paluno - The Ruhr Institute for Software Technology
Prof. Dr. Michael Goedicke

Miniprojekt 2

Abgabefrist: Dienstag, 19. November 2019, 23:59 Uhr

In diesem Miniprojekt befinden sich drei Klassen, `Train` und `Waggon`, die eine verkettete Liste (Train) und deren Listenelemente (Waggon) darstellen, sowie die Klasse `Miniprojekt2`, welche beide Klassen verwendet und in ihrer `main`-Methode eine anschauliche Ausgabe eines Verwendungsszenarios ausgibt. Die Klasse `Waggon` ist bereits vollständig, so dass Sie ausschließlich die Klasse `Train` bearbeiten müssen.

Die Klassen-, Variablen- und Methodennamen dürfen nicht verändert werden!

Bitte beachten Sie, dass für die jeweilige Testatzulassung die Abgabe einer (nicht notwendigerweise korrekten) Lösung zum Miniprojekt erforderlich ist!

Aufgabe 1: Klasse Waggon

Studieren Sie die Klasse Waggon und ihre Methoden. Ein Verständnis der Klasse ist notwendig für die Bearbeitung von Aufgabe 2.

Aufgabe 2: Klasse Train

Die Klasse Train beschreibt einen Zug, der Passagier-Waggons enthält und stellt technisch eine einfach verkette Liste dar, wobei ein `Waggon`-Objekt ein Listenelement ist.

Hinweise:

- Die Methode `Train.toString()` gibt die Waggon-Namen eines Zugs aus und stellt ihnen eine schematische Lok voran. Bitte beachten Sie, dass dies nur eine Hilfsmethode ist und die Lok auch anzeigt, wenn die Liste leer ist
- Gehen Sie von sinnvollen Parametern innerhalb der zu implementierenden Methoden aus. Ein übergebenes `Waggon`-Objekt ist niemals `null`, ein Index ist niemals kleiner 0.
- Die Reihenfolge in der Sie die Teilaufgaben des Miniprojekts erledigen ist ihnen überlassen. Es kann sinnvoll sein von der gebotenen Reihenfolge abzuweichen.
- Es muss nicht für jede Teilaufgabe eine passende Ausgabe geben. Falls sie weitere Tests oder Ausgaben benötigen, schreiben sie sich in der Main-Methode weitere Testausgaben als Hilfestellung.

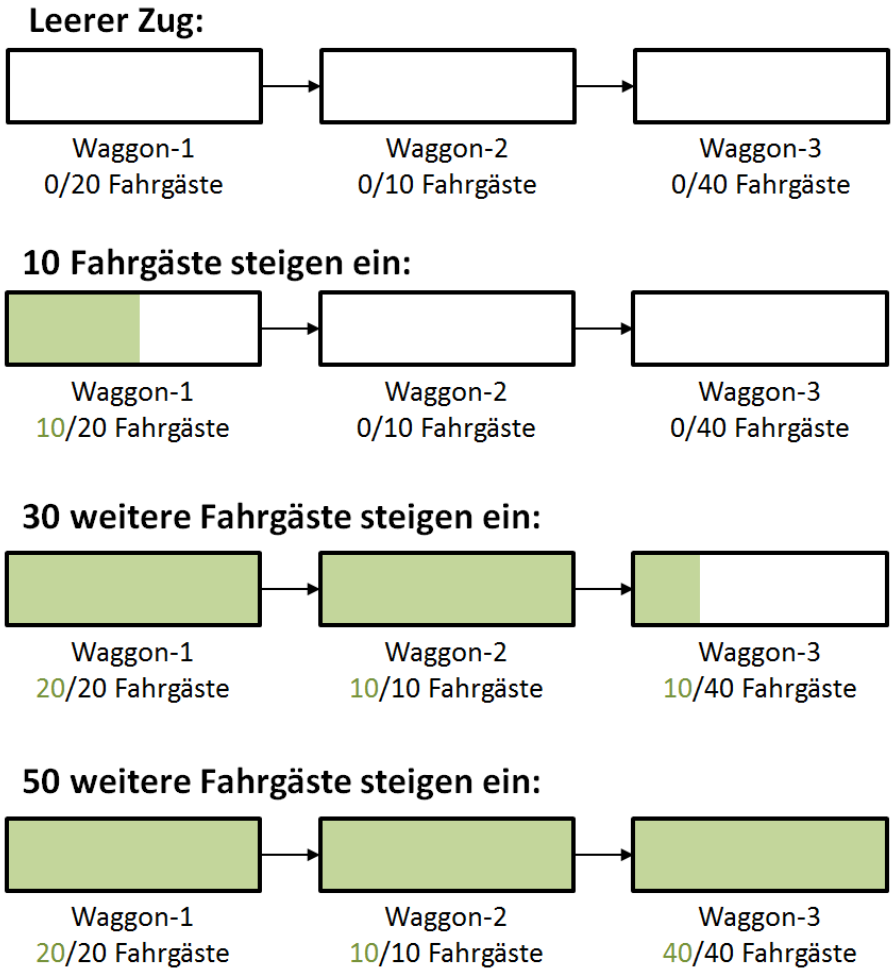
a) **Waggon anhängen:** Die Methode `appendWaggon(Waggon waggon)` soll den übergebenen Waggon `waggon` an das Ende des Zugs anhängen. Ist der Zug leer, soll `waggon` der erste Waggon werden.

