Fachbereich Informatik
FG Eingebettete Systeme und
ihre Anwendungen
Prof. A. Koch

TECHNISCHE
UNIVERSITÄT
DARMSTADT

2. Teilprüfung
Allgemeine Informatik II
Sommersemester 2008
04. September 2008

Name (Nach-, Vorname)	
Matrikel-Nr.	**Lösungsvorschlag**
Unterschrift	
Prüfung Bitte ankreuzen	☐ Bachelor ☐ Master ☐ Diplom Fachrichtung: Wiederholer: ☐ Ja, ___. Wiederholung ☐ Nein
Anzahl abgegebene Zusatzblätter:	

Aufgabe	**max. Punkte**	**Erreicht**
1	12	
2	18	
3	28	
4	7	
5	15	
Summe	**80**	
Note		

Aufgabe 1 Java-Konstrukte (12 Punkte)

Gegeben seien folgende Klassendefinitionen

```
1  public class A {
2      public          void m1() {};
3      protected       void m2() {};
4      private         void m3() {};
5      public static   void m4() {};
6
7      public void x() {
8          ... // Stelle A
9      }
10 }
11
12 public class B extends A {
13     public void x(A p) {
14         ... // Stelle B
15     }
16 }
17
18 public class C extends A {
19     public static void x(C p) {
20         ... // Stelle C
21     }
22
23     private void y(A a) {
24         ... // Stelle D
25     }
26 }
27
28 class D {
29     public void x(A p) {
30         ... // Stelle E
31     }
32 }
```

Fünf Stellen des Java-Programmes sind durch A bis E bezeichnet. Beantworten Sie folgende Fragen, wo durch eine Linie kenntlich gemacht auch mit einer kurzen Begründung.

a) Wir betrachten die neue Anweisung `System.out.println((B) p)`. Ist dies zulässig an der Stelle

Stelle	Ja	Nein	Erklärung
B	□	□	_____
C	□	□	_____

B ok, C falsch (Klasse B->C ist kein Downcast)

b) Wir betrachten neue Anweisungen. Sind diese zulässig an der Stelle

Anweisung	Stelle	Ja	Nein	Erklärung
`super.x()`	B	□	□	_____
`super.x()`	C	□	□	_____
`super.y()`	D	□	□	_____
`super.x()`	E	□	□	_____

B ok, C falsch (x ist in Superklasse eine Instanzmethode, in Klasse C eine Klassenmethode, D falsch (Klasse C hat keine Superklasse, die y() implementiert, E falsch (Klasse D hat keine Superklasse, die x() implementiert)

c) Welche Methodenaufrufe können an der Stelle A eingesetzt werden, ohne einen Fehler zu verursachen?

Methodenaufruf	Ja	Nein
`m1()`	☐	☐
`m2()`	☐	☐
`m3()`	☐	☐
`m4()`	☐	☐

Alle OK.

d) Welche Methodenaufrufe können an der Stelle C eingesetzt werden, ohne einen Fehler zu verursachen?

Methodenaufruf	Ja	Nein
`m1()`	☐	☐
`m2()`	☐	☐
`m3()`	☐	☐
`m4()`	☐	☐

Nur m4 OK, alle anderen nicht zugelassen (u.a. Instanzmethoden aus statischer Methode aufgerufen, m3 ist auch nicht sichtbar)

e) Welche Methodenaufrufe können an der Stelle D eingesetzt werden, ohne einen Fehler zu verursachen?

Methodenaufruf	Ja	Nein
`m1()`	☐	☐
`m2()`	☐	☐
`m3()`	☐	☐
`m4()`	☐	☐

OK: m1, m2, m4; Falsch: m3 (private)

Aufgabe 2 Wörter zählen (18 Punkte)

Geben Sie den Quelltext einer Java-Methode

```
public static int wordCount(String s) { ...  }
```

an, die als Ergebnis die Anzahl der Wörter im Eingabe-String `s` zurückliefert. In `s` sind als Zeichen zugelassen eine beliebige Zahl der Buchstaben 'A' ... 'Z' (aus denen die Worte bestehen) und eine beliebige Zahl an Leerzeichen.

