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

Semestralklausur
Informatik I - Programmierung
9. 1. 2004

Vorname: ____________________

Nachname: ____________________

Matrikelnummer: ____________________

Studiengang (bitte ankreuzen):

○ Informatik Diplom ○ Informatik Lehramt ○ Mathematik Diplom
○ CES ○ Werkstoffinformatik ○ Computermathematik
○ 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	12	
Aufgabe 2	11	
Aufgabe 3	12	
Aufgabe 4	18	
Summe	53	
Note	-	

Vorname	Name	Matr.-Nr.

Aufgabe 1 (Programmanalyse, 6 + 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 extends B {
  public A() {
    super();
    increase(this);
  }
  public void increase(B b) {
    b.summe = b.summe + 1;
  }
  public String toString() {
    return "id = "+id+", summe = "+summe;
  }
}
public abstract class B {
  protected int summe = 0;
  public static int id = 1;
  public B() {
    id = id + 1;
  }
  public void increase(B b) {
    b.summe = b.summe + id;
  }
}
public class M {
  public static void main(String[] args) {
    A a = new A();
    System.out.println(a);       // AUSGABE:

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

    b = a;
    a.summe = b.summe + A.id;
    System.out.println(a);       // AUSGABE:

    System.out.println(b);       // AUSGABE:
  }
}
```

(b) Zu der Klasse `A` wird die folgende Methode hinzugefügt. Welche drei Fehler treten beim Compilieren auf? Begründen Sie Ihre Antwort kurz.

```
public A kopie() {
    B c = new B();
    B.summe = c.increase(this);
    return c;
}
```

Vorname	Name	Matr.-Nr.

Vorname	Name	Matr.-Nr.

Aufgabe 2 (Verifikation, 7 + 4 Punkte)

Der Algorithmus P berechnet durch Aufsummierung natürlicher Zahlen das Produkt $\frac{n(n+1)}{2}$.

a) Vervollständigen Sie die folgende Verifikation des Algorithmus P 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.

Algorithmus: P
Eingabe: $\texttt{n} \in \mathbb{N} = \{0, 1, 2, \ldots\}$
Ausgabe: `res`
Vorbedingung: $\texttt{n} \geq 0$
Nachbedingung: $\texttt{res} = \frac{n(n+1)}{2}$

$\langle\, n \geq 0 \,\rangle$

$\langle\, n \geq 0 \;\wedge$ ____________________ $\rangle$

```
i = 0;
```

$\langle\, n \geq i \;\wedge$ ____________________ $\rangle$

```
res = 0;
```

$\langle\, n \geq i \;\wedge$ ____________________ $\rangle$

$\langle\, n \geq i \;\wedge$ ____________________ $\rangle$

```
while (n > i) {
```

$\langle\, n \geq i \wedge n > i \;\wedge$ ________________ $\rangle$

$\langle$ __________________________ $\rangle$

```
  i = i + 1;
```

$\langle\, n \geq i \;\wedge$ ____________________ $\rangle$

$\langle\, n \geq i \;\wedge$ ____________________ $\rangle$

```
  res = res + i;
```

$\langle\, n \geq i \;\wedge$ ____________________ $\rangle$

