Objektorientierte Programmierung — Blatt 4

Institut für Softwaretechnik und Programmiersprachen | Sommersemester 2024
Matthias Tichy, Raphael Straub und Florian Sihler

Abgabe (git) bis
26. Mai 2024

First Look at Classes and Objects

6 ★ 3 🟦 9 🟢

- Ein erster Blick auf Klassen und Objekte
- Das Abstraktionskonzept verstehen

Aufgabe 1: Lets represent our Data in Java

🟦🟦🟦

In dieser Aufgabe sollen Sie, basierend auf der folgenden Beschreibung in natürlicher Sprache, passende Klassen ableiten und erstellen. Erstellen Sie alle Klassen, die Sie für die folgende Beschreibung benötigen und begründen Sie gerne, warum Sie diese für nötig oder gerade nicht-nötig halten. Die erstellten Klassen sollen alle nötigen Attribute und Methoden enthalten, aber keine Funktionalität. Das bedeutet, dass Sie die Methode zwar definieren, aber nicht implementieren sollen. Leiten Sie die Klassen aus folgendem Text ab:

Auf dem rauen Wüstenplaneten Arrakis fährt der gewitzte Händler Kaleb mit seinem futuristischen Eiswagen „Desert Delight“ von Oase zu Oase, um den Einheimischen und Reisenden eine seltene Erfrischung anzubieten. Der Eiswagen ist ausgerüstet mit einer Kühlkammer, die drei verschiedenen Eissorten enthält. Zu den Sorten gehören das wertvolle „Melange-Eis“ mit dem seltenen Gewürz Melange, sowie Sorten wie „Sanddünen-Schokolade“ und „Oasen-Fruchtmix“.

Kaleb führt eine Liste aller Sorten und ihrer aktuellen Bestandsmengen in der Kühlkammer. Jede Sorte hat Eigenschaften wie Name, Preis und Lagerbestand, der durch die Lagerverwaltung überwacht wird. Wenn der Lagerbestand einer Sorte zur Neige geht, muss Kaleb seine Route neu planen, um seine Vorräte aufzufüllen.

Kaleb hat auch eine Liste seiner Kunden mit ihren persönlichen Vorlieben und Bestellungen. Dabei gibt es verschiedene Kunden: von Nomaden, die spontan Eis kaufen, bis hin zu Fremen, die Melange-Eis lieben und deshalb lange im Voraus bestellen. Jede Vorbestellung spezifiziert die Sorte und Menge und kann zu einem beliebigen Zeitpunkt abgeholt werden.

Aufgabe 2: We need better Code

🟦🟦🟦

Betrachten Sie den folgenden Code, den Sie auch unter `src/main/java/oop/exercise2/SpiceHarvester.java` in Ihrem Repository finden können:

```java
public class SpiceHarvester {
    private String id;
    private String name;
    private String model;
    private String manufacturer;
    private String serialNumber;

    private String currentLocation;
    private String destination;
```

```
12      private float latitude;
13      private float longitude;
14      private float altitude;
15
16      private int enginePower;
17      private int engineEfficiency;
18      private int engineTemperature;
19      private int coolingSystemStatus;
20
21
22      private int cargoCapacity;
23      private int currentCargoLoad;
24      private String cargoType;
25
26
27      private boolean isCargoFull;
28      private boolean isCargoEmpty;
29      private int crewCapacity;
30      private String[] crewMembers;
31      private boolean isCrewOnboard;
32      private boolean isCrewAssigned;
33
34      private int ambientTemperature;
35      private int ambientHumidity;
36      private int windSpeed;
37      private int currentCrewCount;
38      private int windDirection;
39      private int sandDensity;
40  }
```

Leider ist dieser Code nicht besonders gut strukturiert. Können Sie ihn verbessern? Schreiben Sie eine bessere Version des Codes in dem Sie die existierenden Attribute in sinnvolle Klassen aufteilen. Entschieden Sie darüber hinaus für jedes Attribut, ob es `final` sein sollte oder nicht.

