RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

LEHR- UND FORSCHUNGSGEBIET INFORMATIK II
RWTH Aachen · D-52056 Aachen · GERMANY
http://www-i2.informatik.rwth-aachen.de/lufgi2
Prof. Dr. Jürgen Giesl

Diplomvorprüfung / Zwischenprüfung Informatik I - Programmierung

4. 10. 2004

Vorname: _____

Nachname: _____

Matrikelnummer: _____

Studiengang (bitte ankreuzen):

- ○ Informatik Diplom ○ Informatik Lehramt
- ○ Sonstige: _____

- Schreiben Sie bitte auf jedes Blatt **Vorname**, **Name** und **Matrikelnummer**.
- Geben Sie Ihre Antworten bitte in lesbarer und verständlicher Form an. Schreiben Sie bitte nicht mit roten Stiften.
- Bitte beantworten Sie die Aufgaben auf den **Aufgabenblättern**. Benutzen Sie ggf. auch die Rückseiten der **zur jeweiligen Aufgabe gehörenden** Aufgabenblätter.
- Antworten auf anderen Blättern können nur berücksichtigt werden, wenn **Name, Matrikelnummer und Aufgabennummer** deutlich darauf erkennbar sind.
- Was nicht bewertet werden soll, kennzeichnen Sie bitte durch **Durchstreichen**.
- Werden Täuschungsversuche beobachtet, so wird die Klausur mit **nicht bestanden** bewertet.
- Geben Sie bitte am Ende der Klausur **alle Blätter zusammen mit den Aufgabenblättern ab**.

	Anzahl Punkte	Erreichte Punkte
Aufgabe 1	15	
Aufgabe 2	13	
Aufgabe 3	14	
Aufgabe 4	25	
Aufgabe 5	17	
Aufgabe 6	16	
Summe	100	
Note	-	

Vorname	Name	Matr.-Nr.

Aufgabe 1 (Programmanalyse, 9 + 6 Punkte)

a) Geben Sie die Ausgabe des Programms für den Aufruf `java M` an. Schreiben Sie hierzu jeweils die ausgegebenen Zeichen hinter den Kommentar "`AUSGABE:`".

```
public class A {
    public int k = 2;
    public A() {
        k = 1;
    }
    public void f(int x) {
        k = k * x;
    }
}

public class B extends A {
    public void f(int x) {
        k = k + x;
    }
    public void g(int x) {
        super.f(x);
        f(x);
    }
}


public class M {
    public static void main(String[] args) {

        A a = new A();
        System.out.println(a.k);               // AUSGABE:

        B b = new B();
        System.out.println(b.k);               // AUSGABE:

        a.f(2);
        System.out.println(a.k);               // AUSGABE:

        b.f(2);
        System.out.println(b.k);               // AUSGABE:

        b.g(2);
        System.out.println(b.k);               // AUSGABE:

        a = b;

        a.f(2);
        System.out.println(a.k + ", " + b.k);  // AUSGABE:

        b.f(2);
        System.out.println(a.k + ", " + b.k);  // AUSGABE:
    }
}
```

Vorname	Name	Matr.-Nr.

b) Es wurde eine neue Klasse `C` geschrieben. Welche drei Fehler treten beim Compilieren auf? Begründen Sie Ihre Antwort kurz.

```
public abstract class C extends A {

    public C() {
        k = 3;
    }

    void f(int x) {
        A a;
        B b = null;
        C c = new C();
        a = b;
        b = a;
    }

}
```

Vorname	Name	Matr.-Nr.

4

Aufgabe 2 (Verifikation, 10 + 3 Punkte)

Der folgende Algorithmus berechnet die Summe aller Zahlen eines Arrays.

a) Vervollständigen Sie die Verifikation des folgenden Algorithmus im Hoare-Kalkül, indem Sie die unterstrichenen Teile ergänzen. Hierbei dürfen zwei Zusicherungen nur dann direkt untereinander stehen, wenn die untere aus der oberen folgt. Hinter einer Programmanweisung darf nur eine Zusicherung stehen, wenn dies aus einer Regel des Hoare-Kalküls folgt.

