Softwareentwicklung II – Praktikum

Listen

Elektronische Abgabe bis 26.04.2015, 22:00 Uhr

Hinweise

Halten Sie die Java-Konventionen ein, insbesondere bei der Wahl Ihrer Bezeichner.

Kommentieren Sie Ihre Lösung.

Sichern Sie Ihre Klassen mit geeigneten Unit-Tests ab.

Entwickeln Sie von Beginn an eine ausführbare Hilfsklasse `MainTest.java` mit. Probieren Sie mittels dieser Klasse von Anfang an die von Ihnen programmierten Einzelschritte aus.

Vorgehensweise

Legen Sie in Ihrem Workspace ein neues Java-Projekt für die Aufgaben zu Listen an.

Spielen Sie diejenigen der folgenden Aufgabenstellungen, die die Struktur der Listen (Züge) verändern, zunächst manuell mit Papier und Bleistift anhand von einigen Beispielen durch, bis sie verstanden haben, was prinzipiell zu tun ist.

Abstrahieren Sie dann von den Beispielen zu einem Algorithmus. Notieren Sie diesen auf geeignete Weise, z.B. als Struktogramm. Setzen Sie Ihren Algorithmus anschließend in ein funktionsfähiges Java-Programm um. Probieren Sie dieses aus.

Implementieren Sie geeignete Unit-Tests, um Ihre Lösungen systematisch zu überprüfen. Achten Sie dabei auf eine geeignete Auswahl der Testfälle. Wählen Sie Ihre Testfälle so, dass sie jeden prinzipiell möglichen Ausführungspfad Ihres Programms abdecken.

Kontext

Im Folgenden implementieren Sie ein Programm zur Verwaltung von Zügen.

Dabei erstellen Sie eine Klasse `Wagon.java` zur Repräsentation von Zugwaggons, eine Klasse `Locomotive.java` zur Repräsentation von Lokomotiven und einen Aufzählungstypen `MotivePower.java` zur Repräsentation der verschiedenen Antriebsarten einer Lokomotive. Des Weiteren implementieren Sie eine Klasse `Main.java`, in der Sie verschiedene Züge anlegen und sukzessive um einzelne Wägen erweitern. Dabei wird der Entstehungsprozess eines jeden Zuges in Form einer grafischen Ausgabe dokumentiert.

Diese Übungsaufgabe ist angelehnt an die Grundidee einer der Übungsaufgaben zu Kapitel 4 aus dem Buch von Prof. Dr. Reinhard Schiedermeier.

Hinweis

Lösen Sie die einzelnen Aufgaben schrittweise verzahnt, sodass Sie Ihre einzelnen Lösungsschritte zu den Klassen jeweils testen können.

Bitte geben Sie neben Ihrer Lösung jeweils auch die Testklassen mit ab.

Aufgabe 1: Antriebsarten

Eine Lokomotive kann verschiedene Antriebsarten haben.

a) Implementieren Sie einen geeigneten Datentypen MotivePower.java, um diese Antriebsarten zu repräsentieren. Möglich seien dabei die Antriebsarten `STEAM`, `DIESEL` und `ELECTRIC` (Level 3, Anwenden).

b) Erweitern Sie den Datentypen um eine Methode `toString()`, die als Ergebnis zum jeweiligen Wert eine textuelle Repräsentation zurückliefert (Level 6, Kreieren).

   Diese habe jeweils einen Umfang von 8 Zeichen. Zu kurze Wörter füllen Sie hinten mit Leerzeichen auf.

   Des Weiteren habe die textuelle Repräsentation jeweils einen großen Anfangsbuchstaben, gefolgt von Kleinbuchstaben.

Aufgabe 2: Waggons

Implementieren Sie eine Klasse `Wagon.java` zur Repräsentation von Zugwaggons. Die Klasse `Wagon` realisiere dabei die folgenden Eigenschaften.

a) Jeder Waggon wird durch eine ganzzahlige Nummer identifiziert. Ein einzelner (d. h. noch nicht in einen Zug eingebundener) Waggon hat dabei die Nummer 0 (Level 3, Anwenden).

b) Ein Waggon ist entweder ein Personenwaggon oder ein Güterwaggon.