Aufgabe 3: Locomotive Loop

In dieser Aufgabe wollen wir einen Güterzug im Kreis fahren lassen und dabei Spice zwischen den Bahnhöfen transportieren. Wir stellen dabei nur Tests für die letzte Teilaufgabe zur Verfügung. Sie können allerdings selbst weitere Tests schreiben, um sicherzustellen, dass ihr Code korrekt funktioniert.

Hinweis: Die Klassendiagramme in dieser Aufgabe stellen nur die notwendigen Attribute und Methoden dar. Es steht ihnen frei weitere Methoden oder Attribute hinzuzufügen. Damit die Tests korrekt funktionieren, müssen Sie allerdings sicherstellen, dass die Attribute und Methoden die gleichen Namen beziehungsweise Signaturen haben.

a) Der Zug 

Implementieren Sie eine Klasse `Train`, die einen Zug repräsentiert. Unsere Züge haben eine maximale Kapazität an Spice, die sie transportieren können, sowie die aktuell geladene Menge an Spice. Außerdem hat der Zug eine Menge an Treibstoff die er verbraucht, um sich zu bewegen. Die Bewegung des Zuges implementieren wir dabei in einer späteren Teilaufgabe.

b) Das Spice-Transportschein-Zertifikat

Jeder Zug hält eine Liste an sogenannten *Spice-Transportschein-Zertifikaten*™ (patent-pending). Ein solcher Transportschein enthält die Information, welche Menge des transportierten Spices an einem bestimmten Bahnhof aufgenommen oder abgeladen werden soll. Folglich enthält ein Transportschein die Menge des Spice und den Bahnhof, an dem das Spice aufgenommen oder abgeladen werden soll. Erstellen Sie eine Klasse `SpiceTransportCertificate`, die einen Transportschein repräsentiert. Erweitern Sie anschließend die Klasse `Train` um eine Liste von Transportscheinen. Verwenden Sie für hierfür eine `ArrayList<SpiceTransportCertificate>`.

Train
- maxCapacity: int
- currentSpice: int
- fuel: int
- spiceCertificatList: List<SpiceTransportCertificate>

SpiceTransportCertificate
- amount: int
- stationName: String

c) Der Bahnhof

Erstellen Sie eine Klasse `Station`, die einen Bahnhof repräsentiert. Ein Bahnhof hat einen Namen, eine Menge an Spice (die er anbietet), und einen weiteren Bahnhof, zu dem er eine Verbindung hat. Zusätzlich hält ein Bahnhof die Information darüber, wie viel Treibstoff benötigt wird, um zum anderen, verbundenen Bahnhof zu gelangen.

Gehen Sie davon aus, dass ein Bahnhof unbegrenzt viel Spice lagern kann.

Station
- name: String
- spiceAmount: int
- nextStation: Station
- fuelRequired: int

d) Die Navigation

Erstellen Sie eine Klasse `TrainNetwork`, die das gesamte Schienennetz repräsentiert. Hierfür ist es ausreichend, einen einzelnen Bahnhof als Startpunkt zu speichern. Sie können davon ausgehen, dass das Schienennetz ein Kreis ist, d.h. jeder Bahnhof hat eine Verbindung zu einem anderen Bahnhof und der letzte Bahnhof hat eine Verbindung zum ersten Bahnhof.

TrainNetwork
- startStation: Station
- train: Train

+ startJourney(): JourneyResult

Implementieren Sie die Methode `startJourney()` (annotiert mit `@Exercise(task = 3, subTask = 'j')`, wobei das `j` hier für „journey“ stehen soll), die den Zug durch das Schienennetz navigiert. Der Zug soll am Startbahnhof des Schienennetzes beginnen und dann die Gütertransporte durchführen, die auf dem Transportschein stehen. Das bedeutet an jedem Bahnhof soll ein Transportschein (des entsprechenden Bahnhofs) abgearbeitet und das Spice entsprechend umgelagert werden.

