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?

   `public` Elemente oder Methoden sind für jeglichen Java-Code zugreifbar, für den ihre Klasse zugreifbar ist. `private` *Elemente oder Konstruktoren sind nur innerhalb der Klassendeklaration zugreifbar (Siehe auch Vorlesungsscript S. 3-90f und Lehrbuch S. 154f). Die Verwendung von* `private` *dient der Kapselung von Instanzvariablen. D.h. die Datenstruktur innerhalb einer Klasse kann jederzeit geändert werden, ohne daß sie nach außen hin sichtbar wird.*

   Bsp.: Bei der Überarbeitung der Klasse `AdressElement` *hat sich der Entwickler entschlossen, die Angaben PLZ und Ort in einer Zeichenkette zu speichern (warum auch immer...). Das Ergebnis sieht man in der Klasse* `AdressElement`*, neue Version. Das Verhalten der Klasse nach außen hat sich durch diese Veränderung nicht geändert, d.h. andere Stellen im Programm müssen nicht modifiziert werden.*

```
// alte Version
public class AdressElement {
  private String name;
  private String strasse;
  private String PLZ;
  private String ort;

  public  void setzeAdresse (String name, String strasse,
                             String PLZ, String ort) {
     this.name    = name;
     this.strasse = strasse;
     this.PLZ     = PLZ;
     this.ort     = ort;
  }
}

// neue Version
public class AdressElement {
  private String name;
  private String strasse;
```

```
  private String PLZOrt;

  public  void setzeAdresse (String name, String strasse,
                             String PLZ, String ort) {
     this.name    = name;
     this.strasse = strasse;
     this.PLZOrt  = PLZ + " " + Ort;
  }
}
```

2. Welche Bedeutung hat das Schlüsselwort *this*? Warum wird this in der Methode setzeAdresse benötigt?

   Der Name einer Instanzvariablen kann durch lokale Variablen oder einen Methoden- oder Konstruktorparameter desselben Namens verdeckt werden (Dies ist in der Methode `setzeAdresse` *der Fall!). Unter Verwendung des Schlüsselwortes* `this` *kann man dennoch auf die verdeckte Variablen zugreifen (Siehe auch Vorlesungsscript und Lehrbuch S. 95).*

3. Was ist ein Konstruktor?

   Ein Konstruktor ist eine spezielle Methode, mit der man die Instanzvariablen initialisiert. Bei der Erzeugung eines Objekts wird automatisch der entsprechende Konstruktor aufgerufen. So initialisiert der Konstruktor `AdressElement`*, der in Zeile 66 aufgerufen wird, die Instanzvariablen* `name`*,* `strasse`*,* `ort`*,* `anzahlFelder` *und* `felder`*.*

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?

   Instanz- und Klassenvariablen können mit dem final-Modifizierer als symbolische Konstanten deklariert werden. In diesem Fall muß die Deklaration einen Initialisierer enthalten. Alternativ kann für final Instanzvariablen in jedem Konstruktor bzw. für final Klassenvariablen in einem static Initialisierer genau eine Zuweisung vorgenommen werden. Im letzten Fall nennt man die Variablen unspezifiziert final.

   Die erste Zeile der Methode `dummy1` *wird deshalb fehlerfrei kompiliert, weil lediglich das Array* `AdressElement.felder` *final ist, die einzelnen Einträge innerhalb des Feldes sind dagegen ganz normale Variablen. Die auskommentierten Zeilen werden dagegen nicht übersetzt, da hier eine Zuweisung an als final deklarierte Variablen erfolgt.*

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!

   Die Freigabe des für ein Objekt reservierten Speicherplatzes erfolgt automatisch durch den Java-Garbage-Collector. Dieser überprüft existierende Objekte periodisch daraufhin, ob sie noch referenziert werden. Wenn keinerlei Referenzen auf ein Objekt mehr existieren, kann dieses zerstört werden und sein Speicherplatz anderweitig verwandt werden. Im Gegensatz zu Sprachen mit expliziter Speicherfreigabe wie C und C++ gibt es in Java somit keine Probleme mit Speicherlecks.

