RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

LEHR- UND FORSCHUNGSGEBIET INFORMATIK II
RWTH Aachen · D-52056 Aachen · GERMANY
http://programmierung.informatik.rwth-aachen.de

Prof. Dr. Jürgen Giesl
Darius Dlugosz, Thomas v. d. Maßen, Antje Nowack

Übung *Informatik I - Programmierung* - Blatt 0

Dieses Blatt wird nicht eingesammelt. Es wird in den Übungsgruppen am 24./25. Oktober besprochen.

Aufgabe 1

Sie stehen vor einem Getränkeautomaten (ähnlich dem vor Mensa V) und möchten eine Flasche Limonade erwerben. Formulieren Sie einen Algorithmus, welcher beschreibt, wie Sie die Flasche Limonade erhalten. Achten Sie dabei auf möglicherweise auftretende Sonderfälle. Notieren Sie diesen Algorithmus in Form von Pseudo-Code (d.h., in strukturierter natürlicher Sprache).

Aufgabe 2

Gegeben sei die folgende Sprache:

$$L = \{w \in \{a, b\}^* | \sharp_a(w) = 3\}$$

das heisst, es können Wörter beliebiger Länge erzeugt werden, welche die Buchstaben a und b enthalten und in denen der Buchstabe a in einem Wort genau 3-mal vorkommt. Die folgenden Wörter sind beispielsweise in der Sprache enthalten:

abbbababb bbaaba aabbbba

Folgende Wörter sind nicht Bestandteil der Sprache:

abb abaabbba bbbb

a) Geben Sie das kürzeste Wort der oben angegebenen Sprache L an.

b) Geben Sie eine Grammatik G an, welche die Sprache L erzeugt.

Aufgabe 3

a) Was ist Syntax? Was ist Semantik? Erläutern Sie den Unterschied.

b) Impliziert gleiche Syntax auch gleiche Semantik? Geben Sie ein Beispiel.

c) Erläutern Sie folgende Aussage: Ein syntaktisch korrektes Programm ist nicht immer korrekt.

RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

LEHR- UND FORSCHUNGSGEBIET INFORMATIK II
RWTH Aachen · D-52056 Aachen · GERMANY
http://programmierung.informatik.rwth-aachen.de

Prof. Dr. Jürgen Giesl
Darius Dlugosz, Thomas v. d. Maßen, Antje Nowack

Übung *Informatik I - Programmierung* - Blatt 0

Dieses Blatt wird nicht eingesammelt. Es wird in den Übungsgruppen am 24./25. Oktober besprochen.

Aufgabe 1

Sie stehen vor einem Getränkeautomaten (ähnlich dem vor Mensa V) und möchten eine Flasche Limonade erwerben. Formulieren Sie einen Algorithmus, welcher beschreibt, wie Sie die Flasche Limonade erhalten. Achten Sie dabei auf möglicherweise auftretende Sonderfälle. Notieren Sie diesen Algorithmus in Form von Pseudo-Code (d.h., in strukturierter natürlicher Sprache).

Aufgabe 2

Gegeben sei die folgende Sprache:

$$L = \{w \in \{a, b\}^* | \sharp_a(w) = 3\}$$

das heisst, es können Wörter beliebiger Länge erzeugt werden, welche die Buchstaben a und b enthalten und in denen der Buchstabe a in einem Wort genau 3-mal vorkommt. Die folgenden Wörter sind beispielsweise in der Sprache enthalten:

abbbababb bbaaba aabbbba

Folgende Wörter sind nicht Bestandteil der Sprache:

abb abaabbba bbbb

a) Geben Sie das kürzeste Wort der oben angegebenen Sprache L an.

b) Geben Sie eine Grammatik G an, welche die Sprache L erzeugt.

Aufgabe 3

a) Was ist Syntax? Was ist Semantik? Erläutern Sie den Unterschied.

b) Impliziert gleiche Syntax auch gleiche Semantik? Geben Sie ein Beispiel.

c) Erläutern Sie folgende Aussage: Ein syntaktisch korrektes Programm ist nicht immer korrekt.

RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

LEHR- UND FORSCHUNGSGEBIET INFORMATIK II
RWTH Aachen · D-52056 Aachen · GERMANY
http://programmierung.informatik.rwth-aachen.de

LuFG
Informatik II

Prof. Dr. Jürgen Giesl
Darius Dlugosz, Thomas v. d. Maßen, Antje Nowack

Übung *Informatik I - Programmierung* - Blatt 1

Die Übungsblätter sollen in Gruppen von je 3 Studierenden bearbeitet werden. Diese 3 Studierenden müssen aus derselben Übungsgruppe kommen. Lösungen können bis zum 30. Oktober, 17.00 h, in den Kasten für Ihre Übungsgruppe im Foyer in der Ahornstr. 55 eingeworfen werden.
Bitte vergessen Sie nicht, die **Nummer Ihrer Übungsgruppe** sowie die **Namen** und **Matrikelnummer** der Mitglieder Ihrer 3er-Gruppe auf Ihre Abgabe zu schreiben.

Aufgabe 1 (2+2+2 Punkte)

Gegeben sei die folgende Sprache:

$$L = \{w \in \{a,b\}^* : \sharp_a(w) \text{ ist ungerade oder } \sharp_b(w) \text{ ist gerade}\}$$

das heißt, L enthält genau die endlichen Wörter, in denen die Anzahl der a's ungerade ist oder die Anzahl der b's gerade ist (oder beides gilt).
Die folgenden Wörter sind beispielsweise in der Sprache enthalten:

abbbab aabbab baaab ε

Folgende Wörter sind nicht Bestandteil der Sprache:

aab bbb baabb

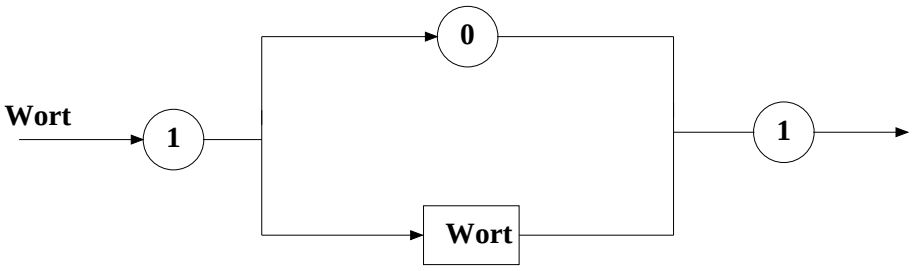
a) Geben Sie eine kontextfreie Grammatik G an, welche die Sprache L erzeugt.

b) Geben Sie ein Syntaxdiagramm ohne Nichtterminale an, das die Sprache L definiert.

c) Geben Sie eine Grammatik in EBNF mit nur einer Gleichung an, die L definiert.

Aufgabe 2 (2 Punkte)

Gegeben sei folgendes Syntaxdiagramm:



a) Geben Sie an, welche Aussagen für Wörter, die dem obigen Syntaxdiagramm entsprechen, zutreffen bzw. nicht zutreffen. Begründen Sie Ihre Antworten.

   (a) Jedes Wort besteht aus einer ungeraden Anzahl von Zeichen.

   (b) Jedes Wort enthält doppelt soviele Einsen wie Nullen.

   (c) Die Ziffernsumme ist in jedem Wort gerade

   (d) Nullen und Einsen treten immer abwechselnd auf.

b) Welche Sprache definiert das Syntaxdiagramm? Begründen Sie Ihre Antwort.

Aufgabe 3 (4 Punkte)

Existiert zu jedem Syntaxdiagramm eine Grammatik in EBNF, die dieselbe Sprache definiert? Beweisen Sie Ihre Antwort, indem Sie allgemein zu jedem Syntaxdiagramm eine äquivalente Grammatik in EBNF definieren oder durch ein Gegenbeispiel.

Aufgabe 4 (4 Punkte)

Gegeben sei folgende (nicht-kontextfreie) Grammatik $G = (N, T, P, S)$ mit $N = \{S, A, B\}$, $T = \{a, b\}$ und P definiert durch:

$$\begin{array}{lcl} S & \rightarrow & aAb|aBbb|\varepsilon \\ aA & \rightarrow & aaAb|a \\ Ab & \rightarrow & aAbb|b \\ aB & \rightarrow & aaBbb|a \\ Bb & \rightarrow & aBbbb|b \end{array}$$

Hierbei steht "$x \rightarrow y_1|y_2$" abkürzend für die Regeln $x \rightarrow y_1$ und $x \rightarrow y_2$.

a) Welche Sprache definiert die Grammatik G?

b) Geben Sie eine kontextfreie Grammatik an, die zu G äquivalent ist.

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
Darius Dlugosz, Thomas v. d. Maßen, Antje Nowack

Übung *Informatik I - Programmierung* - Blatt 2

Die Übungsblätter sollen in Gruppen von je 3 Studierenden bearbeitet werden. Diese 3 Studierenden müssen aus derselben Übungsgruppe kommen. Lösungen können bis zum Dienstag, den 6. November, 17.45 h, in den Kasten für Ihre Übungsgruppe im Foyer (neben Aula 2) in der Ahornstr. 55 eingeworfen werden.
Bitte vergessen Sie nicht, die **Nummer Ihrer Übungsgruppe** sowie die **Namen** und **Matrikelnummer** der Mitglieder Ihrer 3er-Gruppe auf Ihre Abgabe zu schreiben.

Wichtig: Alle Programme sind **sowohl** auf **Papier** in den Kasten einzuwerfen **als auch** elektronisch mit Hilfe des **Online-Systems** abzugeben.

Hinweis: Die Übungen, die am Donnerstag, den 01.11.01, aufgrund des Feiertages ausfallen, finden zu folgenden Terminen statt:

- Gruppe 15 - Fr, 2.11, 10:15 - 11:45, RS 3
- Gruppe 16 - Fr, 2.11,10:15 - 11:45, Fk 9
- Gruppe 17 - Fr, 2.11, 10:15 - 11:45, SG 203
- Gruppe 18 - Fr, 2.11, 17:30 - 19:00, SG 13
- Gruppe 19 - Mo, 5.11, 15:30 - 17:00, 6019
- Gruppe 20 - Fr, 2.11, 16:30 - 18:00, SG 422
- Gruppe 21 - Fr, 2.11, 10:15 - 11:45, SFo 4
- Gruppe 22 - Fr, 2.11, 16:30 - 18:00, SG 23
- Gruppe 23 - Fr, 2.11, 17:30 - 19:00, 5052
- Gruppe 24 - Fr, 2.11, 16:45 - 18:15, 6019
- Gruppe 25 - Fr, 2.11, 10:15 - 11:45, 6019
- Gruppe 26 - Fr, 2.11, 17:30 - 19:00, SG 512
- Gruppe 27 - Fr, 2.11, 17:30 - 19:00, SG 203
- Gruppe 28 - Fr, 2.11, 16:45 - 18:15, 5054
- Gruppe 29 - Fr, 2.11, 16:30 - 18:00, Fk 9
- Gruppe 30 - Fr, 2.11, 17:30 - 19:00, RS 107

Aufgabe 1 (4 Punkte)

Bestimmen Sie, falls möglich, den Typ und das Ergebnis der folgenden Java Ausdrücke:

(1) `23 + 17`

(2) `(byte) 127 + 1`

(3) `'A' - 0`

(4) `(byte) (127 + 1)`

(5) `10d / 3f`

(6) `0 && true`

(7) `10 / 3`

(8) `"A" - 0`

(9) `3 > 2 ? "no" : "yes"`

(10) `10 / 3.`

(11) `(3 > 2) ? "no" : 1`

(12) `1 || 0`

Begründen Sie Ihre Antwort.

Hinweis: Der Typ des Ergebnisses des zweistelligen Operators `+` ist `int`, falls die Argumente vom Typ `byte`, `short` oder `int` sind. Dabei müssen nicht beide Argumente denselben Typ haben.

Aufgabe 2 (4 Punkte)

Schreiben Sie ein Java-Programm `Box`, das zwei ganze Zahlen `a` $\in \{3, ..., 80\}$ und `b` $\in \{3, ..., 50\}$ als Eingabe erwartet und ein Rechteck der Größe `a`×`b` ausgibt. Dabei wird der Rahmen des Rechtecks mit dem Zeichen `*` und das Innere des Rechtecks mit `o` gezeichnet. Bei ungültigen Eingaben soll das Programm eine entsprechende Meldung ausgeben.

Beispiel: Für `a` = 5 und `b` = 4 gibt das Programm

```
*****
*ooo*
*ooo*
*****
```

aus.

Aufgabe 3 (4 Punkte)

Schreiben Sie ein Java-Programm `Cross`, welches eine ganze Zahl `a` zwischen 1 und 80 als Eingabe erwartet und die Diagonalen des Quadrates mit der Seitenlänge `a` ausgibt, wobei die Diagonalen mit dem Zeichen `x` gezeichnet werden. Bei ungültigen Eingaben soll das Programm eine entsprechende Meldung ausgeben.

Beispiel: Für die Zahl 5 erfolgt als Ausgabe

```
x   x
 x x
  x
 x x
x   x
```

und für die Zahl 2

```
xx
xx
```

Aufgabe 4 (4 Punkte)

Die trigonometrische Funktion Cosinus hat die folgende Reihenentwicklung:

$$\cos(x) = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i}}{(2i)!}$$

d.h.

$$\cos(x) = (-1)^0 \frac{x^{2*0}}{(2*0)!} \; + \; (-1)^1 \frac{x^{2*1}}{(2*1)!} \; + \; (-1)^2 \frac{x^{2*2}}{(2*2)!} \; + \; (-1)^3 \frac{x^{2*3}}{(2*3)!} \; + \; \ldots$$

bzw.

$$\cos(x) = 1 \; - \; \frac{x^2}{2} \; + \; \frac{x^4}{24} \; - \; \frac{x^6}{720} \; + \; \ldots$$

Implementieren Sie die Funktion in Java. Dabei sollen zwei Zahlen `x` und `n` vom Benutzer eingegeben werden und Ihr Programm liefert den Wert der Reihenentwicklung von Cosinus für `x` durch Berechnung der ersten `n` Reihenglieder (Summanden der Reihe). Benutzen Sie die Methode `pow(a,b)` aus der Klasse `java.lang.Math`, für die $\texttt{pow(a,b)} = \texttt{a}^{\texttt{b}}$ gilt. Die Benutzung weiterer Bibliotheksfunktionen zur Berechnung des Ergebnisses ist nicht erlaubt.
Testen Sie Ihr Programm mit verschiedenen Eingabewerten und vergleichen Sie die Ergebnisse mit den Werten der Bibliotheksfunktion `java.lang.Math.cos`.

Aufgabe 5 (2 + 4 Punkte)

(a) Geben Sie die Darstellung der folgenden ganzen Zahlen im 6-Bit- und 8-Bit-Zweierkomplement an:

0, 1, -1, 8, -8, 32, -32

Welches ist die kleinste und welches die größte ganze Zahl, die im 6-Bit-Zweierkomplement dargestellt werden kann?

(b) Schreiben Sie ein Java Programm `Int2Bin`, das für eine eingegebene ganze Zahl ihre Darstellung im 8-Bit-Zweierkomplement ausgibt. Das Programm soll Zahlen, die aufgrund ihrer Größe nicht in 8-Bit-Zweierkomplement dargestellt werden können, erkennen und sich dabei sinnvoll verhalten (z.B. durch eine entsprechende Meldung).

Beispiel: Für die Zahl 54 liefert das Programm
`00110110`
als Ausgabe.

RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

LEHR- UND FORSCHUNGSGEBIET INFORMATIK II
RWTH Aachen · D-52056 Aachen · GERMANY
http://programmierung.informatik.rwth-aachen.de

Prof. Dr. Jürgen Giesl
Darius Dlugosz, Thomas v. d. Maßen, Antje Nowack

Übung *Informatik I - Programmierung* - Blatt 3

Die Übungsblätter sollen in Gruppen von je 3 Studierenden bearbeitet werden. Diese 3 Studierenden müssen aus derselben Übungsgruppe kommen. Lösungen können bis zum 13. November, 17.00 h, in den Kasten für Ihre Übungsgruppe im Foyer in der Ahornstr. 55 eingeworfen werden.
Bitte vergessen Sie nicht, die **Nummer Ihrer Übungsgruppe** sowie die **Namen** und **Matrikelnummer** der Mitglieder Ihrer 3er-Gruppe auf Ihre Abgabe zu schreiben.

Wichtig: Alle Programme sind **sowohl** auf Papier in den Kasten einzuwerfen **als auch** elektronisch mit Hilfe des **Online-Systems** abzugeben.

Aufgabe 1 (1 + 1 Punkte)

a) Kann jede while-Schleife in eine gleichwertige for-Schleife umgeschrieben werden? Wenn ja, geben Sie allgemein zu einer while-Schleife eine gleichwertige for-Schleife an. Wenn nein, belegen Sie dies durch ein Beispiel.

b) Kann jede while-Schleife in eine gleichwertige do-while-Schleife umgeschrieben werden? Wenn ja, geben Sie allgemein zu einer while-Schleife eine gleichwertige do-while-Schleife an. Wenn nein, belegen Sie dies durch ein Beispiel.

Aufgabe 2 (1 + 1 + 1 Punkte)

Schreiben Sie jeweils ein Java-Programm, das für eine Eingabe von $n \in \mathbb{N}$ die n-te Fibonacci-Zahl berechnet und dafür außer dem jeweils angegebenen Schleifentyp keinen anderen verwendet:

a) while-Schleife

b) for-Schleife

c) do-while-Schleife

Hinweis:
Die n-te Fibonacci-Zahl $\mathrm{fib}(n)$ $(n \geq 0)$ ist wie folgt definiert:

$$\begin{array}{lcll} \mathrm{fib}(0) & = & 0 & \\ \mathrm{fib}(1) & = & 1 & \\ \mathrm{fib}(n+2) & = & \mathrm{fib}(n+1) + \mathrm{fib}(n) & \text{für } n \geq 0 \end{array}$$

Aufgabe 3 (5 Punkte)

Schreiben Sie ein Java-Programm, welches das Mastermind Spiel simuliert. Dieses Spiel wird von einem Spieler und dem Computer nach folgenden Regeln gespielt:

a) Der Computer bestimmt eine zufällige positive ganze Dezimalzahl mit drei *verschiedenen* Ziffern $\in \{1, ..., 9\}$ (Geheimzahl), die der Spieler erraten muß. Für alle solche Zahlen ist die Wahrscheinlichkeit, daß sie vom Computer gewählt werden, jeweils gleich groß (d.h., die zufällige Auswahl ist gleichverteilt).

b) Der Spieler versucht nun, die Geheimzahl zu erraten, indem er in einer begrenzten Anzahl von Versuchen (maximal 10) dreiziffrige positive ganze Dezimalzahlen eingibt.

c) Nach jedem Versuch meldet ihm der Computer:

   (a) wieviele Ziffern an der richtigen Stelle stehen und

   (b) wieviele Ziffern in der Geheimzahl zwar enthalten sind, aber an einer anderen Stelle stehen

   Beispiel: Geheimzahl = 123, Rateversuch = 521
   $\Rightarrow$ 1 Ziffer an der richtigen Position,
   1 weitere Ziffer enthalten, aber an der falschen Stelle.

d) Das Spiel endet, sobald der Spieler die Geheimzahl erraten hat (gewonnen) oder nach dem 10. Rateversuch des Spielers (verloren).

