Praktikum Programmiertechnik (Technische Informatik)
SS 2016, Hochschule für Angewandte Wissenschaften (HAW), Hamburg
Prof. Dr. Philipp Jenke, Prof. Dr. Axel Schmolitzky, Norbert Kasperczyk-Borgmann

Aufgabenblatt 5: Aufzählungstypen, Varargs, Strings

1. Aufgabe 5.1: Schatzkästchen

Schwerpunkte: Enums, Varargs
Aufgabe: Ein Schatzkästchen ist ein Behälter für Edelsteine verschiedener Sorten. In dieser Aufgabe gibt es die drei Sorten Diamant, Rubin und Smaragd.

- Definieren Sie einen Aufzählungstyp `Edelstein` für diese drei Edelsteinarten.
- Schreiben Sie eine Klasse `EdelsteinBox`, die ein Schatzkästchen repräsentiert.

`EdelsteinBox` soll die folgenden öffentlichen Methoden anbieten:

- Einen Konstruktor dem man eine beliebige Anzahl von Edelsteinen als Argumente mitgeben kann.
- `int getAnzahl(Edelstein sorte)`: liefert die Anzahl an enthaltenen Edelsteinen der übergebenen Sorte
- `String toString()`: gibt einen String mit einer Textdarstellung dieses Schatzkästchens zurück (siehe Methode ausgeben()).
- `EdelsteinBox ausgeben()`: gibt auf der Konsole eine Zeile mit einer Textdarstellung dieses Schatzkästchens aus. Für jeden Diamanten wird ein D gedruckt, für jeden Rubin ein R und für jeden Smaragden ein S. Alle Buchstaben folgen lückenlos aufeinander. Die Buchstabenfolge wird in runde Klammern gesetzt. Ein Schatzkästchen mit 1 Diamanten, 2 Rubinen und 3 Smaragden wird zum Beispiel ausgegeben als (DRRSSS)
- `EdelsteinBox hineinlegen(Edelstein sorte, int n)` fügt diesem Schatzkästchen n Edelsteine der Sorte `sorte` hinzu.
- `EdelsteinBox herausnehmen(Edelstein sorte, int n)`: entnimmt diesem Schatzkästchen n Edelsteine der Sorte `sorte`. Wenn weniger als n Edelsteine der Sorte `sorte` enthalten sind, werden alle entnommen.
- `EdelsteinBox umwandeln(Edelstein vonSorte, Edelstein zuSorte)`: wandelt alle Edelsteine der Sorte `vonSorte` in Edelsteine der Sorte `zuSorte` um.
- `EdelsteinBox leeren()`: entleert dieses Schatzkästchen. Es enthält dann keine Edelsteine mehr.

Hinweis: Viele Methoden liefern das eigene Schatzkästchen-Objekt als Ergebnis zurück, um die Anwendung zu vereinfachen. Die Anweisung

```
new EdelsteinBox()
    .hineinlegen(Edelstein.DIAMANT, 1)
    .hineinlegen(Edelstein.RUBIN, 2)
    .hineinlegen(Edelstein.SMARAGD, 3)
    .umwandeln(Edelstein.RUBIN, Edelstein.DIAMANT)
    .ausgeben();
```

liefert: (DDDSSS)
Beispiele:

- `(DRRSSS): hineinlegen(Edelstein.RUBIN, 3) --> (DRRRRRSSS)`
- `(DRRSSS): herausnehmen(Edelstein.SMARAGD, 2) --> (DRRS)`
- `(DRRSSS): herausnehmen(Edelstein.DIAMANT, 2) --> (RRSSS)`
- `(DRRSSS): umwandeln(Edelstein.RUBIN, Edelstein.DIAMANT)  --> (DDDSSS)`
- `(DRREEE): leeren() --> ()`

Aufgabe 5.2: Eisenbahnzüge

Schwerpunkte: Verkettung von Objekten
Aufgabe: Schreiben Sie Klassen, die Eisenbahnzüge repräsentieren. Ein Eisenbahnzug besteht aus einer `Lokomotive` und einer beliebigen Anzahl `Wagen`, möglicherweise auch überhaupt keinen.

