Technische Universität Darmstadt
Fachbereich Informatik
Prof. Johannes Fürnkranz

Allgemeine Informatik II im SS 2007

Übungsblatt 8

Bearbeitungszeit: 20.06. bis 26.06.2006

Aufgabe 1: Lebensmittel-Pakete

In den bisherigen Übungsaufgaben haben wir nur Klassen verwendet, die im selben Verzeichnis lagen. Bei einem größeren Projekt mit vielen Klassen wird dann allerdings die Orientierung schwieriger und die Verwaltung der Klassen schwieriger.

Um diesen Schwierigkeiten entgegenzuwirken, wird das Konzept der **`Packages`** eingeführt. Ein **`Package`** ist eine Sammlung mehrerer logisch zusammengehörender Klassen. **`Packages`** sind mit Java-Bezeichnern benannt, wie auch Klassen, Variablen, etc. In der Praxis wählt man üblicherweise englische Namen aus kleinen Buchstaben (z. B. **`java.util.ArrayList`**).

`Packages` können geschachtelt werden. Die Schachtelungs-Hierarchie lässt sich als Baumstruktur darstellen. Die Namen geschachtelter **`Packages`** werden mit Punkten getrennt (siehe oben). Außerdem müssen die Namen von Klassen, Interfaces und **`Subpackages`** eindeutig sein. Und jede Klasse kann nur Mitglied genau eines **`Packages`** sein.

`Packagestrukturen` werden normalerweise auf Verzeichnisstrukturen des Betriebssystems abgebildet (es gibt auch andere Varianten, z. B. Archivdateien). Sie können Klassen aus einem **`Package`** verwenden, in dem Sie die **`import`**-Klausel (z.B. **`import java.util.ArrayList`**;) verwenden.

Struktur-Beispiel:

```
1     > package test.pack;
2     >
3     > import java.util.ArrayList;
4     > import ...;
5     >
6     > public class Test class ... {
7     > ...
8     > }
```

Probieren Sie nun das Anlegen und Arbeiten mit **`Packages`** in einem kleinen, einfachen Beispiel aus:

1. Öffnen Sie BlueJ.
2. Erzeugen Sie in einem neuen Projekt ein **`Package lebensmittel`** (rechte Maustaste).
3. Erzeugen Sie in **`lebensmittel`** die beiden **`Packages suessigkeiten`** und **`obst`**.
4. Schreiben Sie Klassen **`Apfel`**, **`Orange`**, **`Schokolade`**, **`Kekse`** und **`Brot`** mit jeweils einer statischen Methode **`print()`**, die den Namen des jeweiligen Lebensmittels auf dem Bildschirm ausgibt. Ordnen Sie die Klassen in die von Ihnen angelegten neuen Ordner passend gemäß der Lebensmittelgruppe ein.

5. Jede Klasse muss mit einer Package-Deklaration beginnen. (z. B. **`package lebensmittel.obst;`**). Dies erledigt BlueJ für Sie. Schauen Sie sich diese Zeile jedoch genau an, denn Sie müssen im folgenden diese Befehle selbst einfügen.
6. Legen Sie in Ihrem BlueJ-Projekt eine Klasse **`Einkaufskorb`** mit einer **`main`**-Methode an. Sie könnte bei einem Aufruf folgende Ausgabe liefern:

```
1     > Ich nehme in meinem Einkaufskorb mit:
2     >
3     > Brot
4     > Schockolade
5     > Orange
```

7. Schreiben Sie im Kopf der Klasse **`Einkaufskorb`** evtl. notwendige **`import`**-Anweisungen (z.B. **`import lebensmittel.obst.Orange`**;).
8. Erweitern Sie ihre Package-Struktur um weitere Lebensmittelgruppen-**`Subpackages`** und Lebensmittel-**`Klassen`**.

Ein Beispiel für größere Packagestrukturen ist das Java SDK API (siehe **`http://java.sun.com/javase/6/docs/api/`**).