Eingabe: Ein Array `a` der Länge `n` von Integer-Zahlen,
d.h. `a` enthält die Zahlen `a[0]`, ..., `a[n-1]`
Ausgabe: `res`
Vorbedingung: $\texttt{n} \geq 0$
Nachbedingung: $\texttt{res} = \sum_{\texttt{j}=0}^{\texttt{n-1}} \texttt{a[j]}$, d.h., $\texttt{res} = \texttt{a[0]} + \ldots + \texttt{a[n-1]}$

$\langle\ \texttt{n} \geq 0\ \rangle$

$\langle\ \texttt{n} \geq 0\ \wedge$ _____ $\rangle$

```
res = 0;
```

$\langle$ _____ $\rangle$

$\langle$ _____ $\rangle$

```
i = n;
```

$\langle$ _____ $\rangle$

$\langle$ _____ $\rangle$

```
while (i > 0) {
```

$\langle$ _____ $\rangle$

$\langle$ _____ $\rangle$

```
  i = i - 1;
```

$\langle$ _____ $\rangle$

```
  res = res + a[i];
```

$\langle$ _____ $\rangle$

```
}
```

$\langle$ _____ $\rangle$

$\langle\ \texttt{res} = \sum_{\texttt{j}=0}^{\texttt{n-1}} \texttt{a[j]}\ \rangle$

Vorname	Name	Matr.-Nr.

5

b) Beweisen Sie die Terminierung des Algorithmus. Geben Sie hierzu eine Variante für die `while`-Schleife an. Zeigen Sie, dass es sich tatsächlich um eine Variante handelt und beweisen Sie damit unter Verwendung des Hoare-Kalküls die Terminierung.

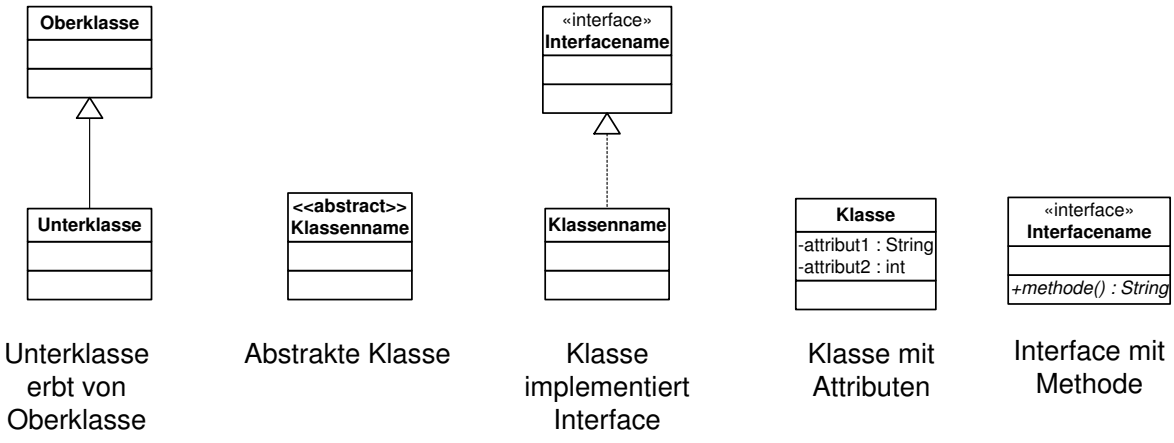
Vorname	Name	Matr.-Nr.

Aufgabe 3 (Datenstrukturen in Java, 6 + 8 Punkte**)**

Ihre Aufgabe ist es, eine objektorientierte Datenstruktur zur Verwaltung von Sensoren zu entwerfen. Diese soll in Java implementierbar sein. Bei der vorangehenden Analyse wurden folgende Eigenschaften der verschiedenen Sensorarten ermittelt:

- Ein Gassensor ist ein Alarmsensor, der einen Messwert (Konzentration eines beliebigen Gases in der Umgebungsluft als Prozentsatz) sowie einen einstellbaren Schwellwert besitzt. Weiterhin kennt er die Art des entdeckten Gases (als Text).
- Ein Schocksensor ist ein Alarmsensor, der einen Messwert (detektierte Erschütterung ausgedrückt als Vielfaches der Erdbeschleunigung g) sowie einen einstellbaren Schwellwert besitzt. Darüberhinaus speichert der Sensor die für einen Menschen kritische Erschütterung (wiederum als Vielfaches der Erdbeschleunigung g).
- Ein Temperatursensor ist ein Alarmsensor, der einen Messwert (Temperatur in Grad Celsius) sowie einen einstellbaren Schwellwert besitzt. Weiterhin speichert der Sensor, ob in den letzten 24 Stunden Frost herrschte.
- Ein Feuchtesensor ist ein Sensor, der einen Messwert (relative Feuchtigkeit der Umgebungsluft als Prozentsatz) besitzt. Außerdem speichert er den aktuellen Taupunkt (Temperatur in Grad Celsius).
- Temperatur- und Feuchtesensoren sind gleichzeitig Wettermessgeräte und besitzen als solche die Möglichkeit, eine Tendenz der Messwerte zu berechnen und als Text zurückzuliefern.

