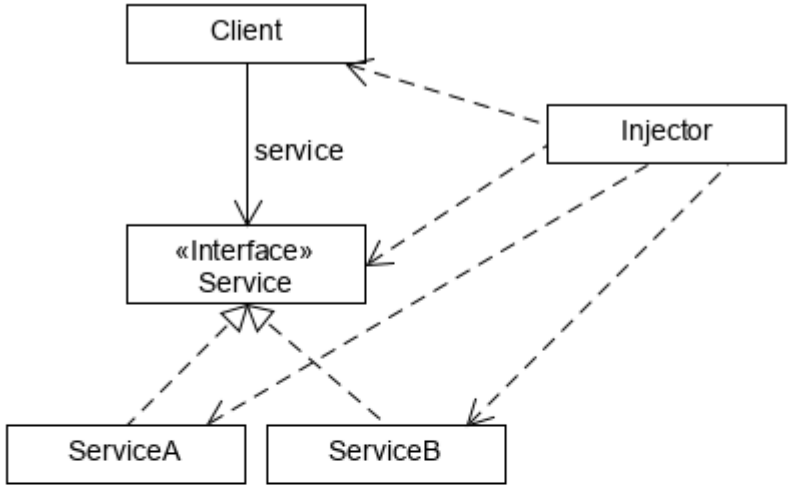
- Problem
  - Eine Klasse braucht eine Referenz auf ein anderes Objekt. Dieses Objekt muss ein bestimmtes Interface definieren, je nach Konfiguration aber mit einer anderen Funktionalität.
- Lösung
  - Anstelle, dass die Klasse das abhängige Objekt selber erzeugt, wird dieses Objekt von aussen (Injector) gesetzt.

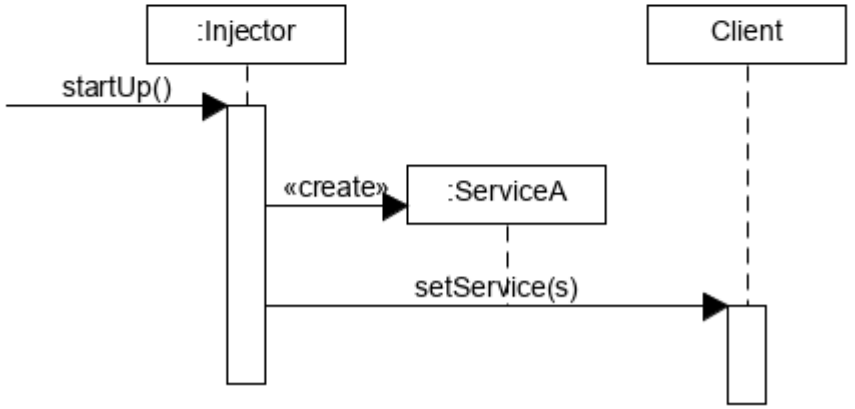




Dependency Injection (DI): Hinweise (1/2)

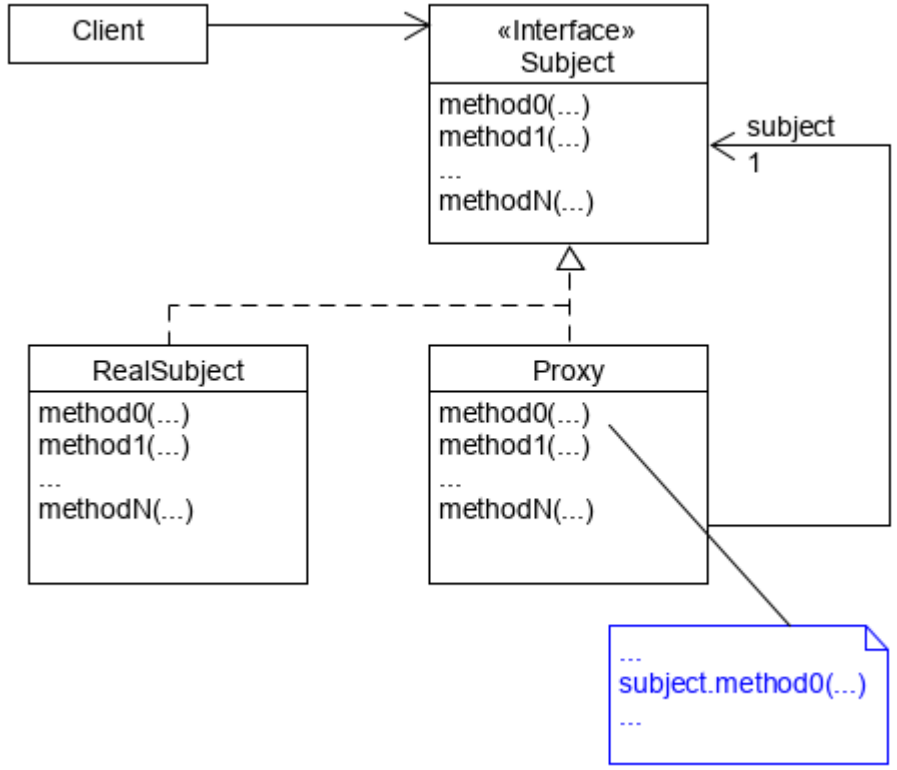
- Hinweise
  - Ersatz für das Factory Pattern.
  - Direkter Widerspruch zum GRASP Creator Prinzip.
  - Viele Frameworks unterstützen inzwischen DI (z.B. Spring), kann aber problemlos auch ohne Framework angewendet werden.
  - Erleichtert das Schreiben von Testfällen, insbesondere den Gebrauch von Mocks.





