Software Engineering 2
Übungsblatt 3

- Entwurfsmuster 2 -

Ausgabe: 13.10.2022 **Besprechung: 20.10.2022 und 24.10.2022**

Ablauf der Übungen
- Erfolgreiches Bearbeiten von 2/3 der Pflichtübungsaufgaben und persönliche Vorstellung mindestens einer Aufgabenlösung (auf Nachfrage) berechtigen zur Klausurteilnahme.
- Eine Aufgabe gilt als erfolgreich bearbeitet, wenn eine korrekte Lösung bzw. ein nachvollziehbarer Lösungsversuch termingerecht abgegeben wurden.
- Die Aufgaben sollen in 2er Gruppen bearbeitet werden.
- Die Abgabe erfolgt per Moodle als PDF-Datei vor der Besprechung in der Übung.
- Das PDF-Dokument enthält Namen und Matrikelnummern der an der Lösung beteiligten Personen.
- Das (identische) PDF-Dokument wird von allen an der Lösung beteiligten Personen auf Moodle hochgeladen.
- Das Ergebnis soll in der Übung vorgestellt und diskutiert werden.

Aufgabe 3.1: Proxy Pattern

Eine Wetterstation misst die aktuelle Temperatur in Fahrenheit und stellt den Messwert über eine Methode `getTemperatur()` zur Verfügung. Da in vielen Ländern aber Celsius als Temperatureinheit verbreitet ist, soll der Temperaturwert in Celsius umgerechnet zur Verfügung gestellt werden, ohne dabei die Klasse Wetterstation zu ändern.

Umrechnungsformel: `tempInCelsius = (tempInFahrenheit - 32) * 5/9`

Modellieren Sie die Problemstellung mit Hilfe des Proxy Pattern und geben Sie die Java-Implementierung an.

Aufgabe 3.2: Proxy Pattern vs. Decorator Pattern

Das Proxy Pattern und das Decorator Pattern erweitern jeweils Objekte um Zuständigkeiten. Welche Gemeinsamkeiten und welche Unterschiede besitzen die beiden Pattern?

Aufgabe 3.3: Command Pattern

In einem Hochschulgebäude sollen sich in jedem Raum die Jalousien und die Beleuchtung über eine Fernbedienung steuern lassen. Die Fernbedienung soll später um weitere Steuerungsfunktionen im SmartHome-Bereich erweitert werden können. Die Steuerung der Jalousien und der Beleuchtung stammen von verschiedenen Herstellern und besitzen folglich unterschiedliche proprietäre Ansteuerungsschnittstellen.

a) Modellieren Sie die Problemstellung mit dem Command-Pattern. Achten Sie dabei darauf, dass die Fernbedienung (Methoden `ein(), aus())` und die Jalousie-(`herunterfahren(), hochfahren()`) bzw. Beleuchtungssteuerungen (`anschalten(), ausschalten()`) entkoppelt sind.
b) Veranschaulichen Sie die dynamischen Abläufe durch ein UML-Sequenzdiagramm.
c) Implementieren Sie Ihre Lösung in Java. Dabei kann die Fernbedienung natürlich einfach als Standard-Java-Klasse (ohne Grafikelemente) realisiert werden, deren beide Methoden `ein()` und `aus()` die Ausführung direkt an die konkreten Kommando-Objekte delegieren.

Aufgabe 3.4: Dependency Injection

Dependency Injection ist ein Prinzip der objektorientierten Programmierung durch das sich die Abhängigkeiten zwischen Objekten zur Laufzeit verwalten lassen. Während bei der klassischen objektorientierten Programmierung jedes Objekt dafür verantwortlich ist, seine Abhängigkeiten (Erzeugung von Objekten und deren Referenzen) selbst zu verwalten, überträgt Dependency Injection das Erzeugen und Verknüpfen von Objekten an ein separates Objekt. Das separate Objekt "injiziert" die Abhängigkeit durch die Übergabe der Referenz als Parameter.

Gegeben sind die beiden folgenden Klassen:

```java
class SomeApplication{
    Logger log = new Logger();

    public void notify(String message){
        log.write(message);
    }
}
```

```java
class Logger
{
    public void write(String message){
        System.out.println(message);
    }
}
```

a) Welche Klasse hängt von der anderen Klasse ab?

b) Es gibt mehrere Realisierungsvarianten für Dependency Injection. Recherchieren Sie im Internet nach den Varianten. Verringern Sie die Abhängigkeit zwischen den Klassen durch

   1. Constructor Injection
   2. Setter Injection
   3. Method Injection

   Geben Sie jeweils das Klassendiagramm sowie die veränderte Implementierung an.

c) Wie ließe sich die Abhängigkeit zwischen den Klassen weiter reduzieren?