Universität Mannheim
Lehrstuhl für Praktische Informatik IV
Prof. Dr. W. Effelsberg
Christoph Kuhmünch, Gerald Kühne

Praktische Informatik I
WS 1999/2000

Übungsblatt 7 **Ausgabe: Mi, 08.12.99** **Abgabe: Di, 14.12.99, 18 Uhr**

Aufgabe 1: Modularität in Java: Klassen und Objekte [14 Punkte]

Das nachfolgend abgebildete Programm stellt den Anfang der Implementierung einer Adressverwaltung dar (das Programm ist bewußt mit wenigen Kommentaren versehen worden):

```
 1 class AdressElement {
 2   private String name;
 3   private String strasse;
 4   private String ort;
 5   private final int anzahlFelder;
 6   private final String[] felder;  // Namen der Felder
 7
 8   public AdressElement () {
 9     this.name    = "";
10     this.strasse = "";
11     this.ort     = "";
12     this.anzahlFelder = 3;
13     this.felder= new String[this.anzahlFelder];
14     this.felder[0] = "Name";
15     this.felder[1] = "Strasse";
16     this.felder[2] = "Ort";
17   }
18
19   public void dummy1 () {
20     this.felder[0] = "Vorname Name";
21     // this.anzahlFelder = 3;
22     // this.felder = new String [17];
23   }
24
25   public void dummy2 () {
26     char ca1[]= {'h','a','l','l','o'};
27     System.out.println(ca1[0]);
28     {
29       char ca2[]= ca1;
30       ca2[0]= 'X';
31       System.out.println(ca1[0]);
32     }
33   }
34
35   public void setzeAdresse (String name, String strasse, String ort) {
36     this.name    = name;
37     this.strasse = strasse;
38     this.ort     = ort;
39   }
40
41   public void liesAnzahlFelder (int anzahlFelder) {
42     anzahlFelder = this.anzahlFelder;
43   }
44
45   public void liesAdresse (String name, String strasse, String ort) {
46     name    = this.name;
```

```
47     strasse = this.strasse;
48     ort     = this.ort;
49   }
50
51   public static void zeigeFelder () {
52     System.out.println ("Name;Strasse;Ort\n");
53     //   for (int i = 0; i < anzahlFelder; i++) {
54     //     System.out.println (felder[i]);
55     //   }
56   }
57
58   public String toString() {
59     String s = name + ";" + strasse + ";" + ort + "\n";
60     return s;
61   }
62 }
63
64 public class AdressenListe {
65   public static void main (String[] args) {
66     AdressElement a = new AdressElement ();
67     a.zeigeFelder ();
68   }
69 }
```

(a) [9 Punkte] Beantworten Sie die folgenden Fragen!

1. Welche Bedeutung haben die Schlüsselwörter *private* und *public* in Java?
2. Welche Bedeutung hat das Schlüsselwort *this*? Warum wird this in der Methode setzeAdresse benötigt?
3. Was ist ein Konstruktor?
4. Welche Bedeutung hat das Schlüsselwort *final*? Warum wird die Zuweisung in der ersten Zeile der Methode dummy1 fehlerfrei kompiliert, während die nachfolgenden beiden auskommentierten Zeilen zu Fehlermeldungen führen würden, wenn sie nicht auskommentiert wären?
5. An welcher Stelle könnte in der Methode dummy2 der Speicher, auf den die Referenz ca2 zeigt, gelöscht werden? Erläutern Sie in diesem Zusammenhang auch den Begriff Garbage Collection!
6. Welche Bedeutung hat das Schlüsselwort *static*? Warum funktioniert die auskommentierte For-Schleife in der Methode zeigeFelder nicht? Wie würde man das Programm sinnvollerweise verbessern, sodaß die For-Schleife ebenfalls funktionieren würde?
7. Wo liegt der semantische Fehler in der Methode liesAnzahlFelder? Wie könnte man den semantischen Fehler korrigieren, ohne daß die Methode per return-Anweisung einen Rückgabewert liefert? Wieso tritt dieses Problem bei `liesAdresse` ebenfalls auf, obwohl hier Referenztypen verwendet werden?
8. Was versteht man unter dem Begriff “Implizite Objekterzeugung”? Nennen Sie eine Stelle im Programm, an der Objekte implizit erzeugt werden!
9. Zeile 67 wird vom Compiler zwar fehlerfrei übersetzt, die gewählte Syntax des Funktionsaufrufs ist jedoch irreführend. Erläutern Sie das Problem und geben Sie einen Verbesserungsvorschlag an.