```
}
```

$\langle\, n \geq i \wedge n \not> i \;\wedge$ ________________ $\rangle$

$\langle\, res = \frac{n(n+1)}{2} \,\rangle$

Vorname	Name	Matr.-Nr.

b) Beweisen Sie die Terminierung des Algorithmus P. 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.

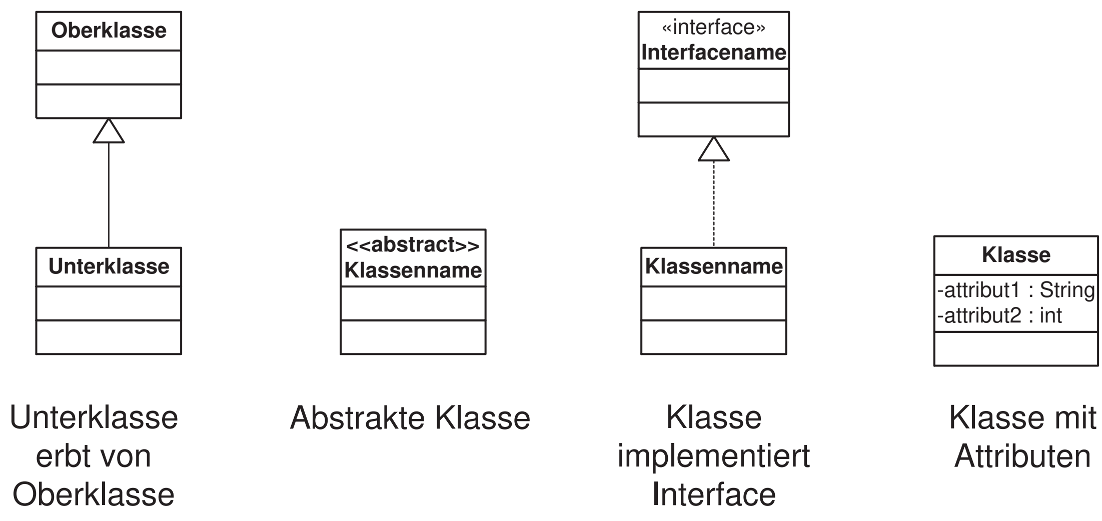
Vorname	Name	Matr.-Nr.

Aufgabe 3 (Datenstrukturen in Java, 3 + 9 Punkte)

Ihre Aufgabe ist es, eine objekt-orientierte Datenstruktur zur Verwaltung von Fahrzeugen zu entwerfen. Bei der vorangehenden Analyse wurden folgende Eigenschaften der verschiedenen Fahrzeugarten ermittelt:

- Ein Fahrrad ist dadurch charakterisiert, dass es eine bestimmte Anzahl von Gängen sowie eine Farbe und zwei Räder besitzt.
- Ein Personenkraftwagen (PKW) ist ein Kraftfahrzeug (KFZ) mit einer bestimmten Leistung (PS) und dadurch charakterisiert, dass eine maximale Anzahl von Personen mit dem PKW transportiert werden darf. Darüberhinaus hat ein PKW eine Farbe und vier Räder.
- Ein Lastkraftwagen (LKW) ist ein KFZ mit einer bestimmten Leistung (PS) und dadurch charakterisiert, dass er ein maximal zulässiges Ladegewicht, eine Farbe, sowie sechs Räder besitzt.
- Ein Motorrad ist ein KFZ mit einer bestimmten Leistung (PS) und dadurch charakterisiert, dass es einen Soziussitz (Beifahrersitz) haben kann oder nicht. Darüberhinaus hat ein Motorrad eine Farbe und zwei Räder.

a) Entwerfen Sie eine geeignete Klassenhierarchie für die oben aufgelisteten Arten von Fahrzeugen. 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 ihren Namen und die Namen und Typen Ihrer Attribute an. Methoden von Klassen und Interfaces müssen nicht angegeben werden.

Vorname	Name	Matr.-Nr.

Vorname	Name	Matr.-Nr.

b) Implementieren Sie in `Java` eine Methode `getMaxLKW`, die ein Array von Fahrzeugen als Eingabe übergeben bekommt. Die Methode soll den LKW mit dem größten maximal zulässigen Ladegewicht ermitteln und diesen als Ergebnis zurückliefern. Falls in dem Array kein LKW vorkommt, soll `null` zurückgeliefert werden. Gehen Sie hierbei davon aus, dass für alle Attribute geeignete Selektoren existieren und verwenden Sie für den Zugriff auf die benötigten Attribute die passenden Selektoren.

Vorname	Name	Matr.-Nr.

Aufgabe 4 (Programmierung in Java, 3 + 5 + 10 Punkte)

a) Das folgende Interface `Value` beschreibt die Methoden, die benötigt werden, um Objekte bezüglich ihrer Größe zu vergleichen und in Strings umzuwandeln. Schreiben Sie eine Klasse `IntValue` mit einem Attribut vom Typ `int`, die das Interface implementiert. Die Implementierung von `isSmaller` soll dabei Objekte anhand der Größe ihres `int`-Attributs vergleichen. Die Implementierung von `toString` soll jedes Objekt in einen String wandeln, der dem `int`-Attribut entspricht (d.h., falls das `int`-Attribut den Wert `123` hat, so soll `toString` den String "`123`" liefern). Gehen Sie in der gesamten Aufgabe davon aus, dass Methoden mit Eingabeparametern vom Typ `Value` nicht mit dem Argument `null` aufgerufen werden.

```
public interface Value {
  public boolean isSmaller (Value other);
  public String toString ();
}
```

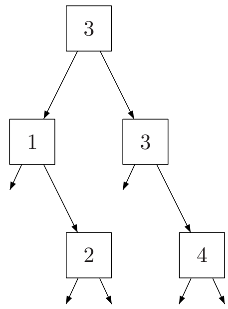
Vorname	Name	Matr.-Nr.

b) Betrachten Sie die folgenden weiteren Klassen.

