DESIGN PATTERN

Entwurfsmuster

Christian Schirmer

Entwurfsmuster

- Die Entwurfsmuster gliedern sich in mehrere Unterkategorien
  - *Erzeugungsmuster*
  - *Strukturmuster*
  - *Verhaltensmuster*
  - *Datenmuster*
  - *Gui-Muster*
  - *Muster verteilter Architekturen*

Okay, aber wofür?

- Entwurfsmuster stellen wiederverwendbare Lösungen zu unterschiedlichen Problemen dar.
  - *Irgendwer hat doch da schon mal was gemacht?*
    - Wie verhindere ich, dass mehr als eine Verbindung zur Datenbank aufgebaut wird.
      - *Das Singleton Muster beschreibt genau diesen Lösungsansatz.*
    - Wie verhindere ich Seiteneffekte durch das Verändern der Inhalte eines Objektes.
      - *Die Klasse wird nach dem Immutable Pattern erstellt. – Korrekterweise ist das Immutable Pattern kein echtes Pattern, sondern eher ein Konzept. Wie auch immer, es hat nicht den Weg in den Heiligen Kreis der Design Pattern geschafft…. bis jetzt.*
    - Wenn für die Erstellung eines Objektes Komplexe Schritte notwendig sind, diese aber losgelöst durchgeführt werden soll.
      - *Das Builder Pattern. Am Ende soll ein Konkretes Objekt entstehen, unabhängig davon aus wie vielen einzelnen Teilen es besteht. Der Client soll sich nicht mit den Einzelheiten befassen. Ein Erstellungsprozess*

Pattern in diesem Kurs

- Entwurfsmuster
  - *Singleton*
  - *Builder*
  - *Adapter auch Hüllklasse oder der Wrapper*
  - *Factory*
  - *DAO – Data Access Object*

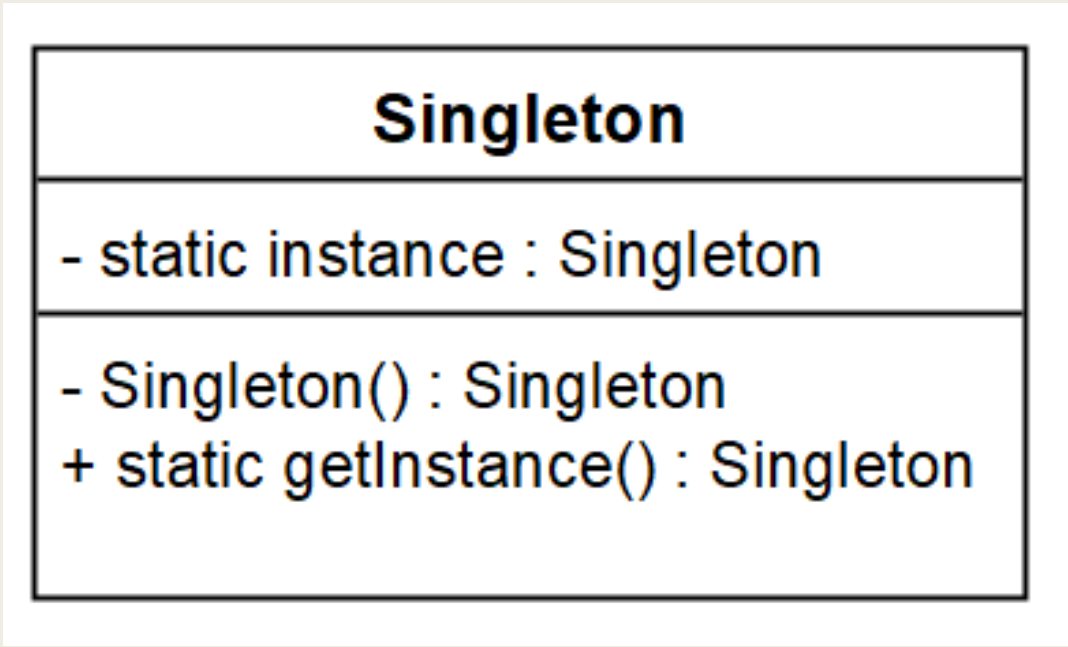
- Umsetzungskonzepte
  - *Immutable Objects*
  - *Enum*