(b) [6 Punkte] Implementieren Sie die Funktionalität für das Einfügen, Löschen und Suchen von Adressen auf der Grundlage des obigen Programmfragments! Verwenden Sie dazu ein Array der Größe 100 innerhalb der Klasse AdressenListe (Gehen Sie davon aus, daß

niemals mehr als 100 Adressen verwaltet werden müssen). Die einzelnen Elemente des Arrays sind dabei Objekte vom Typ AdressElement. Verwenden Sie für jede Klasse eine eigene Datei (`AdressElement.java` und `AdressenListe.java`) und geben Sie diese inklusive der `.class`-Dateien per abox ab.

Erweitern Sie die statische main-Methode der Klasse AdressenListe in der Art, daß sie eine rudimentäre Kommandosprache zum Zugriff auf die Adressen ermöglicht. In dieser Sprache gibt es die folgenden Befehle:

`add Name Strasse Ort`	Einfügen eines neuen Eintrages, Duplikate sind erlaubt. Keine Ausgabe.
`delete Name`	Löschen aller Einträge des entsprechenden Namens. Keine Ausgabe.
`search Name`	Ausgabe aller Einträge des entsprechenden Namens.
`print`	Ausgabe aller Einträge der Liste.

Search und print müssen als Ergebnis jeweils eine Adresse pro Zeile ausgeben. Die Felder sollen durch Semikolon (;) getrennt werden. Ist das Suchergebnis bzw. die Adreßliste leer, soll das Wort “LEER” ausgegeben werden.

Anmerkung: Das Programm muß als Kommandozeilenparameter den Namen einer Datei erwarten, die eine Folge von Befehlen der Kommandosprache enthält. Ein Beispiel für eine solche Kommandodatei finden Sie im Web[1].

Hier ein Beispiel für einen Kommandozeilenaufruf und der Ergebnisausgabe des von Ihnen zu entwickelnden Programmes:

```
kommandos.txt:
add Meier Meierstrasse Meiershausen
add Mueller Muellerstrasse Muellersheim
print
search Meier
delete Meier
search Meier

 > java AdressenListe kommandos.txt
Meier;Meierstrasse;Meiershausen
Mueller;Muellerstrasse;Muellersheim
Meier;Meierstrasse;Meiershausen
LEER
```

[1] http://www.informatik.uni-mannheim.de/informatik/pi4/stud/veranstaltungen/ws199900/pi1/ueb/blatt7/kommandos.txt

Aufgabe 2: Fibonacci-Zahlen [6 Punkte]

Die Fibonacci-Funktion auf natürlichen Zahlen ist wie folgt definiert:

$$fib(n) = \begin{cases} 1 & , \quad falls\ n = 1 \\ 1 & , \quad falls\ n = 2 \\ fib(n-1) + fib(n-2) & , \quad sonst \end{cases} \tag{1}$$

(a) [3 Punkte] Implementieren Sie einen *rekursiven* Algorithmus zur Berechnung des Fibonacci-Funktionswertes einer Zahl *n* in Java.

Benutzen Sie als Klassennamen `FiboRekursiv`. Das Programm muß als Kommandozeilenparameter eine Integer-Zahl erwarten und den zugehörigen Fibonacci-Funktionswert zurückliefern.

Hier ein Beispiel für einen Kommandozeilenaufruf und der Ergebnisausgabe des von Ihnen zu entwickelnden Programms:

```
> java FiboRekursiv 1
1
```

(b) [3 Punkte] Implementieren Sie einen *iterativen* Algorithmus zur Berechnung des Fibonacci-Funktionswertes in Java.

Benutzen Sie als Klassennamen `FiboIterativ`. Das Programm muß als Kommandozeilenparameter eine Integer-Zahl erwarten und den zugehörigen Fibonacci-Funktionswert zurückliefern.

Hier ein Beispiel für einen Kommandozeilenaufruf und der Ergebnisausgabe des von Ihnen zu entwickelnden Programms:

```
> java FiboIterativ 1
1
```