Hinweise:

a) Berücksichtigen Sie dabei, daß keine Ziffer doppelt vorkommen darf! Die Zahlen 212 und 442 sind also keine gültigen Geheimzahlen.

b) Das folgende Java-Programm liefert eine zufällige ganze Zahl zwischen 1 und n (jeweils einschließlich). Zum Beispiel liefert `x = Zufall.Zufallszahl(5);` eine Zahl zwischen 1 und 5, welche in der Variablen `x` abgelegt wird. Die zufällige Auswahl ist dabei gleichverteilt.

```
import java.util.Random;

public class Zufall {

  public static int Zufallszahl (int n) {

    int ergebnis;
    // Erzeugt eine Zufallszahl
    Random ZufGen = new Random();

    // Projiziert die oben erzeugte Zufallszahl in den
    // Wertebereich {1, ..., n}
    ergebnis = Math.abs(ZufGen.nextInt() % n) + 1;

    return ergebnis;
  }
}
```

c) Das Spiel soll dann mittels einer do-while Schleife durchgeführt werden. Falls der Spieler sämtliche Ziffern richtig geraten hat, soll die do-while-Schleife mittels einer break-Anweisung verlassen werden.

d) Beim Rateversuch soll nicht überprüft werden, ob alle Ziffern verschieden sind.

e) Geben Sie das Ergebnis des Spielers aus, d.h., in wieviel Versuchen die Geheimzahl erraten wurde, oder ob sie nach zehn Versuchen nicht erraten wurde.

f) Geben Sie weiterhin im Fall, daß der Spieler verloren hat, die Geheimzahl aus.

Aufgabe 4 (1 + 3 + 3 + 3 Punkte)

a) Erweitern Sie den Hoare-Kalkül um Regeln zur Behandlung von do-while- und for-Schleifen, ohne (eventuell existierende) Transformationen von do-while bzw. for-Schleifen in while-Schleifen zu verwenden.

b) Welche der folgenden Zusicherungen sind beweisbar mit Hilfe des Hoare-Kalküls? Begründen Sie Ihre Antwort und geben Sie gegebenenfalls eine Herleitung im Hoare-Kalkül an.

(a) $< x = 0 >$ `x = x + 1;` $< x = 1 >$

(b) $< x = 0 >$ `x = x + 1;` $< x > 0 >$

(c) $<$ true $>$ `y = x + 1;` $< y > x >$

(d) $< x = 0 >$ `x = x + 1;` $< x = 0 >$

(e) $< x = 1 >$ `if (x > 0) x = x + 1;` $< x = 2 >$

(f) $<$ true $>$ `while (true) { x = x; }` $< x = 42 >$

c) Gegeben sei der folgende Algorithmus P zur Berechnung von $2 \cdot n$ (für $n \in \mathbb{N}$).

Algorithmus: P
Eingabe: `n`
Vorbedingung: `n` $\in \mathbb{N} = \{0, 1, 2, ...\}$

```
i = 0;
res = 0;
while (i < n) {
  res = res + 2;
  i = i + 1;
}
```

Nachbedingung: `res = 2 * n`
Ausgabe: `res`

(a) Beweisen Sie die partielle Korrektheit von P bezüglich der Vor- und Nachbedingung mit dem Hoare-Kalkül.

(b) Finden Sie eine geeignete Schleifeninvariante für die while-Schleife.

(c) Beweisen Sie die Terminierung durch Angabe einer geeigneten Variante.

d) Gegeben sei der folgende Algorithmus P zur Berechnung der Summe $\Sigma_{k=0}^{n} k^2$ (für $n \in \mathbb{N}$).

Algorithmus: P
Eingabe: `n`
Vorbedingung: `n` $\in \mathbb{N} = \{0, 1, 2, ...\}$

```
i = n;
res = 0;
while (i > 0) {
  j = i * i;
  res = res + j;
  i = i - 1;
}
```

Nachbedingung: `res` $= \Sigma_{k=0}^{n} k^2$
Ausgabe: `res`

(a) Beweisen Sie die partielle Korrektheit von P bezüglich der Vor- und Nachbedingung mit dem Hoare-Kalkül.

(b) Finden Sie eine geeignete Schleifeninvariante für die while-Schleife.

(c) Beweisen Sie die Terminierung durch Angabe einer geeigneten Variante.

RHEINISCH-WESTFÄLISCHE TECHNISCHE HOCHSCHULE AACHEN

LEHR- UND FORSCHUNGSGEBIET INFORMATIK II
RWTH Aachen · D-52056 Aachen · GERMANY
http://programmierung.informatik.rwth-aachen.de

Prof. Dr. Jürgen Giesl
Darius Dlugosz, Thomas v. d. Maßen, Antje Nowack

Übung *Informatik I - Programmierung* - Blatt 4

Die Übungsblätter sollen in Gruppen von je 3 Studierenden bearbeitet werden. Diese 3 Studierenden müssen aus derselben Übungsgruppe kommen. Lösungen können bis zum 20. November, 17.45 h, in den Kasten für Ihre Übungsgruppe im Foyer in der Ahornstr. 55 eingeworfen werden.
Bitte vergessen Sie nicht, die **Nummer Ihrer Übungsgruppe** sowie die **Namen** und **Matrikelnummer** der Mitglieder Ihrer 3er-Gruppe auf Ihre Abgabe zu schreiben.

Wichtig: Alle Programme sind **sowohl** auf Papier in den Kasten einzuwerfen **als auch** elektronisch mit Hilfe des **Online-Systems** abzugeben.

Aufgabe 1 (3 + 1 + 1 Punkte)

Gegeben sei der folgende Algorithmus P, welcher aus einem gegebenen Array A, welches natürliche Zahlen enthält, die Summe aller Elemente berechnet.

Algorithmus: P
Eingabe: `int[] a`
Vorbedingung: `a.length = n`

```
j = 0;
sum = 0;
while (j < n) {
    sum = sum + a[j];
    j = j + 1;
}
```

Nachbedingung: $sum = \sum_{i=0}^{a.length-1} a[i]$
Ausgabe: sum

a) Beweisen Sie die partielle Korrektheit von P bezüglich der Vor- und Nachbedingung mit dem Hoare-Kalkül.

b) Geben Sie die Schleifeninvariante für die `while`-Schleife an.

c) Beweisen Sie die Terminierung durch Angabe einer geeigneten Variante.

Aufgabe 2 (3 + 2 Punkte)

Ein Bruch b läßt sich mit Hilfe eines Arrays d in seiner Dezimalform mit k Nachkommastellen wie folgt darstellen:

$$b = \sum_{i=0}^{k-1} d[i] * 10^{-i-1}$$

a) Realisieren Sie ein Programm in Java, welches dem Benutzer gestattet, einen **echten** Bruch b in seiner Dezimalform mit zehn Nachkommastellen einzugeben. Die einzelnen Stellen sollen in einem Array gespeichert werden. Zusätzlich soll eine positive, ganze Zahl x eingelesen werden. Das Programm soll den Bruch $\frac{b}{x}$ berechnen und das Ergebnis in einem Array speichern. Dabei sollen nur die ersten zehn Nachkommastellen berücksichtigt werden. Das Ergebnis soll schließlich auf dem Bildschirm ausgegeben werden.
Beispiel:
Die Darstellung von $b = \frac{1}{64}$ in einem Array:

0	1	5	6	2	5	0	0	0	0

Teilt man diesen Bruch nun durch 8, so sieht das Ergebnis-Array wie folgt aus:

0	0	1	9	5	3	1	2	5	0

b) Implementieren Sie ein Programm in Java, welches die ersten 10 Nachkommastellen der negativen Potenzen $x^{-1}, x^{-2}, ..., x^{-10}$ einer durch den Benutzer einzugebenden, positiven, ganzen Zahl x berechnet und auf dem Bildschirm ausgibt. Verwenden Sie dazu die Array-Darstellung sowie die Routine zur Berechnung der Division eines Bruchs durch eine positive ganze Zahl aus Aufgabenteil a).

Aufgabe 3 (3 Punkte)

In dieser Aufgabe soll ein Programm realisiert werden, welches in einem beliebigen Satz die Vokale nach dem folgenden Muster ersetzt (dieses gilt sowohl für Klein- als auch für Grossbuchstaben): $a \rightarrow e, e \rightarrow i, i \rightarrow o, o \rightarrow u, u \rightarrow a$. Die einzelnen Wörter des Satzes sollen als Kommandozeilenargumente der `main`-Methode übergeben werden.
Beispiel:
Durch den Aufruf
`java Vokalersetzung In diesem Satz werden die Vokale vertauscht.` wird das folgende Array erzeugt:

In	diesem	Satz	werden	die	Vokale	vertauscht.

Nach Vertauschung der Vokale wird das folgende Ergebnis-Array auf dem Bildschirm ausgegeben:

On	doisim	Setz	wirdin	doi	Vukeli	virteascht.

Aufgabe 4 (4 Punkte)

Sei $A \in M^{m,n}$ eine $m \times n$ - Matrix und B $\in M^{n,p}$ eine $n \times p$ - Matrix. Das Produkt zweier solcher Matrizen läßt sich wie folgt berechnen:

$A \cdot B = C \in M^{m,p}$ mit

$$C = \begin{pmatrix} c_{1,1} & \cdots & c_{1,p} \\ \vdots & \ddots & \vdots \\ c_{m,1} & \cdots & c_{m,p} \end{pmatrix}$$

wobei $1 < i < m$, $1 < s < p$ und $c_{i,s} = \sum_{j=1}^{n} a_{ij} b_{js}$
Zur Berechnung von $c_{i,s}$ wird also jedes Element der i-ten Zeile von A mit dem entsprechenden Element der s-ten Spalte von B multipliziert und über die entstehenden n Produkte summiert. Beispiel:

$$\begin{pmatrix} 2 & 1 & 4 \\ 1 & -1 & 3 \end{pmatrix} \cdot \begin{pmatrix} 4 & -5 \\ -2 & -3 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 10 & -9 \\ 9 & 1 \end{pmatrix}$$

Realisieren Sie ein Programm in Java, welches das Produkt zweier Matrizen A und B berechnet und auf dem Bildschirm ausgibt. Dazu sollen die Dimensionen und die Elemente der Matrizen A und B nach dem Start des Programms durch den Benutzer eingegeben werden können.

Aufgabe 5 (4 Punkte)

Implementieren Sie in Java eine Klasse *Kugel*. Eine Kugel soll einen Radius, eine Farbe und ein Gewicht besitzen. Realisieren Sie ebenfalls Methoden, die den Durchmesser, den Umfang, die Oberfläche und das Volumen einer Kugel liefern. Testen Sie die Klasse Kugel mit einem Hauptprogramm. Erzeugen Sie folgende Kugelobjekte durch Zuweisung der folgenden Attributausprägungen:

- Kugel 1: Radius = 10; Farbe = blau; Gewicht = 56;
- Kugel 2: Radius = 7.5; Farbe = rot; Gewicht = 12;
- Kugel 3: Radius = 23; Farbe = gelb; Gewicht = 115;

Berechnen Sie mit Hilfe der definierten Methoden Umfang, Durchmesser, Oberfläche und Volumen der drei Kugeln und geben Sie diese auf dem Bildschirm aus.

RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

LEHR- UND FORSCHUNGSGEBIET INFORMATIK II
RWTH Aachen · D-52056 Aachen · GERMANY
http://programmierung.informatik.rwth-aachen.de

Prof. Dr. Jürgen Giesl
Darius Dlugosz, Thomas v. d. Maßen, Antje Nowack

Übung *Informatik I - Programmierung* - Blatt 5

Die Übungsblätter sollen in Gruppen von je 3 Studierenden bearbeitet werden. Diese 3 Studierenden müssen aus derselben Übungsgruppe kommen. Lösungen können bis zum 27. November, 17.45 h, in den Kasten für Ihre Übungsgruppe im Foyer in der Ahornstr. 55 eingeworfen werden.
Bitte vergessen Sie nicht, die **Nummer Ihrer Übungsgruppe** sowie die **Namen** und **Matrikelnummer** der Mitglieder Ihrer 3er-Gruppe auf Ihre Abgabe zu schreiben.

Wichtig: Alle Programme sind **sowohl** auf Papier in den Kasten einzuwerfen **als auch** elektronisch mit Hilfe des **Online-Systems** abzugeben.

Aufgabe 1 (3 Punkte)

Realisieren Sie ein Programm in Java, welches eine positive, ganze Dezimalzahl in die römische Zahlendarstellung konvertiert und auf dem Bildschirm ausgibt. Die Dezimalzahl soll dabei durch den Benutzer eingegeben werden können. Für die römischen Zahlen gelte die übliche Darstellung mit den folgenden Ziffern:

Römische Ziffer	I	V	X	L	C	D	M
Dezimalwert	1	5	10	50	100	500	1000

Die hier betrachteten römischen Zahlen sollen keine Abkürzungen, wie z.B. IX für die Dezimalzahl 9 oder MCM für die Dezimalzahl 1900 verwenden. Beispiele:

Dezimal	1	2	3	4	5	6	7	8	9	10	11	12	15	21	563
Römisch	I	II	III	IIII	V	VI	VII	VIII	VIIII	X	XI	XII	XV	XXI	DLXIII

Implementieren Sie geeignete Methoden zur Eingabe der darzustellenden Dezimalzahl und zur Berechnung und Ausgabe der römischen Zahl.

Aufgabe 2 (3 + 2 Punkte)

a) Gegeben sei das folgende Java-Programm, wobei die Klasse Kugel bereits in Aufgabe 5 des 4. Übungsblattes definiert wurde.

```
public class Referenzen {

   static int a = 15;
   static int radius = 5;
   static Kugel k1, k2;

   public static void Mystery(Kugel k2, Kugel k1) {
      Kugel k3;
      byte a = 100;
      k2.farbe = "orange";
      /* *** MARKIERUNG *** */
      k2 = k1;
      /* MARKIERUNG */
      k1.gewicht = a;
      k2.gewicht = radius;
      /* *** MARKIERUNG *** */
      k3 = new Kugel();
      k2 = k3;
      /* *** MARKIERUNG *** */
   }

   public static void main (String[] args) {
      radius = 20;
      k1 = new Kugel();
      k2 = new Kugel();
      Kugel k3 = new Kugel();

      /* *** MARKIERUNG *** */

      k1.farbe = "rot";
      k1.radius = 10;
      k1.gewicht = 50;
      k2.farbe = "blau";
      k2.radius = 11;
      k2.gewicht = 60;
      k3.farbe = "gelb";
      k3.radius = 12;
      k3.gewicht = 70;
      /* *** MARKIERUNG *** */
      k1.farbe = k2.farbe;
      /* *** MARKIERUNG *** */
      k2.farbe = "gelb";
      k2.radius = radius;
      /* *** MARKIERUNG *** */

      Mystery(k2, k3);

      radius = a;
```

```
        k2 = k1;
        /* *** MARKIERUNG *** */
        k1 = new Kugel();
        /* *** MARKIERUNG *** */
    }
}
```

Visualisieren Sie den Ablauf des Programms, indem Sie die Zustände des Speichers und der Objekte (das heißt die Ausprägungen ihrer Attribute) an jeder markierten Stelle des Programms in einer neuen Grafik darstellen. Die grafische Repräsentation soll an die in der Vorlesung verwendeten Notation angelehnt sein.

b) Nun soll die Klasse *Kugel* aus der Übungsaufgabe 5 des 4. Übungsblattes um Prinzipien der Datenabstraktion erweitert werden. Dazu ist es nötig, die Attribute zu kapseln und geeignete Zugriffsoperationen (Selektoren) zu definieren, um auf die Attribute zuzugreifen. Realisieren Sie ebenfalls eine Methode `toString()`, um die Ausprägungen der Attribute einer Kugel auf dem Bildschirm darzustellen.

Aufgabe 3 (2 + 3 + 3 Punkte)

In dieser Aufgabe soll eine Adressverwaltung in Java realisiert werden. Dazu soll eine Klasse *Adresse* implementiert werden, welche die Attribute einer Adresse kapselt. Desweiteren soll eine Klasse *Adressenverwaltung* realisiert werden, welche es ermöglicht, Adressen zu speichern.

a) Implementieren Sie die Klasse *Adresse*. Eine Adresse zeichnet sich durch die folgenden Attribute aus:

- Name
- Vorname
- Straße
- Postleitzahl
- Ort

Zudem werden geeignete Methoden benötigt, um auf die oben angegebenen Attribute zuzugreifen und die enthaltenen Daten in einen String für die Ausgabe auf dem Bildschirm zu wandeln(`toString()`-Methode).

b) Implementieren Sie die Klasse *Adressendatenbank*. Diese soll in der Lage sein, maximal 20 Adressen zu speichern. Ebenso soll die Klasse Methoden bereitstellen, welche es ermöglichen, nach einer Adresse zu suchen, Adressen hinzuzufügen, zu löschen und zu ändern sowie alle gespeicherten Adressen anzuzeigen.

c) Realisieren Sie ein geeignetes Hauptprogramm, welches die Adressverwaltung steuert. Der Benutzer soll dabei aus einem Menü die folgenden Operationen auswählen können:

- Adresse hinzufügen
- Adresse löschen
- Adresse ändern
- Adresse zu einem vorgegebenen Namen suchen (Beachten Sie, dass es mehrere Personen mit dem gleichen Namen geben kann, die natürlich alle aufgelistet werden sollen). Ein Vergleich auf Strings kann mit der Methode `equals(String)` durchgeführt werden. Beispiel: `if (text1.equals(text2)) { ... }`
- Alle Adressen auflisten

Hinweis: Für das Einlesen der Adressdaten wird eine Operation benötigt, um Zeichenketten einzulesen. Auf der Internet-Seite *http://programmierung.informatik.rwth-aachen.de* steht dafür eine aktualisierte Version der Klasse *IO* zur Verfügung, die nun auch eine Methode `public static String TextEingabe()` bereitstellt.

Aufgabe 4 (6 + 3 Punkte)

Herr Müller möchte seine Girokonten elektronisch verwalten. Da er vorhandenen Banking-Programmen nicht traut, beschließt er, selbst ein objektorientiertes Programm dafür zu entwickeln. Nach eingehender Analyse findet er heraus, dass er die folgenden Klassen dafür implementieren muß: *Konto, Buchung* und *Datum*. Ein Konto besitzt neben einem Inhaber, einer Kontonummer und einer Bankleitzahl ebenfalls eine Buchungsliste, in der maximal 100 Buchungen, nach Datum sortiert, gespeichert werden können.
Eine Buchung besteht aus einem Beschreibungstext, einem Betrag und einem Datum. Ein Datum zeichnet sich durch die Angabe des Tags, des Monats und des Jahres aus.

a) Implementieren Sie die Klassen *Konto, Buchung* und *Datum* und überlegen Sie sich geeignete Operationen für diese Klassen.
Hinweis: Zum Sortieren der Buchungen kann der in der Vorlesung vorgestellte Sortieralgorithmus verwendet werden. Dazu ist es sinnvoll, eine Vergleichsoperation zu definieren, die prüft, ob ein Datum größer als ein anderes Datum ist.

b) Realisieren Sie ein geeignetes Hauptprogramm, welches es ermöglicht, maximal 10 Konten zu eröffnen und deren Buchungen zu verwalten, das heißt:

   - ein neues Konto kann erzeugt werden
   - das aktive Konto kann gewählt werden
   - neue und auch ältere Buchungen können einem Konto hinzugefügt werden
   - der Saldo und alle Buchungen eines Kontos können sortiert ausgegeben werden
   - die Anzahl aller verwalteten Konten kann abgefragt werden

   indem dem Benutzer ein entsprechendes Menü zur Verfügung gestellt wird.