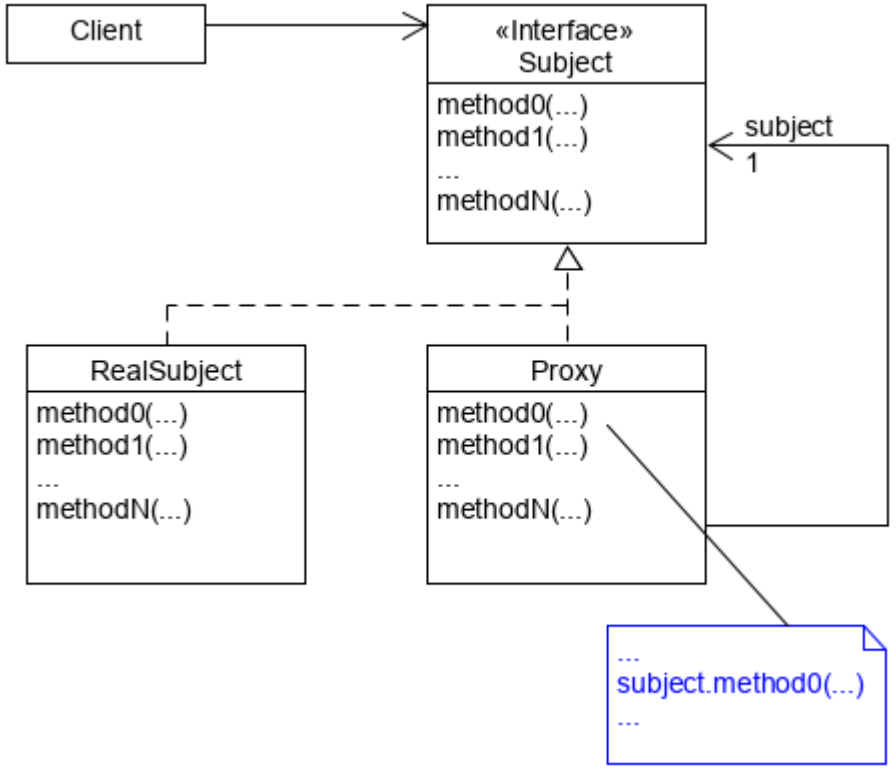
Proxy: Problem und Lösung

- Problem
  - Ein Objekt ist nicht oder noch nicht im selben Adressraum verfügbar.
- Lösung
  - Ein Stellvertreter Objekt («Proxy») mit demselben Interface wird anstelle des richtigen Objekts verwendet.
  - Das «Proxy» Objekt leitet alle Methodenaufrufe zum richtigen Objekt weiter.