   Dokumentieren Sie diese Art des Waggons in einem Attribut `isPassengerWagon` eines geeigneten Datentyps (Level 3, Anwenden).

c) Ein Güterwaggon kann mit Gefahrgut beladen werden. Dokumentieren Sie diese Information in einem Attribut `carriesDangerousGoods` eines geeigneten Datentyps (Level 3, Anwenden).

d) Ob ein Waggon ein Personenwaggon ist oder nicht und ob der Waggon Gefahrgut transportiert oder nicht wird jeweils beim Erzeugen des Waggons festgelegt. Danach sei diese Information unveränderlich.

   Ein Personenwaggon darf nicht mit Gefahrgut beladen werden.

   Der Versuch, einen Personenwaggon mit Gefahrgut zu beladen, resultiert in der textuellen Fehlermeldung „Personenwaggon darf kein Gefahrgut enthalten! Waggon nicht beladen!“.

   Erstellen Sie einen geeigneten Konstruktor, der Parameter mit diesen beiden Informationen empfängt, einen neuen Waggon entsprechend initialisiert und dabei die genannten Forderungen umsetzt (Level 6, Kreieren).

e) Prinzipiell soll es möglich sein, hinten an einen Waggon einen anderen Waggon anzuhängen. Erweitern Sie dazu die Klasse `Wagon.java` um ein geeignetes Attribut.

   Standardmäßig habe ein neuer Waggon noch keinen anderen Waggon als Nachfolger (Level 4, Analysieren).

f) Sichern Sie die genannten Attribute vor unbefugtem Zugriff von außen.

Ein Waggon biete die folgenden Verhaltensweisen an.

g) Erstellen Sie für die beiden booleschen Attribute zwei geeignete lesende Zugriffsmethoden (Level 3, Anwenden).

h) Erstellen Sie für die beiden Attribute mit der Nummer des Wagens und mit der Referenz auf den nachfolgenden Wagen im Zug zwei geeignete schreibende Zugriffsmethoden (Level 3, Anwenden).

In der grafischen Darstellung wird ein Waggon je nach Art wie folgt dargestellt:

Güterwaggon mit Gefahrgut	Güterwaggon ohne Gefahrgut	Personenwaggon
`**********`	`**********`	`**********`
`* f00 dg *`	`* f00 *`	`* p00 *`
`**********->`	`**********->`	`**********->`
`*** ***`	`*** ***`	`*** ***`

Der Wagennummer eines Güterwaggons ist also ein „`f`“ (für „freight“) vorangestellt, der Wagennummer eines Personenwaggons dagegen ein „`p`“. Des Weiteren werden Güterwaggons mit Gefahrgut mittels „`dg`“ gekennzeichnet.

i) Erstellen Sie eine Methode `printWagon(int level)`, die einen einzelnen Waggon ausgibt. Planen Sie dabei mit ein, dass Waggons später nicht einzeln, sondern innerhalb von ganzen Zügen auszugeben sind.

   Zur Darstellung eines Zuges wird zunächst von jedem Waggon die Deckelplatte (`level 3`) ausgegeben, dann die Waggonseitenwände inklusive der passenden Beschriftung für jeden Waggon (`level 2`), dann die Bodenplatten und die Anhängerkupplungen der einzelnen Waggons (`level 1`) und zuletzt die Räder (`level 0`) (Level 6, Kreieren).

j) Erstellen Sie eine Methode `trainCarriesDangerousGoods()`, die für eine Folge von Waggons ermittelt, ob einer der enthaltenen Waggons Gefahrgut enthält (Level 6, Kreieren).

Um Züge zu bilden, werden einzelne Waggons zu einer Folge von Waggons aneinandergehängt. Personen- und Güterwaggons dürfen dabei in ein und demselben Zug auftreten.

In einem Zug, in dem auch Personenwaggons hängen, dürfen die eventuell enthaltenen Güterwaggons jedoch kein Gefahrgut transportieren. Wird versucht, einen Waggon mit Gefahrgut in einen Zug einzukoppeln, der auch Personenwaggons enthält, wird eine entsprechende Fehlermeldung „Zug enthält Personenwaggons; Gefahrgutwaggon kann nicht eingefügt werden“ ausgegeben und der Güterwaggon nicht eingekuppelt.