Prof. Dr. Jürgen Giesl
Darius Dlugosz, Thomas v. d. Maßen, Antje Nowack

Übung *Informatik I - Programmierung* - Blatt 6

Die Übungsblätter sollen in Gruppen von je 3 Studierenden bearbeitet werden. Diese 3 Studierenden müssen aus derselben Übungsgruppe kommen. Lösungen können bis zum 4. Dezember, 17.45 h, in den Kasten für Ihre Übungsgruppe im Foyer in der Ahornstr. 55 eingeworfen werden.
Bitte vergessen Sie nicht, die **Nummer Ihrer Übungsgruppe** sowie die **Namen** und **Matrikelnummer** der Mitglieder Ihrer 3er-Gruppe auf Ihre Abgabe zu schreiben.

Wichtig: Alle Programme sind **sowohl** auf Papier in den Kasten einzuwerfen **als auch** elektronisch mit Hilfe des **Online-Systems** abzugeben.

Aufgabe 1 (3 + 1 + 2)

Gegeben sei der folgende Algorithmus P zur Berechnung des Produkts von zwei natürlichen Zahlen.
Algorithmus: P
Eingabe: `x, y` $\in \mathbb{N} = \{0, 1, 2, ...\}$
Vorbedingung: $x \geq 0 \wedge y \geq 0$

```
res = 0;
for (int i = 0 ; i < x ; i = i+1) {
  for (int j = 0 ; j < y ; j = j+1) {
    res = res + 1;
  }
}
```

Nachbedingung: `res = x * y`
Ausgabe: `res`

a) Beweisen Sie die partielle Korrektheit von P bezüglich der Vor- und Nachbedingung mit dem Hoare-Kalkül.

b) Finden Sie geeignete Schleifeninvarianten für die for-Schleifen.

c) In der Vorlesung wurde gezeigt, wie man durch Angabe einer geeigneten Variante die Terminierung einer while-Schleife nachweisen kann. Wie läßt sich diese Vorgehensweise generell auf for-Schleifen übertragen?
Führen Sie die Vorgehensweise für das obige Beispiel durch.

Hinweis:
Verwenden Sie zur Lösung dieser Aufgabe folgende Regel für for-Schleifen:

$$\frac{<\psi> I <\varphi> \qquad\qquad <\varphi \wedge B> P\ F <\varphi>}{<\psi> \texttt{ for } (I; B; F)\ \{P\} <\varphi \wedge \neg B>}$$

Aufgabe 2 (2 + 1)

Gegeben seien folgende Methoden:

1. `f()`
2. `f(int x)`
3. `f(boolean x)`
4. `f(double x, int y)`
5. `f(int x, double y)`
6. `f(int x, int y)`
7. `f(double x, double y)`

a) Ist die obige Überladung erlaubt? Begründen Sie Ihre Antwort. Wenn die Überladung nicht zulässig ist, so ändern Sie sie derart ab, daß sie zulässig wird.

b) Geben Sie jeweils an, ob eine der obigen Methoden ausgeführt werden kann, und wenn dies möglich ist, welche Methode ausgeführt wird. Falls sie in a) eine Modifikation angegeben haben, verwenden Sie statt des obigen Systems Ihre Modifikation in diesem Aufgabenteil. Begründen Sie jeweils Ihre Antworten.

   i. `f(3.0)`
   ii. `f(2,3)`
   iii. `f(true, 3)`
   iv. `f(true)`
   v. `f(2, 3.0)`
   vi. `f(2.0, 3)`

Aufgabe 3 (1 + 4 + 1 + 2 + 1)

In dieser Aufgabe geht es um eine Realisierung des VierGewinnt-Spiels.
Die Regeln des Spiels sind folgende:
Das Spiel wird von zwei Spielern auf einem $(n \times n)$-Spielfeld ($n \in \mathbb{N}$) gespielt. Abwechselnd werden Spielsteine (Spieler 1 hat Steine der Farbe 1, und Spieler 2 hat Steine der Farbe 2) in eine Spalte des Spielfelds "eingeworfen", welche der aktuelle Spieler wählt. Die Steine "fallen" immer nach unten durch. Gewonnen hat der Spieler, der als erster mindestens vier Steine der eigenen Farbe in einer Reihe (horizontal, vertikal oder diagonal) abgelegt hat. Es kann gewählt werden, welcher Spieler (1 oder 2) beginnt.

a) Entwerfen Sie die Schnittstelle für einen abstrakten Datentyp *Spielkonfiguration*, welcher eine Konfiguration bzw. einen Zustand des Spiels kapselt. Eine Spielkonfiguration zeichnet sich aus durch:

- den Spieler, der am Zug ist
- den Zustand des Spielfeldes

Weiterhin sollen geeignete Methoden für folgende Aufgaben öffentlich zur Verfügung gestellt werden:

- Initialisierung durch einen Konstruktor mit zwei Parametern.
  Der erste Parameter gibt an, welcher Spieler in der Spielkonfiguration der aktuelle ist. Der zweite gibt die Höhe des Spielbrettes an. Bei der Initialisierung soll das Spielbrett leer (d.h., ohne Steine) sein.
- Überführung der Spielkonfiguration in einen String für die Ausgabe auf dem Bildschirm (toString()-Methode).
- Zugriff auf den aktuellen Spieler.
- Ausführen eines Spielzuges (mit einem Parameter zur Angabe der gewählten Spalte, in die der Stein fallen soll) und somit Modifikation der Spielkonfiguration.
- Überprüfen, ob der Einwurf des letzten Steins in eine Spalte zu einem Gewinn geführt hat (die Spaltennummer wird als Parameter übergeben).
- Überprüfen, ob ein Stein in eine Spalte (die Nummer der Spalte muß angegeben werden) geworfen werden kann (Spielbarkeit).
- Überprüfen, ob das Spielfeld voll ist (alle Positionen belegt).

Weitere Methoden dürfen nach außen nicht sichtbar sein (aber natürlich "privat" realisiert werden).

b) Implementieren Sie den Datentyp *Spielkonfiguration*, dessen Schnittstelle Sie in a) entworfen haben.

c) Erstellen Sie mit Hilfe von javadoc eine Schnittstellendokumentation für den Datentyp *Spielkonfiguration*.

d) Implementieren Sie die Klasse *Spiel*, die nur die Methode `Spiel(int Spieler, int n)` enthält. Diese Methode soll es ermöglichen, ein VierGewinnt-Spiel aufzurufen. Der Spieler, der der Zahl des ersten Parameters entspricht, soll beginnen. Gespielt wird auf einem $(n \times n)$-Spielbrett. Die Spieler geben abwechselnd ihre Züge ein.

e) Implementieren Sie eine Klasse *VierGewinnt*, die nur ein Hauptprogramm enthält, welches zunächst nach dem Spieler fragt, der beginnen soll, und dann unter Verwendung der Methode `Spiel` aus der Klasse *Spiel* ein Spiel durchführt.

Aufgabe 4 (1 + 2 + 1 + 2 + 1 + 2)

a) Entwerfen Sie die Schnittstelle für einen abstrakten Datentyp *Komplex* zum Rechnen mit komplexen Zahlen, deren Real- und Imaginärteil jeweils als double-Zahl repräsentiert sind. Hierbei soll folgendes Konzept für *Komplex* zugrunde liegen:

- Real- und Imaginärteil können gesetzt werden
- Real- und Imaginärteil können abgefragt werden
- der Betrag einer komplexen Zahl kann berechnet werden

Attribute dürfen in diesem Aufgabenteil *nur primitive Datentypen* besitzen.

b) Implementieren Sie den Datentyp *Komplex*, dessen Schnittstelle Sie in a) entworfen haben.

c) Erstellen Sie mit Hilfe von javadoc eine Schnittstellendokumentation für den Datentyp *Komplex*.

d) Implementieren Sie eine weitere Klasse *Komplexe_Operationen*, die die folgenden Operationen realisiert:

- Addition zweier komplexer Zahlen
- Subtraktion zweier komplexer Zahlen
- Multiplikation zweier komplexer Zahlen
- Berechnung der konjugiert komplexen Zahl zu einer gegebenen komplexen Zahl

e) Schreiben Sie ein geeignetes Hauptprogramm, welches die Eingabe von zwei komplexen Zahlen ermöglicht und zu jeder der beiden Zahlen ihren Betrag, die konjugiert komplexe Zahl sowie die Summe, die Differenz und das Produkt der beiden Zahlen ausgibt.

f) Implementieren Sie einen zweiten abstrakten Datentyp *Komplex*, welcher im Unterschied zur Implementierung in Teil a) *nur ein Attribut* besitzt. Dieser Datentyp darf sich nach außen nicht vom ersten unterscheiden, d.h., die Klasse *Komplexe_Operationen* und Ihr Hauptprogramm müssen auch mit der veränderten Implementierung von *Komplex* arbeiten.
Erstellen Sie wiederum eine Schnittstellendokumentation mit Hilfe von javadoc. Beachten Sie, daß diese Schnittstellendokumentation und diejenige aus Teil c) bei einer korrekten Lösung der Aufgabe identisch sind.

Hinweis:
Seien $u := a + bi$ und $v := c + di$ zwei komplexe Zahlen. Dann gilt

$$
\begin{array}{lcll}
u + v & := & (a + c) + (b + d)i & \\
u - v & := & (a - c) + (b - d)i & \\
u \cdot v & := & (ac - bd) + (ad + bc)i & \\
u^* & := & a - bi & \text{(wobei } u^* \text{ die konjugiert komplexe Zahl zu } u \text{ ist)} \\
|u| & := & \sqrt{a^2 + b^2} & \text{(wobei } |u| \text{ der Betrag von } u \text{ ist)}
\end{array}
$$

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
Darius Dlugosz, Thomas v. d. Maßen, Antje Nowack

Übung *Informatik I - Programmierung* - Blatt 7

Die Übungsblätter sollen in Gruppen von je 3 Studierenden bearbeitet werden. Diese 3 Studierenden müssen aus derselben Übungsgruppe kommen. Lösungen können bis zum Dienstag, den 11. Dezember, 17.45 h, in den Kasten für Ihre Übungsgruppe im Foyer (neben Aula 2) in der Ahornstr. 55 eingeworfen werden.
Bitte vergessen Sie nicht, die **Nummer Ihrer Übungsgruppe** sowie die **Namen** und **Matrikelnummer** der Mitglieder Ihrer 3er-Gruppe auf Ihre Abgabe zu schreiben.

Wichtig: Alle Programme sind **sowohl** auf **Papier** in den Kasten einzuwerfen **als auch** elektronisch mit Hilfe des **Online-Systems** abzugeben.

Aufgabe 1 (3 Punkte)

Schreiben Sie ein Java-Programm, das die n-te Fibonacci-Zahl mit ***linearer*** Rekursion berechnet. Das Programm darf keine Schleifen und auch keine Sprünge enthalten, gegenseitige Aufrufe mehrerer Methoden (verschränkte Rekursion) sind ebenfalls nicht erlaubt.
Lineare Rekursion bedeutet, dass jede Ausführung einer Methode höchstens zu einem rekursiven Aufruf führt (dessen Ausführung kann natürlich erneut zu einem rekursiven Aufruf führen).

Zur Erinnerung: Die n-te Fibonacci-Zahl $fib(n)$ ist wie folgt definiert:

$$fib(n) = \begin{cases} 0 & , falls\ n < 1 \\ 1 & , falls\ n = 1 \\ fib(n-1) + fib(n-2) & , falls\ n > 1 \end{cases}$$

Hinweis: Überlegen Sie, wie $fib(n)$ und $fib(n-1)$ in "einem" Schritt (und nur mit einem rekursiven Aufruf) berechnet werden können, wenn der Wert von $fib(n-2)$ bereits feststeht. Insbesondere sind $fib(0)$ und $fib(1)$ von Beginn an bekannt.

Aufgabe 2 ((1+1) + (1+1) + (2+1) Punkte)

Überführen Sie die folgenden Algorithmen jeweils in eine *endrekursive* und eine *iterative* Version, die die Berechnung auf analoge Weise durchführen (insbesondere müssen dieselbe Fallunterscheidungen verwendet werden).

(a)
```
public static int foo(int x, int y) {
  if (x == 0) return y;
```

```
  else return 1 + foo(x-1, y);
}
```

(b)
```
public static int boo(int x, int y) {
  if (x == 0) return 0;
  else {
    if ((x % 2) == 0) return boo(x/2 , 2*y);
    else return y + boo((x-1)/2, 2*y);
  }
}
```

(c)
```
public static int rham(int n) {
  if (n == 1) return 1;
  else if (n > 1) {
    if ((n % 2) == 0) return rham(n/2);
    else return rham((n-1)/2) + rham((n+1)/2);
  }
  else return 0;
}
```

(Die Methode `rham` wird auch als Rham's Funktion bezeichnet.)

Hinweis: Bei der Transformation in die endrekursive Version darf sich die Anzahl der formalen Parameter ändern.

Aufgabe 3 (6 Punkte)

In dieser Aufgabe soll das *m-beschränkte n-Damen Problem* algorithmisch gelöst werden.
Problembeschreibung: Auf einem $a \times a$-Schachbrett sollen n Damen so platziert werden, dass sie sich gegenseitig nicht schlagen können. Bekanntlich kann eine Dame eine andere Figur schlagen, wenn sie dieselbe horizontale oder vertikale Position hat oder wenn sie in derselben Diagonalen steht. Die Beschränkung besteht darin, dass jede Dame sich maximal m Felder von ihrer aktuellen Position bewegen kann, d. h. das auch zwei (und mehr) Damen in derselben horizontalen oder vertikalen Linie oder derselben Diagonalen stehen dürfen, wenn zwischen ihnen mindestens m Felder frei sind.

Ein Lösungsansatz, den N. Wirth vorschlägt, beruht auf einer Technik, die *Backtracking* (Zurücksetzen) genannt wird. Hierbei handelt es sich um rekursive Algorithmen, die nach Lösungen suchen, indem sie Kandidaten nach einer Strategie erzeugen, testen und falls sie keine geeigneten Kandidaten finden können, zu einem früheren Zustand der Berechnung zurückkehren, um

dort weitere Kandidaten zu erzeugen und zu testen.

Implementieren Sie ein Java-Programm, das für die Eingaben a, m und n eine Lösung für das m-beschränkte n-Damen Problem mit Hilfe von Backtracking rekursiv berechnet und auf dem Bildschirm ausgibt (z. B. in Form von Positionen, auf denen die Damen platziert werden).

Hinweis: Beim (m-beschränkten) n-Damen-Problem geht man also für $k < n$ so vor, dass die Position der $(k + 1)$-ten Dame betrachtet wird, falls schon k Damen korrekt platziert sind. Existiert keine geeignete Position für die $(k + 1)$-te Dame, so muss die Position der k-ten Dame zurückgesetzt und neu ermittelt werden.

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
Darius Dlugosz, Thomas v. d. Maßen, Antje Nowack

Übung *Informatik I - Programmierung* - Blatt 8

Die Übungsblätter sollen in Gruppen von je 3 Studierenden bearbeitet werden. Diese 3 Studierenden müssen aus derselben Übungsgruppe kommen. Lösungen können bis zum Dienstag, den 18. Dezember, 17.45 h, in den Kasten für Ihre Übungsgruppe im Foyer (neben Aula 2) in der Ahornstr. 55 eingeworfen werden.
Bitte vergessen Sie nicht, die **Nummer Ihrer Übungsgruppe** sowie die **Namen** und **Matrikelnummer** der Mitglieder Ihrer 3er-Gruppe auf Ihre Abgabe zu schreiben.

Wichtig: Alle Programme sind **sowohl** auf **Papier** in den Kasten einzuwerfen **als auch** elektronisch mit Hilfe des **Online-Systems** abzugeben.

Aufgabe 1 (5 Punkte)

Implementieren Sie in Java eine Datenstruktur `Zeichenkette`, die eine endliche, jedoch beliebig lange Kette von Elementen des Typs `char` repräsentiert. Zu einer gegebenen Zeichenkette kann ein weiteres Zeichen hinzugefügt werden. Die Reihenfolge, in der die Zeichen hinzugefügt werden, bleibt in der Struktur der Zeichenkette erhalten. Die Datenstruktur soll dabei rekursiv sein und folgende Methoden zur Verfügung stellen:

- `public void add(char zeichen)`
  fügt am Ende der Zeichenkette das Zeichen `zeichen` ein
- `public int laenge()`
  liefert die Länge der Zeichenkette
- `public boolean beginntMit(Zeichenkette prefix)`
  liefert `true`, falls die Zeichenkette mit der Zeichenkette `prefix` beginnt, `false` sonst
- `public boolean endetMit(Zeichenkette suffix)`
  liefert `true`, falls die Zeichenkette mit der Zeichenkette `suffix` endet, `false` sonst
- `public boolean enthaelt(Zeichenkette zeichenfolge)`
  liefert `true`, falls `zeichenfolge` in der aktuellen Zeichenkette enthalten ist, `false` sonst (beispielweise ist *ei* in k*ei*l enthalten, aber nicht in *er*innerung)

Keine der Methoden von `Zeichenkette` darf Schleifen oder Sprünge enthalten. Hierbei ist die Verwendung von Arrays und jeglicher Java-Bibliotheksmethoden nicht erlaubt!

Aufgabe 2 (7 + 2 Punkte)

Die rekursive Datenstruktur *Baum* kann wie folgt beschrieben werden. Ein Baum besteht aus *Knoten*. Ein Knoten A besitzt eine endliche Anzahl von *Kinderknoten* (Nachfolger von A). Bis auf einen Knoten hat jeder Knoten des Baumes genau einen (unmittelbaren) *Vorgänger(knoten)*. Der Knoten ohne Vorgänger wird die *Wurzel* des Baumes genannt. Die Knoten ohne Nachfolger sind die *Blätter* des Baumes.

Diese Datenstruktur soll nun dazu verwendet werden, den Abstammungsbaum der Nachkommen von Noah nachzubilden. Der Abstammungsbaum ist in Genesis 10:1-25 oder auf der letzten Seite des Übungsblattes zu finden.

(a) Implementieren Sie in Java die rekursive Datenstruktur Baum. Jeder Knoten des Baumes kann dabei eine *beliebige* Anzahl von Kinderknoten haben. Die Knoten des Baumes besitzen unter anderem ein Attribut für den Namen einer Person. Die Klasse, die den Baum repräsentiert, soll die folgende Methode enthalten:

- `public void add(String parent, String child)`
  `parent` ist der Name einer Person, `child` der Name seines Kindes.
  Befindet sich im Baum ein Knoten A mit dem Namen `parent`, so wird ein neuer Knoten mit dem Namen des Kindes zur Kinderknoten von A hinzugefügt, sonst bleibt der Baum unverändert. Für den Fall, dass mehrere Knoten des Baumes den Namen `parent` besitzen, soll das Suchen des Knotens A wie in Abbildung 1 dargestellt geschehen (Abb.1 zeigt das Suchverfahren an einem Beispiel; bei anderen Bäumen arbeitet es analog). Hierbei geben die Nummern die Reihenfolge an, in der die Knoten nach dem Namen `parent` durchsucht werden. Der erste Knoten, der bei dieser Suche den Namen `parent` enthält, ist der gesuchte Knoten.

