

ÜBUNG SZETTEL MIKROARCHITEKTUR

BLUEJ-BUCH KLASSENENTWURF

Arbeite das Buchkapitel „Klassenentwurf“ (K8). Die Folien dazu sind online. Erledige alle Übungen aus dem Kapitel. Dokumentiere deine Erledigungen.

BLUEJ-BUCH ENTWURF VON ANWENDUNGEN

Erledige das BlueJ-Buch-Kapitel „Entwurf von Anwendungen“ (K15). Die Folien dazu sind online. Erledige alle Übungen aus dem Kapitel. Dokumentiere deine Erledigungen.

ENTWURFSMUSTER 1

Arbeite dich in das Thema Entwurfsmuster ein. Du sollst wissen, was Entwurfsmuster sind, wie sie beschrieben werden und wozu man sie benötigt.

Du bekommst dazu auch eine Einführung im Unterricht anhand des Strategy-Entwurfsmusters.

ENTWURFSMUSTER 2

Arbeite dich in folgende Entwurfsmuster ein:

- Chain of Responsibility
- TemplateMethod
- Decorator
- Builder
- Adapter
- Observer

Einarbeiten heißt:

- Muster lernen
- Muster anhand einer minimalen Aufgabenstellung selbst implementieren
- Alles dokumentieren.

AUSGANGSPUNKT FÜR DIE FOLGENDEN AUFGABEN

Die im Laufe der folgenden Aufgaben zu erstellende Software bildet einen einfachen Währungsrechner nach, der unter Anwendung verschiedener Entwurfsmuster implementiert werden muss. Die abstrakte Klasse WR sowie die Interfaces IUmrechnen und ISammelumrechnung sind als Ausgangspunkt für die Arbeit sind im Anhang gegeben.

```
public interface IUmrechnen{
    double umrechnen(String variante, double betrag);
    double getFaktor();
    boolean zustandig(String variante);
}

public interface ISammelumrechnung{
    public double sammelumrechnen(double[] betraege, String variante);
}

public abstract class WR implements IUmrechnen{
```

```
public double umrechnen(String variante, double betrag){  
    ;  
}  
}
```

Hinweis: Wenn du die folgenden Entwurfsmuster anhand einer zur Währungsrechner-Thematik alternativen Domäne implementieren möchtest, dann ist das natürlich in Absprache mit der LV-Leitung möglich.

CHAIN OF RESPONSIBILITY

Erstellen Sie eine Verantwortlichkeitskette mit mindestens zwei Währungsrechnern (z.B. EUR2YEN, EURO2Dollar). Neben dem Standardverhalten einer Zuständigkeitskette soll es möglich sein, neue Währungsrechner in die Kette aufzunehmen bzw. bestehende aus der Kette zu löschen (jeweils am Ende der Kette). Die Mutterklasse der Chain ist die abstrakte WR-Klasse. Diese implementiert die Ketten-Weiterleitung in der Methode `umrechnen()`. Die Zuständigkeit sowie der Umrechnungsfaktor für die Umrechnung wird in den Unterklassen geklärt (Template Method).

TEMPLATE METHOD

Implementieren Sie die Template-Methode `umrechnen()` in der abstrakten Klasse WR. Delegieren Sie ausschließlich die für die jeweiligen Umrechner spezifischen Methoden (`zustaendig()`, `faktor()`) zur Implementierung an die konkreten Währungsrechner-Unterklassen von WR (Hook-Methoden der Unterklassen).

DECORATOR

Implementieren Sie ein System von Decorators, das Währungsrechner des Typen IUmrechnen dekorieren kann. Folgende Decorators sind z.B. sinnvoll und möglich:

- Belegung eines Umrechnungsvorganges mit Gebühren (z.B. 0,5 % des Umrechnungsbetrages)
- Belegung eines Umrechnungsvorganges für Umrechnungen von Euro nach Währung X (nicht in die andere Richtung) mit fixen Gebühren von 5 Euro.

Hinweis: Leiten Sie dazu zunächst einen abstrakten Decorator von WR ab und implementieren Sie alle abstrakten Methoden sowie `umrechnen()` als transparente Weiterleitungen an den dekorierten Nächsten. Implementieren Sie dann in zwei weiteren Subklassen dieser abstrakten Decorator-Klasse die `umrechnen` Methode neu, sodass die gewünschte Verhaltensänderung des Systems resultiert.

BUILDER

Implementieren Sie das Builder-Pattern (<https://dzone.com/articles/design-patterns-the-builder-pattern>) für einen konkreten Währungsrechner nach Wahl. Der Builder soll es ermöglichen, den Umrechnungsfaktor eines Währungsrechners sowie das nächste Kettenglied in der Chain of Responsibility zu setzen.

ADAPTER

Externe Anwendungen benötigen eine Implementierung der Schnittstelle ISammelumrechnung (siehe oben), um Sammelumrechnungen durchführen zu können. Stellen Sie einen Adapter bereit, der Sammelumrechnungen in der geforderten Form (siehe Methodensignatur) zur Verfügung stellt und dazu die Funktionalität eines IUmrechnen (kann dekoriert sein, kein ein konkreter Währungsrechner sein, kann eine Kette sein) verwendet.

OBSERVER

An einem Währungsrechner sollen sich mehrere Observer registrieren können, die bei einer Umrechnung benachrichtigt werden sollen. Jedes Mal, wenn eine Umrechnung stattfindet, sollen alle Observer benachrichtigt

werden. Alle Informationen der Umrechnungen (Ausgangsbetrag, Ausgangswährung, Zielwährung, Zielbetrag) sollen mit der Benachrichtigung versendet werden. Beispielfhaft sollen zwei Observer implementiert werden:

- 1) Atom-Feed-Observer: Erzeugt einen Atom-Feed mit allen Umrechnungsinformationen und Zeitstempel (verwende dazu: <https://mvnrepository.com/artifact/rome/rome>)
- 2) Log-Observer: Erzeugt eine Log-Text-Datei mit allen Umrechnungsinformationen und Zeitstempel.

Hinweis: Das Observable-Interface soll in der abstrakten WR-Klasse implementiert werden.

SOLID UND SONSTIGE DESIGNPRINZIPIEN

Lerne im Detail die die **SOLID-Prinzipien**:

<https://dev.to/ham8821/solid-principles-to-start-with-object-oriented-programming-1e49>

http://openbook.rheinwerk-verlag.de/oo/oo_03_prinzipien_000.htm#Xxx999144

<https://en.wikipedia.org/wiki/SOLID>

<https://www.youtube.com/watch?v=TMuno5RZNeE>

<https://www.youtube.com/playlist?list=PL4CE9F710017EA77A>

Lerne mindestens 3 **weitere Prinzipien** des objektorientierten Entwurfs kennen (DRY, YAGNI, KISS etc. siehe Moodle-Kurs <https://moodle.tsn.at/course/view.php?id=24763#section-10>).

Wenn du alles im Detail gelernt hast, **weise an verschiedenen Stellen deiner Implementierungen** aus den vorhergehenden Aufgaben **die Realisierung oder die Verletzung** der SOLID-Prinzipien bzw. der übrigen Designprinzipien (DRY, YAGNI, KISS etc. siehe Moodle-Kurs) nach.