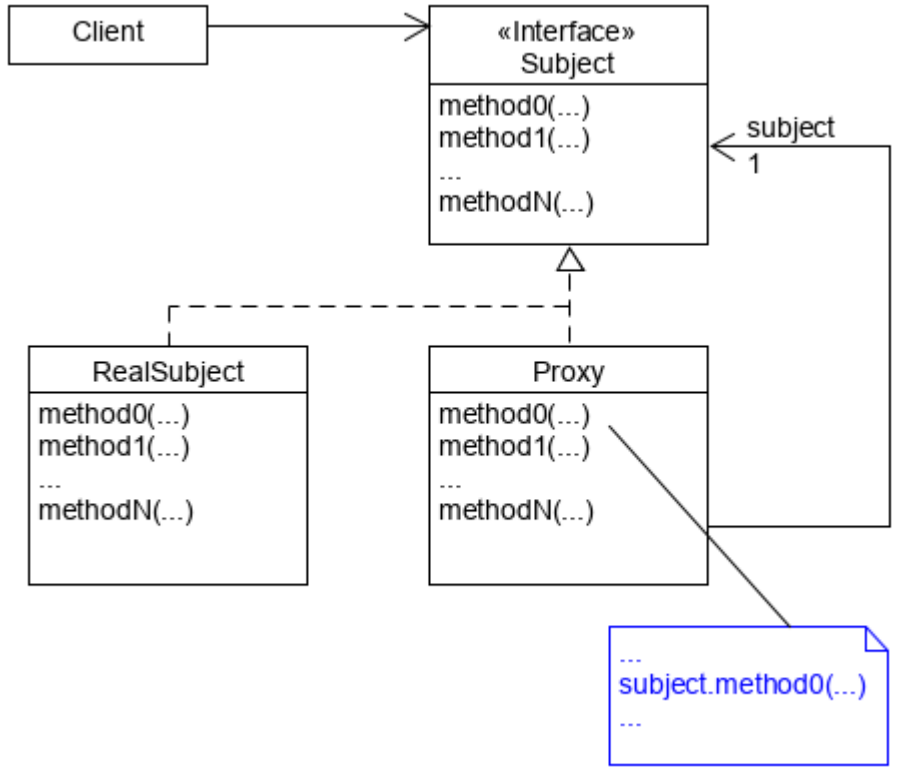
Proxy: Einsatz

- Einsatz als (Struktur ist dieselbe!)
  - «Remote Proxy» ist ein Stellvertreter für ein Objekt in einem anderen Adressraum und übernimmt die Kommunikation mit diesem.
  - «Virtual Proxy» verzögert das Erzeugen des richtigen Objekts auf das erste Mal, dass dieses benutzt wird.
  - «Protection Proxy» kontrolliert den Zugriff auf das richtige Objekt.



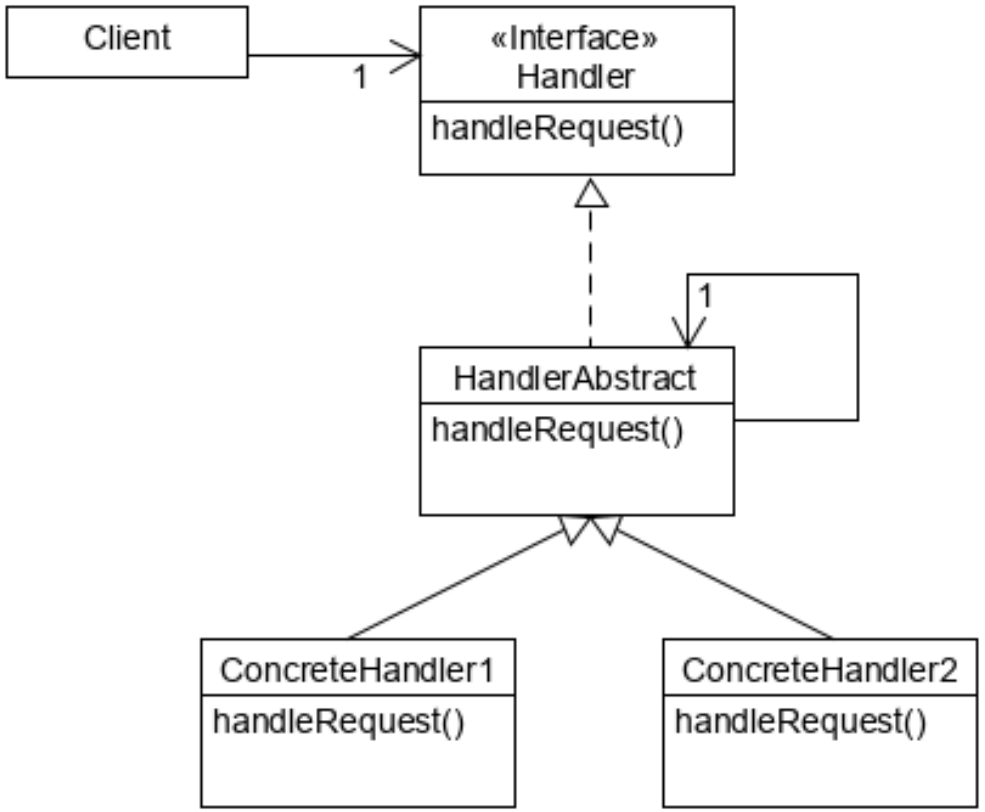
Proxy: Hinweise

- Sieht ähnlich aus wie ein Adapter, der Unterschied ist aber, dass der «Adaptee», in diesem Fall das RealSubject, auch dasselbe Interface implementiert wie der «Adapter» resp. Subject
- Vom Aufbau her identisch mit dem Decorator Pattern (siehe LE09), hat aber einen anderen Zweck.