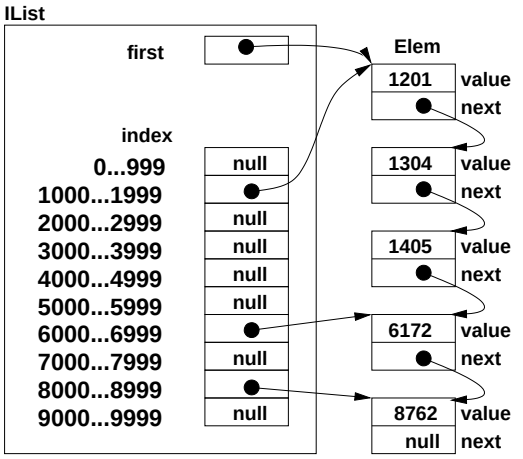
Hinweise: Die `String`-Methode `charAt(int n)` liefert das an *n*-ter Stelle stehende Zeichen eines Strings, die Methode `length()` die Länge des `String`s in Zeichen. Weitere Methoden aus der Java-Bibliothek dürfen Sie nicht verwenden!

Eine mögliche Lösung sieht so aus:

```
1     static public int wordCount(String text)
2      {
3          int n=0, count=0;
4          boolean inWord = false;
5
6         while (n < text.length()) {
7
8             if (!inWord && text.charAt(n) != ' ') {
9               ++count;
10              inWord = true;
11             } else
12                 inWord = text.charAt(n) != ' ';
13
14            ++n;
15         }
16
17         return count;
18     }
19 }
```

Aufgabe 3 Indizierte Liste (28 Punkte)

Um Suchzugriffe auf eine Liste zu beschleunigen, kann man einen Index einsetzen. Dabei wird neben der Liste ein Array verwendet, welches für einen gegebenen Wertebereich einen Verweis in den entsprechenden Abschnitt der Liste hinein abspeichert. Bei einer Suche braucht die Liste dann nicht von Anfang an durchlaufen zu werden, sondern der Durchlauf kann beim ersten Element des entsprechenden Wertebereichs beginnen.



Eine Suchoperation im abgebildeten Beispiel, z.B. nach dem Wert 6172, bräuchte durch Benutzung des Index nicht die drei Elemente 1201, 1304 und 1405 zu durchlaufen, sondern bestimmt aus dem Wertebereich des gesuchten Wertes (6172 liegt zwischen 6000 und 6999, damit also im 6. Bereich des Index) den entsprechenden Einsprungpunkt in die Liste (der hier unmittelbar zum gesuchten Wert 6172 führt).

Realisieren Sie eine solche Liste als Klasse `IList` mit folgenden Vorgaben:

- `IList` speichert neben dem Anfang der Liste `first` das Index-Array `index` ab.
- Die Liste soll als Datenwerte natürliche Zahlen aus dem Intervall $0 \ldots 9999$ enthalten.
- Der Index unterteilt dieses Intervall in zehn Bereiche von jeweils 1000 unterschiedlichen Datenwerten
- Der gleiche Datenwert kann auch mehrfach in der Liste vorkommen
- Die Liste ist definiert als einfache Verkettung von aufsteigend nach Datenwert geordneten Einträgen der Form

```
1 class Elem {
2     int value; // der gespeicherte Datenwert 0 ... 9999
3     Elem next; // Verweis auf das nächste Datenelement
4 }
```

- Implementieren Sie folgende Operationen für `IList`

  a) Das Anlegen einer leeren `IList` der Form `IList il = new IList()`

  b) Das Einfügen eines neuen Elementes mit einer Methode `void insert(int value)` entsprechend den obigen Anforderungen. Hierbei muss der Index ggf. aktualisiert werden!

c) Das Ausgeben (mittels `system.out.println`) von allen Listenelementen in einem gegebenen Wertebereich *untergrenze* ... *obergrenze* mit einer Methode

`void printInterval(int untergrenze, int obergrenze)`

Dabei muss der Index für die Bestimmung des passenden Einsprungpunktes in die Liste genutzt werden. Im Beispiel würde `printInterval(42, 1400)` also die Werte 1201 und 1304 ausgeben.

- Zur Vereinfachung braucht der Index lediglich für die Realisierung von `printInterval` ausgenutzt zu werden. `insert` darf die Liste auch von Anfang an traversieren, muss den Index aber korrekt aktualisieren.

```
1  public class IList
2  {
3      private Elem first;
4      private Elem[] index;
5
6      IList() {
7          index = new Elem[10];
8      }
9
10     void insert(int value) {
11         int idx = value / 1000;
12
13         Elem e = new Elem();
14         e.value = value;
15         // 1. Fall: Listenkopf ändert sich
16         if (first == null || first.value >= value) {
17             e.next = first;
18             index[idx] = e;
19             first = e;
20         } else {
21             // 2. Fall: Listenrumpf ändert sich
22             Elem cur = first;
23             while (cur.next != null && cur.next.value < value)
24               cur = cur.next;
25             e.next = cur.next;
26             if (index[idx] == cur.next)
27               index[idx] = e;
28             cur.next = e;
29         }
30     }
31
32     void printInterval(int lower, int upper) {
33         int idx = lower / 1000;
34         Elem cur;
35
36         do {
37             cur = index[idx];
38         } while (cur == null && ++idx < 10);
39
40         while (cur != null && cur.value < lower)
41              cur = cur.next;
42
43         while (cur != null && cur.value <= upper) {
44             System.out.println(cur.value);
45             cur = cur.next;
46         }
47     }
48
49 }
```