b) **n-ten Waggon bestimmen:** Die Methode `getWaggonAt(int index)` soll den Waggon mit dem Index `index` zurückgeben. Ist `index` größer oder gleich der Länge des Zuges, soll `null` zurückgegeben werden, da an dieser Stelle kein Waggon existiert.

c) **Waggons zählen:** Implementieren Sie die Methode `getSize()` so, dass sie die Anzahl der an den Zug angehängten Waggons zurückgibt.

d) **Kapazität zählen:** Implementieren Sie die Methode `getCapacity()` so, dass sie die Gesamtkapazität des Zuges angibt. Die Gesamtkapazität des Zuges ergibt sich aus der Summe der Kapazitäten der einzelnen Waggons.

e) **Fahrgäste einsteigen lassen:** Die Methode `boardPassengers(int numberOfPassengers)` soll `numberOfPassengers` Fahrgäste in den Zug einsteigen lassen. Dafür soll die Anzahl der Passagiere innerhalb der einzelnen Waggons (maximal bis zu ihrer jeweiligen Kapazität) erhöht werden. Nehmen Sie an, dass der Zug nur einen Einstieg vorne hat und sich die Waggons, beginnend mit dem ersten, von vorne nach hinten füllen. Hat der Zug nicht genügend Kapazität für alle zusteigenden Passagiere, so werden die überzähligen Fahrgäste ignoriert.



