5 Strategie-Muster

Um das Konzept des Strategie-Musters zu verdeutlichen wird ein Programm implementiert, das den Namen des aktuell verwendeten Betriebssystems ausgibt. Schreiben Sie eine Schnittstelle Strategie, die eine Methode `getOS()` deklariert.

```
public interface Strategie {
    public String getOS();
}
```

Schreiben Sie eine Klasse **`LinuxStrategie`**, welche die Schnittstelle **`Strategie`** implementiert.

```
public class LinuxStrategie implements Strategie {
    public String getOS() {
        return "Linux";
    }
}
```

Erstellen Sie analog eine Klasse **`WindowsStrategie`** und eine Klasse **`MacStrategie`**.

Schreiben Sie eine Klasse **`Betriebssystem`** als Kontext, die ein Attribut vom Typ **`Strategie`** besitzt. Implementieren Sie einen Konstruktor mit einem Übergabeparameter vom Typ Strategie. Überladen sie die Methode `toString()` der Klasse **`Betriebssystem`**, die die Methode `getOS()` der Strategie aufruft:

```
public String toString(){
    return strategie.getOS();
}
```

Testen Sie Ihr Programm, indem Sie eine Klasse **`BetriebbsystemTest`** schreiben. Um zu überprüfen welches Betriebssystem auf Ihrem Rechner läuft, verwenden Sie die Methode `System.getProperty("os.name")` der Java API:

```
boolean isWin =
    System.getProperty("os.name").startsWith("Windows");
boolean isMac = System.getProperty("os.name").startsWith("Mac");
boolean isLinux =
    System.getProperty("os.name").startsWith("Linux");
```

Erstellen Sie eine Strategie abhängig von dem Betriebssystem:

```
Strategie str;
if (isWin) {
    str =  new WindowsStrategie();
} else if (isMac) {
    str = new MacStrategie();
} else if (isLinux) {
    str = new LinuxStrategie();
} else {
    str = null;
    System.out.println("OS nicht feststellbar");
}
```

Überprüfen Sie das Ergebnis Ihres Programms indem Sie ein Objekt der Klasse **Betriebssystem** erstellen und die Methode `toString()` aufrufen:

```
Betriebssystem bs = new Betriebssystem(str);
System.out.println(bs);
```

Zeichnen Sie das zum Programm zugehörige Klassendiagramm. Orientieren Sie sich dabei an dem Beispiel aus der Vorlesung.

Aufgaben

1. Für ein Online-Shop sind verschiedene Zahlungsmöglichkeiten zu implementieren. Verwenden sie dazu das Strategie-Muster. Erstellen sie zunächst eine Schnittstelle **Zahlungsstrategie**.

2. Schreiben Sie zwei neue Klassen **KreditkartenStrategie** und **PayPalStrategie**, die die Schnittstelle **Zahlungsstrategie** implementieren. Orientieren sie sich dabei an dem UML-Klassendiagramm in Abb. 1. Die `zahle(preis:int)`-Methode soll je nach gewählter Strategie, den Preis, Namen und die Kreditkartennummer oder den Preis und die Email in der Konsole ausgeben.

3. Erstellen sie eine Klasse **Warenkorb**, die ein privates Attribut vom Typ Zahlungsstrategie definiert.

4. Testen sie die erstellten Klassen in der `main`-Methode einer neuen Klasse **WarenkorbTest**. Erstellen Sie dazu Warenkorb Objekte, die verschiedene Zahlungsstrategien als Übergabeparameter bekommen.

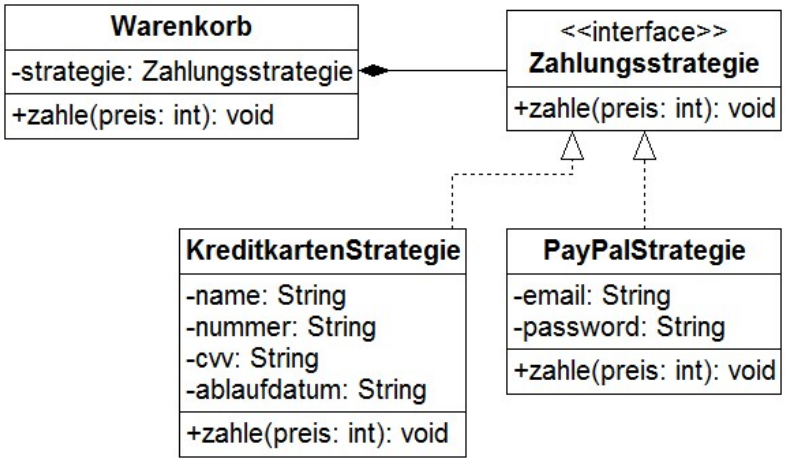

Abbildung 1: Klassendiagramm des Online-Shops

Iterator-Muster

Arbeiten Sie für die nächsten Aufgaben mit dem Binärbaum aus Übung 4 Aufgabe 3 weiter. Dazu können Sie einen neuen Branch ihres Git-Projektes erstellen oder mit ihren lokal gespeicherten Daten zu dieser Aufgabe weiterarbeiten.