   In der Methode `dummy2` *zeigen sowohl* `ca1` *als auch* `ca2` *auf dasselbe Objekt und somit auch auf denselben Speicherbereich. Folglich darf der Speicher erst freigegeben werden, wenn sowohl* `ca2` *als auch* `ca1` *zerstört sind.* `ca2` *wird am Blockende in Zeile 32 zerstört,* `ca1` *aber erst am Ende der Methode, also in Zeile 33. Der*

Garbage-Collector kannalso frühstens dann den Speicherbereich wieder freigeben, wenn die Methode `dummy2` *beendet ist.*

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, so daß die For-Schleife ebenfalls funktionieren würde?

   Variablen können als Klassenvariablen, die Java einmal pro Klasse anlegt oder als Instanzvariablen, die für jedes Objekt neu erzeugt werden, deklariert werden. Ebenso ist es möglich, eine Methode als Klassenmethode, die ohne Objekt aufgerufen wird, oder als Instanzmethode, die immer für ein bestimmtes Objekt aufgerufen wird, zu deklarieren (Schader S. 92). Klassenmethoden und Klassenvariablen werden mit dem Schlüsselwort `static` *deklariert.*

   Im Folgenden sind die notwendigen Änderungen im Quelltext mit einem Ausrufezeichen gekennzeichnet.

```
     1 class AdressElement {
     2   private String name;
     3   private String strasse;
     4   private String ort;
 !   5   private final static int anzahlFelder = 3;
 !   6   private final static String[] felder = { "Name", "Strasse", "Ort" };
     7
     8   public AdressElement () {
     9     this.name    = "";
    10     this.strasse = "";
    11     this.ort     = "";
 !  12
 !  13
 !  14
 !  15
 !  16
    17   }
    18
    19   public void dummy1 () {
 !  20     felder[0] = "Vorname Name";
    21    }
    ..
    41   public void liesAnzahlFelder (int anzahlFelder) {
 !  42     anzahlFelder = AdressElement.anzahlFelder;
    43   }
    ..
    51   public static void zeigeFelder () {
 !  52     // System.out.println ("Name\nStrasse\nOrt\n");
 !  53        for (int i = 0; i < anzahlFelder; i++) {
 !  54          System.out.println (felder[i]);
    55        }
    56   }
```

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?

   In der Methode liesAnzahlFelder wird ein elementarer Typ (int) als Parameter übergeben, d.h. innerhalb der Methode wird auf einer lokalen Kopie des Parameters gearbeitet (call by value), die beim Beenden der Methode gelöscht wird. Der eigentliche Parameter anzahlFelder wird im Methodenrumpf nicht verändert.

   Nachfolgend eine korrigierte Version von liesAnzahlFelder:

```
public void liesAnzahlFelder (int[] anzahlFelder) {
  anzahlFelder[0] = AdressElement.anzahlFelder;
}
```

Bei der Methode `liesAdresse` tritt dieses Problem in der gleichen Weise auf. Strings sind Referenztypen, d.h. in `liesAdresse` werden zwar Referenzen auf Objekte vom Typ String als Parameter übergeben, aber innerhalb der Methode wird auf lokalen Kopien dieser Referenzen (nicht der Objekte!) gearbeitet. Die Zuweisung `name = this.name` führt lediglich dazu, daß der lokalen Variablen `name` eine andere Referenz zugewiesen wird. Die lokalen Variablen werden bei Beendigung der Methode gelöscht, d.h. etwaige Berechnungen oder Zuweisungen bzgl. dieser Variablen sind nach außen nicht sichtbar.

8. Was versteht man unter dem Begriff "Implizite Objekterzeugung"? Nennen Sie eine Stelle im Programm, an der Objekte implizit erzeugt werden!

   Objekte können implizit erzeugt werden, d.h. das Schlüsselwort `new` wird nicht benutzt. Beispiele hierfür sind String-Objekte zur Aufnahme der Werte von Zeichenketten oder in Zusammenhang mit der Auswertung des Operators `+`. Siehe dazu Zeile 59, in der verschiedene Zeichenketten zusammengefügt 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.

   Die Methode zeigeFelder ist als Klassenmethode deklariert. Daher ist es sinnvoll, dieses auch in den entsprechenden Methodenaufrufen zu kennzeichnen: `AdressElement.zeigeFelder()`.

(b) [5 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].