Aufgabe 2: Eisenbahnzüge

Eisenbahnzüge bestehen aus einer Lokomotive und mehreren Zugwagons. Bei den Lokomotiven gibt es als spezielle Ausprägung die Diesel- und die Elektro-Lokomotive. Bei den Zugwagons gibt es die Spezialisierungen Schlafwagen und Speisewagen.

Lokomotiven charakterisieren sich außerdem durch eine Typ-Nummer und ihrer Länge, während Wagons Länge, Passagierkapazität und Wagennummer als Eigenschaft besitzen.

Ihre Aufgabe ist es, Züge mit Hilfe von Java-Klassen, die in Paketen liegen, zu beschreiben. Als Vorgabe können Sie sich ein vorbereitetes BlueJ-Projekt mit der Klasse **`Locomotive`** im Package **`train.locomotive`** und die Klasse **`Car`** im Package **`train.car`** herunterladen.

Gehen Sie wie folgt vor:

1. Erstellen Sie im vorgegebenen Projekt die Klassen **`DieselLocomotive`** und **`ElectricLocomotive`**. Sie sollen von **`Locomotive`** erben (**`extends`**) und im Package **`train.locomotive`** sein. In beiden Klassen müssen Sie den **`Konstruktor`** und die Methode **`toString()`** überschreiben.
2. Erstellen Sie die Klassen **`DiningCar`** und **`SleepingCar`**. Sie sollen von **`Car`** erben und im Package **`train.car`** sein. In beiden Klassen müssen Sie den **`Konstruktor`** und die Methode **`toString()`** überschreiben.
3. Schreiben Sie im Package **`train`** eine Klasse **`Train`**. Diese Klasse soll folgende Methoden anbieten:
    a. Den **`Train-Konstruktor`**, der eine Lokomotive als Parameter erwartet und einen ziemlich kurzen Zug - nur aus einer Lok ohne Wagen bestehend - zusammenbaut.
    b. Methode **`void add(Car car)`** hängt einen Wagon am Ende des Zuges an.
    c. Methode **`int getCapacity()`** berechnet und gibt die gesamte Passagierkapazität zurück.
    d. Methode **`int getLength()`** berechnet und gibt die gesamte Zuglänge zurück.

e. Methode **`Car removeLast()`** hängt den letzten Wagon vom Zug ab und gibt ihn als Ergebnis zurück. Falls es keinen Wagon gibt, wird null zurückgegeben.
f. Methode **`String toString()`** gibt einen Text folgender Art zurück:
`ElectricLocomotive Typ: 3672 <-> Car Nr 1 <-> DiningCar Nr 3 <-> Car Nr 4 <-> SleepingCar Nr 7`

Verwenden Sie zur Verwaltung der Wagons in einer Liste die Klasse **`ArrayList`** oder Vector aus dem Package **`java.util`**. Sie benötigen die Methoden **`void add(Object o)`**, **`Object get(int index)`**, **`Object remove(int index)`** und **`size()`**. Bei den Methoden **`get`** und **`remove`** müssen Sie einen Typecast vornehmen, um die Methoden der **`Car`**-Objekte aufrufen zu können. (Beispiel: **`Car car = (Car) cars.get(3)`**).

4. Erstellen Sie in ihrem Projekt eine Klasse **`TrainTest`**, die die Funktion ihrer Klassen ausprobiert.

Hinweis: Dass Klassen von einer anderen Klasse erben können, kennen Sie schon von den KarelJ Aufgaben aus der Veranstaltung Allgemeine Informatik 1. Sie haben in diesem Zusammenhang bereits auch den Unterschied zwischen dynamischen und statischen Typ und die Polymorphie kennen gelernt. Sie wissen also, dass folgendes Codefragment ohne Probleme funktioniert:

```
1     > public class SleepingCar extends Car {
2     > ...
3     > }
4     > ...
5     >
6     >
7     > // In irgendeiner Methode steht :
8     > Car car = new SleepingCar (20,50,2) ;
9     >
10    >
11    > /* Diese Zuweisung funktioniert, da der dynamische
12    > Typ SleepingCar zum statischen Typ Car auf Grund
13    > der Vererbungsbeziehung konform ist */
14    >
```