Technische Universität München
Fakultät für Informatik
Programmieren in Java

WS 1998/99
Übungsblatt 3
28. Oktober 1998

Übungen zum Programmieren in Java

Vorbemerkung zu den Programmieraufgaben:

Im Verzeichnis `/usr/proj/javakurs/Tag3/skelette` finden Sie vorbereitete Programmgerüste, die Sie in Ihr Verzeichnis kopieren können, um dann dort Ihre Lösungen einzutragen.

Aufgabe 1 Objektorientierte Modellierung

Für einen Tierpark soll ein Informationssystem aufgebaut werden, das Besuchern Informationen über einzelne im Tierpark lebende Tiere und über die verschiedenen Tierarten zur Verfügung stellt. Dabei sollen den Besuchern neben den körperlichen Merkmalen (Größe, Farbe usw.) der Tiere auch bestimmte Verhaltensweisen der Tierarten (z.B. bei der Nahrungsaufnahme) gezeigt werden können.

a) Wie lassen sich einzelne Tiere und Tierarten objektorientiert modellieren? Wie lassen sich körperliche Merkmale und Verhalten der Tiere beschreiben? Was zeichnet dabei ein Individuum gegenüber seiner Art aus?

b) Der Tierpark besitzt u.a. einige Pferde.

   Nennen Sie einige Eigenschaften zur genaueren Charakterisierung von Pferden. Erstellen Sie auf dieser Grundlage eine Objektbeschreibung für eine Stute namens Rosinante.

c) Neben Pferden leben im Tierpark noch viele andere Arten, z.B. Zebras und Löwen.

   Erweitern Sie Ihr Modell um diese Tierarten. Nutzen Sie dabei vorhandene Gemeinsamkeiten und Unterschiede der einzelnen Tierarten zur Vereinfachung bzw. Konkretisierung ihres Modells.

d) Das System soll die Nahrungsaufnahme der einzelnen Tiere differenziert darstellen können.

   Ergänzen Sie Ihr Modell entsprechend. Versuchen Sie dabei, die einfache Nutzung des Systems als Ziel bei der Modellierung zu berücksichtigen.

Aufgabe 2 (P) Abstrakte Mengen

Die von Übungsblatt 2 bekannten Listen sind eine von vielen Möglichkeiten, den abstrakten Begriff einer Menge von Objekten durch eine konkrete Datenstruktur zu realisieren. In den folgenden Aufgaben wollen wir verschiedene Arten solcher Konkretisierungen von Mengen betrachten. Dabei beschränken wir uns der Einfachheit halber auf Mengen von `Integer`–Objekten und geben uns damit zufrieden, daß eine solche Menge vom Typ `IntegerSet` die folgenden Methoden zur Verfügung stellt:

```
// Konstruktor fuer eine leere Menge:
public IntegerSet();

// Konstruktor, der die Kopie einer Menge liefert:
public IntegerSet(IntegerSet s);

// Abfrage, ob Menge leer ist:
public boolean isEmpty();

// Einfuegen eines Elementes:
public void insert(Integer i);

// Abfrage, ob Element enthalten ist:
public boolean contains(Integer i);

// Abfrage nach Groesse der Menge:
public int size();

// Methode fuer Mengeniteration:
public Enumeration elements();

// Abfrage nach Teilmengeneigenschaft:
public boolean isSubset(IntegerSet s);

// Operator fuer Gleichheit:
public boolean isEqual(IntegerSet s);

// Ausgabefunktion:
public String toString();
```

a) Welche dieser Methoden lassen sich nur realisieren, wenn man die zugrundeliegende Datenstruktur kennt, und welche lassen sich abstrakt d.h. datenstrukturunabhängig und nur unter Verwendung anderer Methoden formulieren?

b) Entwickeln Sie aus diesen Überlegungen eine abstrakte Java-Klasse

```
public class AbstractIntegerSet{ ... }
```

c) Formulieren Sie die Mengendarstellung über Listen als Ableitung dieser Klasse:

```
public class ListIntegerSet extends AbstractIntegerSet { ... }
```

Aufgabe 3 (P) Mengendarstellung als Feld

Formulieren Sie eine weitere Klasse zur Mengendarstellung

```
public class ArrayIntegerSet extends AbstractIntegerSet { ... }
```

in der die Elemente der Menge nicht in einer Liste sondern in einem Feld (array) gehalten werden. Welche der in 2b) realisierten datenstrukturunabhängigen Methoden läßt sich hier besonders effizient implementieren und sollte deshalb die datenstrukturunabhängige Methode überschreiben?

Aufgabe 4 (P) Sortierte Mengendarstellungen

a) Einige abstrakte Methoden von 2b) lassen sich wesentlich effizienter realisieren, wenn vorausgesetzt wird, daß die Mengeniteration die Elemente der Menge der Größe nach in aufsteigender Reihenfolge liefert. Geben Sie dazu eine weitere abstrakte Klasse

```
public abstract class AbstractOrderedIntegerSet
                              extends AbstractIntegerSet { ... }
```

an.

b) Geben Sie Konkretisierungen von `AbstractOrderedIntegerSet` an, deren zugrundeliegende Datenstruktur

(i) sortierte Listen von `Integer`–Objekten sind:

```
public class OrderedListIntegerSet
                    extends AbstractOrderedIntegerSet { ... }
```

(ii) boolesche Felder sind, deren Eintrag unter dem Index `i` angeben, ob `Integer(i)` in der dargestellten Menge enthalten ist oder nicht:

```
public class BoolArrayIntegerSet
                    extends AbstractOrderedIntegerSet { ... }
```