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

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
```

Lösungsvorschlag:

```
//
/*****************************************************************
 * Autor:        Gerald Kuehne, Christoph Kuhmuench
 * Datum:        08.12.1999
 * Name:         AdressElement
 * Beschreibung: Klasse zur Ablage einer Adresse mit den Elementen
 *               Name, Strasse, Ort
 *****************************************************************/

class AdressElement {
  // Datenelemente fuer die Adresse
  private String name;
  private String strasse;
  private String ort;

  // Initialisierung der Datenelemente
  public AdressElement () {
    this.name    = "";
    this.strasse = "";
    this.ort     = "";
  }

  // Adresse ablegen
  public void setzeAdresse (String name, String strasse, String ort) {
    this.name    = name;
    this.strasse = strasse;
    this.ort     = ort;
  }

  // Zugriff auf die Datenelemente Name, Strasse, Ort
  public String liesName () {
    return name;
  }

  public String liesStrasse () {
    return strasse;
  }

  public String liesOrt () {
    return ort;
```

```
  }

  // Konversion von AdressElement -> String
  // Beispiel:
  // AdressElement e;
  // System.out.println (e); // impliziter Aufruf der toString-Methode
  public String toString() {
    String s = name + ";" + strasse + ";" + ort;
    return s;
  }
}
//
//
/*****************************************************************
 * Autor:        Gerald Kuehne, Christoph Kuhmuench
 * Datum:        08.12.1999
 * Name:         AdressenListe
 * Beschreibung: Simple Adressverwaltung
 *****************************************************************/

import java.util.StringTokenizer;
import java.io.*;

public class AdressenListe {
  private final static int MAX_ANZAHL = 100; // maximale Anzahl der
                                             // Adressen

  // Datenstruktur fuer die Adressenspeicherung
  private AdressElement[] adressen = new AdressElement[MAX_ANZAHL];

  /*************************************************************
   * Name:         einfuegen
   * Beschreibung: Einfuegen eines Datensatzes
   * Parameter:    name, strasse, ort - Elemente des Datensatzes
   *************************************************************/
  public void einfuegen (String name, String strasse, String ort) {
    boolean adresseEingefuegt = false; // konnte Adresse eingefuegt werden?

    for (int i = 0; i < MAX_ANZAHL; i++) {
      if (adressen[i] == null) {            // freies AdressElement?
        adressen[i] = new AdressElement (); // AdressElement anlegen
        adressen[i].setzeAdresse (name, strasse, ort); // und auffuellen
        adresseEingefuegt = true;           // Adresse wurde eingefuegt
        break;
      }
    }
    if (!adresseEingefuegt) {
      System.out.println ("Fehler: Adressenliste ist voll");
    }
  }

  /*************************************************************
   * Name:         suchen
   * Beschreibung: Suchen und ausgeben aller Datensaetze, die einen
   *               bestimmten Namen enthalten
   * Parameter:    name - zu suchender Name
   *************************************************************/
  public void suchen (String name) {
    boolean adresseGefunden = false; // wurde eine passende Adresse gefunden?
    for (int i = 0; i < MAX_ANZAHL; i++) {
      if (adressen[i] != null) {     // ist AdressElement belegt?
```

```
      String tmpName  = adressen[i].liesName ();
      if (tmpName.equals (name)) { // Namen vergleichen
        adresseGefunden = true;    // passende Adresse wurde gefunden
        System.out.println (adressen[i]);
      }
    }
  }
  if (!adresseGefunden)
    System.out.println ("LEER");
}

/*****************************************************************
 * Name:         loeschen
 * Beschreibung: Loeschen aller Datensaetze, die einen bestimmten
 *               Namen enthalten
 * Parameter:    name
 *****************************************************************/
public void loeschen (String name) {
  for (int i = 0; i < MAX_ANZAHL; i++) {
    if (adressen[i] != null) {
      String tmpName = adressen[i].liesName ();
      if (tmpName.equals (name))
        adressen[i] = null;  // Objektreferenz loeschen
    }
  }
}

/*****************************************************************
 * Name:         drucken
 * Beschreibung: Ausgabe aller Datensaetze
 *****************************************************************/
public void drucken () {
  boolean adresseVorhanden = false; // enthaelt die Liste eine Adresse?
  for (int i = 0; i < MAX_ANZAHL; i++) {
    if (adressen[i] != null) {
      adresseVorhanden = true; // Adresse gefunden
      System.out.println (adressen[i]);
    }
  }
  if (!adresseVorhanden)
    System.out.println ("LEER");
}