a) Entwerfen Sie unter Berücksichtigung der Prinzipien der Datenkapselung eine geeignete Klassenhierarchie für die oben aufgelisteten Arten von Sensoren. Achten Sie darauf, dass gemeinsame Merkmale in (evtl. abstrakten) Oberklassen zusammengefasst werden. Notieren Sie Ihren Entwurf grafisch und verwenden Sie dazu die folgende Notation:



Geben Sie für jede Klasse ausschließlich den jeweiligen Namen der Klasse und die Namen ihrer Attribute an. Geben Sie für jedes Interface ausschließlich den Namen des Interface sowie die Namen seiner Methoden an.

Vorname	Name	Matr.-Nr.

7

b) Implementieren Sie in Java eine Methode `checkSensors`, die ein Array von Sensoren übergeben bekommt. Die Methode soll *true* zurückliefern, falls mindestens einer der Sensoren ein Alarmsensor ist, bei dem der Messwert den Schwellwert überschreitet. Ansonsten wird *false* zurückgeliefert. Gehen Sie dabei davon aus, dass für alle Attribute geeignete Get- und Set-Methoden (Selektoren) existieren und verwenden Sie für den Zugriff auf die benötigten Attribute die passenden Selektoren.

Vorname	Name	Matr.-Nr.

Aufgabe 4 (Programmierung in Java, 5 + 4 + 8 + 8 Punkte)

In dieser Aufgabe sollen Programme unter Beachtung der Datenkapselung entworfen werden.

a) Gegeben ist das folgende Interface.

```
public interface Vergleichbar {
    public int vergleiche(Vergleichbar other);
}
```

Falls `s` und `t` zwei Objekte vom Typ `Vergleichbar` sind, so soll `s.vergleiche(t)` den folgenden Wert liefern:

$$\texttt{s.vergleiche(t)} == \begin{cases} 1, & \text{falls } \texttt{s} \text{ größer als } \texttt{t} \text{ ist} \\ 0, & \text{falls } \texttt{s} \text{ und } \texttt{t} \text{ gleich groß sind,} \\ -1, & \text{sonst.} \end{cases}$$

Schreiben Sie eine Klasse `VerglInteger` in Java, die `int`-Werte kapselt und das Interface `Vergleichbar` implementiert. Konstruktoren sowie Get- und Set-Methoden für die Attribute (Selektoren) müssen nicht implementiert werden. Zum Vergleich, ob ein `int`-Wert "größer" als ein anderer ist, soll hierbei die übliche Relation $>$ auf den ganzen Zahlen verwendet werden.

Beispiel:
Wir betrachten Objekte `eins`, `zwei`, ..., `fuenf`, die die Zahlen `1`, `2`, ..., `5` kapseln. Falls es einen geeigneten Konstruktor gibt, könnten sie durch `Vergleichbar eins = new VerglInteger(1)`, `Vergleichbar zwei = new VerglInteger(2)` etc. erzeugt werden. Dann soll folgendes gelten: `eins.vergleiche(zwei)==-1`, `fuenf.vergleiche(drei)==1`, `vier.vergleiche(vier)==0`.

Vorname	Name	Matr.-Nr.

10

b) Die folgenden zwei Klassen `OrdList` und `Element` dienen zur Darstellung von sortierten Listen. In diesen Listen können Objekte vom Typ `Vergleichbar` gespeichert werden. Ein Beispiel für eine sortierte Liste mit Objekten vom Typ `VerglInteger` ist die Liste `[eins, zwei, zwei, drei, vier]`. Hingegen ist `[eins, zwei, drei, zwei, vier]` keine sortierte Liste.

Im allgemeinen gilt: Eine Liste $[a_1, a_2, \ldots a_n]$ ist genau dann sortiert, wenn für je zwei Elemente a_i und a_j mit $i < j$ gilt: a_i`.vergleiche(`a_j`) <= 0`.

Die einzelnen Listenelemente werden durch Objekte der Klasse `Element` implementiert, die jeweils ein Attribut `wert` für den gespeicherten Wert und ein Attribut `next` für das nächste Element der sortierten Liste besitzen.

