Übungsblatt 06: Entwurfsmuster

Software Engineering (WiSe 24/25)

Abgabe: **Fr. 20.12.2024, 10:00 Uhr**
Besprechung: Fr. 13.12.2024

Bitte beachten Sie die Hinweise zur Abgabe in Moodle (siehe "Abgabe Blatt 06")

Aufgabe 1: Anwendung eines Entwurfsmusters (20 Punkte)

Betrachten Sie den folgenden Code. Es wurde bereits eine *Collection* entwickelt (siehe `CollectionImpl` in Listing 1), die es erlaubt, über ihre Elemente zu iterieren. Ebenfalls wurde ein simples User Interface (UI) entwickelt (siehe `CollectionUI` in Listing 2), dass die Elemente einer Collection anzeigt, jedoch verwendet diese UI das Interface `CollectionIterator` (siehe Listing 3), um über die Elemente einer Collection zu iterieren.

Ziel ist es nun, dass die `CollectionUI` auch Instanzen von `CollectionImpl` verarbeiten kann, ohne dass die Klassen `CollectionUI` und `CollectionImpl` und das Interface `CollectionIterator` geändert werden müssen (siehe Listing 4).

Listing 1: `CollectionImpl`

```
1  package de.hu.se.aufgabe1;
2
3  public class CollectionImpl {
4
5      private String[] elements;
6      private int index = 0;
7
8      public CollectionImpl(String[] elements) {
9          this.elements = elements;
10     }
11
12     public boolean hasMoreElements() {
13         return this.index < this.elements.length;
14     }
15
16     public String nextElement() {
17         String elem = this.elements[this.index];
18         this.index++;
19         return elem;
20     }
21
22 }
```

Listing 2: `CollectionUI`

```
1  package de.hu.se.aufgabe1;
2
3  public class CollectionUI {
4
5      public void showCollection(CollectionIterator collectionIterator) {
6          while (collectionIterator.hasNext()) {
7              System.out.println(collectionIterator.next());
8          }
9      }
10
11 }
```

Listing 3: `CollectionIterator`

```java
package de.hu.se.aufgabe1;

public interface CollectionIterator {

    public boolean hasNext();

    public String next();

}
```

Listing 4: `Client`

```java
package de.hu.se.aufgabe1;

public class Client {

    public static void main(String[] args) {
        CollectionImpl c = new CollectionImpl(new String[]{"elem1", "elem2"});

        // We can directly use the class CollectionImpl to list all elements
        // but we want to use the UI!

        // UI is expecting a CollectionIterator.
        // TODO How can we use a CollectionImpl object in the UI?
        CollectionIterator collectionIterator = null;

        CollectionUI ui = new CollectionUI();
        ui.showCollection(collectionIterator);
    }
}
```

(a) *(8 Punkte)* Diese Aufgabe ist in der Moodle Übung 06 zu bearbeiten.

(b) *(12 Punkte)* Erweitern Sie den Code um dieses Entwurfsmuster, so dass die `CollectionUI` die Elemente von `CollectionImpl`-Instanzen anzeigen kann. Die vorgegebenen Klassen (`CollectionImpl` und `CollectionUI`) und das Interface (`CollectionIterator`) dürfen hierbei **nicht** geändert werden. Ergänzen bzw. stellen Sie eine Klasse mit einer Main-Methode bereit, die Ihre Lösung beispielhaft ausführt.

Aufgabe 2: Anwendung eines Entwurfsmusters (40 Punkte)

Das Kassensystem eines Cafés soll weiterentwickelt werden. Bisher wurden bereits die Klassen für Getränke entwickelt, insbesondere für Kaffee (siehe `Coffee` in Listing 7) und entkoffeinierter Kaffee (siehe `Decaf` in Listing 8), die beide von der abstrakten Klasse `Beverage` (siehe Listing 9) erben. Insbesondere ist die Methode `costs()`, die den Preis eines Getränks berechnet, abstrakt und muss für jede Art von Getränk festgelegt und implementiert werden.

Für jedes der beiden Arten von Getränken (`Coffee` and `Decaf`) kann ein Gast eine beliebige Anzahl und Kombination von folgenden Zusätzen bestellen, die jeweils zusätzlich kosten:

- Milch (Preis: 0.15)
- Sojamilch (0.20)
- Zucker (0.10)
- Sahne (0.30)

Ziel ist es, den bestehenden Code unter Verwendung eines Entwurfsmusters zu erweitern, insbesondere sollen die Eigenschaften eines Getränks *dynamisch und flexibel* um die ausgewählten Zusätze zu dem Getränk ergänzt werden. Entsprechend soll das Verhalten zur Preisberechnung und die Beschreibung eines Getränks um die gewählten Zusätze ergänzt werden. Es ist keine Option, die verschiedenen Möglichkeiten und Kombinationen von Getränken und Zusätzen aufgrund deren Anzahl aufzuzählen (z.B. `CoffeeWithMilk`, `CoffeeWithTwoMilk`, etc.).