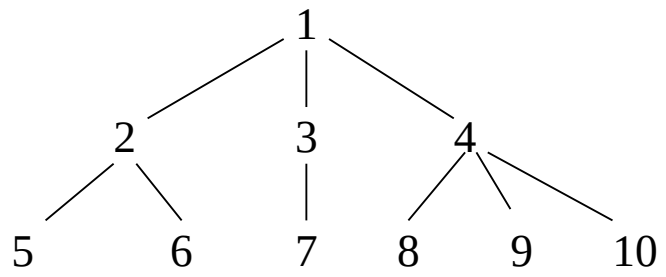


Abbildung 1. *(Knoten 1 ist die Wurzel des Baumes, Knoten 2, 3, 4 sind die Nachfolger des Knotens 1, Knoten 5 und 6 sind die Nachfolger von 2, usw.)*

Weiterhin soll die Klasse einen Konstruktor zur Verfügung stellen, der mit dem Namen einer Person aufgerufen wird und einen Baum erzeugt, dessen Wurzel mit diesem Namen beschriftet ist und der außer der Wurzel keinen weiteren Knoten enthält.

(b) Implementieren Sie für die Datenstruktur aus (a) zusätzlich folgende Methode. Die Methode soll rekursiv sein.

- `public void print()`
  gibt, beginnend mit der Wurzel, den gesamten Baum auf dem Bildschirm in folgender Form aus:

```
A
  B_1
    C_1
      ... (Nachfolger von C_1)
    ... (Nachfolger C_2 bis C_(m-1) von B_1)
    C_m
  B_2
  ... (weitere Nachfolger B_i von A samt ihren Nachfolger)
  B_n
   ...
```

  wobei `A` die Beschriftung der Wurzel des Baumes ist, `B_1`, `B_2`, ..., `B_n` diejenigen der Nachfolger von `A`, und `C_1`, ..., `C_m` die der Nachfolger von `B_1`.

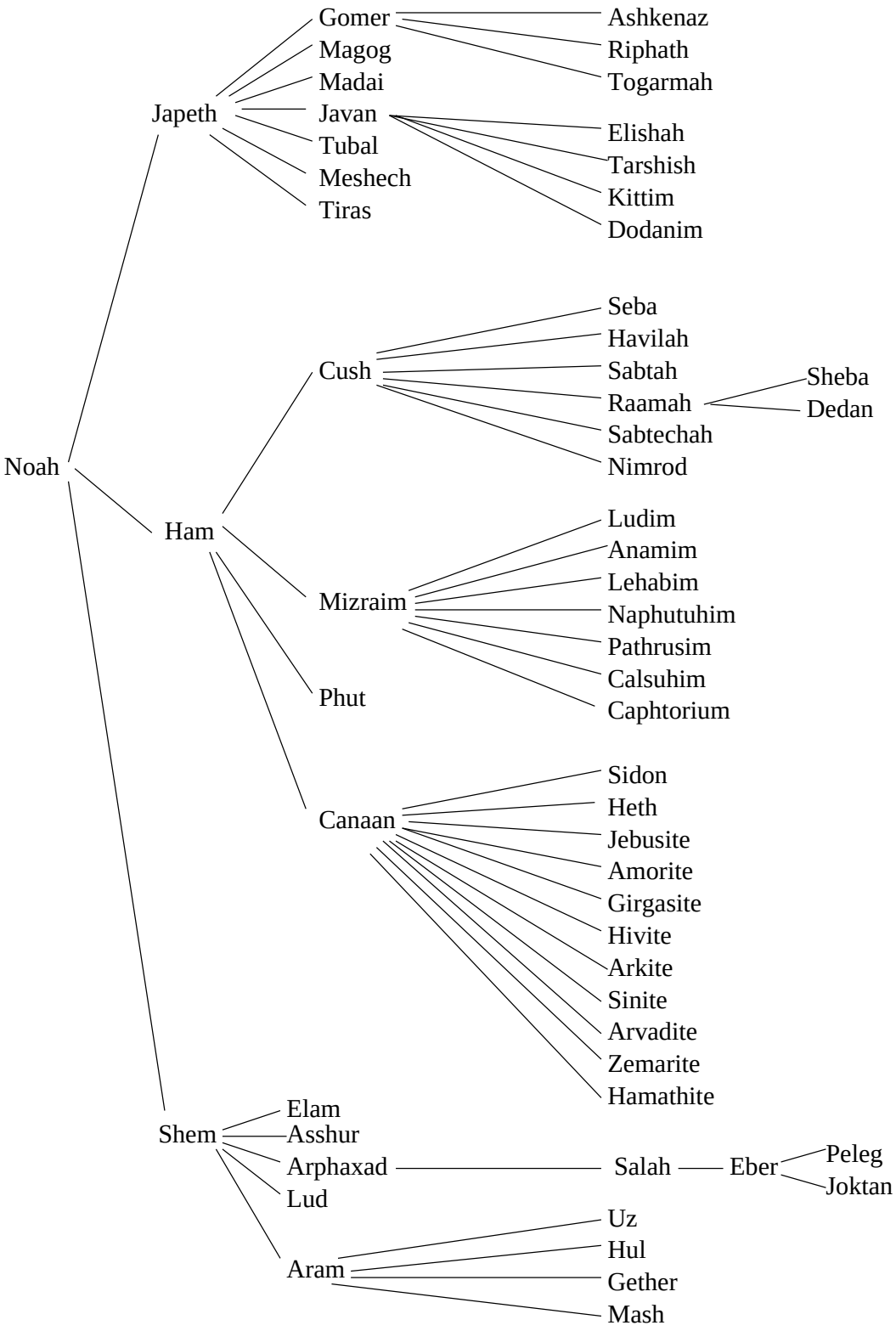
  Für den Abstammungsbaum der Nachkommen Noah's erhält man somit die folgende Ausgabe:

```
Noah
  Shem
    Aram
      Mash
      Gether
      Hul
      Uz
    Lud
    Arphaxad
      Salah
        Eber
          Joktan
          Peleg
    Asshur
    Elam
  Ham
    Canaan
      Hamathite
      Zemarite
      Arvadite
      Sinite
      Arkite
      Hivite
      Girgasite
```

```
    Amorite
    Jebusite
    Heth
    Sidon
  Phut
  Mizraim
    Caphtorim
    Casluhim
    Pathrusim
    Naphtuhim
    Lehabim
    Anamim
    Ludim
  Cush
    Nimrod
    Sabthechah
    Raamah
      Dedan
      Sheba
    Sabtah
    Havilah
    Seba
Japheth
  Tiras
  Meshech
  Tubal
  Javan
    Dodanim
    Kittim
    Tarshish
    Elishah
  Madai
  Magog
  Gomer
    Togarmah
    Riphath
    Ashkenaz
```

In dieser Aufgabe ist die Verwendung von Arrays und jeglicher Java-Bibliotheksmethoden bis auf die der Klasse `String` nicht erlaubt!

Hinweis: Auf der WWW-Seite zur Vorlesung finden Sie unter dem Punkt *Übungen* die Datei *NoahsTree.java*, mit der Sie Ihre Implementierung testen können. In dieser Datei müssen Sie `Tree` durch den von Ihnen gewählten Klassennamen für den Baum ersetzen.


Noah
Japeth
Gomer
Ashkenaz
Riphath
Togarmah
Magog
Madai
Javan
Elishah
Tarshish
Kittim
Dodanim
Tubal
Meshech
Tiras
Ham
Cush
Seba
Havilah
Sabtah
Raamah
Sheba
Dedan
Sabtechah
Nimrod
Mizraim
Ludim
Anamim
Lehabim
Naphutuhim
Pathrusim
Calsuhim
Caphtorium
Phut
Canaan
Sidon
Heth
Jebusite
Amorite
Girgasite
Hivite
Arkite
Sinite
Arvadite
Zemarite
Hamathite
Shem
Elam
Asshur
Arphaxad
Salah
Eber
Peleg
Joktan
Lud
Aram
Uz
Hul
Gether
Mash

RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

LEHR- UND FORSCHUNGSGEBIET INFORMATIK II
RWTH Aachen · D-52056 Aachen · GERMANY
http://programmierung.informatik.rwth-aachen.de

Prof. Dr. Jürgen Giesl
Darius Dlugosz, Thomas v. d. Maßen, Antje Nowack

Übung *Informatik I - Programmierung* - Blatt 9

Die Übungsblätter sollen in Gruppen von je 3 Studierenden bearbeitet werden. Diese 3 Studierenden müssen aus derselben Übungsgruppe kommen. Lösungen können bis zum 8. Januar, 17.45 h, in den Kasten für Ihre Übungsgruppe im Foyer in der Ahornstr. 55 eingeworfen werden.
Bitte vergessen Sie nicht, die **Nummer Ihrer Übungsgruppe** sowie die **Namen** und **Matrikelnummer** der Mitglieder Ihrer 3er-Gruppe auf Ihre Abgabe zu schreiben.

Wichtig: Alle Programme sind **sowohl** auf Papier in den Kasten einzuwerfen **als auch** elektronisch mit Hilfe des **Online-Systems** abzugeben.

Aufgabe 1 (9 Punkte)

In dieser Aufgabe soll die elektronische Girokontenverwaltung aus Aufgabe 4 des 5. Übungsblatts verändert werden. Die Klasse *Konto* soll nun in der Lage sein, beliebig viele Buchungen speichern zu können. Dazu ist es notwendig, die vorhandene Implementierung mit der Array-Verwaltung durch eine **doppelt-verkette** lineare Liste zu ersetzen. Beachten Sie dabei die folgenden Punkte:

- Definieren Sie für die Klasse *Konto* geeignete Konstruktoren die es erlauben, ein leeres aber initialisiertes Konto sowie ein Konto mit einer Belegung der Attribute zu instantiieren.
- Implementieren Sie die Verwaltung der Buchungen durch eine doppelt-verkette lineare Liste. Durch die doppelte Verkettung soll es möglich sein, in beide Richtungen durch die Liste zu navigieren. Dies bedeutet, dass jedes Listenelement einen Verweis auf seinen Nachfolger und auf seinen Vorgänger besitzt. Die Realisierung der Buchungsliste sollte in einer eigenen Klasse - analog zur Vorlesung - implementiert werden. Überlegen Sie sich Operationen, die für den nächsten Punkt und somit zur Verwaltung der Buchungsliste benötigt werden. Sie können den Quellcode aus dem Lösungsvorschlag der Aufgabe 4 des 5. Übungsblatts verwenden.
- Passen Sie die folgenden Methoden der Klasse *Konto* aus Aufgabe 4b des 5. Übungsblatts entsprechend an:
  - `public void buchungHinzufuegen(Buchung b)`
  - `public void druckeBuchungsliste()`
    Diese Methode soll die Buchungsliste vorwärts und rückwärts ausgeben!
  - `public double berechneSaldo()`

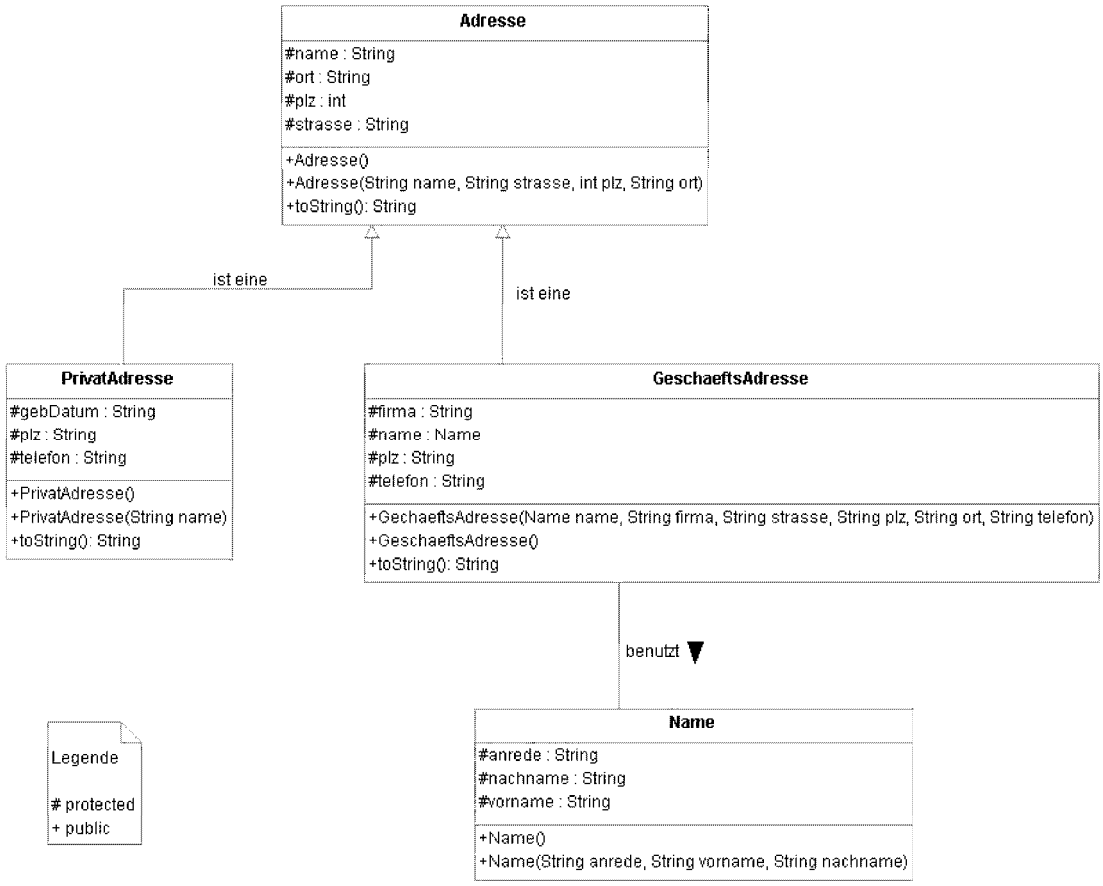
Hinweis: Diese Methoden können selbstverständlich Methoden der Klasse, welche die Buchungsliste realisiert, aufrufen. Beachten Sie, dass eine neue Buchung direkt an der richtigen Position (d. h. die chronologische Ordnung soll erhalten bleiben) eingefügt werden soll.

- Beachten Sie, dass Sie keine Änderungen an den Klassen *Buchung* oder *Datum* durchführen sollen.

Realisieren Sie alle Zugriffe auf die Liste mittels Rekursion, d.h. ohne Verwendung von Schleifen oder Sprüngen.

Aufgabe 2 (4 Punkte)

Gegeben sei das folgende Klassendiagramm:



Ebenso ist eine Klasse *Adressentest* gegeben:

```
1: public class Adressentest {
2:
```

```
 3:   public static void main (String[] args) {
 4:      Adresse adr = new Adresse("Muster", "Rheinstrasse 4", 52060, "Aachen");
 5:      PrivatAdresse padr = new PrivatAdresse();
 6:      GeschaeftsAdresse gadr = new GeschaeftsAdresse();
 7:
 8:      padr = gadr;
 9:      padr = (Adresse)gadr;
10:      gadr = adr;
11:      padr.name = "Peter Muster";
12:      padr.plz = "52074";
13:      padr.gebDatum = "13.4.1968";
14:
15:      adr = padr;
16:      gadr.telefon = "+49 241 123456";
17:      System.out.println("Name: " + adr.name);
18:      System.out.println("Telefon: " + adr.telefon);
19:
20:      Adresse a = new GeschaeftsAdresse();
21:      adr.telefon = "0241-10275";
22:      gadr.plz = "D-52074";
23:      gadr.name = "Simon Schmidt";
24:      padr.name = "Simon Schmidt";
25:      a.plz = "D-52074";
26:      a.name = new Name("Herr", "Klaus", "Wienert");
27:      gadr.name = new Name("Herr", "Klaus", "Wienert");
28:   }
29: }
```

Geben Sie für die Anweisungen der Zeilen 4-27 in der main-Methode an, ob diese gültig sind, d.h. ob sie ausführbar sind oder ob sie zu einem Fehler führen würden. Falls eine Anweisung nicht gültig ist, geben Sie den Grund dafür an. Sollte eine Anweisung gültig sein, so erläutern Sie, auf welches Attribut bzw. auf welche Methode welcher Klasse zugegriffen wird.

Aufgabe 3 (3 + 6 + 2 + 2 Punkte)

Ein Graphikeditor soll die folgenden geometrischen Figuren darstellen können:

- Rechtecke
- Kreise
- Dreiecke
- Quadrate
- Ellipsen

Alle Figuren besitzen einen Flächeninhalt und einen Umfang. Ebenso besitzen alle Figuren eine x- und eine y-Koordinate sowie eine Linien- und eine Füllfarbe.

a) Modellieren Sie die oben angegebenen Klassen und fassen sie gleiche Funktionalität in sinnvollen Oberklassen zusammen, d.h. so viele Attribute und Mathoden wie möglich sollen vererbt werden. Zeichnen Sie ein Klassendiagramm gemäß Aufgabe 2, welches die benötigten Klassen mit ihren Attributen und Methodensignaturen zeigt sowie die Vererbungsbeziehungen zwischen den Klassen darstellt. Alle Klassen sollen u.a. die folgenden Methoden bereitstellen:

   - `public double berechneFlaecheninhalt()`
   - `public double berechneUmfang()`

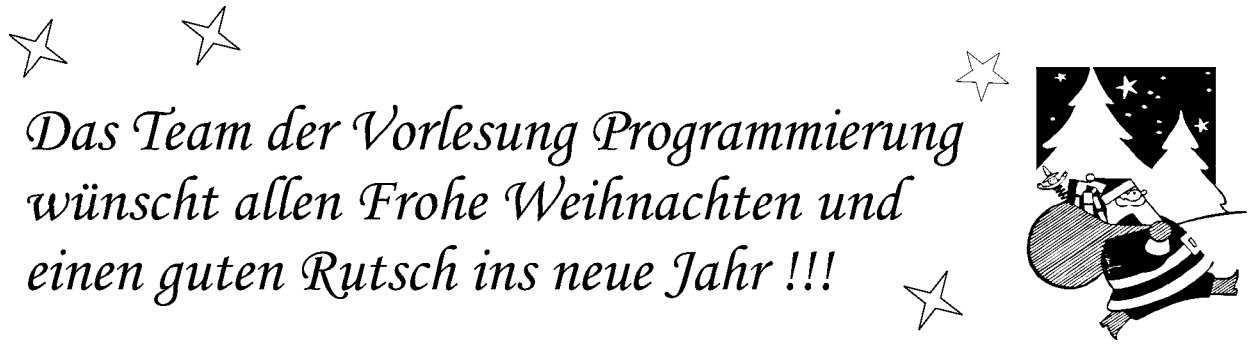
b) Realisieren Sie die von Ihnen modellierten Klassen in Java. Verwenden Sie dabei die Konzepte, die im Kapitel II.4 der Vorlesung vorgestellt worden sind. Entwerfen Sie insbesondere geeignete Konstruktoren, um die Attribute der jeweiligen Klasse mit beliebig vorgegebenen Werten zu initialisieren. Verwenden Sie dabei soweit wie möglich bereits existierende Konstruktoren der Oberklassen unter Benutzung von `super(...)`. Sollte es in einer Oberklasse nicht möglich sein, die Berechnungen von Umfang und Flächeninhalt durchzuführen, da noch zu wenige Informationen über die geometrische Figur vorhanden sind, so soll der Benutzer nach den Werten für den Flächeninhalt und Umfang gefragt werden.

c) Implementieren Sie eine Methode in einem Hauptprogramm, welche als Parameter ein Array erhält, welches beliebige geometrische Figuren enthält. Die Methode soll die Anzahl der jeweiligen geometrischen Figuren (d.h. die Anzahl der Rechtecke, der Quadrate, ...), die in dem Array gespeichert sind sowie die Summe aller Flächen berechnen und ausgeben.

d) Erzeugen Sie für alle Klassen eine gemeinsame Schnittstellendokumentation mit `javadoc`.