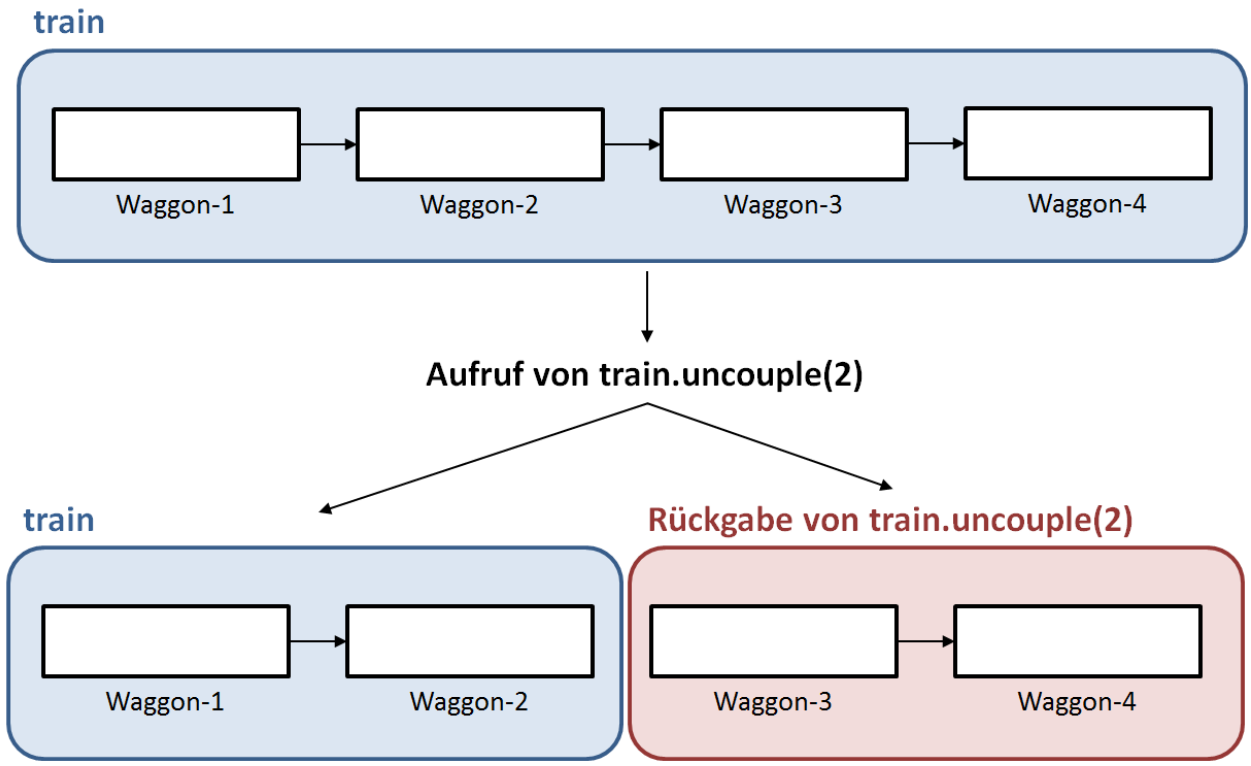
f) **Fahrgäste zählen:** Implementieren Sie die Methode `getPassengerCount()` so, dass sie die Anzahl der sich im Zug befindlichen Fahrgäste zurückgibt. Die Anzahl der Fahrgäste ergibt sich aus der Summe der Fahrgäste in den einzelnen Waggons.

g) **Waggon einfügen:** Die Methode `insertWaggon(Waggon waggon, int index)` soll den Waggon `waggon` als `index`-ten Waggon in den Zug einfügen, wobei der erste Waggon in einem Zug den Index 0 hat. Ist `index` größer oder gleich der Länge des Zuges, soll `waggon` hinten angehängt werden.

h) **Waggon entfernen:** Die Methode `removeWaggon(Waggon waggon)` soll `waggon` aus dem Zug entfernen. Existiert `waggon` nicht, so verändert sich der Zug nicht. Beachten Sie besonders die Sonderfälle:
- Entfernen des letzten Waggons
- Entfernen des einzigen Waggons

i) **Waggons abkoppeln:** Die Methode `uncoupleWaggons(int index)` koppelt Waggons vom Zug ab. Dabei behält das `Train`-Objekt, auf dem die Methode aufgerufen wurde, die ersten `index` Waggons. Die Methode soll zusätzlich einen neuen Zug erstellen, dem die Waggons ab `index` angehören und das daraus resultierende `Train`-Objekt zurückgeben.

Hinweis: Die Methode muss nur für `0 < index < train.getSize()` definiert sein und kann in anderen Fällen bspw. einfach `null` zurückgeben.



j) **Wagenreihenfolge umdrehen:** Die Methode `turnOver()` soll die Wagenreihenfolge umdrehen. Nach dem Aufruf der Methode soll der letzte Waggon des Zuges, auf dem die Methode aufgerufen wurde, der erste Waggon sein, der vorletzte der zweite Waggon, usw.