RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

LEHR- UND FORSCHUNGSGEBIET INFORMATIK 2
RWTH Aachen · D-52056 Aachen · GERMANY
http://programmierung.informatik.rwth-aachen.de/
Prof. Dr. Jürgen Giesl

LuFG Informatik II

Prof. Dr. Jürgen Giesl
Fabian Emmes, Carsten Fuhs, Carsten Otto, Prof. Dr. Peter Schneider-Kamp, Stephan Swiderski

Präsenzübung
Programmierung
7. 1. 2009

Vorname: _____

Nachname: _____

Matrikelnummer: _____

Studiengang (bitte ankreuzen):

○ Informatik Bachelor	○ Informatik Diplom	○ Informatik Lehramt
○ Mathematik Bachelor	○ Mathematik Diplom	
○ CES Bachelor	○ CES Diplom	
○ 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 oder mit Bleistiften.
- 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 **0 Punkten** bewertet.
- Geben Sie bitte am Ende der Klausur **alle Blätter** zusammen mit den Aufgabenblättern ab.

	Anzahl Punkte	Erreichte Punkte
Aufgabe 1	14	
Aufgabe 2	12	
Aufgabe 3	14	
Aufgabe 4	26	
Summe	66	
Prozentzahl		

Vorname	Name	Matr.-Nr.

Aufgabe 1 (Programmanalyse, 8 + 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 „`OUT:`“.

```
public class A {
    public int x = 1;
    public A(int x) {
        this.x += x;
    }
    public A(double x) {
        x += x;
    }
    public void f(double x) {
        this.x = this.x + (int) (x + B.y);
    }
}

public class B extends A {
    public static int y = 3;
    public int x = 0;
    public B(double x) {
        super((int) x);
    }
    public void f(int y) {
        this.x = y * 2;
        B.y = this.x;
    }
    public void f(double y) {
        this.x = (int) y + B.y;
    }
}

public class M {
    public static void main(String[] args) {
        A a = new A(B.y);
        System.out.println(a.x);                // OUT:
        a.f(1);
        System.out.println(a.x);                // OUT:
        B b = new B(3.0);
        System.out.println(b.x);                // OUT:
        A z = b;
        System.out.println(z.x);                // OUT:
        z.f(-5.0);
        System.out.println(b.x + " " + z.x);    // OUT:
        z.f(-6);
        System.out.println(b.x + " " + z.x);    // OUT:
    }
}
```

Vorname	Name	Matr.-Nr.

b) Wir schreiben zusätzlich zu `A` und `B` eine neue Klasse `C`. Welche drei Fehler treten beim Compilieren auf? Begründen Sie Ihre Antwort kurz.

```
public class C extends B {
    public String x;

    public C() {
        super();
    }

    public C(int x) {
        super(x);
        this.x = "Viel Erfolg!";
    }

    public int f(int x) {
        B b = new C(1);
        b.x = "Frohes Neues!";
        A a = b;
        b.f(2.0);
        return x;
    }
}
```

Vorname	Name	Matr.-Nr.

Aufgabe 2 (Verifikation, 10 + 2 Punkte**)**

Seien a und b jeweils Arrays der Länge n, wobei a aus den Zahlen $a[0], \ldots, a[n-1]$ und b aus den Zahlen $b[0], \ldots, b[n-1]$ besteht. Dann berechnet der Algorithmus P die Summe $\sum_{i=0}^{n-1} a[i] * b[i]$ (Standardskalarprodukt).

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. **Hinweis:** Für $\ell > m$ und einen beliebigen mathematischen Ausdruck p gilt $\sum_{i=\ell}^{m} p = 0$.

Algorithmus: P
Eingabe: Zwei Arrays a und b der Länge n, die die Zahlen $a[0], \ldots, a[n-1]$ bzw. $b[0], \ldots, b[n-1]$ enthalten
Ausgabe: res
Vorbedingung: $n \geq 0$
Nachbedingung: $res = \sum_{i=0}^{n-1} a[i] * b[i]$

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

$\langle$ _____ $\rangle$

$k = 0;$

$\langle$ _____ $\rangle$

$res = 0;$

$\langle$ _____ $\rangle$

$\langle$ _____ $\rangle$

`while` $(k < n)$ {

$\langle$ _____ $\rangle$

$\langle$ _____ $\rangle$

$k = k + 1;$

$\langle$ _____ $\rangle$

$\langle$ _____ $\rangle$

$res = res + a[k-1] * b[k-1];$

$\langle$ _____ $\rangle$

}

$\langle$ _____ $\rangle$

$\langle\ res = \sum_{i=0}^{n-1} a[i] * b[i]\ \rangle$

Vorname	Name	Matr.-Nr.

b) Beweisen Sie die Terminierung des Algorithmus P.