Aufgabe 4 Traversieren von Listen (7 Punkte)

Gegeben sei eine Liste mit der aus der Vorlesung bekannten Struktur, definiert durch die Klassen

```
1  class List {
2    Elem first;
3    ...
4  };
5
```

```
6  class Elem {
7    int value;
8    Elem next;
9  }
```

Implementieren Sie zwei statische Methoden zur Ausgabe der `value`-Variablen der Listenelemente, wobei das erste Element der Liste an die Methode übergeben wird.

a) `static void printIterative(Elem first)` soll diese Aufgabe *iterativ* erledigen.

b) `static void printRecursive(Elem first)` soll dagegen *rekursiv* arbeiten.

```
1     static public void printIterative(Elem first) {
2         for (Elem cur = first; cur != null; cur = cur.next)
3           System.out.println(cur.value);
4     }
5
6     static public void printRecursive(Elem first) {
7         if (first != null) {
8             System.out.println(first.value);
9             printRecursive(first.next);
10        }
11    }
```

Aufgabe 5 Klassen und Interfaces (15 Punkte)

a) Definieren Sie eine abstrakte Klasse `Fahrzeug`, bestehend aus

- einer Methode `geschwindigkeit`, die die aktuelle Geschwindigkeit des Fahrzeugs im km/h zurückgeben soll. Diese Methode soll nur in den Unterklassen implementiert werden.
- einer Methode `stop`, die die aktuelle Geschwindigkeit des Fahrzeugs auf Null setzt. Diese Methode soll nur in den Unterklassen implementiert werden.
- eine Datenkomponente `name` für den Namen des Fahrzeugs, der nur in dieser Klasse sichtbar sein darf.
- eine Methode `getName`, die den Namen als Rückgabewert hat. Diese Methode soll in der Klasse `Fahrzeug` implementiert werden.
- eine Methode `setName`, die den als String übergebenen Namen des Fahrzeugs setzt. Auch diese Methode soll in der Klasse `Fahrzeug` implementiert werden.

b) Definieren Sie ein Interface `Land` mit einer rückgabewertlosen Methode `fahren`. Durch deren Aufruf soll später das Landfahrzeug in Bewegung versetzt werden.

c) Definieren Sie ein Interface `Frachttransport` mit Methoden `einladen` und `ausladen`, die jeweils das Frachtgewicht in Tonnen als Parameter bekommen sollen. Eine weitere Methode `freieKapazität` soll die noch verfügbare Frachtkapazität in Tonnen liefern.

d) Definieren Sie aufbauend auf der Klasse `Fahrzeug` und den Interfaces eine Klasse `LKW` für einen Lastwagen mit einer Frachtkapazität von 12,5 Tonnen, der sich beim Fahren mit einer Geschwindigkeit von 40,5 km/h bewegt. Der Name des Fahrzeugs soll über den Konstruktor gesetzt werden. Implementieren Sie alle benötigten Methoden!

Hinweis: Die Implementierungen ihrer Methoden sollen sinngemäß das richtige tun, aufwendige Prüfungen (Überschreitung der Frachtkapazität, mehr Ausladen als eingeladen wurde etc.) sind *nicht* erforderlich.

```
1  abstract class Fahrzeug {
2      abstract double geschwindigkeit();
3      abstract void stop();
4      private String name;
5      public String getName() { return name;}
6      public void setName(String name) { this.name = name;}
7  }
8
9  interface Land {
10    public void fahren();
11 }
12
13 interface Frachttransport {
14    void einladen(double x);
15    void ausladen(double x);
16    double freieKapazität();
17 }
18
19 class LKW extends Fahrzeug implements Land, Frachttransport {
20
21   private double v;
```

```
22    private double capacity;
23
24    LKW(String name) { setName(name); v = 0; capacity = 12.5;}
25    public double geschwindigkeit() { return v; }
26    public void stop() { v = 0; }
27    public void fahren() {v = 40.5;}
28    public void einladen(double x) {capacity -= x; }
29    public void ausladen(double x) { capacity += x; }
30    public double freieKapazität() { return capacity; }
31  }
```