Innerhalb eines Zuges sind unmittelbar hinter der Lokomotive erst alle Personenwaggons eingehängt. Die Güterwaggons befinden sich dagegen am Ende des Zuges.

Die Waggons eines Zuges werden durch laufende Nummern. Personenwaggons und Güterwaggons werden dabei getrennt nummeriert, jeweils beginnend bei `1`. Innerhalb einer Waggonart steigen dabei die Nummern der Waggons, wenn man sich im Zug von der Lokomotive weg ans Ende des Zuges bewegt.

Damit bereits in einem Zug enthaltene Waggons nicht umnummeriert werden müssen, werden neue Waggons jeweils als letzter Waggon der jeweiligen Waggonart in einen bestehenden Zug eingekuppelt. Ein neuer Güterwaggon wandert also immer ans Ende eines bereits bestehenden Zuges. Ein neuer Personenwaggon wird dagegen hinter den bisher im Zug enthaltenen Personenwaggons, aber vor den Güterwaggons in den Zug eingefügt. Ein neuer Personenwaggon kann dabei nur dann in einen bestehenden Zug eingefügt werden, wenn dieser Zug noch keine Güterwaggons umfasst, die Gefahrgut geladen haben. In diesem Fall wird der Personenwaggon nicht eingekuppelt, sondern eine entsprechende Fehlermeldung „Personenwaggon kann nicht eingefügt werden, da Zug bereits Gefahrgut enthält“ ausgegeben.

k) Implementieren Sie eine Methode `addWagon()`, die als Parameter einen neuen Waggon erhält, diesen in eine bestehende, vom aktuellen Waggon ausgehende Waggonfolge einfügt

und als Ergebnis eine Referenz auf den (möglicherweise neuen) Anfang dieser Waggonfolge zurückliefert. Die Methode stelle dabei sicher, dass die oben definierten Eigenschaften für die Reihenfolge der Wagenfolge eingehalten werden (Level 4, Analysieren).

- Diese Methode prüft zunächst auf der Grundlage der oben genannten Bedingungen, ob der neue Waggon überhaupt in die Waggonfolge eingefügt werden darf.
- Falls der neue Waggon nicht in den Zug eingefügt werden darf, wird eine entsprechende Fehlermeldung ausgegeben (siehe oben).
- Falls der neue Waggon ein Personenwaggon ist, der erste Waggon der bestehenden Waggonfolge jedoch ein Güterwaggon ist, fügt die Methode den neuen Waggon vor dem Anfang der aktuellen Waggonfolge als neuen ersten Waggon derselben ein und gibt als Ergebnis die Referenz auf diesen neuen Anfangswaggon zurück. Der neue Waggon wird bei dieser Integration in die bestehende Waggonfolge mit einer geeigneten Nummer versehen.
- Treffen die bisher aufgeführten Fälle nicht zu, so soll der neue Waggon erst weiter hinten in die Waggonfolge eingekuppelt werden. Diese Aufgabe wird an eine private Hilfsmethode `addWagon()` delegiert, die als Parameter den neu einzuhängenden Waggon sowie die Nummer des aktuellen ersten Waggons der Waggonfolge erhält. Diese Methode liefert kein Ergebnis an ihren Aufrufer zurück.

l) Implementieren Sie abschließend diese private Hilfsmethode `addWagon()`. Stellen Sie darin sicher, dass der einzufügende Waggon an der richtigen Stelle eingekuppelt wird und die richtige Waggonnummer erhält (Level 6, Kreieren).

Aufgabe 3: Lokomotiven

Implementieren Sie eine Klasse `Locomotive.java` zur Repräsentation von Lokomotiven. Die Klasse `Locomotive` realisiere dabei die folgenden Eigenschaften und Verhaltensweisen.

a) Jeder Lokomotive ist genau eine der im Datentyp `MotivePower.java` definierten Antriebsarten zugeordnet (Level 3, Anwenden).

b) Eine Lokomotive ist als Zugmaschine für einen Zug konzipiert. Entsprechend kann an eine Lokomotive bei Bedarf ein Waggon angehängt werden. (An diesem Waggon können wiederum weitere Waggons hängen.) An einer einzelnen Lokomotive hängt zunächst erst mal kein Waggon dran. Auch eine einzelne Lokomotive sei im folgenden Sprachgebrauch bereits ein Zug (Level 3, Anwenden).