DESIGNPATTERN

Entwurfsmuster

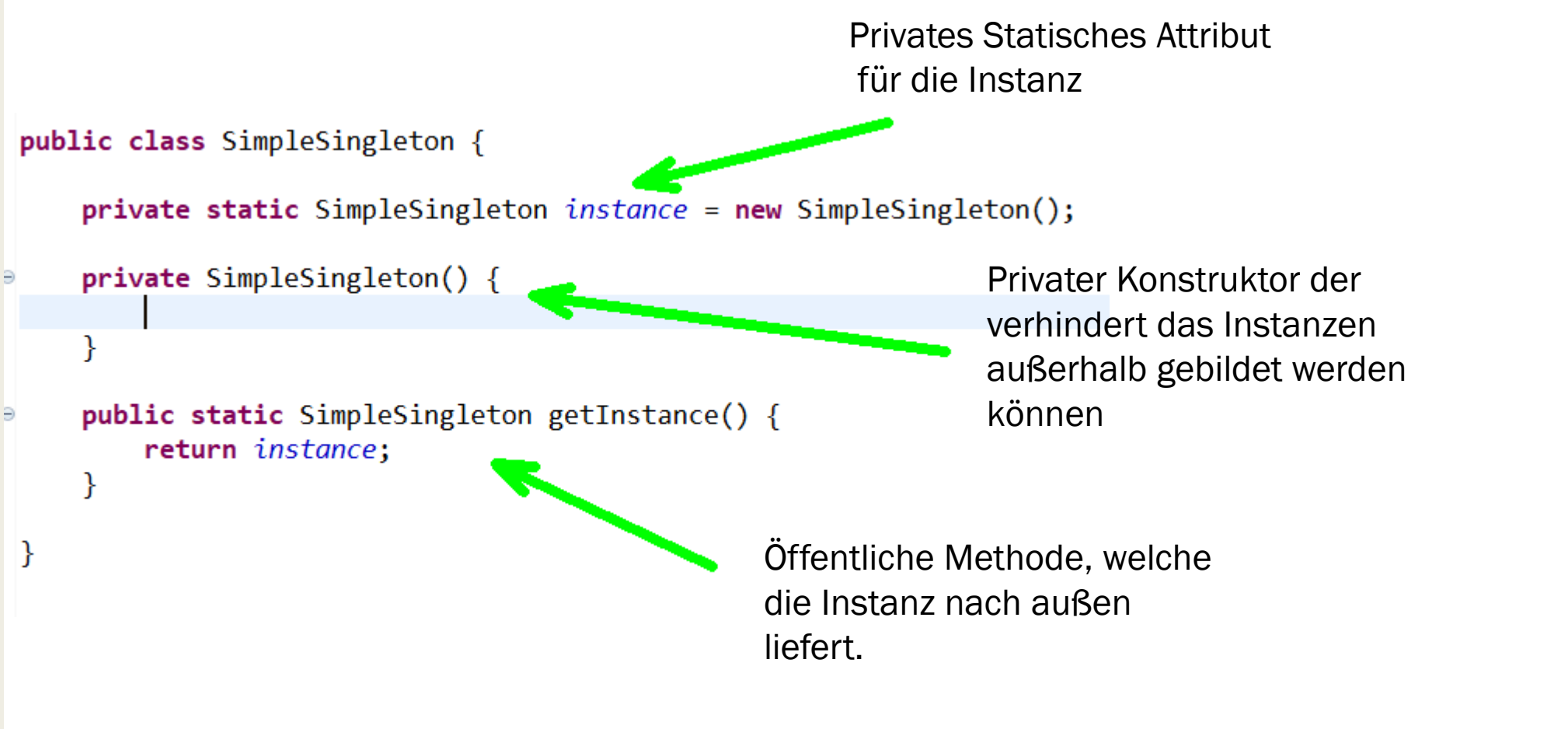
SINGLETON

Singleton – UML Klassendiagramm



- Das Singleton Pattern ist ein relativ Simples, wenn auch wirkungsvolles Pattern.
  - *Ein Privates Statisches Attribut vom Typ der Klasse selbst.*
  - *Alle Konstruktoren werden auf private gesetzt.*
  - *Eine Öffentliche zugriffsmethode, womit die Statische Instanz nach außen gelieẞert wird.*

Simple Singleton



Simple Singleton

- Allerdings ist das Simple Singleton sehr eifrig mit dem erstellen einer Instanz der Klasse.
  - *Sofern die Klasse vom einem Classloader erfasst wird, wird die Instanz angelegt.*
  - *Wenn diese allerdings nie benötigt wird im Programmdurchlauf, haben wir unnötigerweise Speicher verbraucht.*

Simple Singleton - II

- Besser ist es, abzuwarten ob eine Instanz benötigt wird und diese erst dann zu erzeugen.

```
public class SimpleSingleton02 {

    private static SimpleSingleton02 instance = null;

    private SimpleSingleton02() {

    }

    public static SimpleSingleton02 getInstance() {
        if (instance == null) {
            instance = new SimpleSingleton02();
        }
        return instance;
    }
```

Vorerst Referenziert instance auf „null“

Wenn getInstance() aufgerufen wird, wird geprüft ob eine Instanz vorhanden ist, wenn nicht wird eine erstellt.

