zu Kapitel 9

Aufgabe 1
(Armee.java)

Drei Entwickler sollen zusammen für die Bundeswehr ein Projekt bearbeiten: Soldaten sollen nach ihrer Größe sortiert werden.

Sie einigen sich auf folgende Arbeitsteilung: Der eine soll den Sortieralgorithmus BubbleSort als statische Funktion sortiere() einer Klasse BubbleSort implementieren. Die beiden vereinbaren folgendes Interface

```
package bubblesort;

public interface Sortierbar
{
      public int istGroesser(Sortierbar element);
}

public class BubbleSort
{
      public static void sortiere (Sortierbar[] liste)
      {
         // muss noch implementiert werden.
      }
}
```

Ein anderer soll in einer Klasse Armee ein Array von (der Einfachheit halber 10) Soldaten mittels des von seinem Kollegen implementierten BubbleSort-Algorithmus sortieren. Ein dritter Entwickler hatte schon vor einiger Zeit eine Klasse Mensch implementiert, welche (wieder der Einfachheit halber) hier nur einen Namen besitzt. Diese Klasse Mensch bietet einen Konstruktor, welcher den Namen initialisiert, und eine public-Funktion getName(), mit welcher der ansonsten geschützte Name gelesen werden kann. Da auch ein Soldat einen Namen hat, entschließt sich Entwickler Nr. 3 einen Soldaten von der Klasse Mensch abzuleiten und nur noch eine Groesse (in cm), nach der sortiert werden kann, hinzuzufügen.

Erledigen Sie die Aufgabe von allen drei Entwicklern, die übrigens in getrennten Packages arbeiten sollen.

Hinweis: Sie können den Bubblesort Algorithmus von früher verwenden und müssen ihn lediglich in ein geeignetes Package stecken.

Aufgabe 2
(Modul.java)

Bilden Sie zu einer gegebenen Menge ganzer Zahlen die Restklassen Modulo m. Schreiben Sie dazu eine Klasse *Modul*, welche aus *m* Restklassen besteht. Die Restklassen sollen als Objekte der inneren Klasse *Rest* realisiert werden. Die Klasse *Rest* beinhaltet diejenigen ganzzahligen Elemente der Grundmenge, die bei Division durch *m* denselben Rest lassen.

a) Dazu soll die Klasse *Modul* einen geeigneten Konstruktor bekommen, welcher aus der Angabe des gewünschten Moduls *m* und einem gegebenen Feld ganzer Zahlen, die Restklassen Modulo m bildet. Die Klasse *Rest* benötigt ihrerseits einen passenden Konstruktor, sowie z.B. eine Methode `void put(int num);` um der Restklasse zahlen hinzufügen zu können. Die Methode `String toString()` sollte außerdem Verwendung finden, um die Elemente der Restklasse, als durch Komma getrennte Liste,

zu formatieren. Nachdem alle Zahlen ihren Restklassen zugeordnet wurden, soll der ganze Modul mithilfe einer simplen print()-Methode ausgegeben werden. Die Ausgabe für die ersten 100 natürlichen Zahlen im Modul 10 könnte etwa so aussehen:

```
[0] = { 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 }
[1] = { 1, 11, 21, 31, 41, 51, 61, 71, 81, 91 }
[2] = { 2, 12, 22, 32, 42, 52, 62, 72, 82, 92 }
[3] = { 3, 13, 23, 33, 43, 53, 63, 73, 83, 93 }
[4] = { 4, 14, 24, 34, 44, 54, 64, 74, 84, 94 }
[5] = { 5, 15, 25, 35, 45, 55, 65, 75, 85, 95 }
[6] = { 6, 16, 26, 36, 46, 56, 66, 76, 86, 96 }
[7] = { 7, 17, 27, 37, 47, 57, 67, 77, 87, 97 }
[8] = { 8, 18, 28, 38, 48, 58, 68, 78, 88, 98 }
[9] = { 9, 19, 29, 39, 49, 59, 69, 79, 89, 99 }
```

b) Erzeugen sie die Grundmenge nun mittels Zufallszahlen und stellen Sie ihre Restklassen grafisch dar, indem Sie für jede Restklasse in einer Zeile Sternchen ausgeben, die angeben, wie viele Elemente diese hat.

c) Schreiben Sie eine statische innere Klasse *MultiTafel*, welche die Multiplikationstafel für einen Modul realisiert und ausgibt. Für den Modul 10 könnte die Ausgabe wie folgt aussehen:

```
        [ 0 ] [ 1 ] [ 2 ] [ 3 ] [ 4 ] [ 5 ] [ 6 ] [ 7 ] [ 8 ] [ 9 ]
[ 0 ]     0     0     0     0     0     0     0     0     0     0
[ 1 ]     0     1     2     3     4     5     6     7     8     9
[ 2 ]     0     2     4     6     8     0     2     4     6     8
[ 3 ]     0     3     6     9     2     5     8     1     4     7
[ 4 ]     0     4     8     2     6     0     4     8     2     6
[ 5 ]     0     5     0     5     0     5     0     5     0     5
[ 6 ]     0     6     2     8     4     0     6     2     8     4
[ 7 ]     0     7     4     1     8     5     2     9     6     3
[ 8 ]     0     8     6     4     2     0     8     6     4     2
[ 9 ]     0     9     8     7     6     5     4     3     2     1
```

d) Erweitern Sie die Klasse *MultiTafel*, um die Methode `int getInverse(int num);` welche die Inverse zu einer ganzen Zahl *num* liefert oder -1, falls die Zahl kein Inverses in diesem Modul besitzt. Geben Sie anschließend zu jeder Restklasse des Moduls aus, welche Restklasse die Inverse dazu ist. Für den Modul 10 könnte die Ausgabe etwa so aussehen:

```
[ 0 ]    -
[ 1 ]  [ 1 ]
[ 2 ]    -
[ 3 ]  [ 7 ]
[ 4 ]    -
[ 5 ]    -
[ 6 ]    -
[ 7 ]  [ 3 ]
[ 8 ]    -
[ 9 ]  [ 9 ]
```