c) Über einen Zähler ist zu verwalten, wie viele Lokomotiven bisher insgesamt erzeugt wurden. Ein weiterer Zähler dokumentiert, wie viele dieser Lokomotiven mit Dampf getrieben werden (Level 3, Anwenden).

d) Erstellen Sie einen Konstruktor, der als Parameter eine Antriebsart erhält und eine neue Lokomotive entsprechend initialisiert, sodass auch die bisher genannten Bedingungen und Eigenschaften erfüllt sind (Level 4, Analysieren).

e) Erstellen Sie einen zweiten Konstruktor, der zusätzlich zur Antriebsart auch noch einen Waggon als Eingabeparameter erhält. Abgestützt auf den zuvor definierten einfacheren Konstruktor initialisiert auch dieser zweite Konstruktor eine Lokomotive und hängt den Waggon als ersten Waggon an die Lokomotive an (Level 4, Analysieren).

f) Implementieren Sie die Methode `addWagon()`, die als Parameter einen neuen Waggon empfängt, der in einen Zug einzufügen ist. Dabei gelten für die Anordnung und die Nummerierung der Waggons im Zug die in der Klasse `Wagon` festgelegten Regeln.

Auch eine Lokomotive soll grafisch dargestellt werden können in der folgenden Form:

```
 **    ******
 **     *   *
************
* Steam    *
************-->
 ***     ***
```

Dabei wird der Rumpf der Lokomotive mit der entsprechenden Antriebsart beschriftet.

g) Implementieren Sie eine private Hilfsmethode `toString(int level)`, die analog zur zeilenweisen Darstellung eines Waggons eine zeilenweise Darstellung der Lokomotive als Zeichenkette zurück liefert (Level 6, Kreieren).

   Stellen Sie sicher, dass bei der Darstellung eines Zuges nicht nur die Lokomotive, sondern auch alle angehängten Waggons wiedergegeben werden.

h) Die öffentlich aufrufbare Methode `toString()` steuert die zeilenweise Ausgabe einer Lokomotive. Sie stützt sich dabei auf die gleichnamige private Hilfsmethode ab (Level 6, Kreieren).

i) Erstellen Sie die Methode `printTrain()`, die die Darstellung des kompletten Zuges als Zeichenkette auf der Console ausgibt (Level 3, Anwenden).

j) Implementieren Sie die Methode `printStatistics()`, die die Gesamtanzahl der erzeugten Lokomotiven sowie die Anzahl der darin enthaltenen Dampfloks ausgibt, zusammen mit geeigneten textuellen Kommentaren (siehe Ausgabe am Ende des Arbeitsblattes) (Level 3, Anwenden).

Aufgabe 4: Hauptprogramm (Level 3, Anwenden)

Implementieren Sie eine Klasse `Main.java` zur Steuerung Ihrer Applikation.

a) Legen Sie im Hauptprogramm eine Referenz namens `train` an, die auf eine Lokomotive bzw. auf den damit verbundenen Zug verweist.

b) Erzeugen Sie einen ersten Zug, der mit einer Dampflok beginnt. Fügen Sie der Reihe nach die folgenden Waggons in Ihren Zug ein. Geben Sie nach jedem Schritt den Zug aus.

   - Güterwaggon mit Gefahrgut
   - Güterwaggon ohne Gefahrgut
   - Personenwaggon mit Gefahrgut
   - Personenwaggon ohne Gefahrgut

c) Erzeugen Sie einen zweiten Zug, der mit einer Diesellok beginnt, an die sofort ein Personenwaggon ohne Gefahrgut gehängt wird. Fügen Sie der Reihe nach die folgenden Waggons in Ihren Zug ein. Geben Sie nach jedem Schritt den Zug aus.

   - Güterwaggon ohne Gefahrgut
   - Güterwaggon mit Gefahrgut
   - Personenwaggon ohne Gefahrgut
   - Güterwaggon ohne Gefahrgut
   - Personenwaggon ohne Gefahrgut

d) Erzeugen Sie einen dritten Zug, der mit einer Elektrolok beginnt. Fügen Sie der Reihe nach die folgenden Waggons in Ihren Zug ein. Geben Sie nach jedem Schritt den Zug aus.

