Software Engineering 2
Übungsblatt 2

- Entwurfsmuster -

Ausgabe: 06.10.2022 **Besprechung: 13.10.2022 und 17.10.2022**

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.

Aufgabe 2.1: Interface versus Abstrakte Klasse

Wie Sie auf den Folien bestimmt gemerkt haben, werden bei den Entwurfsmustern die Oberklassen als Interface oder als abstrakte Klasse modelliert. Wann sollte ein Interface, wann eine abstrakte Klasse verwendet werden?

Aufgabe 2.2: Open-Closed Principle (Offen/Geschlossen-Prinzip)

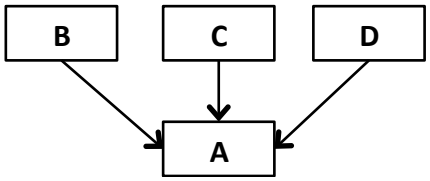
Ein weiteres wichtiges Entwurfsprinzip für objektorientierte Software ist das Open-Closed Principle (OCP):

> "Modules should be both open (for extension) and closed (for modification)."
> [Bertrand Meyer: *Object Oriented Software Construction*. Prentice Hall, 1988]
>
> "Komponenten sollten sowohl offen (für Erweiterungen) als auch verschlossen (für Änderungen) sein."

Eine Komponente ist *verschlossen*, wenn sich deren Schnittstelle mit den angebotenen Operationen nicht verändert und auch deren Implementierung unverändert bleibt. Komponenten sind so zu entwerfen, dass sie sowohl stabil benutzbar (geschlossen) als auch offen für Erweiterungen sind.

a) Zum Beispiel realisiert das Observer-Muster (siehe SE1) das OCP, da neue Beobachter hinzugefügt werden können, ohne den Code des Modells zu ändern. Welche OO-Techniken ermöglichen das OCP?

b) Erweitern Sie folgenden Entwurf: Die Klassen B, C und D verwenden die Klasse A. Bei der Weiterentwicklung des Systems kommen zwei neue Klassen E und F hinzu, die ebenfalls die Klasse A benutzen könnten, wenn Klasse A einige zusätzliche Funktionen anbieten würde.



Aufgabe 2.3: Abstract Factory Pattern (Pflichtaufgabe)

In einem System sollen Parks verwaltet werden. Es gibt verschiedene Arten von Parks: Zum einen Klosterparks, in denen Kräuter angepflanzt werden. Zum anderen Stadtparks, in denen Rosen wachsen. Die Umrandung unterscheidet sich: Der Klosterpark ist von einer Steinmauer, der Stadtpark von einer Hecke umgeben. Der Boden ist auch unterschiedlich: Im Klosterpark liegen Steinplatten, im Stadtpark wächst Gras. Kloster- und Stadtpark sind zwei unterschiedliche Produktfamilien. Jede Produktfamilie hat gleiche Merkmale (Pflanzen, Umrandung, Boden), die aber unterschiedlich ausgestaltet sind.
Ein erster Lösungsansatz zum Anlegen, Pflegen und Verwalten (u.a. Mitarbeiter*innen einsetzen, Bewässerung steuern, Besucher*innen zählen etc.) von Parks könnte wie folgt aussehen:

```
public class Park {
    private enum Parktyp { Klosterpark, Stadtpark };
    private Parktyp park;
    private Boden boden;
    private Pflanze pflanze;
    private Umrandung umrandung;

    public void bodenLegen() {
        switch (park) {
            case Klosterpark:
                boden = new Steinplatte();
            break;

            case Stadtpark:
                boden = new Gras();
            break;
        }
    }

    public void pflaenzeSetzen() {
        switch (park) {
            case KlosterPark:
                pflanze = new Kraeuter();
            break;

            case Stadtpark:
                pflanze = new Rose();
            break;
        }
    }

    public void umranden() {
    // ... wie die obigen Methoden

//  die eigentlichen operativen Funktionen im Park
    public void zuweisenAufgabeMitarbeiter() {
    // ...
    }

    public void steuernBewaesserung() {
    // ...
    }

    public void steuernBeleuchtung() {
    // ...
    }

    public void zaehlenBesucher() {
    // ...
    }

    public void verwaltenFinanzen) {
    // ...
    }

    // ...
    // ...
```

Welche Nachteile hat obige Implementierung? Verbessern Sie den Entwurf der Klasse `Park` durch Anwendung des **Abstract Factory Pattern**. Geben Sie hierzu das resultierende Klassendiagramm und die Implementierung der Java-Klassen an.
Welche Änderungen sind notwendig, um mit "Japanischer-Park" eine neue Parkart hinzuzufügen?

Aufgabe 2.4: Decorator Pattern (Pflichtaufgabe)
Bei einem PKW-Hersteller gibt es wenige Grundmodelle (z.B. Modell_A, Modell_B, Modell_C) mit einer Fülle von optionalen Sonderausstattungen (z.B. Navigationssystem, Lederausstattung, Klimaanlage). Der Gesamtpreis für ein bestelltes Fahrzeug errechnet sich aus dem festen Grundpreis des Grundmodells plus der Preise der gewählten Sonderausstattung.

(a) Modellieren Sie obigen Sachverhalt mit dem **Decorator Pattern**.

(b) Implementieren Sie das Modell in Java. Neben der Preisberechnung mit der Methode `getPreis()` soll es auch eine Methode `getBeschreibung()` geben, die das bestellte Fahrzeug mit seiner Ausstattung beschreibt, z.B.

```
Bestelltes Fahrzeug: Ein Fahrzeug des Modell_B und eine
Klimaanlage und eine Lederausstattung
```