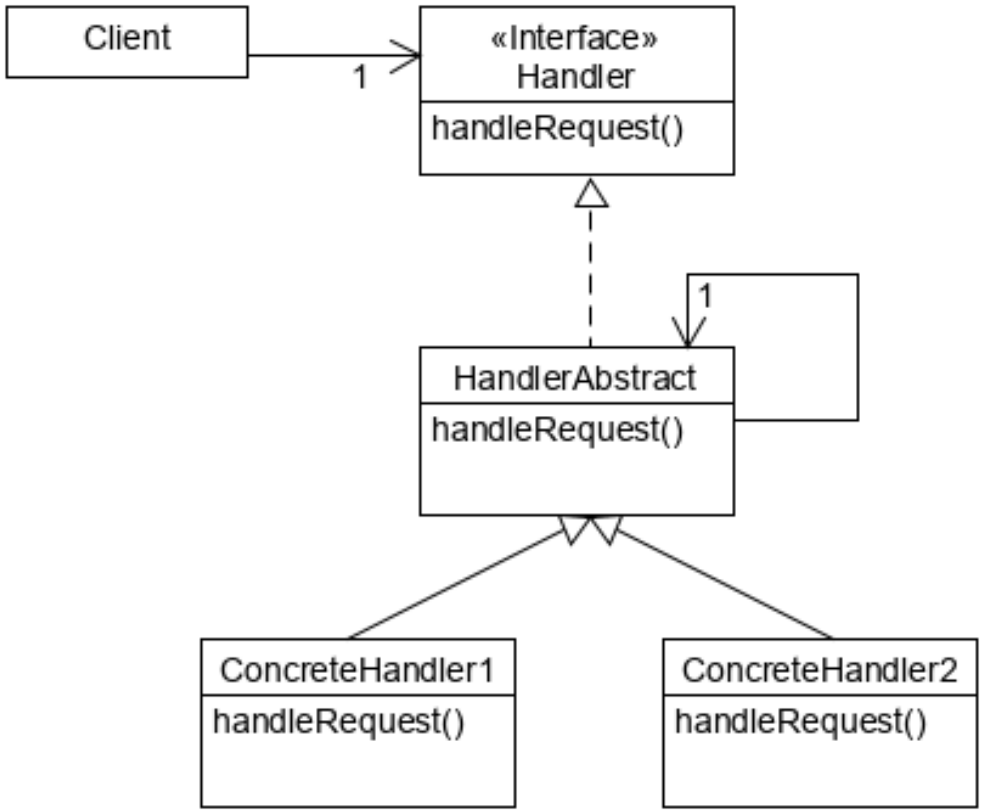
Chain of Responsibility: Problem und Lösung

- Problem
  - Für eine Anfrage gibt es potentiell mehrere Handler, aber von vornherein ist es nicht möglich (oder nur sehr schwer), den richtigen Handler herauszufinden.
- Lösung
  - Die Handler werden in einer einfach verketteten Liste hintereinandergeschaltet.
  - Jeder Handler entscheidet dann, ob der die Anfrage selber beantworten möchte oder sie an den nächsten Handler weiterleitet.



Chain of Responsibility: Hinweise

- Hinweise
  - Als Variante davon leitet jeder Handler die Anfrage an den nächsten Handler weiter, unabhängig davon, ob er sie selber behandelt oder nicht.
  - Es könnte sein, dass gar kein Handler die Anfrage behandelt.



Agenda

1. Einführung in Design Patterns
2. Repetition GRASP
3. Design Pattern
4. **Wrap-up und Ausblick**

Wrap-up

- Design Patterns sind wichtige Werkzeuge um gut strukturierten, wartbaren Code zu schreiben.
- Die Kombination von Singleton, Factory und Adapter wurde traditionell oft eingesetzt, um externe Dienste anzusprechen.
- Anstelle von Singleton und Factory ist vermehrt Dependency Injection (DI) vorzuziehen.
- Ein Proxy kapselt den Zugriff auf ein anderes Objekt vollständig ab und ist wie ein Stellvertreter.
- Chain of Responsibility ist dann angebracht, wenn eine Aufgabe potentiell von mehreren Handlern übernommen werden kann, aber für eine konkrete Aufgabe im voraus nicht klar ist, welcher Handler wirklich zuständig ist.

Ausblick

- In der nächsten Lerneinheit werden wir:
  - weitere Design Patterns kennenlernen und anwenden.

Quellenverzeichnis

[1] Larman, C.: UML 2 und Patterns angewendet, mitp Professional, 2005

[2] Seidel, M. et al.: UML @ Classroom: Eine Einführung in die objektorientierte Modellierung, dpunkt.verlag, 2012

[3] Martin, R. C.: Clean Architecture: A Craftsman's Guide to Software Structure and Design, mitp Professional, 2018

[4] Gamma, E et al.: Design Patterns: Elements of Reusable Object-Oriented Software Addison Wesley Longman, 1995

[5] McDonald, J: DZone Refcardz: Design Patterns, www.dzone.com, 2008