9 Template-Muster

Sie sollen für eine Bank Software zum Analysieren von Aktienkursen entwickeln. Der Finanzdienstleister der Bank reicht Ihnen eine Festplatte mit allen Aktienkursen, doch schnell stellen Sie fest dass die Daten zum einen als .csv Dateien und zum anderen als .txt Dateien gelagert werden. Daten in den CSV-Dateien sind per „,“ getrennt, Daten in den .txt Dateien mit „|“. Sie entscheiden sich, das Template-Muster anzuwenden um beide Dateien einlesen und prüfen zu können.

Aufgaben

1. Erstellen Sie die Klassen `Aktienkurs` und `Tageskurs`, welche die Werte eines Kurses enthalten.
2. Setzen Sie das Template-Muster um. Die Methode `einlesen` des DatenHandlers soll zuerst prüfen, ob die angegebene Datei existiert, dann den Datentypen validieren, und zuletzt die Daten transformieren. Die Methode `validiereDatentyp` testet vor Verarbeitung der Datei, ob der korrekte Datentyp gelesen wird. `transformiereDaten` überführt den Inhalt der Dateien in jeweils ein Objekt vom Typ `Aktienkurs`.
3. Lesen Sie zum Testen die Beispieldateien die in Moodle bereitgestellt sind ein und lassen Sie sich einige Werte in der Konsole ausgeben.

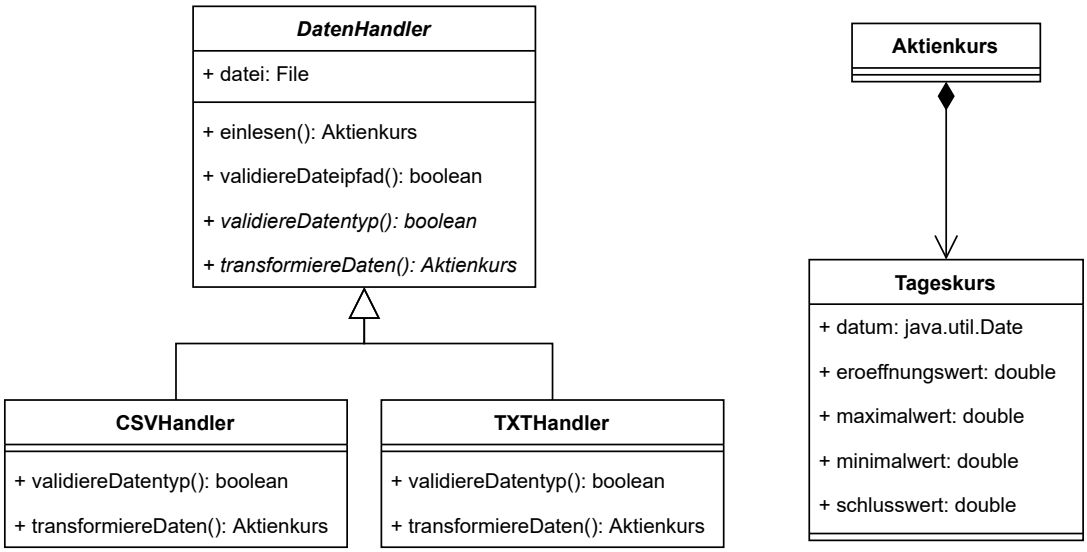

Abbildung 1: Klassendiagramm

Composite-Muster

Die Verzeichnisstruktur Ihres Rechners kann als ein Beispiel für das Composite Muster betrachtet werden. Üblicherweise ist die Verzeichnisstruktur baumartig. Sie beginnt bei einer Wurzel (root) und jeder Knoten dieses Baumes kann entweder eine Datei oder ein Verzeichnis sein.

Implementieren Sie ein Programm, das alle Dateien und Verzeichnisse, die zu einem beliebigen Verzeichnis gehören, auf der Console ausgibt. Zusätzlich soll die Größe eines Verzeichnisses berechnet und ausgegeben werden.