Ein Objekt der Klasse `OrdList` hat nur ein Attribut `kopf`, das auf den Anfang der Liste zeigt. Für die Liste `[eins, zwei, zwei, drei, vier]` ist dies das Objekt vom Typ `Element`, dessen `wert` das Objekt `eins` ist, und dessen `next`-Attribut auf das Objekt vom Typ `Element` zeigt, welches den Anfang der Restliste `[zwei, zwei, drei, vier]` darstellt. Der Konstruktor `OrdList()` erzeugt eine leere (und damit trivialerweise sortierte) Liste.

```
public class Element {

    private Vergleichbar wert;
    private Element next;

    public Element(Vergleichbar wert, Element next) {
        this.wert = wert;
        this.next = next;
    }
    public void setNext(Element next) {
        this.next = next;
    }
    public Element getNext() {
        return next;
    }
    public Vergleichbar getWert() {
        return wert;
    }
}

public class OrdList {

    private Element kopf;

    public OrdList() {
        kopf = null;
    }

    ...
}
```

Vorname	Name	Matr.-Nr.

11

Implementieren Sie in der Klasse `OrdList` die Methode

```
public int laenge() { ... },
```

die die Länge der Liste, d.h. die Anzahl der Elemente in der sortierten Liste berechnet.

Beispiel:
Falls `list` die Liste `[eins, zwei, zwei, drei, vier]` darstellt, so gilt `list.laenge() == 5`.

c) Implementieren Sie in der Klasse `OrdList` die Methode

```
public void fuegeEin(Vergleichbar neu) { ... },
```

die ein neues `Element` mit dem Wert `neu` in die sortierte Liste so einfügt, dass die resultierende Liste auch sortiert ist.

Beispiel:
Falls `list` die Liste `[eins, zwei, zwei, drei, vier]` darstellt, so gilt `list.fuegeEin(drei) == [eins, zwei, zwei, drei, drei, vier]`.

Vorname	Name	Matr.-Nr.

d) Implementieren Sie in der Klasse `OrdList` die Methode

```
public int enthaelt(Vergleichbar gesucht) { ... },
```

die berechnet, wie oft der übergebene Wert `gesucht` in der sortierten Liste vorkommt. Aus Effizienzgründen sollen nur so viele Elemente der Liste untersucht werden, wie nötig. Die Methode `enthaelt` und ggf. benötigte Hilfsfunktionen sollen ohne Verwendung von Schleifen realisiert werden. Sie dürfen aber *Rekursion* benutzen.

Beispiel:
Falls `list` die Liste `[eins, zwei, zwei, drei, vier]` darstellt, so gilt `list.enthaelt(drei) == 1`, `list.enthaelt(zwei) == 2` und `list.enthaelt(fuenf) == 0`. Für den Aufruf `list.enthaelt(zwei)` soll die Berechnung bei Erreichen des vierten Elements (das der `drei` entspricht) abgebrochen werden.

Vorname	Name	Matr.-Nr.

Aufgabe 5 (Funktionale Programmierung in Haskell, $3 + 3 + 4 + 2 + 1 + 4$ Punkte)

a) Geben Sie den allgemeinsten Typ der Funktionen `second`, `f` und `g` an.

```
second (x, y, z) = y

f x y = f y x

g x y = y ++ (map x y)
```

b) Bestimmen Sie das Ergebnis der Auswertung für die beiden Ausdrücke

```
let (minus_vier, f) = (-4, \x -> if x < 0 then (-x) else x)
    in f minus_vier +  f (f minus_vier)
```

```
map (\x -> filter (\y -> 3 < y) x) [ [1,2,3], [4,7], [0,5] ]
```

Vorname	Name	Matr.-Nr.

15

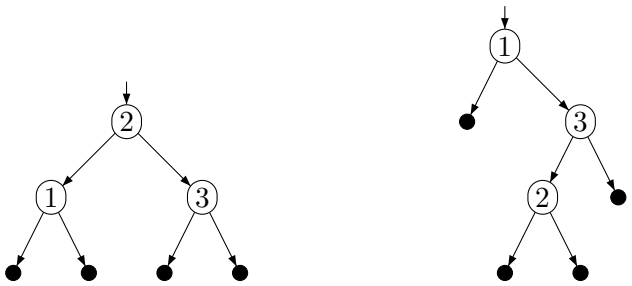
c) Schreiben Sie eine Funktion `split :: Int -> [a] -> (a, [a], [a])` in Haskell, die eine natürliche Zahl n und eine mindestens $n+1$-elementige Liste $[a_0, \ldots, a_n, \ldots, a_m]$ mit $0 \leq n \leq m$ als Eingabe erwartet, und dann die Liste am Element a_n aufspaltet. Das Ergebnis soll das Element a_n, die Liste davor und die Liste danach enthalten. Das Resultat ist also das Tripel $(a_n, [a_0, \ldots, a_{n-1}], [a_{n+1}, \ldots, a_m])$. So ergibt `split 1 [2,5,4,7]` das Resultat `(5, [2], [4,7])` und `split 3 [2,5,4,7]` ergibt `(7, [2,5,4], [])`.