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 Lokomotive und Wagen, jeweils mit den angegebenen Eigenschaften und sinnvollen Methoden. Die oben genannten Eigenschaften sind unveränderlich.

Das interessante Problem ist das Zusammenstellen eines Zuges aus den Einzelteilen. Der erste Wagen hängt direkt an der Lokomotive. Geben Sie der Klasse Lokomotive deshalb eine Objektvariable ersterWagen vom Typ Wagen, 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 Wagen eine Objektvariable naechsterWagen des gleichen Typs Wagen, wieder mit Gettern und Settern. Diese Objektvariable speichert ein anderes Objekt derselben Klasse oder null beim letzten Wagen.

Definieren Sie schließlich eine Klasse Zug, die den ganzen Zug repräsentiert. Ein Zug-Objekt kennt seine Lokomotive (Objektvariable lok), aber nicht die Wagen. Diese können aber, einer nach dem anderen, auf dem Weg über die Lokomotive erreicht werden.

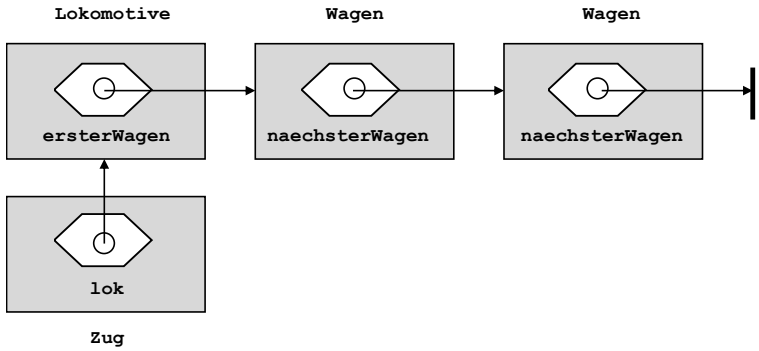

Abbildung 2: Aufbau eines Zuges mit Lokomotive und Wagen.

Die Klasse Zug bietet die folgenden Methoden (ergänzen Sie sinnvolle Parameter und Rückgabewerte):

- Konstruktor: Der Zug-Konstruktor erwartet eine Lokomotive und baut einen ziemlich kurzen Zug, der nur aus der Lokomotive, noch ohne Wagen besteht.
- `wagenHinzufuegen()`: Hängt für diesen Zug einen gegebenen Wagen an das Ende an.
- `erstenWagenEntfernen()`: Hängt den ersten Wagen aus diesem Zug aus und liefert den ausgehängten Wagen als Ergebnis zurück. Die restlichen Wagen rücken nach vorne. Falls es keinen Wagen gibt, ist das Ergebnis null.
- `zugAnhaengen()`: Akzeptiert als Parameter einen anderen Zug und hängt alle Wagen des anderen Zuges in der gleichen Reihenfolge an diesen Zug an. Im anderen Zug bleibt nur die Lokomotive zurück. Nutzen Sie für diese Methode geschickt die vorher definierten Methoden.
- `getWagenAnzahl()`: Liefert die Anzahl der Wagen in diesem Zug (ohne Lokomotive).
- `getKapazitaet()`: Liefert die gesamte Passagierkapazität dieses Zuges, das heißt die Summe der Passagierkapazitäten aller Wagen.
- `getLaenge()`: Liefert die Gesamtlänge dieses Zuges, d. h. die Summe der Länge der Lokomotive und aller Wagen.
- `toString()`: Liefert eine Beschreibung dieses Zuges mit allen Bestandteilen (Typ der Lok, Anzahl Wagen, Gesamtlänge, gesamte Passagierkapazität sowie für jeden Wagen Seriennummer, Wagenlänge und Passagierkapazität); kann zur Konsolenausgabe verwendet werden.

Schreiben Sie eine Anwendungsklasse, die mehrere Züge zusammenstellt und die Methoden verwendet/testet.