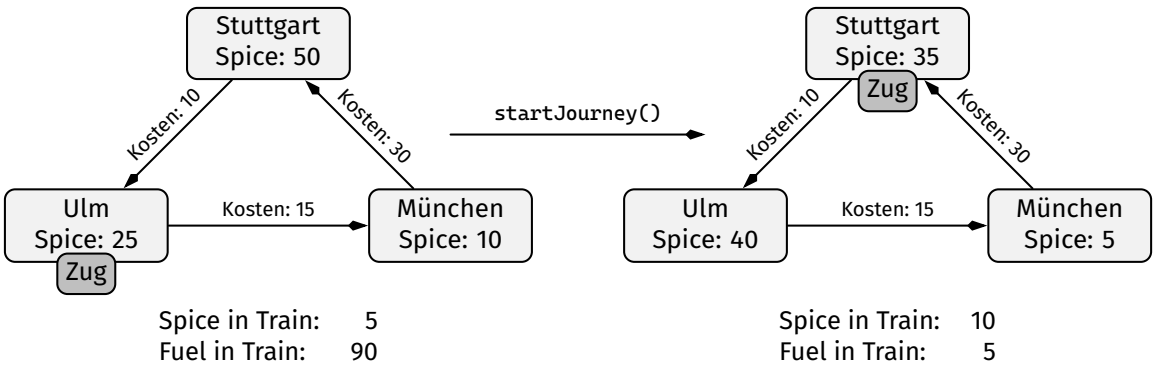

Abbildung 1: Ein beispielhaftes Schienennetz (`getTrainNetworkSmall()` in Teilaufgabe 3e).

Nachdem das Spice verladen ist, soll der Zug zum nächsten Bahnhof fahren. Die Reise ist beendet, wenn alle Gütertransporte abgearbeitet sind, oder wenn der Treibstoff nicht mehr zum nächsten Bahnhof ausreicht.

Falls der Zug keinen Transportschein für den entsprechenden Bahnhof hat, soll er einfach weiterfahren. Da er hierbei nicht bremsen und wieder beschleunigen muss, verbraucht er nur die Hälfte des Treibstoffs (abgerundet zur nächsten Ganzzahl) zum nachfolgenden Bahnhof.

Falls der Zug mehrere Gütertransporte für einen Bahnhof hat, so darf er nur einen einzigen Transportschein abarbeiten. Hierbei soll der Transportschein gewählt werden, der als Erstes in der Liste steht.

Sollte die maximale Kapazität durch einen Transportschein überschritten werden, soll der Zug so viel Spice transportieren wie möglich und den Rest zurücklassen. Dafür bekommt der Zug einen neuen Transportschein, der den Rest des Spice enthält, damit dies zu einem späteren Zeitpunkt abgeholt oder geliefert werden kann. Dieser wird hinten an die Liste angehängt. Das Gleiche passiert auch, falls der Zug mehr Spice abholen soll, als der Bahnhof zur Verfügung stellt. Ebenso, soll so viel wie möglich abgeladen werden und ein neuer Transportschein erstellt werden, falls der Zug nicht genug Spice zur Verfügung hat.

Sobald die Reise beendet ist, soll die `startJourney()` Methode ein `JourneyResult`-Objekt zurückgeben. Es ist Ihnen überlassen, wie Sie die Informationen in dem `JourneyResult`-Objekt speichern. Allerdings sollen die Informationen über die folgenden Methoden abrufbar sein:

```
public class JourneyResult {
   @Exercise(task = 3, subTask = '1')
   public int stationsVisited();
   @Exercise(task = 3, subTask = '2')
   public List<SpiceTransportCertificate> remainingCargo();
   @Exercise(task = 3, subTask = '3')
   public int spiceInTrain();
   @Exercise(task = 3, subTask = '4')
   public int fuelInTrain();
   @Exercise(task = 3, subTask = '5')
   public int spiceOfStation(String stationName);
}
```

Beachten Sie, dass Transportscheine, am ersten Bahnhof abgearbeitet werden, bevor der Zug losfährt, dies aber nicht in die Anzahl der besuchten Bahnhöfe einfließen soll. Beachten Sie außerdem, dass die `@Exercise`-Annotationen notwendig sind, um die Tests der folgenden Aufgabe korrekt auszuführen.

e) Der Test 

Wir brauchen für unsere Tests `TrainNetwork` Objekte. Erstellen sie eine Klasse `TrainNetworkSampleProvider` die die folgenden Methoden enthält:

```
public class TrainNetworkSampleProvider {
   @Exercise(task = 3, subTask = 'a')
   public static TrainNetwork getTrainNetworkSmall(){...}

   @Exercise(task = 3, subTask = 'b')
   public static TrainNetwork getTrainNetworkLarge(){...}

   @Exercise(task = 3, subTask = 'c')
   public static TrainNetwork getTrainNetworkComplex(){...}
}
```

Diese Methoden sollen `TrainNetwork` Objekte zurückgeben, die für die Testfälle in der letzten Teilaufgabe verwendet werden können. Im Folgenden finden Sie Erklärungen in natürlicher Sprache, die beschreiben wie diese `TrainNetwork` Objekte aussehen sollen.