Singleton und Threads

- Soweit so gut. Doch was machen wir in einer Multithreaded umgebung?
  - *Annahme, 2 Threads benötigen eines unserer Singleton Klassen.*
    - Thread 1 prüft ob eine Instanz vorliegt. Da dies der erste Aufruf von getInstance() ist, liegt keine vor und wird erstellt.
      - *Genau am Anfang der Zeile „instance = new SimpleSingleton02(); „ wird der Thread vom Threadscheduler unterbrochen. (Threads, Threadpool sowie der Scheduler kommen später im OCP noch dran.)*
      - *Nun wurde dem 2. Thread Ausführungszeit gegeben und dieser benötigt ebenfalls unser Singleton. Die Prüfung von if (instance == null) ergibt true. Also wird eine Instanz angelegt.*
        - Thread 2 arbeitet mit dieser Instanz und wird vom Scheduler gestoppt, damit andere Threads arbeiten können.
      - *Thread 1 bekommt wieder ausführungszeit. Nun arbeitet Thread 1 seine Code ab, welcher wieder eine Instanz erzeugt. – Das Chaos möge seinen lauf nehmen. T1 verändert weitere Attribute an der Instanz und ….*
      - *T2 ist wieder da. Trifft nun auf eine veränderte Instanz und es können die merkwüdigsten Nebeneffekte auftreten.*

Singleton – Threadsafe gestalten

- Lange rede kurzer sinn. In einer Multithreaded Umgebung müssen wir unsere Singleton Klasse Threadsafe konstuieren.

```
public class SingletonThreadsafe {

    private static SingletonThreadsafe instance = null;

    private SingletonThreadsafe() {

    }

    public static synchronized SingletonThreadsafe getInstance() {
        if (instance == null) {
            instance = new SingletonThreadsafe();
        }
        return instance;
    }
```

Durch das Schlüsselwort „synchronized“ wird in Multithreaded Umgebungen ein Paralleler ablauf unterbunden.

Es gibt noch weiteres

- Vollständig weitergeführt müsste noch beachtet werden, das Instanzen auch durch
  - *Serialisierung*
  - *Clonebarkeit*
  - *Reflection*
  - *Laden mit mehreren class loadern*

  Erzeugt werden können.

Es gibt unzählige Literatur die sich mit dem Thema auseinandersetzen. Die erstmal wichtigen Punkte wurden besprochen.

ADAPTER PATTERN

Adapter Pattern

- In der Sammlung der Design Pattern der „Gang of Four“ werden 2 Realisierungsalternativen beschrieben.
  - *Die erste stellt eine Adapter mit Delegation dar. (Objektadapter)*
  - *Die zweite ist ein Adapter mit Vererbung. (Klassenadapter)*

- Wir widmen uns der Umsetzung des Adapter mit Delegation.
  - *Der Klassenadapter setzt auch Mehrfachvererbung, welche in Java per Konzept nicht umsetzbar ist.*

Adapter Klasse

- UML-Klassendiagramm einer Umsetzung des Adapterpattern (Objectadapter)