```
public abstract class OrderedCollection {
    protected Element root = null;
    public abstract Value minimum ();
    public abstract void insert (Value v);

    public String toString () {
        if (root == null) return "[]";
        else return ("[" + root + "]");
    }
}

public class List extends OrderedCollection {...}
public class Tree extends OrderedCollection {...}

public abstract class Element {
    protected Value value;
    protected Element (Value value) {this.value = value;}
    public Value getValue () {return value;}
    public abstract String toString ();
}

public class ListElement extends Element {
    private ListElement next;
    public ListElement (Value value, ListElement next) {...}
    public void setNext (ListElement next) {this.next = next;}
    public ListElement getNext () {return next;}
    public String toString () {...}
}

public class Node extends Element {
    private Node left,right;
    public Node (Value value, Node left, Node right) {...}
    public void setLeft (Node left) {this.left = left;}
    public Node getLeft () {return left;}
    public void setRight (Node right) {this.right = right;}
    public Node getRight () {return right;}
    public String toString () {...}
}
```

Die Klasse `OrderedCollection` dient dazu, Sammlungen von `Value`-Objekten zu speichern, wobei diese Objekte aufgrund der Ordnung `isSmaller` sortiert werden. Im obigen Programm betrachten wir zwei Möglichkeiten, wie solche Sammlungen implementiert werden können. Die Klasse `List` realisiert geordnete Listen, bei denen die Elemente aufsteigend sortiert sind (d.h., Listen wie z.B. `[1,2,3,3,4]`). Die Klasse `Tree` realisiert geordnete binäre Bäume, bei denen die Knoten des linken Teilbaums immer nur Werte enthalten, die kleiner oder gleich groß wie der Wert des Wurzelknotens sind. Die Knoten des rechten Teilbaums enthalten nur Werte, die größer oder gleich groß wie der Wert des Wurzelknotens sind. Der folgende Baum ist ein Beispiel für einen geordneten Binärbaum.

Vorname	Name	Matr.-Nr.



Die Elemente einer `OrderedCollection` sind Objekte der Klasse `Element`. Hierbei enthalten Listen Objekte der Klasse `ListElement` und Binärbäume enthalten Objekte der Klasse `Node`. Für eine Liste `l` ist `l.root` das erste Listenelement (d.h., bei der obigen Liste ist das `value`-Attribut von `l.root` ein `IntValue`-Objekt mit dem Attributwert `1` und das `next`-Attribut ist das zweite Listenelement). Für einen Baum `t` ist `t.root` der Wurzelknoten (d.h., bei dem obigen Baum ist das `value`-Attribut von `t.root` ein `IntValue`-Objekt mit dem Attributwert `3` und die `left`- und `right`-Attribute sind die links und rechts folgenden Baumelemente).

Ergänzen Sie in den Klassen `ListElement` und `Node` die mit "`...`" markierten fehlenden Implementierungen der Konstruktoren und der `toString`-Methoden. Hierbei sollten Sie `toString` so implementieren, dass die Elemente bei geordneten Listen und Bäumen auch geordnet ausgegeben werden. Falls `o` die obige Liste oder der obige Baum ist, so sollte `o.toString ()` den String "`[1,2,3,3,4]`" liefern. Bei einer Liste mit dem einzigen Wert `1` würde man "`[1]`" erhalten. Achten Sie in Ihrer Implementierung auf die richtige "Kommasetzung".

Vorname	Name	Matr.-Nr.

Vorname	Name	Matr.-Nr.

c) Ergänzen Sie die mit "`...`" markierten fehlenden Implementierungen der Klassen `List` und `Tree` unter Beachtung der Prinzipien der Datenkapselung. Gehen Sie hierbei davon aus, dass die Sammlungen (d.h., die Listen und Bäume) wie oben beschrieben geordnet sind. Hierbei sollte `minimum` den kleinsten Wert der Sammlung zurückliefern und `insert` soll einen neuen Wert so in die Sammlung einfügen, dass Sie hinterher immer noch geordnet ist. Die `insert`-Methoden müssen *rekursiv* programmiert werden, d.h., ohne Verwendung von Schleifen. Kennzeichnen Sie Methoden mit dem Schlüsselwort "`static`", falls angebracht.

Vorname	Name	Matr.-Nr.