Listing 7: `Coffee`

```java
package de.hu.se.aufgabe2;

public class Coffee extends Beverage {

    public Coffee() {
        description = "House Blend Coffee";
    }
```

```
8
9       @Override
10      public double costs() {
11          return 1.10;
12      }
13  }
```

Listing 8: `Decaf`

```
1   package de.hu.se.aufgabe2;
2
3   public class Decaf extends Beverage {
4
5       public Decaf() {
6           description = "Decaf Coffee";
7       }
8
9       @Override
10      public double costs() {
11          return 1.20;
12      }
13  }
```

Listing 9: `Beverage`

```
1   package de.hu.se.aufgabe2;
2
3   public abstract class Beverage {
4       String description = "Unknown Beverage";
5
6       public String getDescription() {
7           return description;
8       }
9
10      public abstract double costs();
11  }
```

(a) *(16 Punkte)* Diese Aufgabe ist in der Moodle Übung 06 zu bearbeiten.

(b) *(24 Punkte)* Erweitern Sie den Code um dieses Entwurfsmuster, so dass die vorgegebenen Klassen `Coffee` und `Decaf` dynamisch und flexibel um Eigenschaften ergänzt werden können, die eine beliebige Auswahl an Zusätzen berücksichtigen. Ergänzen bzw. stellen Sie eine Klasse mit einer Main-Methode bereit, die Ihre Lösung beispielhaft ausführt.

Aufgabe 3: Anwendung eines Entwurfsmusters (40 Punkte)

Betrachten Sie den folgenden Code, der einen Einkaufswagen für Lebensmittel (siehe `ShoppingCart` in Listing 16) beschreibt. Lebensmittel (siehe `ShoppingItem` in Listing 17) sind entweder Grundnahrungsmittel (siehe `Food` in Listing 18) oder Luxuslebensmittel (siehe `LuxuryFood` in Listing 19).

Mit Hilfe dieser Klassen lässt sich eine Objektstruktur erzeugen, die einen gefüllten Einkaufswagen beschreibt. Ziel ist es, diese Objektstruktur um Verhalten zu erweitern, ohne dass neue Attribute und Referenzen zu diesen Klassen hinzugefügt werden. Konkret soll für einen gefüllten Warenkorb die Mehrwertsteuer berechnet werden. Da die Software in Supermärkten in Deutschland (19% auf Luxuslebensmittel und 7% auf Grundnahrungsmittel) und Italien (22% auf Luxuslebensmittel und 10% auf Grundnahrungsmittel) eingesetzt wird, soll für beide Länder die Mehrwertsteuerberechnung unterstützt werden. In Zukunft soll es auch leicht möglich sein, weitere Länder zu unterstützen. Daher sollte die Implementierung des Verhaltens zur Berechnung der Mehrwertsteuer möglichst außerhalb der vorgegebenen Klassen bzw. Objektstruktur erfolgen.

Listing 16: `ShoppingCart`

```
1   package de.hu.se.aufgabe3;
2   import java.util.ArrayList;
3   import java.util.Collection;
4   import java.util.Collections;
5
6   public class ShoppingCart {
7
8       private Collection<ShoppingItem> items = new ArrayList<ShoppingItem>();
9
```

```
10      public void addItem(ShoppingItem item) {
11          items.add(item);
12      }
13
14      public Collection<ShoppingItem> getItems(){
15          return Collections.unmodifiableCollection(items);
16      }
17  }
```

Listing 17: `ShoppingItem`

```
1   package de.hu.se.aufgabe3;
2
3   public abstract class ShoppingItem {
4
5       private double price;
6
7       public ShoppingItem(double price) {
8           this.price = price;
9       }
10
11      public double getPrice() {
12          return price;
13      }
14
15      public void setPrice(double price) {
16          this.price = price;
17      }
18  }
```

Listing 18: `Food`

```
1   package de.hu.se.aufgabe3;
2
3   public class Food extends ShoppingItem {
4
5       public Food(double price) {
6           super(price);
7       }
8   }
```

Listing 19: `LuxuryFood`

```
1   package de.hu.se.aufgabe3;
2
3   public class LuxuryFood extends ShoppingItem {
4
5       public LuxuryFood(double price) {
6           super(price);
7       }
8   }
```

(a) *(16 Punkte)* Diese Aufgabe ist in der Moodle Übung 06 zu bearbeiten.

(b) *(24 Punkte)* Erweitern Sie den Code um dieses Entwurfsmuster, ohne dass neue Attribute und Referenzen zu diesen Klassen hinzugefügt werden, während die Berechnung der Mehrwertsteuer möglichst außerhalb der vorgegebenen Klassen bzw. Objektstruktur erfolgen soll. Ergänzen bzw. stellen Sie eine Klasse mit einer Main-Methode bereit, die Ihre Lösung beispielhaft ausführt.