Hinweis: Die folgenden (Näherungs-)Formeln können zur Berechnung von Flächeninhalt und Umfang verwendet werden:

- Fläche eines allgemeinen Dreiecks: $A = \sqrt{s * (s-a) * (s-b) * (s-c)}$, wobei s für den halben Umfang steht
- Fläche einer Ellipse: $A = \pi * a * b$, wobei a und b die Längen der beiden Halbachsen sind
- Umfang einer Ellipse: $U \approx \pi * (3 * (a+b) - \sqrt{(a+3*b) * (3*a+b)})$



RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

LEHR- UND FORSCHUNGSGEBIET INFORMATIK II
RWTH Aachen · D-52056 Aachen · GERMANY
http://programmierung.informatik.rwth-aachen.de

Prof. Dr. Jürgen Giesl
Darius Dlugosz, Thomas v. d. Maßen, Antje Nowack

Übung *Informatik I - Programmierung* - Blatt 10

Die Übungsblätter sollen in Gruppen von je 3 Studierenden bearbeitet werden. Diese 3 Studierenden müssen aus derselben Übungsgruppe kommen. Lösungen können bis zum 15. Januar, 17.45 h, in den Kasten für Ihre Übungsgruppe im Foyer in der Ahornstr. 55 eingeworfen werden.
Bitte vergessen Sie nicht, die **Nummer Ihrer Übungsgruppe** sowie die **Namen** und **Matrikelnummer** der Mitglieder Ihrer 3er-Gruppe auf Ihre Abgabe zu schreiben.

Wichtig: Alle Programme sind **sowohl** auf Papier in den Kasten einzuwerfen **als auch** elektronisch mit Hilfe des **Online-Systems** abzugeben.

Aufgabe 1 (5 Punkte)

Listen können durch folgende Grammatik in EBNF definiert werden:

Liste	=	(LeereListe | NichtLeereListe)
NichtLeereListe	=	Vergleichbar Liste

wobei LeereListe für die leere Liste steht und Vergleichbar ein Nichtterminal ist, welches hier nicht näher beschrieben wird. Es steht für Elemente einer Klasse, die die abstrakte Klasse `Vergleichbar` (s. Vorlesung) erweitert (bzw. das Interface `Vergleichbar` implementiert).
Implementieren Sie entsprechende Klassen `Liste`, `NichtLeereListe` und `LeereListe`, die genau der obigen Definition von Listen entsprechen. Hierbei müssen Sie ohne Klassen wie `Element` auskommen.
Es sollen Methoden zur Verfügung gestellt werden, die Folgendes ermöglichen:

- Aneinanderhängen (Konkatenieren) von zwei Listen
- Löschen eines Wertes (wenn er enthalten ist)
- Test, ob ein (als Parameter übergebener) Wert enhalten ist
- Einfügen eines Wertes jeweils hinten und vorne

Die oben beschriebenen Methoden verändern nicht das Objekt, mit dem sie aufgerufen werden, sondern liefern jeweils eine neue Liste zurück.

Hinweis: Realisieren Sie `Liste` als abstrakte Klasse, und implementieren Sie in dieser bereits die oben beschriebenen Methoden, soweit dies möglich ist. Die Klassen `LeereListe` und `NichtLeereListe` erweitern jeweils die Klasse `Liste` (d.h., sie sind Unterklassen der Klasse `Liste`).

Aufgabe 2 (2 Punkte)

Vergleichen Sie die Verwendung von Interfaces mit der Verwendung abstrakter Klassen.

Aufgabe 3 (1 + 4 + 1 + 1 + 1 Punkte)

a) Schreiben Sie die in der Vorlesung vorgestellte abstrakte Klasse `Vergleichbar` als Interface. Schreiben Sie nun ein weiteres Interface `Ord`, welches eine abstrakte Methode `groesser` zur Verfügung stellt (und von `Vergleichbar` unabhängig ist).

b) Schreiben Sie eine abstrakte Klasse `Suchbaum`. Ein Suchbaum soll hierbei ein binärer Baum sein, der Werte enthält, welche zu einer Klasse gehören, die sowohl `Ord` als auch `Vergleichbar` implementiert, und dessen Teilbäume alle folgende Eigenschaft erfüllen:
Alle Werte im linken Teilbaum sind kleiner oder gleich dem Wert der Wurzel, und alle Werte im rechten Teilbaum sind größer als der Wert der Wurzel.

Folgende Methoden sollen zur Verfügung gestellt werden:

- Einfügen eines Wertes an der richtigen Stelle
- Test, ob ein übergebener Wert im Baum enthalten ist
- Bestimmung des Wertes der Wurzel
- eine Methode, die den linken Teilbaum liefert
- eine Methode, die den rechten Teilbaum liefert
- eine `toString`-Methode, welche einen String liefert, der den Baum in Infixnotation repräsentiert.
(In der Infixnotation repräsentiert man erst den linken Teilbaum, dann den Wert der Wurzel und dann den rechten Teilbaum.)

c) Erweitern Sie die in der Vorlesung vorgestellte Klasse `Bruch` zu einer Implementierung von `Ord` und von `Vergleichbar`.

d) Schreiben Sie ein geeignetes Hauptprogramm zum Testen Ihrer Implementierung der Klasse `Suchbaum`. Die Elemente eines Suchbaums sollen hierbei aus der Klasse `Bruch` sein.

e) Erstellen Sie eine gemeinsame Schnittstellendokumentation für `Ord`, `Vergleichbar` und die Klasse `Suchbaum` mit Hilfe von javadoc.

Aufgabe 4 (1 + 1 + 2 + 1 Punkte)

a) In der Vorlesung wurde die Klasse `Liste` vorgestellt, welche Werte aus der Klasse bzw. dem Interfacetyp `Vergleichbar` enthält. Ergänzen Sie die Klasse `Liste` um Methoden

- `boolean empty ()` zum Testen, ob die Liste leer ist

- `Vergleichbar ersterWert ()` zur Bestimmung des ersten Wertes in der Liste (wenn die Liste nicht leer ist)
- `void LoescheErstes ()` zum Löschen des ersten Wertes der Liste

b) Erstellen Sie ein Interface für eine Datenstruktur `Stack`. Dieses soll die Methoden `void push (Vergleichbar zuvergleichen)`, `Vergleichbar pop ()` und `boolean empty ()` zur Verfügung stellen.

c) Einen Stack kann man sich als Stapel von Elementen vorstellen. Stacks speichern Werte in sequentieller Ordnung, der Zugriff ist jedoch auf den obersten Wert beschränkt. Die zulässigen Operationen sind:

push: Neues Element wird am oberen Ende (Top) hinzugefügt.

pop: Oberstes Element wird vom Stack entfernt und zurückgeliefert.

empty: Überprüfung, ob der Stack leer ist.

Implementieren Sie das in b) erstellte Interface durch eine Erweiterung (d.h., Unterklasse) der in a) ergänzten Klasse `Liste`. Die Methoden `push`, `pop` und `empty` sollen dabei die oben erläuterten Bedeutungen besitzen.

d) Schreiben Sie ein geeignetes Hauptprogramm zum Testen Ihrer Implementierung.

RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

LEHR- UND FORSCHUNGSGEBIET INFORMATIK II
RWTH Aachen · D-52056 Aachen · GERMANY
http://programmierung.informatik.rwth-aachen.de

Prof. Dr. Jürgen Giesl
Darius Dlugosz, Thomas v. d. Maßen, Antje Nowack

Übung *Informatik I - Programmierung* - Blatt 11

Die Übungsblätter sollen in Gruppen von je 3 Studierenden bearbeitet werden. Diese 3 Studierenden müssen aus derselben Übungsgruppe kommen. Lösungen können bis zum 22. Januar, 17.45 h, in den Kasten für Ihre Übungsgruppe im Foyer in der Ahornstr. 55 eingeworfen werden.
Bitte vergessen Sie nicht, die **Nummer Ihrer Übungsgruppe** sowie die **Namen** und **Matrikelnummer** der Mitglieder Ihrer 3er-Gruppe auf Ihre Abgabe zu schreiben.

Wichtig: Alle Programme sind **sowohl** auf Papier in den Kasten einzuwerfen **als auch** elektronisch mit Hilfe des **Online-Systems** abzugeben.

Aufgabe 1 (3 + 6 Punkte)

In dieser Aufgabe sollen noch einmal die geometrischen Figuren aus Aufgabe 3 des 9. Übungsblatts betrachtet werden. Sie können dabei auf die Dateien der Musterlösung zurückgreifen oder die Schnittstellen Ihrer Lösung, dem auf den Webseiten bereitgestellten Klassendiagramm, anpassen.

a) Konvertieren Sie die Klassen Polygon und GeoObjekt in abstrakte Klassen und deklarieren Sie sinnvolle abstrakte Methoden für diese Klassen. Erweitern Sie ebenfalls jede Klasse um eine Methode `public void aenderung()`, welche es dem Benutzer gestattet, sämtliche Attribute (ausser den Attributen *umfang* und *flaeche*) der jeweiligen Klasse zu belegen. Dabei ist zu beachten, dass so weit wie möglich auf Methoden der Oberklasse zurückgegriffen werden soll. Gültige einzugebende Farben für die Linie und die Füllung sind dabei: "rot", "gelb", "blau" und "gruen".

b) Jetzt sollen die geometrischen Figuren **Rechteck, Quadrat, Ellipse** und **Kreis** auch graphisch auf dem Bildschirm dargestellt werden. Dazu wird Ihnen eine Klasse *Anzeige.java* zur Verfügung gestellt, welche diese Aufgabe übernimmt. Diese Datei können sie von der Webseite zur Vorlesung Programmierung herunterladen. Damit die Figuren von der graphischen Darstellung getrennt werden können, soll die in Abbildung 1 dargestellte Paketstruktur erstellt werden.

- **Paket gui**: Enthält die Klasse *Anzeige.java*
- **Paket figuren**: Enthält die Klassen aller geometrischen Figuren
- Die Klasse *Figurendarstellung.java* fungiert als Hauptprogramm

Legen Sie die oben angegebene Verzeichnisstruktur an und integrieren Sie die zugehörigen Klassen in die jeweiligen Pakete. Realisieren Sie zudem für die Klasse Rechteck die Methoden `public double getLaenge()` und `public double getBreite()` sowie für die Klasse Ellipse die Methoden `public double getRadiusA()` und `public double getRadiusB()`,

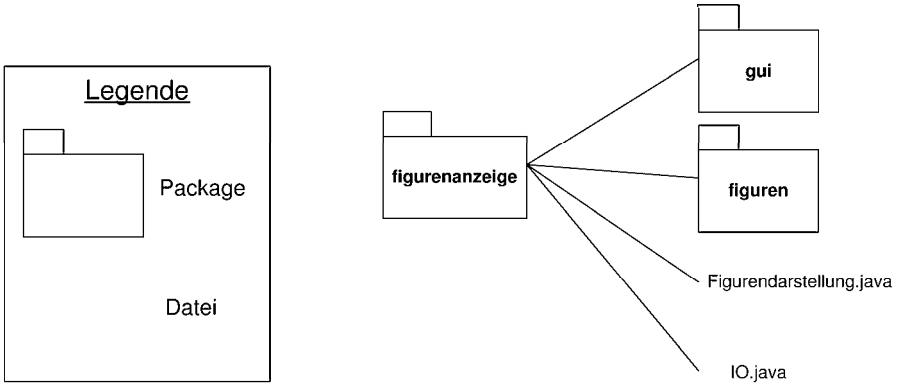


Abbildung 1: Paketstruktur für Aufgabe 1b

da diese von der Klasse *Anzeige* benötigt werden. Schreiben Sie ebenfalls geeignete get-Selektoren für die Attribute *fuellfarbe, linienfarbe, x* und *y* in der Klasse *GeoObjekt*:

- `public String getFuellfarbe()`
- `public String getLinienFarbe()`
- `public int getX()`
- `public int getY()`

Realisieren Sie außerdem die Klasse *Figurendarstellung*, welche dem Benutzer das folgende Menü bereitstellen soll:

```
1) Rechteck einfügen
2) Quadrat einfügen
3) Ellipse einfügen
4) Kreis einfügen
----------------------
5) Zeichnen
```

Durch Aufruf der Menüpunkte 1-4 soll ein neues Objekt der jeweiligen graphischen Figur erzeugt werden und der Benutzer nach den Attributausprägungen der jeweiligen Figur gefragt werden. Die Figur soll dann in einem Array, welches maximal 10 Figuren aufnehmen kann, gespeichert werden (Initialisieren Sie das Array zuvor mit `null`!). Durch Aufruf des Menüpunkts 5 soll eine neue Instanz der Klasse *Anzeige* erzeugt werden. Verwenden Sie dazu den folgenden Konstrukor: `public Anzeige(GeoObjekt[] figuren)`. Als Seiteneffekt werden dann die in dem übergebenen Array enthaltenen Figuren in einem Fenster graphisch angezeigt. Das Ergebnis könnte beispielsweise wie in Abbildung 2 gezeigt, aussehen.

Hinweis: Sollten Sie nicht auf die Musterlösung zurückgreifen wollen, so achten Sie bitte darauf, dass Ihre Klassen der Schnittstellenspezifikation genügen, wie sie in der Musterlösung zu Aufgabe 3a des 9. Übungsblatts und dem auf den Webseiten bereitgestellten Klassendiagramm angegeben ist.

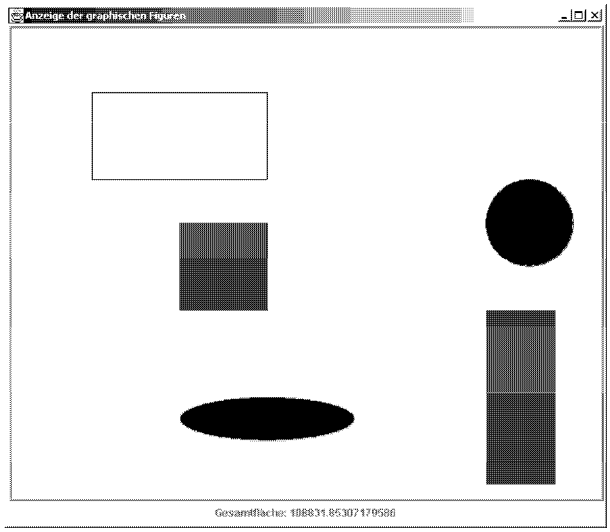


Abbildung 2: Beispielausgabe von den erzeugten graphischen Figuren

Aufgabe 2 (3 Punkte)

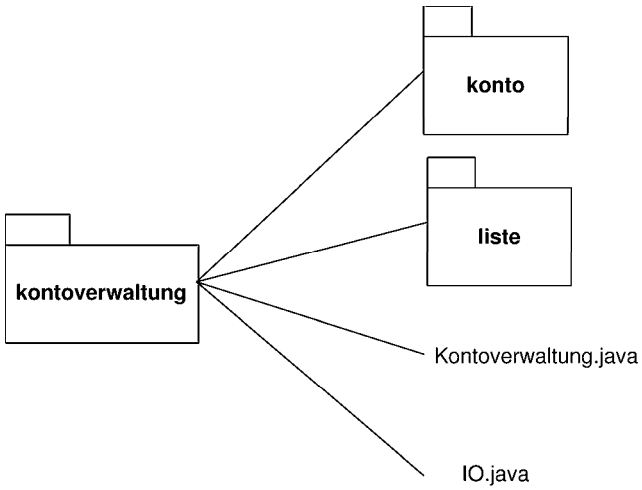
Erläutern Sie den Sinn und Zweck der Aufteilung der Klassen in unterschiedliche Pakete. Welche Vorteile ergeben sich für große Softwareprojekte daraus? Was ist beim Zugriff auf Attribute von Klassen, die sich in unterschiedlichen Paketen befinden, zu beachten?

Aufgabe 3 (6 + 2 Punkte)

In dieser Aufgabe soll die Kontoverwaltung aus Aufgabe 1 des 9. Übungsblatts noch einmal erweitert werden.

a) Jetzt soll es möglich sein, beliebig viele Buchungen sowie beliebig viele Konten zu verwalten. Die Verwaltung soll durch einfach verkettete Listen erfolgen. Da viele Operationen der Buchungsliste und der Kontoliste gleich sind, bietet es sich an, diese in einer gemeinsamen Listenverwaltung zu implementieren. Realisieren Sie zwei Klassen *Buchungsliste* und *Kontoliste* und delegieren sie so viele Operationen wie möglich an die in der Vorlesung vorgestellte Klasse *Liste* (zusammen mit der Klasse *Element* und dem Interface *Vergleichbar*). Beachten Sie, dass die Klassen *Buchung, Datum* und *Konto* nun das Interface *Vergleichbar* implementieren müssen. Erweitern Sie dazu das Interface *Vergleichbar* sowie die Klasse *Liste* um geeignete Operationen.
Hinweis: Die Kontoverwaltung soll natürlich wieder vollständig lauffähig sein. Das heißt, es muss möglich sein, aus einem Hauptprogramm *Kontoverwaltung* beliebig viele Konten zu verwalten, den einzelnen Konten Buchungen (sortiert nach ihrem Datum) hinzuzufügen sowie für jedes Konto den Saldo zu berechnen. Sie können dazu auf die Musterlösung von Aufgabe 1 des 9. Übungsblatts zurückgreifen.

b) Erstellen Sie die folgende Paketstruktur für Ihre Kontoverwaltung:



Dabei enthält das Paket *liste* die Klassen *Liste, Element* und das Interface *Vergleichbar*. Das Paket *konto* soll alle übrigen Klassen, die zur Verwaltung eines Kontos dienen, enthalten.

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
Darius Dlugosz, Thomas v. d. Maßen, Antje Nowack

Übung *Informatik I - Programmierung* - Blatt 12

Die Übungsblätter sollen in Gruppen von je 3 Studierenden bearbeitet werden. Diese 3 Studierenden müssen aus derselben Übungsgruppe kommen. Lösungen können bis zum Dienstag, den 29. Januar, 17.45 h, in den Kasten für Ihre Übungsgruppe im Foyer (neben Aula 2) in der Ahornstr. 55 eingeworfen werden.
Bitte vergessen Sie nicht, die **Nummer Ihrer Übungsgruppe** sowie die **Namen** und **Matrikelnummer** der Mitglieder Ihrer 3er-Gruppe auf Ihre Abgabe zu schreiben.

Wichtig: Alle Programme sind **sowohl** auf **Papier** in den Kasten einzuwerfen **als auch** elektronisch mit Hilfe des **Online-Systems** abzugeben.

Aufgabe 1 (3 Punkte)

Seien `xs`, `ys` Listen von ganzen Zahlen und seien `x` und `y` ganze Zahlen. Welche der folgenden Gleichungen zwischen Listen sind richtig und welche nicht? Begründen Sie bitte Ihre Antwort.

(a) `[] : xs = xs`

(b) `[] : xs = [[], xs]`

(c) `xs : [] = xs`

(d) `xs : [] = [xs]`

(e) `x : y = [x, y]`

(f) `(x : xs) ++ ys = x : (xs ++ ys)`

Hierbei steht `++` für den Verkettungsoperator für Listen. Das Resultat von `xs ++ ys` ist die Liste, die aus der Liste `xs` entsteht, indem an das Ende von `xs` die Elemente der Liste `ys` in der gleichen Reihenfolge wie sie in `ys` stehen, hinzugefügt werden.

Beispiel: `[1, 2, 3] ++ [2, 3] = [1, 2, 3, 2, 3]`

Aufgabe 2 (5 Punkte)

Implementieren Sie in Haskell folgende Funktionen.