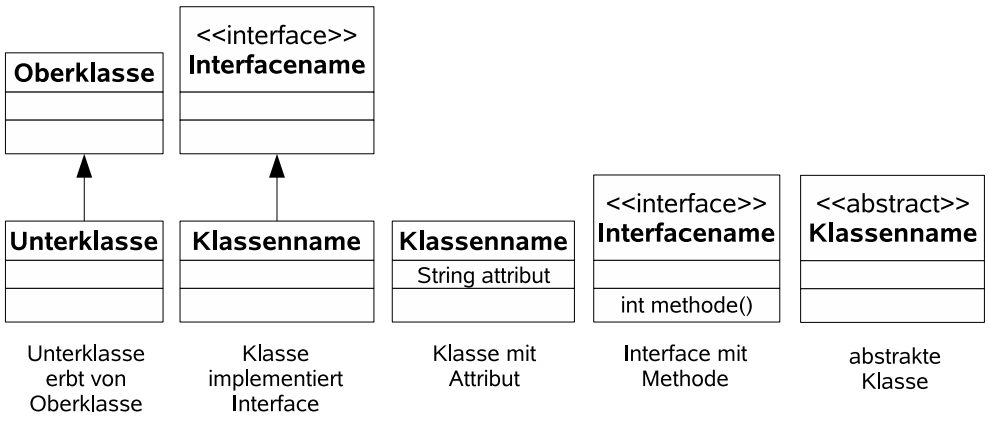
Vorname	Name	Matr.-Nr.

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

Ihre Aufgabe ist es, eine objektorientierte Datenstruktur zur Verwaltung von Räumen an einer Universität zu entwerfen. Bei der vorangehenden Analyse wurden folgende Eigenschaften der verschiedenen Räume ermittelt.

- Jeder Raum wird durch seine Nummer gekennzeichnet.
- Ein Seminarraum ist ein Raum, der sich durch die Fläche seiner Tafel auszeichnet. Außerdem ist für einen Seminarraum relevant, ob dort ein Beamer vorhanden ist.
- Ein Hörsaal ist ein Raum, für den neben der Fläche der Tafel auch die Anzahl der Sitzreihen wichtig ist.
- Neben den Räumen mit Tafel gibt es auch Räume, die als Büros genutzt werden. Für diese Räume ist ein wichtiges Attribut, auf welcher Etage sie sich befinden.
- Ein Professorenbüro ist ein Büro, für das der Name des Professors wichtig ist.
- Ein Rechnerraum ist ein Büro, das mit einer bestimmten Datenübertragungsrate an das Netzwerk der Universität angeschlossen ist.
- Alle Räume mit Tafeln sind entweder Seminarräume oder Hörsäle. Es kann aber Büros geben, die weder Professorenbüros noch Rechnerräume sind.
- Man kann sowohl Professorenbüros als auch Räume mit Tafeln als Prüfungsräume nutzen. Es ist nicht immer leicht zu erkennen, wieviele Plätze man bei einer Prüfung in dem jeweiligen Raum verwenden kann. Daher stellen diese Räume eine Methode zur Verfügung, mit der man die Anzahl der Plätze berechnen kann, die in einer Prüfung verwendet werden können.

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



Geben Sie für jede Klasse ausschließlich den jeweiligen Namen und die Namen und Datentypen ihrer Attribute an. Methoden von Klassen müssen nicht angegeben werden. Geben Sie für jedes Interface ausschließlich den jeweiligen Namen sowie die Namen und Ein- und Ausgabetypen seiner Methoden an.

Vorname	Name	Matr.-Nr.

Vorname	Name	Matr.-Nr.

b) Implementieren Sie in Java eine Methode `pruefung`. Die Methode bekommt als Parameter ein Array von Räumen übergeben. Sie soll die Anzahl von Plätzen zurückgeben, die insgesamt in den Räumen im Array zur Verfügung steht, die für Prüfungen vorgesehen sind.

Gehen Sie dabei davon aus, dass das übergebene Array nicht der `null`-Wert ist. Kennzeichnen Sie die Methode mit dem Schlüsselwort „`static`“, falls angebracht.