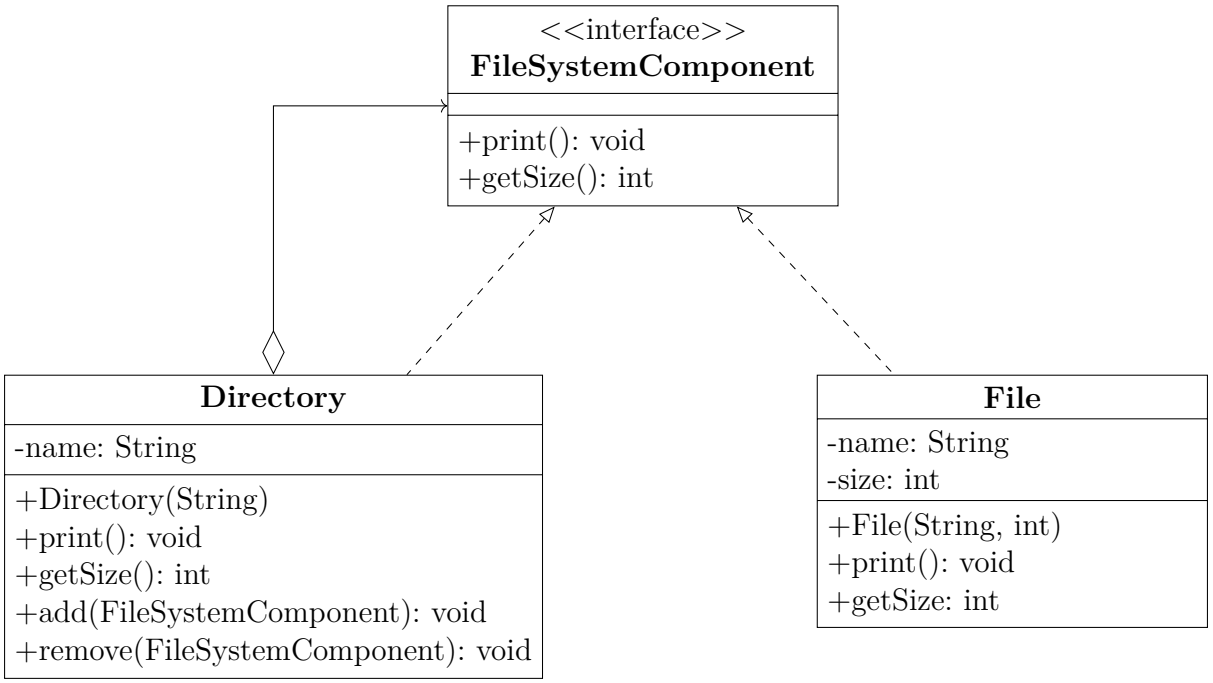

Abbildung 2: Klassendiagramm

Lösung:

FileSystemComponent.java

```
package fileSystem;

public interface FileSystemComponent {

    public void print();
    public int getSize();
}
```

Directory.java

```
package fileSystem;
```

```java
import java.util.ArrayList;

public class Directory implements FileSystemComponent {

    private String name;

    public Directory(String name) {
        this.name = name;
    }

    private ArrayList<FileSystemComponent> dirContents = new
    ↪  ArrayList<FileSystemComponent>();

    public void print() {
        System.out.println(name + ":");
        for (FileSystemComponent component : dirContents) {
            component.print();
        }
    }

    @Override
    public int getSize() {
        int size = 0;
        for (FileSystemComponent component : dirContents) {
            size += component.getSize();
        }
        return size;
    }

    public void add(FileSystemComponent component) {
        dirContents.add(component);
    }

    public void remove(FileSystemComponent component) {
        dirContents.remove(component);
    }

}
```

File.java

```java
package fileSystem;

public class File implements FileSystemComponent {

    private String name;
```

```
    private int size;

    public File(String name, int size) {
        this.name = name;
        this.size = size;
    }

    @Override
    public void print() {
        System.out.println(name);
    }

    @Override
    public int getSize() {
        return size;
    }

}
```

Test.java

```
package fileSystem;

public class Test {

    public static void main(String[] args) {
        FileSystemComponent file1 = new File("uebung01", 384);
        FileSystemComponent file2 = new File("loesung01", 37);
        Directory dir1 = new Directory("uebung");
        dir1.add(file1);
        dir1.add(file2);

        Directory dir2 = new Directory("oom");
        FileSystemComponent file3 = new File("eclipseIDE", 578);
        dir2.add(dir1);
        dir2.add(file3);
        System.out.println("Files in oom");
        dir2.print();
        System.out.println("Directory size: " + dir2.getSize());

        System.out.println("Files in uebung");
        dir1.print();
        System.out.println("Directory size: " + dir1.getSize());
    }
}
```