d) Schreiben Sie eine Funktion `splitHalf :: [a] -> (a, [a], [a])` in Haskell, die eine (nichtleere) Liste an ihrer Hälfte aufspaltet. Beispielsweise hat `splitHalf [2,5,4,7,3]` also das Ergebnis `(4, [2,5], [7,3])`. Bei einer Liste mit gerader Anzahl von Elementen ist es unerheblich, ob die linke oder die rechte Hälfte ein Element mehr besitzt. So hat `splitHalf [9,1,4,5]` also entweder das Ergebnis `(4, [9,1], [5])` oder das Ergebnis `(1, [9], [4,5])`.

Sie dürfen hierbei die Hilfsfunktion `split` aus Aufgabenteil c) sowie die vordefinierten Funktionen `length :: [a] -> Int` und `div :: Int -> Int -> Int` verwenden. Hierbei berechnet `length` die Länge einer Liste und `div` berechnet die ganzzahlige Division. Beispielsweise gilt `length [2,5,4,7,3] == 5` und `div 5 2 == 2`.

Vorname	Name	Matr.-Nr.

e) Ein Binärbaum ist ein Baum, bei dem jeder Knoten entweder zwei Kinder oder kein Kind hat. In Knoten mit Kindern (sogenannten *inneren Knoten*) ist ein Wert gespeichert, während in Knoten ohne Kindern (sogenannten *Blättern*) nichts gespeichert wird. In der Abbildung sieht man zwei Bäume `b1` und `b2`, die Zahlen speichern.



Baum `b1` Baum `b2`

Entwerfen Sie eine Datenstruktur `Tree a` für solche Bäume in Haskell.

f) Schreiben Sie eine Funktion `listToTree :: [a] -> Tree a` in Haskell, die eine Liste in einen balancierten Binärbaum umwandelt. Die in den Knoten des Baums gespeicherten Werte sollen also gerade die Listenelemente sein. Ein Binärbaum heißt *balanciert*, falls für jeden Knoten der unter ihm liegende linke und rechte Teilbaum ungefähr gleich groß sind, d.h. die Anzahl der Knoten im linken und rechten Teilbaum unterscheidet sich um maximal 1. Beispielsweise ist der Baum `b2` nicht balanciert, während der Baum `b1` balanciert ist und ein mögliches Ergebnis von `listToTree [1,2,3]` ist. Tipp: Verwenden Sie die Funktion `splitHalf` aus Aufgabenteil d).

Vorname	Name	Matr.-Nr.

Aufgabe 6 (Logische Programmierung in Prolog, 7 + 3 + 6 Punkte**)**

a) Sie dürfen in der gesamten Teilaufgabe keine vordefinierten Prädikate verwenden.

- Definieren Sie in Prolog ein dreistelliges Prädikat `app`, welches zwei Listen konkateniert. Die Aussage `app(L1, L2, L3)` soll also genau dann wahr sein, wenn die Liste `L3` diejenige Liste ist, die entsteht, wenn man an das Ende von `L1` die Liste `L2` anfügt. Beispielsweise gilt `app( [1,4,3], [8,4], [1,4,3,8,4] )`.

- Definieren Sie ein zweistelliges Prädikat `rev`, welches Listen umdreht. Beispielsweise gilt `rev( [1,2,1,3], [3,1,2,1] )`. Sie dürfen hierbei das Prädikat `app` verwenden.

b) Geben Sie den allgemeinsten Unifikator für die folgenden Term-Paare an, oder begründen Sie, warum dieser nicht existiert.

- `f(g(a), X, Y)` und `f(Y, g(b), X)`

- `f(X, Y, Z)` und `f(g(Y, Y), g(Z,Z), a)`

Vorname	Name	Matr.-Nr.

c) Gegeben sei das folgende Prolog-Programm.

```
p(a,b).
p(X,X) :- p(X,c).
p(c,X) :- p(X,b).
```

- Stellen Sie den Beweisbaum für die Anfrage "`?- p(Y,Z).`" graphisch dar.

- Geben Sie alle Lösungspaare für `Y` und `Z` bei dieser Anfrage an. Welche davon findet Prolog?