- Güterwaggon mit Gefahrgut
- Güterwaggon ohne Gefahrgut
- Personenwaggon ohne Gefahrgut
- Güterwaggon ohne Gefahrgut

e) Erzeugen Sie einen vierten Zug, der wieder mit einer Dampflok beginnt, an die sofort ein Personenwaggon ohne Gefahrgut gehängt wird. Fügen Sie der Reihe nach die folgenden Waggons in Ihren Zug ein. Geben Sie nach jedem Schritt den Zug aus.

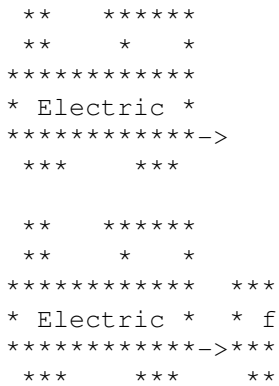
- Güterwaggon ohne Gefahrgut
- Personenwaggon ohne Gefahrgut
- Personenwaggon ohne Gefahrgut
- Güterwaggon ohne Gefahrgut

Die Ausgabe Ihres fertigen Programms sollte in etwa so aussehen:

```
Erster Zug:

 **   ******
 **    *   *
************
* Steam    *
************->
 ***    ***

 **   ******
 **    *   *
************  **********
* Steam    *  * f01 dg *
************->**********->
 ***    ***    ***  ***

 **   ******
 **    *   *
************  **********  **********
* Steam    *  * f01 dg *  * f02    *
************->**********->**********->
 ***    ***    ***  ***    ***  ***

Personenwaggon darf kein Gefahrgut enthalten! Waggon nicht beladen!
Personenwaggon kann nicht eingefügt werden, da Zug bereits Gefahrgut enthält
 **   ******
 **    *   *
************  **********  **********
* Steam    *  * f01 dg *  * f02    *
************->**********->**********->
 ***    ***    ***  ***    ***  ***

Personenwaggon kann nicht eingefügt werden, da Zug bereits Gefahrgut enthält
 **   ******
 **    *   *
************  **********  **********
* Steam    *  * f01 dg *  * f02    *
************->**********->**********->
 ***    ***    ***  ***    ***  ***
```

Zweiter Zug:

Dritter Zug:



```
 **   ******
 **    *   *
************  **********  **********
* Electric *  * f01 dg *  * f02    *
************->**********->**********->
 ***    ***    ***  ***    ***  ***
```

```
Personenwaggon kann nicht eingefügt werden, da Zug bereits Gefahrgut enthält
 **   ******
 **    *   *
************  **********  **********
* Electric *  * f01 dg *  * f02    *
************->**********->**********->
 ***    ***    ***  ***    ***  ***
```

```
 **   ******
 **    *   *
************  **********  **********  **********
* Electric *  * f01 dg *  * f02    *  * f03    *
************->**********->**********->**********->
 ***    ***    ***  ***    ***  ***    ***  ***
```

```
Vierter Zug:
```

```
 **   ******
 **    *   *
************  **********
* Steam    *  * p01    *
************->**********->
 ***    ***    ***  ***
```

```
 **   ******
 **    *   *
************  **********  **********
* Steam    *  * p01    *  * f01    *
************->**********->**********->
 ***    ***    ***  ***    ***  ***
```

```
 **   ******
 **    *   *
************  **********  **********  **********
* Steam    *  * p01    *  * p02    *  * f01    *
************->**********->**********->**********->
 ***    ***    ***  ***    ***  ***    ***  ***
```

```
 **   ******
 **    *   *
************  **********  **********  **********  **********
* Steam    *  * p01    *  * p02    *  * p03    *  * f01    *
************->**********->**********->**********->**********->
 ***    ***    ***  ***    ***  ***    ***  ***    ***  ***
```

```
 **   ******
 **    *   *
************  **********  **********  **********  **********  **********
* Steam    *  * p01    *  * p02    *  * p03    *  * f01    *  * f02    *
************->**********->**********->**********->**********->**********->
 ***    ***    ***  ***    ***  ***    ***  ***    ***  ***    ***  ***
```

```
Insgesamt gibt es 4 Lokomotiven.
Davon sind 2 mit Dampf betrieben.
```