Vorname	Name	Matr.-Nr.

Aufgabe 4 (Programmierung in Java, 6 + 8 + 12 Punkte)

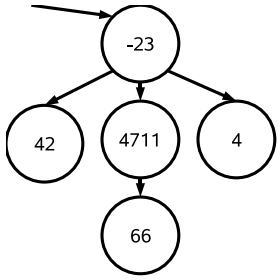
Die Klasse `Knoten` repräsentiert einen Knoten in einem Baum. Das Attribut `kinder` ist ein Array von `Knoten`, welches die Kinderknoten des Knotens enthält. Es kann immer davon ausgegangen werden, dass das Attribut `kinder` nicht `null` ist, das heißt es gibt immer ein `Knoten`-Array. In dem Array selbst kann `null` als Wert auftreten. Zusätzlich hat jeder Knoten die Attribute `inhalt` und `hilfswert`.

```
public class Knoten {
    public int inhalt;
    public int hilfswert;
    public Knoten[] kinder;
}
```

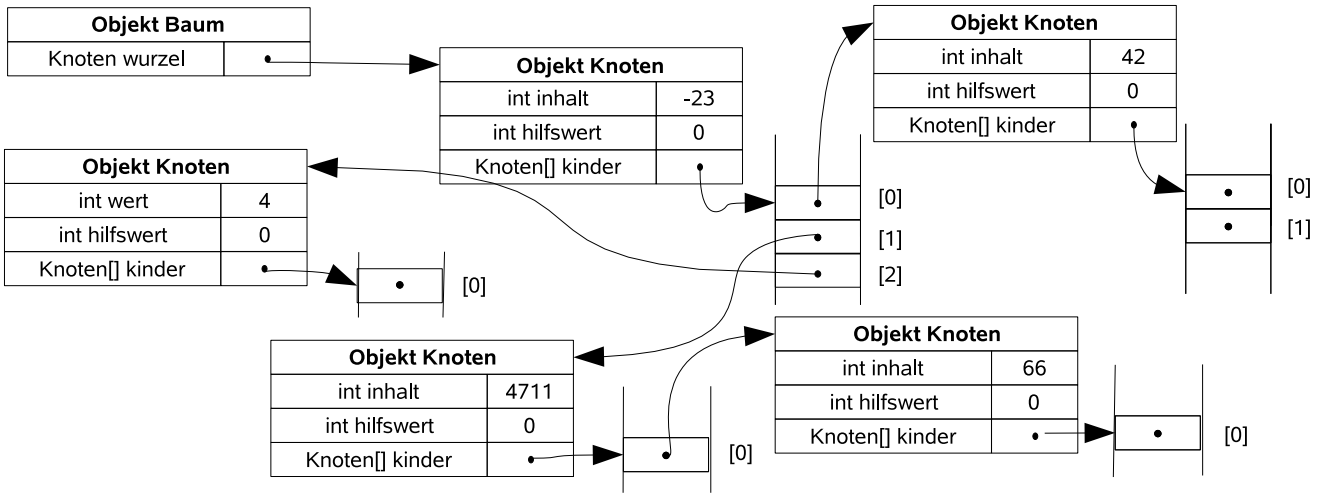
Die Klasse `Baum` repräsentiert einen Baum. Das Attribut `wurzel` der Klasse `Baum` ist der Wurzelknoten. Ein leerer Baum hat keine Knoten, so dass das Attribut `wurzel` den Wert `null` hat.

```
public class Baum  {
    Knoten wurzel;
}
```

Zum Beispiel könnte ein Baum fünf Knoten mit den Werten `-23`, `42`, `4711`, `66` und `4` enthalten, wobei das `wurzel`-Attribut des Baums auf den Knoten mit dem Wert `-23` zeigt:



Der Baum könnte dann zum Beispiel folgende Struktur im Speicher besitzen. Hierbei sind bei den Knoten mit den Werten `42`, `66` und `4` die Einträge im Array `kinder` alle `null`.



Vorname	Name	Matr.-Nr.