/*****************************************************************
 * Name:         main
 * Beschreibung: In main wird eine Kommandodatei eingelesen und
 *               die entsprechenden Operationen werden auf der
 *               Adressenliste ausgefuehrt
 *****************************************************************/
public static void main (String[] args) {

  AdressenListe liste = new AdressenListe ();

  FileReader fr = null;
  BufferedReader in = null;

  try {
    fr= new FileReader(args[0]);
    in= new BufferedReader(fr);
  } catch(FileNotFoundException fnfe) {
```

```
      System.out.println("File not found");
    }

    String kommandoZeile = ""; // enthaelt eine Befehlszeile

    try {
      // Kommadodatei abarbeiten
      // ACHTUNG: Es wird davon ausgegangen, dass die Kommandodatei
      // im korrekten Format vorliegt.
      while ((kommandoZeile = in.readLine()) != null) {
        StringTokenizer st = new StringTokenizer (kommandoZeile);
        String s = st.nextToken(); // Befehl auslesen
        if (s.equals("add")) {
          // Parameter Name, Strasse, Ort
          liste.einfuegen (st.nextToken(), st.nextToken(), st.nextToken());
        } else if (s.equals("delete")) {
          // Parameter Name
          liste.loeschen (st.nextToken());
        } else if (s.equals("search")) {
          // Parameter Name
          liste.suchen (st.nextToken());
        } else if (s.equals("print")) {
          liste.drucken ();
        }
      }
    } catch(java.io.IOException ioe) {
        System.out.println("IO Exception");
    }
  }

}
//
```

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.

Lösungsvorschlag: //

```
/*****************************************************************
 * Autor:        Gerald Kuehne, Christoph Kuhmuench
 * Datum:        08.12.1999
 * Name:         FiboRekursiv
 * Beschreibung: Berechnung der Fibonacci-Zahl fuer n > 0
 *****************************************************************/

public class FiboRekursiv {

  /*****************************************************************
   * Name:          fibo
   * Beschreibung:  Berechnet rekursiv die Fibonacci-Zahl
   * Parameter:     n - Position in der Fibonacci-Reihe (n > 0)
   * Rueckgabewert: Fibonacci-Zahl
   *****************************************************************/
  public static long fibo (long n) {
    if (n < 1) {
      System.out.println ("Fehler: n muss groesser 0 sein!");
      System.exit (1);
    }
    if (n <= 2) { // Abbruchkriterium
      return 1;
    } else {      // rekursive Formel fuer die Fibonacci-Zahlen
      return fibo (n-1) + fibo (n-2);
    }
  }

  public static void main(String[] args) {
    long n = Integer.parseInt(args[0]);
    System.out.println (fibo(n));
  }
}
//
```

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

Lösungsvorschlag: //

```
/*****************************************************************
 * Autor:        Gerald Kuehne, Christoph Kuhmuench
 * Datum:        08.12.1999
 * Name:         FiboIterativ
 * Beschreibung: Berechnung der Fibonacci-Zahl fuer n > 0
 *****************************************************************/

public class FiboIterativ {

  /*****************************************************************
```

```
   * Name:          fibo
   * Beschreibung:  Berechnet iterativ die Fibonacci-Zahl
   * Parameter:     n - Position in der Fibonacci-Reihe (n > 0)
   * Rueckgabewert: Fibonacci-Zahl
   ************************************************************/
  public static long fibo (long n) {
    if (n < 1) {
      System.out.println ("Fehler: n muss groesser 0 sein!");
      System.exit (1);
    }

    long a = 1; // 1. Element der Fibonacci-Reihe
    long b = 1; // 2. Element der Fibonacci-Reihe

    for (long i = 2; i < n; i++) {
      b = a + b; // berechne naechstes Element der Fibonacci-Reihe
      a = b - a; // berechne vorhergehendes Element der Fibonacci-Reihe
    }

    return b;
  }

  public static void main(String[] args) {
    long n = Integer.parseInt(args[0]);
    System.out.println (fibo(n));
  }
}
//
```