(a) `minus :: Int → Int → Int`, die die Subtraktion von zwei natürlichen Zahlen berechnet. Für `x < y` ist `minus x y = 0`.

(b) `mul :: Int → Int → Int`, die die Multiplikation von zwei natürlichen Zahlen berechnet.

(c) `div' :: Int → Int → Int`, die die abgerundete Division von zwei natürlichen Zahlen berechnet (d.h. z.B. `div' 7 2 = 3`).

(d) `gcd' :: Int → Int → Int`, die den größten gemeinsamen Teiler von zwei natürlichen Zahlen berechnet.

(e) `lcm' :: Int → Int → Int`, die das kleinstes gemeinsames Vielfache von zwei natürlichen Zahlen berechnet.

Hierbei dürfen nur der Operator `+` und die Vergleichsoperatoren `==`, `/=`, `>`, `>=`, `<`, `<=` verwendet werden. Lokale Deklarationen (z. B. mit `where`) sind nicht erlaubt. In Teilaufgabe (c) dürfen zusätzlich die Funktionen aus (a) und (b) verwendet werden. In den Teilaufgaben (d) und (e) ist die Benutzung der in (a), (b), (c) deklarierten Funktionen `minus`, `mul` und `div'` erlaubt. In (e) darf auch die Funktion aus (d) verwendet werden. Auf negativen Zahlen dürfen sich Ihre Funktionen beliebig verhalten.

Aufgabe 3 (6 Punkte)

Die Funktion `len`, welche die Länge eine Liste von ganzen Zahlen berechnet, kann wie folgt implementiert werden:

```
len :: [Int] -> Int
len xs = len' 0 xs
    where len' n [] = n
          len' n (x : xs) = len' (n + 1) xs
```

Hierbei wird eine Hilfsfunktion `len'` benutzt, die einen zusätzlichen Parameter hat, in dem das Ergebnis akkumuliert wird. Die Funktion `len'` ist endrekursiv, da die rekursiven Aufrufe auf den rechten Seiten ihrer Deklarationen nicht in Argumenten anderer Funktionen auftreten.

Benutzen Sie die Technik der Endrekursion zur Implementierung der folgenden Funktionen. Geben Sie jeweils die zugehörige Typdeklaration an.

(a) `sum'` addiert alle Elemente einer Liste von ganzen Zahlen.

(b) `fac`, die Fakultätsfunktion (die sich auf negativen Zahlen beliebig verhalten darf).

(c) `reverse'` dreht die Reihenfolge der Elemente in einer Liste von ganzen Zahlen um.

Aufgabe 4 (8 Punkte)

In dieser Aufgabe soll in Haskell ein Programm `wandle` implementiert werden, das als Parameter eine Wortdarstellung (als String) einer ganzen Zahl von 1 bis 9999 erwartet und die zugehörige Zahl auf dem Bildschirm ausgibt.

Bei der Wortdarstellung sollen nur Kleinbuchstaben verwendet werden. Für die Umlaute `ü` und `ö` werden `ue` bzw. `oe` verwendet. Desweiteren beginnen die Zahlen von 100 bis 199 immer mit dem Wortlaut "einhundert" und die Zahlen von 1000 bis 1999 immer mit dem Wortlaut "eintausend". Nach dem Wortlaut "hundert" oder "tausend" folgt unmittelbar der Wortlaut der nachfolgenden Zahl (ohne ein "und" dazwischen), d.h. die Zahl 599 wird "fuenfhundertneunundneunzig" ausgesprochen und nicht "fuenfhundert*und*neunundneunzig". Unzulässige Eingaben sind z.B. "hundert", "einhundertundzwei", "eintausendhundertzwoelf", "eintausendeinhundertzwölf" oder "neuntausendeinhundertundzwanzig".

Beispiele:
Eingabe: `wandle "eins"`
Ausgabe: `1`

Eingabe: `wandle "einhundertzwei"`
Ausgabe: `102`

Eingabe: `wandle "einhundertsechzig"`
Ausgabe: `160`

Eingabe: `wandle "eintausendeinhundertzwoelf"`
Ausgabe: `1112`

Eingabe: `wandle "eintausendneunhundertsechzehn"`
Ausgabe: `1916`

Eingabe: `wandle "neuntausendneunhundertneunundneunzig"`
Ausgabe: `9999`

Eingaben, die keiner Zahl zugeordnet werden können, brauchen nicht gesondert behandelt zu werden.

Die Implementierung darf höchstens 50 Deklarationen enthalten (inklusive der lokalen Deklarationen, wie z.B. mit `where`). Dabei sollen die Funktionen ohne Verwendung von `if` und ohne bedingte definierende Gleichungen (d.h. nur mit Pattern Matching) deklariert werden.

RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

LEHR- UND FORSCHUNGSGEBIET INFORMATIK II
RWTH Aachen · D-52056 Aachen · GERMANY
http://programmierung.informatik.rwth-aachen.de

Prof. Dr. Jürgen Giesl
Darius Dlugosz, Thomas v. d. Maßen, Antje Nowack

Übung *Informatik I - Programmierung* - Blatt 13

Die Übungsblätter sollen in Gruppen von je 3 Studierenden bearbeitet werden. Diese 3 Studierenden müssen aus derselben Übungsgruppe kommen. Lösungen können bis zum 5. Februar, 17.45 h, in den Kasten für Ihre Übungsgruppe im Foyer in der Ahornstr. 55 eingeworfen werden.
Bitte vergessen Sie nicht, die **Nummer Ihrer Übungsgruppe** sowie die **Namen** und **Matrikelnummer** der Mitglieder Ihrer 3er-Gruppe auf Ihre Abgabe zu schreiben.

Wichtig: Alle Programme sind in Haskell zu implementieren und **sowohl** auf Papier in den Kasten einzuwerfen **als auch** elektronisch mit Hilfe des **Online-Systems** abzugeben.

Aufgabe 1 (5 Punkte)

Gegeben seien folgende Deklarationen:

```
data Nats = Zero | Succ Nats deriving Show

inf :: Nats
inf = Succ inf

equal :: Nats -> Nats -> Bool
equal Zero Zero =True
equal (Succ a) (Succ b) = (equal a b)
equal _ _ = False

falls :: Bool -> Int -> Int -> Int
falls True x y = x
falls False x y = y

fun1 :: Nats -> Int
fun1 y = falls (equal y Zero) 3 3

fun2 :: Nats -> Int
fun2 y = 3

fun3 :: Nats -> Int
fun3 y = falls (equal y inf) 0 1
```

```
fun4 :: Nats -> Int
fun4 (Succ (Succ x)) = 5
fun4 (Succ x) = 3
fun4 Zero = 1
```

Betrachten Sie die Auswertung folgender Ausdrücke:

a) `fun1 inf`

b) `fun2 inf`

c) `fun3 inf`

d) `fun4 inf`

e) `fun3 (Succ Zero)`

Zeigen Sie jeweils die Arbeitsweise der strikten Auswertung auf sowie die der Auswertung von Haskell. Geben Sie dazu jeweils die einzelnen Auswertungsschritte an. In welchen Fällen terminiert die Auswertung?

Aufgabe 2 (4 Punkte)

Geben Sie für folgende Ausdrücke ihren Typ an, falls kein Typfehler vorliegt. Begründen Sie in beiden Fällen Ihre Antwort.
Gehen Sie hierbei davon aus, daß `3` und `4` den Typ `Int` haben und `+` den Typ `Int -> Int -> Int` hat.

a) `(3, 4, True)`

b) `[3, 4, True]`

c) `(\x -> x + x) : []`

d) `(\x -> \y -> x + (\y -> y)) 3 4`

e) `\x -> x : [x]`

f) `\x -> x : x`

g) `\x -> [] : x`

h) `\x -> [x] : x`

Hinweis: ':' bindet hierbei stärker als '`->`', d.h. `\x -> a : b` steht für `\x -> (a : b)`.

Aufgabe 3 (3 Punkte)

Geben Sie jeweils ein Beispiel für Funktionen mit folgenden Typen an:

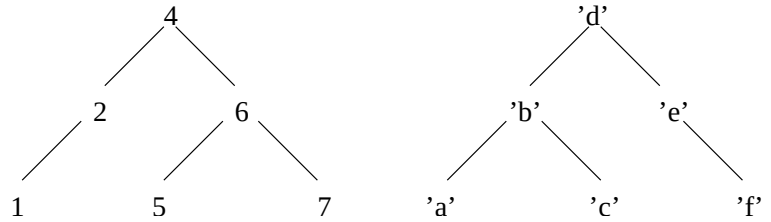
a) `([a] -> b) -> b`

b) `(a -> a) -> (a -> a)`

c) `(a -> b) -> [a] -> [b]`

Aufgabe 4 (4 Punkte)

Ein Baum ist ein Binärbaum, wenn jeder seiner Knoten höchstens zwei Nachfolger besitzt.
Beispiele für Binärbäume sind:



Ein binärer Baum ist ein Suchbaum, wenn er Werte enthält, auf denen eine Relation $<$ existiert und dessen Teilbäume alle folgende Eigenschaft erfüllen:
Alle Werte im linken Teilbaum sind kleiner oder gleich dem Wert der Wurzel, und alle Werte im rechten Teilbaumes sind größer als der Wert der Wurzel.

a) Implementieren Sie in Haskell einen Datentyp für binäre Bäume mit Einträgen eines beliebigen Typs.

b) Schreiben Sie basierend auf dem in a) definierten Datentyp eine Funktion, die eine Liste liefert, die den übergebenen binären Baum in Infixnotation repräsentiert.
(In der Infixnotation repräsentiert man erst den linken Teilbaum, dann den Wert der Wurzel und dann den rechten Teilbaum.)

c) Schreiben Sie basierend auf dem von Ihnen definierten Datentyp für binäre Bäume eine Funktion, die zu einem binären Suchbaum t mit Einträgen vom Typ `Int` und einer Zahl n den binären Suchbaum liefert, der entsteht, wenn n in t eingefügt wird. Hierbei darf ein neuer Wert nur durch Anhängen an einen geeigneten Knoten des Baumes eingefügt werden.

d) Gehen Sie wiederum von einem binären Suchbaum mit Einträgen vom Typ `Int` aus. Schreiben Sie eine Funktion, die zu einem Baum und einer Zahl angibt, ob diese in dem Baum enthalten ist.

Aufgabe 5 (5 Punkte)

Definieren Sie einen Datentyp zur Realisierung von Queues (Warteschlangen).
Eine Queue ist eine endliche Folge von Werten des gleichen Typs, d.h. eine Queue besitzt folgende Form:

q_1	q_2	...	q_n

wobei n eine natürliche Zahl größer oder gleich 0 ist und die Werte q_i vom gleichen Typ sind für $i \in \{1, ..., n\}$. Das erste Element bzw. der Anfang der Queue ist q_1. Das letzte Element bzw. das Ende der Queue ist q_n.
Definieren Sie folgende Funktionen basierend auf dem von Ihnen definierten Datentyp:

- `firstEl` liefert das erste Element der übergebenen Queue.
- `lastEl` liefert das letzte Element der übergebenen Queue.
- `add` liefert die Queue, die entsteht, wenn man den übergebenen Wert am Ende der übergebenen Queue einfügt.
- `get` liefert die Queue, die entsteht, wenn man den ersten Wert der übergebenen Queue löscht.
- `del` liefert die Queue, die entsteht, wenn man alle Vorkommen eines übergebenen Wertes aus der übergebenen Queue löscht.
  (Hierbei kann zusätzlich eine Funktion `eq :: a -> a -> Bool` übergeben werden, die den Gleichheitstest für einen Typ `a` realisiert.)

Die Verwendung von Listen ist in dieser Aufgabe nicht erlaubt.

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
Darius Dlugosz, Thomas v. d. Maßen, Antje Nowack

Übung *Informatik I - Programmierung* - Blatt 14

Die Übungsblätter sollen in Gruppen von je 3 Studierenden bearbeitet werden. Diese 3 Studierenden müssen aus derselben Übungsgruppe kommen. Lösungen können bis zum Dienstag, den 12. Februar, 17.45 h, in den Kasten für Ihre Übungsgruppe im Foyer (neben Aula 2) in der Ahornstr. 55 eingeworfen werden.
Bitte vergessen Sie nicht, die **Nummer Ihrer Übungsgruppe** sowie die **Namen** und **Matrikelnummer** der Mitglieder Ihrer 3er-Gruppe auf Ihre Abgabe zu schreiben.

Wichtig: Alle Programme sind **sowohl** auf **Papier** in den Kasten einzuwerfen **als auch** elektronisch mit Hilfe des **Online-Systems** abzugeben.

Aufgabe 1 (3 Punkte)

Seien `f` und `g` Funktionen mit folgenden Typen

```
f :: Int -> Int
g :: Int -> Int -> Int
```

Die Funktion `h` sei wie folgt definiert

```
h x y = f (g x y)
```

(a) Bestimmen Sie den Typ von `h`.

(b) Welche der folgenden Gleichungen sind richtig und welche nicht? Begründen Sie bitte Ihre Antwort.

(i) `h = comp f g`
(ii) `h x = comp f (g x)`
(iii) `h x y = (comp f g) x y`
(iv) `h x y = (comp f (g x)) y`
(v) `h x y = comp f (g x y)`

Bemerkung: Die Funktion `comp` ist die in der Vorlesung eingeführte Funktion und ist wie folgt definiert:

```
comp :: (b -> c) -> (a -> b) -> (a -> c)
comp f g = \x -> f (g x)
```

Aufgabe 2 (6 Punkte)

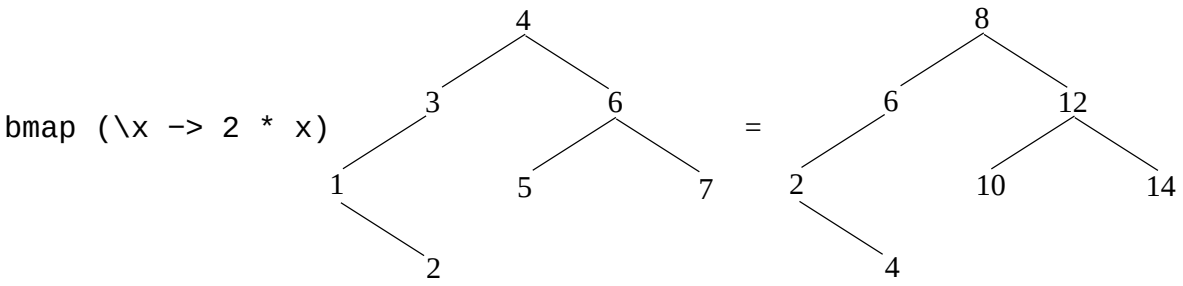
(a) Implementieren Sie in Haskell eine Funktion, welche die Quersumme einer ganzen Zahl berechnet. Die Quersumme einer Zahl ist die Summe ihrer Ziffern.

(b) Implementieren Sie in Haskell eine Funktion, welche für einen String seine Quersumme berechnet. Hierbei sei die Quersumme die Summe der Zahlen, die in dem ASCII-Code den in dem String enthaltenen Zeichen zugeordnet sind. Verwenden Sie dabei die Funktion `map` für Listen.

(c) Implementieren Sie in Haskell für die folgende Datenstruktur `Baum` eine Funktion `bmap` höherer Ordnung, welche auf das Element jedes Knotens des Baumes eine übergebene Funktion anwendet.

```
data Baum a = Knoten a [Baum a]
```

Beispiel:



Geben Sie jeweils den Typ der Funktionen an.

Zur Erinnerung:
Ein String ist eine Liste von Zeichen, d.h. der Typ `String` entspricht dem Typ `[Char]`.

Zu einem Zeichen erhält man mit Hilfe der Funktion `ord :: Char -> Int` ihre in dem ASCII-Code zugeordnete Zahl.

Aufgabe 3 (6 Punkte)

Eine Hash-Datenbank besteht aus Einträgen, die jedem Objekt eines Typs `a` eine Reihe von Objekten eines Typs `b` zuordnet.

Beispiele:

PLZ	Straßen
52074	Halifaxstr., Ahornstr., ...
52062	...
...	

oder

Stadt	PLZ
Aachen	52074, 52062, 52064, ...
Köln	50670, 50679, ...
...	

oder

Student	Freunde
Hugo	Erna, Heinz, Felix, ...
Sabine	Susi, Peter, ...
...	

(a) Definieren Sie in Haskell die Datenstruktur `Hash a b` zur Speicherung von Hash-Datenbanken.

(b) Implementieren Sie in Haskell eine Funktion `insert`, die zu einem Wert `x` des Typs `a`, dem ein Wert `y` des Typs `b` zugeordnet wird, einen Eintrag in eine Hash-Datenbank vornimmt. Dabei soll ein neuer Eintrag erzeugt werden, falls `x` noch nicht in der Hash-Datenbank vorhanden ist. Ansonsten soll die Reihe mit den Objekten, die dem `x` zugeordnet sind, um `y` erweitert werden.

(c) Implementieren Sie in Haskell eine Funktion `hmap` höherer Ordnung, welche eine Funktion `f` und eine Hash-Datenbank als Eingabe erwartet und eine Hash-Datenbank zurückliefert, bei der auf alle Elemente, die einem Wert zugeordnet sind, die Funktion `f` angewendet wird. Welches ist der Typ von `hmap`?

Beispiel: Für die Hash-Datenbank `plz` aus dem Beispiel von oben, die den Städten Postleitzahlen zuordnet, und für die Funktion `f x = x + 1` erhält man nach dem Aufruf `hmap f plz` die folgende Hash-Datenbank.

Stadt	PLZ
Aachen	52075, 52063, 52065, ...
Köln	50671, 50680, ...
...	

Aufgabe 4 (4 Punkte)

Auf einem Tisch t sind mit Nummern gekennzeichnete Blöcke gestapelt.

(a) Erstellen Sie ein Prolog-Programm, in dem die in der Abbildung 1 dargestellte Situation mit Hilfe eines Prädikats `auf` repräsentiert wird. Dabei bedeutet `auf(X,Y)`, dass `X` auf `Y` liegt.

(b) Definieren Sie ein Prädikat `ueber`, welches ausdrückt, dass ein Block über einem anderen Block bzw. auf dem Tisch steht. Hierbei dürfen höchstens zwei Klauseln definiert werden.

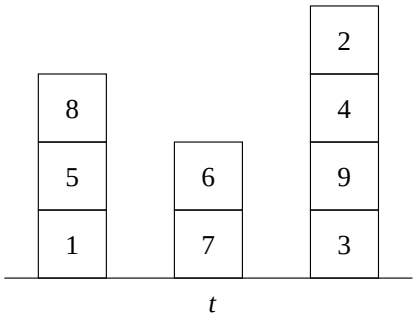


Abbildung 1:

(c) Stellen Sie die nachfolgenden Anfragen und geben Sie alle Lösungen an.

Steht Block 8 über Block 1?
Steht Block 2 über Block 7?
Was steht über Block 3?
Was steht unter Block 4?

Aufgabe 5 (6 Punkte)

(a) Erstellen Sie ein Prolog-Programm, das die nachfolgende umgangssprachliche Beschreibung formalisiert.

Peter liebt Susi.
Hans liebt Susi und Sabine.
Sabine liebt Peter und hasst Hans.
Susi liebt Peter und Felix.
Susi hasst Sabine.
Jeder liebt sich selbst.
Jemand ist ein Pechvogel, wenn seine Liebe mit Hass vergolten wird.

(b) Stellen Sie die nachfolgenden Anfragen in Prolog und geben Sie alle Lösungen an.