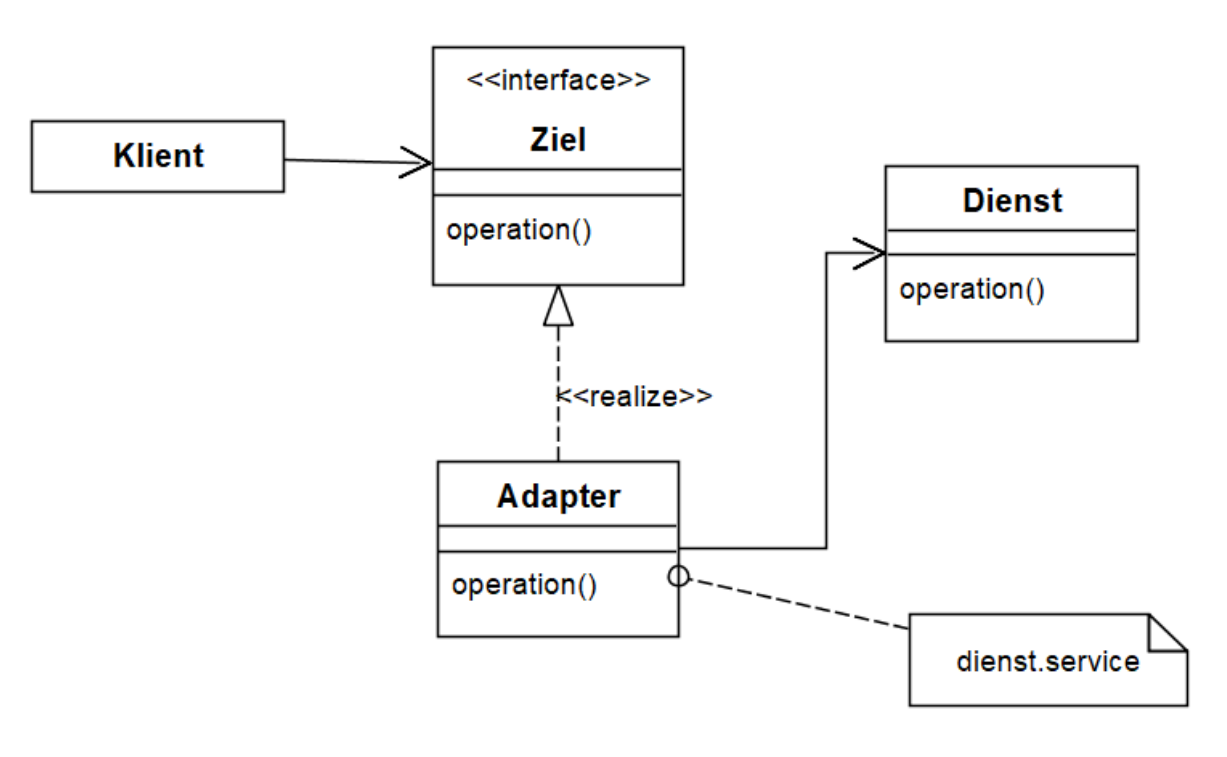


Bild wurd frei nach Wikipedia nachempunden:
https://de.wikipedia.org/wiki/Adapter_(Entwurfsmuster)#/media/File:Objektadapter.svg

Wofür?

- Das beste Anwendungsbeispiel für eine Wrapperklasse ist…
  - *In einem Softwareprojekt werden die Funktionen von sogenannten Third-Party-Librarys eingesetzt. Aus zeitgrunden wird am anfang der Projekt entschieden, die Funktion nicht selbst zu schreiben.*
  - *Die eingesetzte Third-Party-Library wird im Lizenzmodell irgendwann umgestellt, oder die Entwicklung wird eingestellt. Kommerzielle nutzen wird untersagt und und und.*
  - *Es können viele umstände eintreffen wo sich dann das Team ärgert, dass die Third-Party-Lib wieder auf dem Quellcode entfernt werden muss. Überall und an jeder stelle. Eine aufwendige Suche beginnt.*
  - *Wenn am Anfang des Projektes auf Adapterklassen gesetzt worden wäre, müsste jetzt nur an einer Stelle der Code angepasst werden. Die Adapterklasse nimmt die Anfragen Stellvertretend entgegen und deligiert diese dann weiter. Der eigene Quellcode nutzt die Thirs-Party-Lib also nur indirekt.*

Weiterer Nutzen

- Auch denkbar ist es, dass in einem Projekt die Persistenzschicht am Anfang noch nicht klar definiert ist.
  - *Nutzen von XML, CSV, JSON, Serializable und vielen anderen möglichkeiten.*
  - *Eventuell soll auch eine Datenbank angebunden werden an die Persistenz.*

- Stellen Sie sich das Chaos vor, welches entstehen würde wenn das Team das die Persistenzschicht erstellt auf ein anderes System umsteigt. Und Sie als Controller Entwickler sollen nun alles an die „Neue“ Ultra coole Persistenzschicht anpassen.
  - *Deutlich besser wäre da ein Adapter, welcher immer gleich bleibt und als Verbindung zwischen Ihnen und der Persistenzschicht eingesetzt wird.*

KONZEPTE

Keine Pattern in Klassischem sinn.

IMMUTABLE OBJECTS

Immutable Objects

- Bei diesem Konzept werden Klassen so Konzipiert, dass der Inhalt selbst nicht mehr verändert werden kann.
- Es werden keine Setter Methoden nach außen zur verfügung gestellt und alle Attribute werden als final deklariert.
- Um durch Vererbung ein mutable ebenfalls zu verhindern, wird die Klasse selbst als final deklariert.

- Eine berühmte Immutable Klasse in Java ist String.
  - *Strings sind „Immutable“ – wie schön das immer wieder wiederholen zu dürfen :D*
  - *Alle Wrapperklassen aus dem java.lang Pakage, sind ebenfalls Immutable.*
    - Beweis gefällig? Gerne :D