- `getTrainNetworkSmall()`: (siehe Abbildung 1)
  - Ein kleines Schienennetz mit 3 Bahnhöfen.
  - Der erste Bahnhof ist 'Ulm', mit 25 Spice. Der zweite Bahnhof ist 'Stuttgart' mit 50 Spice. Der dritte Bahnhof ist 'München' mit 10 Spice.
  - Die Verbindung von Stuttgart nach Ulm benötigt 10 Treibstoff, von Ulm nach München 15 Treibstoff und von München nach Stuttgart 30 Treibstoff.
  - Der Zug hat eine Kapazität von 10 Spice und started mit 5 Spice.
  - Dem Zug liegen die Folgenden Transportscheine vor: 'Ulm': 25 Spice abliefern, 'Stuttgart': 20 Spice abholen, 'München': 5 Spice abholen.
  - Der Zug hat 90 Treibstoff und startet in Ulm.
- `getTrainNetworkLarge()`:
  - Ein großes Schienennetz mit 6 Bahnhöfen.
  - Der erste Bahnhof ist 'Berlin', mit 100 Spice. Der zweite Bahnhof ist 'Hamburg' mit 75 Spice. Der dritte Bahnhof ist 'Frankfurt' mit 50 Spice. Der vierte Bahnhof ist 'Köln' mit 25 Spice. Der fünfte Bahnhof ist 'Düsseldorf' mit 0 Spice. Der sechste Bahnhof ist 'Dresden' mit 0 Spice.
  - Die Verbindungen und Treibstoffkosten sind wie folgt: Berlin-Hamburg 20, Hamburg-Frankfurt 30, Frankfurt-Köln 25, Köln-Düsseldorf 15, Düsseldorf-Dresden 35, Dresden-Berlin 40.
  - Der Zug hat eine Kapazität von 20 Spice und startet mit 10 Spice.
  - Dem Zug liegen die folgenden Transportscheine vor: 'Berlin': 50 Spice abliefern, 'Hamburg': 25 Spice abholen, 'Frankfurt': 10 Spice abliefern, 'Köln': 5 Spice abholen, 'Düsseldorf': 10 Spice abliefern, 'Dresden': 10 Spice abholen.
  - Der Zug hat 200 Treibstoff und Startet in Berlin.
- `getTrainNetworkComplex()`:
  - Ein komplexes Schienennetz mit 5 Bahnhöfen.
  - Der erste Bahnhof ist 'Wien', mit 60 Spice. Der zweite Bahnhof ist 'Graz' mit 40 Spice. Der dritte Bahnhof ist 'Linz' mit 30 Spice. Der vierte Bahnhof ist 'Salzburg' mit 20 Spice. Der fünfte Bahnhof ist 'Innsbruck' mit 10 Spice.
  - Die Verbindungen und Treibstoffkosten sind wie folgt: Wien-Graz 15, Graz-Linz 20, Linz-Salzburg 25, Salzburg-Innsbruck 30, Innsbruck-Wien 35.
  - Der Zug hat eine Kapazität von 15 Spice und startet mit 8 Spice.
  - Dem Zug liegen die folgenden Transportscheine vor: 'Wien': 10 Spice abholen, 20 Spice abliefern; 'Graz': 5 Spice abliefern, 10 Spice abholen; 'Linz': 5 Spice abholen, 15 Spice abliefern; 'Salzburg': 5 Spice abholen, 10 Spice abliefern; 'Innsbruck': 5 Spice abliefern, 5 Spice abholen, 5 Spice abholen.
  - Der Zug hat 150 Treibstoff und startet in Wien.