Liebt Peter Susi?
Liebt Susi Felix?
Wen liebt Sabine?
Wer liebt Sabine?
Wer liebt jemanden, der ihn auch liebt?
Wer liebt einen Pechvogel?

(c) Definieren Sie ein Prädikat `befreundet(X, Y)`, welches ausdrückt, dass eine Person `X` mit der Person `Y` befreundet ist. Hierbei sind zwei Personen miteinander befreundet, wenn es zwischen ihnen irgendeine Liebesbeziehung gibt oder wenn es eine Person gibt, die mit `X` und `Y` befreundet ist. Verwenden Sie so wenige Klauseln wie möglich (maximal 4).

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

Probeklausur
Informatik I - Programmierung
18. 2. 2002

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 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 ist.
- 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	18	
Aufgabe 2	11	
Aufgabe 3	14	
Aufgabe 4	18	
Aufgabe 5	23	
Aufgabe 6	16	
Summe	100	
Note	-	

Vorname	Name	Matr.-Nr.

Aufgabe 1 (Programmanalyse, 12 + 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 abstract class A {

  public static int anzahl = 0;
  public int nr = 0;

  public A () {
    anzahl = anzahl + 1;}
  public void drucke () {
    System.out.println ("A" + this);}
  public String toString() {
    return "(anzahl: " + anzahl + ", nr: " + nr + ")";}
}

public class B extends A {
  public B () {
    nr = anzahl;}
  public B (int nr) {
    this.nr = nr;}
  public void drucke () {
    System.out.println ("B" + this);}
}

public class M {
  public static void main(String[] argumente) {
    B x = new B(10);
    x.drucke();                                    // AUSGABE:
    A y = new B();
    y.drucke();                                    // AUSGABE:
    A z = y;
    z.nr = z.nr + B.anzahl;
    z.drucke();                                    // AUSGABE:
    y.drucke();}                                   // AUSGABE:
}
```

(b) Die Klasse `B` wird um die Methode `f` erweitert. Finden und erklären Sie die drei Fehler in dieser Methode, die das Compilieren verhindern.

```
public boolean f (A a) {
  A x = new A ();
  B y = a;
  int n = A.nr;
  return n < y.nr;
}
```

Vorname	Name	Matr.-Nr.

Vorname	Name	Matr.-Nr.

Aufgabe 2 (Verifikation, 10 + 1 Punkte)

Der Algorithmus P berechnet den Quotienten zweier natürlicher Zahlen $\mathtt{x}$ und $\mathtt{y}$, wobei $\mathtt{x}$ durch $\mathtt{y}$ teilbar ist.

(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: $\mathtt{x}, \mathtt{y} \in \mathbb{N} = \{0, 1, 2, ...\}$
Ausgabe: $\mathtt{res}$
Vorbedingung: $\mathtt{x} \geq 0 \wedge \mathtt{y} > 0 \wedge \mathtt{y} | \mathtt{x}$ (wobei $\mathtt{y} | \mathtt{x}$ für die Aussage "y teilt x" steht)
Nachbedingung: $\mathtt{y} * \mathtt{res} = \mathtt{x}$

$\langle\ \mathtt{x} \geq 0 \wedge \mathtt{y} > 0 \wedge \mathtt{y} | \mathtt{x}\ \rangle$

$\langle\ \mathtt{x} \geq 0 \wedge \mathtt{y} > 0 \wedge \mathtt{y} | \mathtt{x} \wedge$ _____ $\rangle$

```
z = x;
```

$\langle\ \mathtt{x} \geq 0 \wedge \mathtt{y} > 0 \wedge \mathtt{y} | \mathtt{x} \wedge$ _____ $\rangle$

$\langle\ \mathtt{x} \geq 0 \wedge \mathtt{y} > 0 \wedge \mathtt{y} | \mathtt{x} \wedge$ _____ $\rangle$

```
res = 0;
```

$\langle\ \mathtt{x} \geq 0 \wedge \mathtt{y} > 0 \wedge \mathtt{y} | \mathtt{x} \wedge \mathtt{z} = \mathtt{x} \wedge \mathtt{res} = 0\ \rangle$

$\langle\ \mathtt{x} \geq 0 \wedge \mathtt{y} > 0 \wedge \mathtt{y} | \mathtt{x} \wedge$ _____ $\rangle$

```
while (z != 0) {
```

$\langle\ \mathtt{x} \geq 0 \wedge \mathtt{y} > 0 \wedge \mathtt{y} | \mathtt{x} \wedge \mathtt{z} \neq 0 \wedge$ _____ $\rangle$

$\langle\ \mathtt{x} \geq 0 \wedge \mathtt{y} > 0 \wedge \mathtt{y} | \mathtt{x} \wedge$ _____ $\rangle$

```
  res = res + 1;
```

$\langle \mathtt{x} \geq 0 \wedge \mathtt{y} > 0 \wedge \mathtt{y} | \mathtt{x} \wedge$ _____ $\rangle$

```
  z = z - y;
```

$\langle\ \mathtt{x} \geq 0 \wedge \mathtt{y} > 0 \wedge \mathtt{y} | \mathtt{x} \wedge$ _____ $\rangle$

```
}
```

$\langle\ \mathtt{x} \geq 0 \wedge \mathtt{y} > 0 \wedge \mathtt{y} | \mathtt{x} \wedge$ _____ $\rangle$

$\langle \mathtt{y} * \mathtt{res} = \mathtt{x} \rangle$

Vorname	Name	Matr.-Nr.

5

(b) Geben Sie (ohne Beweis) eine Variante für die `while`-Schleife an, die belegt, dass die Schleife für $\mathtt{x} \geq \mathtt{0} \wedge \mathtt{y} > \mathtt{0} \wedge \mathtt{y} | \mathtt{x}$ terminiert.

Vorname	Name	Matr.-Nr.

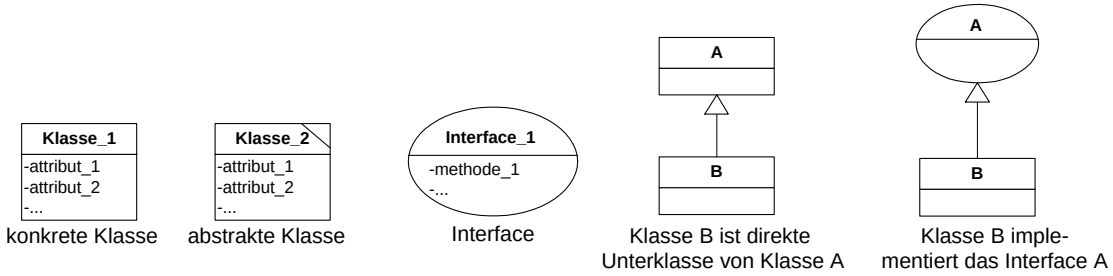
Aufgabe 3 (Datenstrukturen in Java, 5 + 3 + 6 Punkte)

Sie haben die Aufgabe, ein Programm zu entwickeln, mit dem Tonträger verwaltet werden können. Dabei stellen Sie fest, dass es die folgenden vier verschiedenen Arten von Tonträgern gibt:

- Tonband: dieses ist charakterisiert durch Angabe von Titel, Interpret, Plattenfirma, Bandlänge und Spurgröße
- Schallplatte: diese ist charakterisiert durch Angabe von Titel, Interpret, Plattenfirma, Aufzeichnungsart und Abspielgeschwindigkeit
- CD: diese ist charakterisiert durch Angabe von Titel, Interpret, Plattenfirma, Aufzeichnungsart und Kapazität
- Kassette: diese ist charakterisiert durch Angabe von Titel, Interpret, Plattenfirma, Bandlänge und Bandart

(a) Entwerfen Sie eine geeignete Klassenhierarchie, um die oben aufgelisteten Arten von Tonträgern zu implementieren.

Achten Sie dabei darauf, dass gemeinsame Merkmale in abstrakten Klassen zusammengefasst werden. Notieren Sie Ihren Entwurf grafisch und verwenden Sie dazu die folgende Notation (geben Sie lediglich pro Klasse den Namen der Klasse und die Namen ihrer Attribute an und pro Interface den Namen des Interfaces und die Namen seiner Methoden). Verwenden Sie ausschließlich den Typ String für die Attribute.



Vorname	Name	Matr.-Nr.

(b) Implementieren Sie in der Klasse `Kassette` und in allen ihren Oberklassen, die sie definiert haben, je eine Methode `public String toString()`, welche eine String-Repräsentation aller Merkmale eines Tonträgers zurückliefert. Verwenden Sie Vererbungsmechanismen soweit wie möglich.

(c) Implementieren Sie eine Methode `schallplattenfilter`, die ein Array von Tonträgern als Eingabe bekommt. Die Methode soll untersuchen, welche dieser Tonträger Schallplatten sind. Diese sollen als ein Array von Schallplatten (richtiger Länge) als Resultat zurückgeliefert werden.

Vorname	Name	Matr.-Nr.

Aufgabe 4 (Programmierung in Java, 8 + 10 Punkte)

In dieser Aufgabe soll ein Städteverbindungsnetz realisiert werden. Durch eingehende Analyse wurde eine Schnittstellenspezifikation ermittelt, von der hier ein Teil angegeben ist.

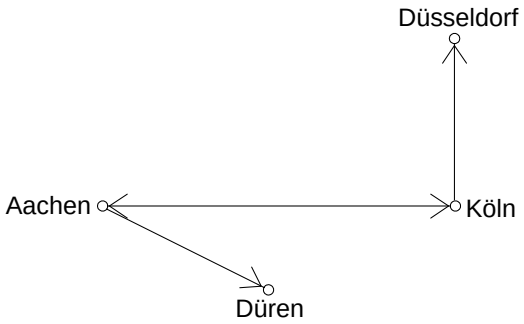
- **Klasse:** `public class Stadt`
  Methode:
```
public boolean gleich (Stadt s)
  /* Prüft, ob s gleich der aktuellen Stadt ist */
```

- **Klasse:** `public class Stadtnetz`
  Attribute:
```
private Stadtelement kopf
```
  Methoden:
```
public void stadtEinfuegen (Stadt s)
  /* Fügt eine neue Stadt in das Stadtnetz ein */
public void verbindungEinfuegen (Stadt von, Stadt nach)
  /* Fügt eine Verbindung zwischen zwei Städten ein */
```

- **Klasse:** `public class Verbindungselement`
  Attribute:
```
private Stadt wert
private Verbindungselement nextVerbindung
```
  Konstruktor:
```
public Verbindungselement (Stadt s, Verbindungselement next)
  /* Erzeugt neues Verbindungselement mit Attributen s und next */
```
  Methoden:
```
public Stadt getWert ()
public void setWert (Stadt wert)
public Verbindungselement getNextVerbindung ()
public void setNextVerbindung (Verbindungselement next)
  /* Selektoren für die Attribute */
```

- **Klasse:** `public class Stadtelement extends Verbindungselement`
  Attribute:
```
private Stadtelement nextStadtelement
```
  Konstruktor:
```
public Stadtelement (Stadt s, Verbindungselement vnext,
                     Stadtelement snext)
  /* Erzeugt neues Stadtelement mit Attributen s, vnext und snext */
```
  Methoden:
```
public Stadtelement getNextStadtelement ()
public void setNextStadtelement (Stadtelement next)
  /* Selektoren für die Attribute */
```

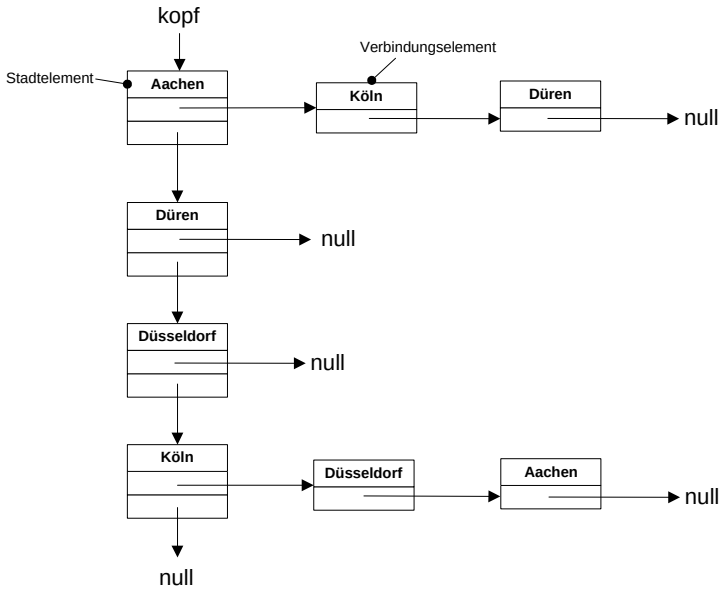
Als Beispiel betrachten wir das folgende Städteverbindungsnetz:



Dieses Städteverbindungsnetz könnte wie folgt realisiert werden.

Ein Stadtnetz ist also im Prinzip eine Liste von Städten, wobei aber zu jeder Stadt `A` eine der Liste der Städte $B_1, \ldots, B_n$ gespeichert wird, die man von ihr aus erreichen kann (d.h., bei denen es Verbindungen von `A` nach B_1, von `A` nach B_2, ..., von `A` nach B_n gibt).

Vorname	Name	Matr.-Nr.



(a) Implementieren Sie die Methode `public void stadtEinfuegen (Stadt s)` der Klasse `Stadtnetz` in Java. Die einzufügende Stadt soll dabei an das Ende der bereits existierenden Liste angehängt werden. Hierbei spielt es keine Rolle, ob die Stadt bereits in dem `Stadtnetz` auftritt. Die Implementierung muss *rekursiv* (ohne Verwendung von Schleifen) realisiert werden. Hierbei dürfen Sie Hilfsmethoden definieren, die aber nach außen nicht sichtbar sein sollen.

Vorname	Name	Matr.-Nr.

10

(b) Implementieren Sie die Methode `public void verbindungEinfuegen(Stadt von, Stadt nach)` der Klasse `Stadtnetz` in Java. Auch hier soll eine neue Verbindung an das Ende der Liste angehängt werden und es spielt keine Rolle, ob die Verbindung bereits in dem `Stadtnetz` auftritt. Gehen Sie davon aus, dass beide übergebenen Städte im Stadtnetz vorhanden sind. Hierbei dürfen Sie wieder Hilfsmethoden definieren, die aber nach außen nicht sichtbar sein sollen.

Vorname	Name	Matr.-Nr.

Aufgabe 5 (Funktionale Programmierung in Haskell, $4 + 2 + 3 + 4 + 2 + 8$ Punkte)

(a) Die Funktionen `f`, `g`, und `h` sind wie folgt definiert:

```
f x = x + 1

g x = [2, 4] ++ x

h x =  \y -> [x] : [[y]]
```

Geben Sie den allgemeinsten Typ von `f`, `g` und `h` an. Gehen Sie hierbei davon aus, dass `2` und `4` den Typ `Int` haben und `+` den Typ `Int -> Int -> Int` hat.

(b) Geben Sie für die folgenden Ausdrücke jeweils das Ergebnis der Auswertung an.

(i) `(\x -> x 2) (\x -> x * 3)`

(ii) `map (\x -> x * 2) [2, 3, 4]`

(iii) `filter (\x -> x + 3 == 7) (map (\x -> x * 2) [2, 3, 4])`

(c) Schreiben Sie eine Funktion `nth :: Int -> [a] -> a`, die zu einer Zahl `x` mit $1 \leq x \leq n$ und einer Liste $[e_1, \ldots, e_n]$ das `x`-te Element e_x liefert. Beispielsweise berechnet `nth 3 ['a','b','c']` das Ergebnis `'c'`. Hierbei dürfen Sie keine in Haskell vordefinierten Funktionen auf Listen außer den Datenkonstruktoren `[]` und `:` verwenden.

(d) Eine Tabelle besteht aus Zeilen und Spalten, die Werte beliebigen Typs aufnehmen können. Jede Position in der Tabelle ist durch ihre Zeilen- und Spaltennummer gekennzeichnet. Eine Zeile kann als Liste von Einträgen realisiert werden und eine Tabelle kann dann als Liste von Zeilen dargestellt werden. Insgesamt lassen sich also Tabellen durch Listen von Listen wie folgt realisieren: Die Tabelle

	1	2	3
1	'a'	'b'	'c'
2	'd'	'e'	'f'

wird dann durch `[ ['a','b','c'], ['d','e','f'] ]` repräsentiert. Der Wert `'b'` steht hierbei in der ersten Zeile und der zweiten Spalte (d.h. an der Position `(1,2)`) und `'f'` steht in der zweiten Zeile und der dritten Spalte, d.h. an Position `(2,3)`.

Implementieren Sie in Haskell eine Funktion `summe :: Int -> [[Int]] -> Int`, die die Summe der Einträge einer vorgegebenen Spalte in der Tabelle liefert. Beispielsweise ergibt also `summe 2 [ [10,2,33,14], [4,5,6,8] ]` das Resultat `7`. In dieser Teilaufgabe betrachten wir nur Tabellen zur Speicherung von `Int`-Werten.

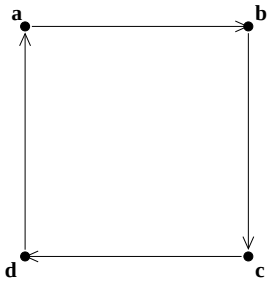
(e) Definieren Sie in Haskell eine solche Datenstruktur zur Repräsentation von Tabellen mit Einträgen beliebigen Typs, welche der Beschreibung in Teil (d) entspricht. Durch Ihre Datenstruktur muss sichergestellt sein, dass an jeder Position der Tabelle nur ein Eintrag stehen kann. Hierbei dürfen Sie jedoch die in Haskell vordefinierten Listen *nicht* verwenden.

(f) Implementieren Sie in Haskell eine Funktion `suche`, die zu einem Eintrag und einer Tabelle eine Position angibt, an der dieser Eintrag in der Tabelle steht. Positionen sind hierbei wieder Paare von Zahlen `(n,m)`, wobei `n` die Zeilen- und `m` die Spaltennummer ist. Falls die Tabelle den Eintrag nicht enthält, soll `(0,0)` zurückgegeben werden. Verwenden Sie hierbei Ihre Datenstruktur aus Teil (e) und geben Sie auch die Typdeklaration von `suche` an. Sie dürfen davon ausgehen, dass der Vergleichsoperator `(==)` für alle Typen vordefiniert ist.
Hinweis: Implementieren Sie gegebenenfalls geeignete Hilfsfunktionen.

Vorname	Name	Matr.-Nr.

Aufgabe 6 (Logische Programmierung in Prolog, $1 + 3 + 2 + 5 + 5$ Punkte)

(a) Gegeben sei folgender Graph mit den Knoten **a**, **b**, **c**, **d**.



Erstellen Sie ein Prolog-Programm, in dem dieser Graph mit Hilfe eines zweistelligen Prädikats `kante` repräsentiert wird.

(b) Definieren Sie in Prolog ein einstelliges Prädikat `pfad`, wobei `pfad(L)` ausdrückt, dass die Liste `L` einen Pfad in dem Graph repräsentiert.
Ein Pfad in einem Graph `G` ist eine Liste $[\mathtt{V}_1, \ldots, \mathtt{V}_n]$ von Knoten aus `G` (mit $\mathtt{n} \geq 0$), so dass eine Kante von $\mathtt{V}_i$ nach $\mathtt{V}_{i+1}$ in `G` existiert für alle $\mathtt{i} \in \{1, .., \mathtt{n} - 1\}$. Im Beispiel-Graphen gälte also z.B. `pfad([a,b,c])`.

Vorname	Name	Matr.-Nr.

(c) Formulieren Sie (basierend auf dem zuvor definierten Prädikat) eine Anfrage, um alle Pfade zu berechnen, die aus mindestens zwei Knoten bestehen und im Knoten `a` beginnen.

(d) Definieren Sie in Prolog ein zweistelliges Prädikat `min`, wobei `min(X,L)` ausdrückt, dass `X` das Minimum einer Liste `L` ist. Gehen Sie hierbei davon aus, dass `L` eine Liste von Zahlen ist. Es gilt also z.B. `min(2, [5,2,7])`.
Hinweis: Es gibt in Prolog vordefinierte Prädikate `<=` und `>=`.

(e) Betrachten Sie folgendes Prolog-Programm:

```
nachfolger(X,succ(X)).
nachfolger(X,Y) :- nachfolger(X,Z), nachfolger(Z,Y).
```

Terminiert das Programm bei der Suche nach den ersten beiden Antworten auf die folgende Anfrage? Geben Sie ggf. die erste bzw. die ersten beiden Antworten an.