Aufgaben

1. Erweitern Sie ihre Implementation passend zu dem UML-Klassendiagramm aus der Vorlesung und Abbildung 3, um das Iterator-Muster umzusetzen. Implementieren Sie einen Iterator für die Tiefensuche (DFS) mithilfe eines Stapels, und einen Iterator für die Breitensuche (BFS) mithilfe einer Warteschlange.

2. Die Standarditeration des Binärbaums soll die Tiefensuche sein. Erweitern Sie das UML-Diagramm um das `Iterable`-Interface und implementieren Sie es.

3. Erweitern Sie Ihren Iterator durch die Funktionen `peek()` und `previous()`. `peek()` soll das aktuelle Element des Iterators zurückgeben ohne die Iteration fortzuführen. `previous()` soll den Iterator "zurückspulen" und das vorherige Element zurückgeben.

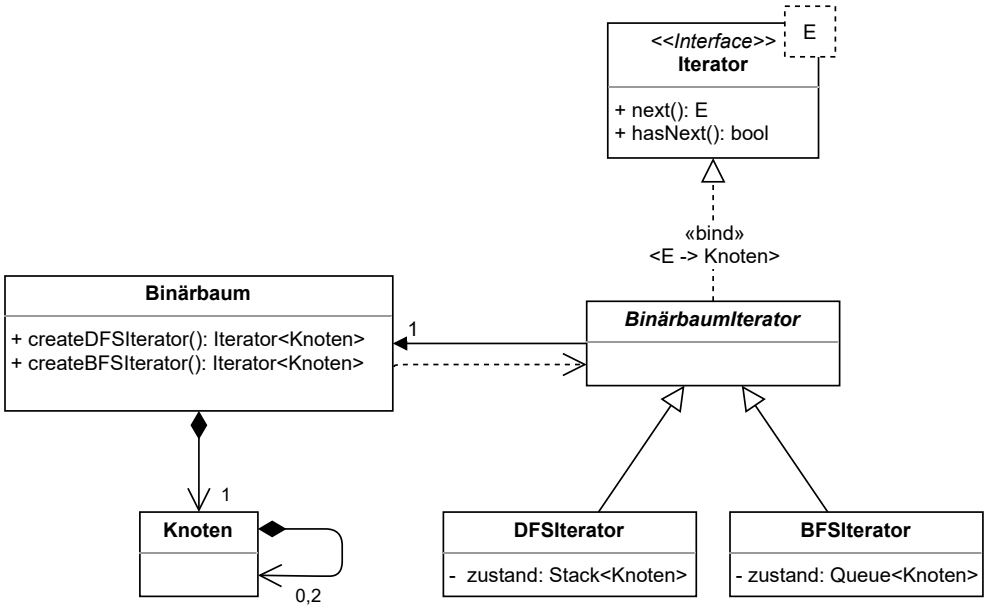

Abbildung 2: UML-Klassendiagramm für Binärbaum-Iteratoren

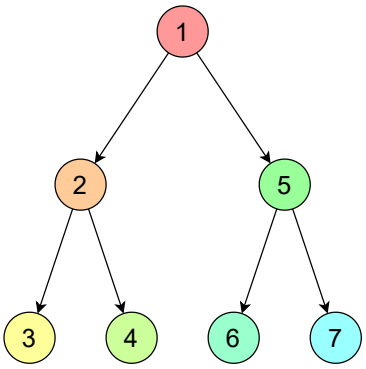

Abbildung 3: Tiefensuche

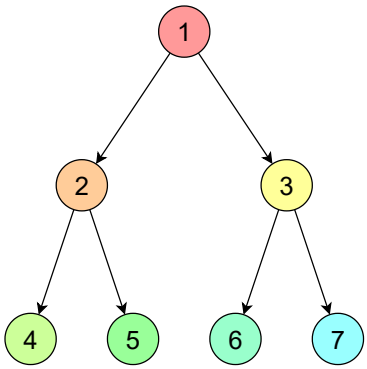

Abbildung 4: Breitensuche