10

Implementieren Sie die folgenden Methoden in der Klasse `Baum`. Dabei dürfen Sie direkt (ohne Verwendung von Selektoren) auf die Attribute aus der Klasse `Knoten` zugreifen. Selbstverständlich dürfen Sie auch weitere benötigte Hilfsmethoden implementieren. Verwenden Sie die Schlüsselworte `public` und `private` auf sinnvolle Weise und kennzeichnen Sie Methoden als `static`, falls angebracht.

a) Die Methode `zaehleKnoten` der Klasse `Baum` zählt alle Knoten in einem Baum. Nach dem Aufruf der Methode `zaehleKnoten` soll in dem Attribut `hilfswert` jedes Knotens die Zahl der von diesem Knoten erreichbaren Knoten stehen (jeder Knoten kann wenigstens sich selbst erreichen). Zusätzlich soll die Methode `zaehleKnoten` die Anzahl der Knoten im Baum als Ergebnis zurückgeben. Sie dürfen eine `for`-Schleife verwenden, um ein Array zu durchlaufen. Ansonsten dürfen Sie aber ausschließlich Rekursion verwenden.

   Bei dem Beispiel-Baum auf der vorigen Seite würde `zaehleKnoten` also das Ergebnis 5 zurückliefern. Der `hilfswert` des Wurzelknotens wäre dann ebenfalls 5, der `hilfswert` des Knotens mit dem `inhalt` 4711 wäre 2, etc.

Vorname	Name	Matr.-Nr.

b) Die Methode `filterAnwenden` der Klasse `Baum` wendet einen Filter auf den Baum an. Ein Filter ist ein Objekt einer Klasse, welche das Interface `Filter` implementiert. Ein Beispiel hierfür ist die Klasse `NegativeFilter`.

```
public interface Filter {
 boolean filtern(Knoten knoten);
}
```

```
public class NegativeFilter implements Filter {
 public boolean filtern(Knoten knoten) {
  return (knoten.inhalt < 0);
 }
}
```

Die Methode `filtern` des Filters soll durch die Methode `filterAnwenden` auf jeden Knoten des Baums *höchstens einmal* angewendet werden. Wenn die Methode `filtern(Knoten knoten)` des Filters für einen Knoten den Wert `true` zurückliefert, sollen dieser Knoten und all seine Kinder aus dem Baum entfernt werden, indem der Knoten durch `null` ersetzt wird. Wenn der Rückgabewert `false` ist, wird der aktuelle Knoten nicht verändert und man wendet die Methode `filter` stattdessen auf die Kinder des aktuellen Knotens an. Implementieren Sie die Methode `public void filterAnwenden(Filter filter)` in der Klasse `Baum`. Verwenden Sie dabei ausschließlich Rekursion. Sie dürfen aber eine `for`-Schleife verwenden, um ein Array zu durchlaufen. Beachten Sie, dass die Methode `filtern` des Filters nicht mit dem Argument `null` aufgerufen werden darf.

Beispielsweise löscht `filterAnwenden(new NegativeFilter())` also alle Knoten (und deren Kinder) aus dem Baum, bei denen der `inhalt` negativ ist.

Vorname	Name	Matr.-Nr.

c) Die Methode `loescheTeilbaeume(int groesse)` der Klasse `Baum` sucht alle Knoten, die einen Unterbaum genau der Größe `groesse` aufspannen und entfernt diese und ihre Kinder. Sie hat die folgende Signatur:

```
public void loescheTeilbaeume(int groesse)
```

Die *Größe* eines Baums ist die Zahl seiner Knoten, d.h., der Baum auf S. 9 hat die Größe 5. Wenn man für diesen Baum `loescheTeilbaeume(2)` aufruft, wird also der Knoten mit dem `inhalt` 4711 sowie sein Kind (der Knoten mit dem `inhalt` 66) gelöscht. Implementieren Sie hierzu eine Klasse `Teilbaumfilter`, die das Interface `Filter` aus Teilaufgabe (b) implementiert. Objekte der Klasse `Teilbaumfilter` sollen ein `int`-Attribut `filterwert` besitzen. Wenn auf einen Baum ein `Teilbaumfilter` *angewendet* wird (durch die Methode `filterAnwenden` aus Teilaufgabe (b)), werden alle Knoten (und ihre Kinder) entfernt, bei denen der `hilfswert` gleich dem `filterwert` des `Teilbaumfilter`s ist.

Implementieren Sie die Methode `loescheTeilbaeume` der Klasse `Baum`, die die von Ihnen implementierte Klasse `Teilbaumfilter` benutzt. Verwenden Sie hierzu die Methoden `zaehleKnoten` aus Teilaufgabe (a) und `filterAnwenden` aus Teilaufgabe (b).