Programmieren 2 (INF)
Übungsblatt 4

In diesem Übungsblatt geht es um verschiedene Konzepte, gleichartige Daten zu verwalten. Für ein konkretes Anwendungsbeispiel wird eine listenartige Struktur entwickelt. Anschließend werden verschiedene Container aus dem Collection-Framework in Bezug auf ihre Laufzeit verglichen.

Aufgabe 1 (Eisenbahnzüge)

Schreiben Sie einige Klassen, die Eisenbahnzüge repräsentieren. Ein Eisenbahnzug besteht aus einer Lokomotive und einer beliebigen Anzahl Wagen, einschließlich überhaupt keinen Wagen. Lokomotiven und Wagen haben die folgenden Eigenschaften, alle ganzzahlig:

Lokomotive: Länge (Meter), Typ (irgendeine Zahl)
Wagen: Länge (Meter), Passagierkapazität (Anzahl Personen)

Definieren Sie die Klassen `Locomotive` und `Car`, jeweils mit sinnvollen Methoden. Die oben genannten Eigenschaften sind unveränderlich.

Das interessante Problem ist das Zusammenstellen eines Zuges aus Einzelteilen. Der erste Wagen hängt direkt an der Lokomotive. Geben Sie der Klasse `Locomotive` deshalb ein Element `first` vom Typ `Car`, dazu eine Getter- und eine Setter-Methode.

An jedem Wagen hängt der jeweils nächste Wagen, oder gar nichts beim letzten Wagen. Definieren Sie in der Klasse `Car` ein Element `next` des gleichen Typs Car, wieder mit Gettern und Settern. Dieses Datenelement speichert ein anderes Objekt der gleichen Klasse oder null beim letzten Wagen.

Definieren Sie schließlich eine Klasse `Train`, die den ganzen Zug repräsentiert. Ein `Train`-Objekt speichert „seine“ Lokomotive, aber nicht die Wagen. Diese können, einer nach dem anderen, auf dem Weg über die Lokomotive erreicht werden. Die Klasse Train bietet die folgenden Methoden:

- `Konstruktor`
  Der Train-Konstruktor erwartet eine Lokomotive und baut einen ziemlich kurzen Zug, der nur aus der Lokomotive, noch ohne Wagen, besteht.
- `hasCars`
  Prüft, ob der Zug über mehr als nur eine Lokomotive verfügt.
- `add`
  hängt in den Zug einen gegebenen Wagen an der spezifizierten Index-Position ein (Index 0 bedeutet direkt hinter der Lokomotive).

- `getPassengers`
  liefert die gesamte Passagierkapazität des Zuges.
- `getLength`
  liefert die Länge des Zuges in Metern.
- `removeFirst`
  hängt den ersten Wagen aus dem Zug aus und liefert den ausgehängten Wagen als Ergebnis zurück. Die restlichen Wagen bleiben am Zug. Falls es keinen Wagen gibt, wird `null` zurückgegeben.
- `relink`
  akzeptiert als Parameter einen anderen Zug und hängt alle Wagen des anderen Zuges in diesen Zug um. Im anderen Zug bleibt nur die Lokomotive zurück. Nutzen Sie für diese Methode geschickt die vorher definierten. Die Methode soll möglichst effizient sein.
- `revert`
  dreht die Abfolge der Wagen in diesem Zug um, das heißt, der vorher letzte Wagen wird zum ersten, und umgekehrt. Auch diese Aufgabe lässt sich effizient ohne Zwischenspeicherung des Zuges lösen!
- `toString`
  liefert eine Beschreibung, aus der alle wichtigen Zug-Daten hervorgehen.

Schreiben Sie schließlich eine Anwendung, die Folgendes abwickelt:

1. Eine Lokomotive „Big Chief“ mit der Nummer 5311 und der Länge 23 m wird erzeugt.
2. Ein Zug namens „Santa Fe“ mit der Lokomotive „Big Chief“ wird erzeugt.
3. An „Santa Fe“ werden drei Wagen mit den Längen 12 m, 15 m, 20 m und den Passagierkapazitäten 50, 75, 100 Personen angehängt.
4. Eine Lokomotive „Steel Horse“ mit der Nummer 5409 und der Länge 21 m wird erzeugt.
5. Ein Zug namens „Rio Grande Express“ mit der Lokomotive „Steel Horse“ wird erzeugt.
6. An den „Rio Grande Express“ werden zwei Wagen mit den Längen 13 m und 18 m sowie den Passagierkapazitäten 60 und 80 Personen angehängt.
7. Alle Wagen von „Santa Fe“ werden in den „Rio Grande Express“ übernommen.
8. Die Wagenreihenfolge im „Rio Grande Express“ wird umgedreht.

Aufgabe 2 (Effizienz von Listenoperationen)

Lesen Sie sich unter http://download.oracle.com/javase/6/docs/api/ die Methodenbeschreibungen zum `java.util.List`-Interface durch. Schreiben Sie dann ein Testprogramm, das die Effizienz von `ArrayList` und `LinkedList` gegenüberstellt. Vergleichen Sie folgende Operationen für ein hinreichend großes N, wie z.B. 50 000:

- Anfügen von N Elementen an das Ende
- Einschieben von N Elementen am Anfang
- sequentieller Zugriff auf jedes der N Elemente über den Index
- sequenzieller Zugriff auf jedes der N Elemente über einen Iterator

Die Laufzeit des Codes soll jeweils automatisch gemessen und ausgegeben werden.