```
?- nachfolger(zero, U).
```

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

Probeklausur Lösungsvorschlag
Informatik I - Programmierung
18. 2. 2002

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 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 ist.
- 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	18	
Aufgabe 2	11	
Aufgabe 3	14	
Aufgabe 4	18	
Aufgabe 5	23	
Aufgabe 6	16	
Summe	100	
Note	-	

Vorname	Name	Matr.-Nr.

Aufgabe 1 (Programmanalyse, 12 + 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 abstract class A {

  public static int anzahl = 0;
  public int nr = 0;

  public A () {
    anzahl = anzahl + 1;}
  public void drucke () {
    System.out.println ("A" + this);}
  public String toString() {
    return "(anzahl: " + anzahl + ", nr: " + nr + ")";}
}

public class B extends A {
  public B () {
    nr = anzahl;}
  public B (int nr) {
    this.nr = nr;}
  public void drucke () {
    System.out.println ("B" + this);}
}

public class M {
  public static void main(String[] argumente) {
    B x = new B(10);
    x.drucke();                                  // AUSGABE: B(anzahl: 1, nr: 10)
    A y = new B();
    y.drucke();                                  // AUSGABE: B(anzahl: 2, nr: 2)
    A z = y;
    z.nr = z.nr + B.anzahl;
    z.drucke();                                  // AUSGABE: B(anzahl: 2, nr: 4)
    y.drucke();}                                 // AUSGABE: B(anzahl: 2, nr: 4)
}
```

(b) Die Klasse `B` wird um die Methode `f` erweitert. Finden und erklären Sie die drei Fehler in dieser Methode, die das Compilieren verhindern.

```
public boolean f (A a) {
  A x = new A ();
  B y = a;
  int n = A.nr;
  return n < y.nr;
}
```

Vorname	Name	Matr.-Nr.

- `A x = new A ();` ist nicht erlaubt, da man keine Objekte einer abstrakten Klasse erzeugen kann.
- `B y = a;` ist nicht erlaubt, da man explizit von der Oberklasse A zur Unterklasse B konvertieren muss. Es müsste also z.B. `B y = (B) a` heißen.
- `A.nr` ist nicht erlaubt, denn `nr` ist ein nicht-statisches (Objekt-)Attribut.

Vorname	Name	Matr.-Nr.

Aufgabe 2 (Verifikation, 10 + 1 Punkte)

Der Algorithmus P berechnet den Quotienten zweier natürlicher Zahlen $\mathtt{x}$ und $\mathtt{y}$, wobei $\mathtt{x}$ durch $\mathtt{y}$ teilbar ist.

(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: $\mathtt{x}, \mathtt{y} \in \mathbb{N} = \{0, 1, 2, ...\}$
Ausgabe: $\mathtt{res}$
Vorbedingung: $\mathtt{x} \geq 0 \wedge \mathtt{y} > 0 \wedge \mathtt{y}|\mathtt{x}$ (wobei $\mathtt{y}|\mathtt{x}$ für die Aussage "$\mathtt{y}$ teilt $\mathtt{x}$" steht)
Nachbedingung: $\mathtt{y} * \mathtt{res} = \mathtt{x}$

$\langle\, \mathtt{x} \geq 0 \wedge \mathtt{y} > 0 \wedge \mathtt{y}|\mathtt{x} \,\rangle$

$\langle\, \mathtt{x} \geq 0 \wedge \mathtt{y} > 0 \wedge \mathtt{y}|\mathtt{x} \wedge \mathtt{x} = \mathtt{x} \,\rangle$

```
z = x;
```

$\langle\, \mathtt{x} \geq 0 \wedge \mathtt{y} > 0 \wedge \mathtt{y}|\mathtt{x} \wedge \mathtt{z} = \mathtt{x} \,\rangle$

$\langle\, \mathtt{x} \geq 0 \wedge \mathtt{y} > 0 \wedge \mathtt{y}|\mathtt{x} \wedge \mathtt{z} = \mathtt{x} \wedge 0 = 0 \,\rangle$

```
res = 0;
```

$\langle\, \mathtt{x} \geq 0 \wedge \mathtt{y} > 0 \wedge \mathtt{y}|\mathtt{x} \wedge \mathtt{z} = \mathtt{x} \wedge \mathtt{res} = 0 \,\rangle$

$\langle\, \mathtt{x} \geq 0 \wedge \mathtt{y} > 0 \wedge \mathtt{y}|\mathtt{x} \wedge \mathtt{y} * \mathtt{res} + \mathtt{z} = \mathtt{x} \,\rangle$

```
while (z != 0) {
```

$\langle\, \mathtt{x} \geq 0 \wedge \mathtt{y} > 0 \wedge \mathtt{y}|\mathtt{x} \wedge \mathtt{z} \neq 0 \wedge \mathtt{y} * \mathtt{res} + \mathtt{z} = \mathtt{x} \,\rangle$

$\langle\, \mathtt{x} \geq 0 \wedge \mathtt{y} > 0 \wedge \mathtt{y}|\mathtt{x} \wedge \mathtt{y} * (\mathtt{res} + 1) + (\mathtt{z} - \mathtt{y}) = \mathtt{x} \,\rangle$

```
  res = res + 1;
```

$\langle \mathtt{x} \geq 0 \wedge \mathtt{y} > 0 \wedge \mathtt{y}|\mathtt{x} \wedge \mathtt{y} * \mathtt{res} + (\mathtt{z} - \mathtt{y}) = \mathtt{x} \,\rangle$

```
  z = z - y;
```

$\langle\, \mathtt{x} \geq 0 \wedge \mathtt{y} > 0 \wedge \mathtt{y}|\mathtt{x} \wedge \mathtt{y} * \mathtt{res} + \mathtt{z} = \mathtt{x} \,\rangle$

```
}
```

$\langle\, \mathtt{x} \geq 0 \wedge \mathtt{y} > 0 \wedge \mathtt{y}|\mathtt{x} \wedge \mathtt{z} = 0 \wedge \mathtt{y} * \mathtt{res} + \mathtt{z} = \mathtt{x} \,\rangle$

$\langle \mathtt{y} * \mathtt{res} = \mathtt{x} \rangle$

Vorname	Name	Matr.-Nr.

5

(b) Geben Sie (ohne Beweis) eine Variante für die `while`-Schleife an, die belegt, dass die Schleife für $\mathtt{x} \geq \mathtt{0} \wedge \mathtt{y} > \mathtt{0} \wedge \mathtt{y} | \mathtt{x}$ terminiert.

Die Variante ist z. Da nach Voraussetzung $y > 0$ gilt, verkleinert sich der Wert von z mit jedem Schleifendurchlauf.

Vorname	Name	Matr.-Nr.

6

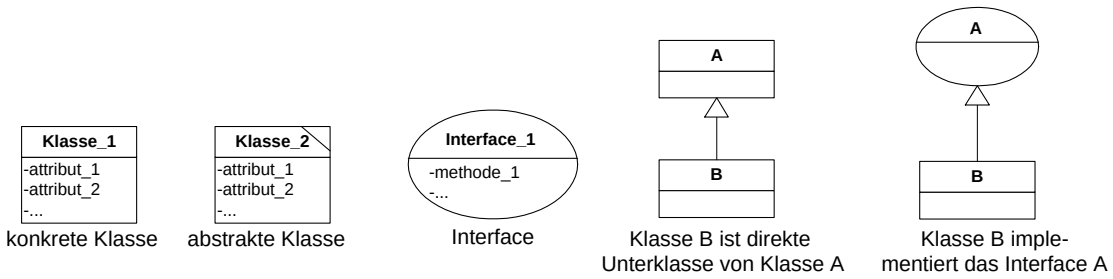
Aufgabe 3 (Datenstrukturen in Java, 5 + 3 + 6 Punkte)

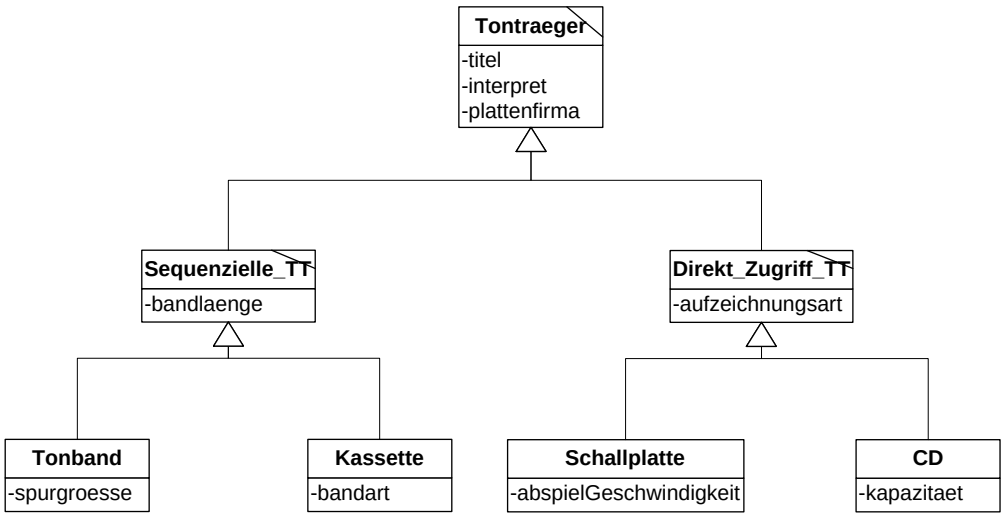
Sie haben die Aufgabe, ein Programm zu entwickeln, mit dem Tonträger verwaltet werden können. Dabei stellen Sie fest, dass es die folgenden vier verschiedenen Arten von Tonträgern gibt:

- Tonband: dieses ist charakterisiert durch Angabe von Titel, Interpret, Plattenfirma, Bandlänge und Spurgröße
- Schallplatte: diese ist charakterisiert durch Angabe von Titel, Interpret, Plattenfirma, Aufzeichnungsart und Abspielgeschwindigkeit
- CD: diese ist charakterisiert durch Angabe von Titel, Interpret, Plattenfirma, Aufzeichnungsart und Kapazität
- Kassette: diese ist charakterisiert durch Angabe von Titel, Interpret, Plattenfirma, Bandlänge und Bandart

(a) Entwerfen Sie eine geeignete Klassenhierarchie, um die oben aufgelisteten Arten von Tonträgern zu implementieren.

Achten Sie dabei darauf, dass gemeinsame Merkmale in abstrakten Klassen zusammengefasst werden. Notieren Sie Ihren Entwurf grafisch und verwenden Sie dazu die folgende Notation (geben Sie lediglich pro Klasse den Namen der Klasse und die Namen ihrer Attribute an und pro Interface den Namen des Interfaces und die Namen seiner Methoden). Verwenden Sie ausschließlich den Typ String für die Attribute.





Vorname	Name	Matr.-Nr.

(b) Implementieren Sie in der Klasse `Kassette` und in allen ihren Oberklassen, die sie definiert haben, je eine Methode `public String toString()`, welche eine String-Repräsentation aller Merkmale eines Tonträgers zurückliefert. Verwenden Sie Vererbungsmechanismen soweit wie möglich.

In `Tontraeger`:

```
public String toString() {
  return titel + ", " + interpret + ", " + plattenfirma;
}
```

In `Sequenzielle_TT`:

```
public String toString() {
  return super.toString() + ", " + bandlaenge;
}
```

In `Kassette`:

```
public String toString() {
  return super.toString() + ", " + bandart;
}
```

(c) Implementieren Sie eine Methode `schallplattenfilter`, die ein Array von Tonträgern als Eingabe bekommt. Die Methode soll untersuchen, welche dieser Tonträger Schallplatten sind. Diese sollen als ein Array von Schallplatten (richtiger Länge) als Resultat zurückgeliefert werden.

```
public static Schallplatte[] schallplattenfilter (Tontraeger[] t) {


   int laenge = 0,
       j = 0;

   if (t == null)
      return null;

   for (int i = 0; i < t.length; i++)
       if (t[i] instanceof Schallplatte)
          laenge++;

   Schallplatte[] s = new Schallplatte[laenge];

   for (int i = 0; i < t.length; i++)
       if (t[i] instanceof Schallplatte) {
          s[j] = (Schallplatte) t[i];
          j++;
       }

   return s;

}
```

Vorname	Name	Matr.-Nr.

Aufgabe 4 (Programmierung in Java, 8 + 10 Punkte**)**

In dieser Aufgabe soll ein Städteverbindungsnetz realisiert werden. Durch eingehende Analyse wurde eine Schnittstellenspezifikation ermittelt, von der hier ein Teil angegeben ist.

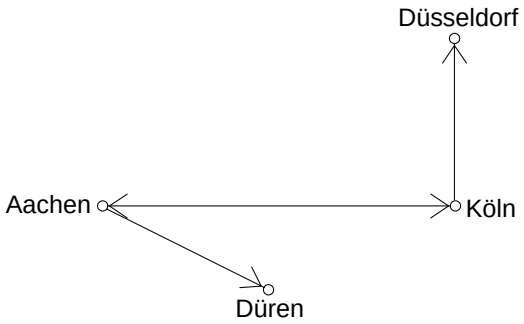
- **Klasse:** `public class Stadt`
  Methode:
```
public boolean gleich (Stadt s)
  /* Prüft, ob s gleich der aktuellen Stadt ist */
```

- **Klasse:** `public class Stadtnetz`
  Attribute: `private Stadtelement kopf`
  Methoden:
```
public void stadtEinfuegen (Stadt s)
  /* Fügt eine neue Stadt in das Stadtnetz ein */
public void verbindungEinfuegen (Stadt von, Stadt nach)
  /* Fügt eine Verbindung zwischen zwei Städten ein */
```

- **Klasse:** `public class Verbindungselement`
  Attribute:
```
private Stadt wert
private Verbindungselement nextVerbindung
```
  Konstruktor:
```
public Verbindungselement (Stadt s, Verbindungselement next)
  /* Erzeugt neues Verbindungselement mit Attributen s und next */
```
  Methoden:
```
public Stadt getWert ()
public void setWert (Stadt wert)
public Verbindungselement getNextVerbindung ()
public void setNextVerbindung (Verbindungselement next)
  /* Selektoren für die Attribute */
```

- **Klasse:** `public class Stadtelement extends Verbindungselement`
  Attribute: `private Stadtelement nextStadtelement`
  Konstruktor:
```
public Stadtelement (Stadt s, Verbindungselement vnext,
                     Stadtelement snext)
  /* Erzeugt neues Stadtelement mit Attributen s, vnext und snext */
```
  Methoden:
```
public Stadtelement getNextStadtelement ()
public void setNextStadtelement (Stadtelement next)
  /* Selektoren für die Attribute */
```

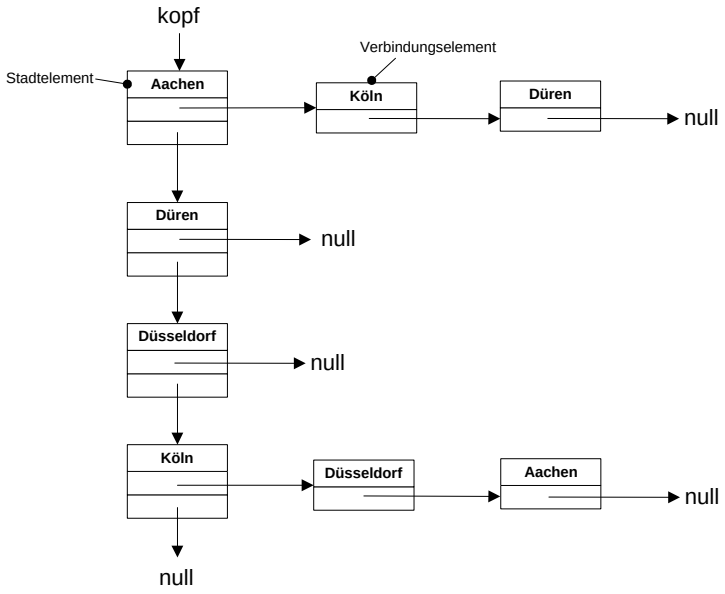
Als Beispiel betrachten wir das folgende Städteverbindungsnetz:



Dieses Städteverbindungsnetz könnte wie folgt realisiert werden.

Ein Stadtnetz ist also im Prinzip eine Liste von Städten, wobei aber zu jeder Stadt `A` eine der Liste der Städte $B_1, \ldots, B_n$ gespeichert wird, die man von ihr aus erreichen kann (d.h., bei denen es Verbindungen von `A` nach B_1, von `A` nach B_2, ..., von `A` nach B_n gibt).

Vorname	Name	Matr.-Nr.



(a) Implementieren Sie die Methode `public void stadtEinfuegen (Stadt s)` der Klasse `Stadtnetz` in Java. Die einzufügende Stadt soll dabei an das Ende der bereits existierenden Liste angehängt werden. Hierbei spielt es keine Rolle, ob die Stadt bereits in dem `Stadtnetz` auftritt. Die Implementierung muss *rekursiv* (ohne Verwendung von Schleifen) realisiert werden. Hierbei dürfen Sie Hilfsmethoden definieren, die aber nach außen nicht sichtbar sein sollen.

```
public void stadtEinfuegen (Stadt s) {

    if   (kopf == null)
         kopf = new Stadtelement (s, null, null);
    else stadtEinfuegen (s, kopf);
}

private static void stadtEinfuegen (Stadt s, Stadtelement element) {

    if   (element.getNextStadtelement() == null)
         element.setNextStadtelement (new Stadtelement (s, null, null));
    else stadtEinfuegen (s, element.getNextStadtelement());

}
```

Vorname	Name	Matr.-Nr.

10

(b) Implementieren Sie die Methode `public void verbindungEinfuegen(Stadt von, Stadt nach)` der Klasse `Stadtnetz` in Java. Auch hier soll eine neue Verbindung an das Ende der Liste angehängt werden und es spielt keine Rolle, ob die Verbindung bereits in dem `Stadtnetz` auftritt. Gehen Sie davon aus, dass beide übergebenen Städte im Stadtnetz vorhanden sind. Hierbei dürfen Sie wieder Hilfsmethoden definieren, die aber nach außen nicht sichtbar sein sollen.

```
public void verbindungEinfuegen (Stadt von, Stadt nach) {

    Verbindungselement verb;
    Stadtelement element = kopf;

    while (! element.getWert().gleich(von))
        element = element.getNextStadtelement();

    verb = element;

    while (verb.getNextVerbindung() != null)
        verb = verb.getNextVerbindung();

    verb.setNextVerbindung (new Verbindungselement(nach, null));
}
```

Vorname	Name	Matr.-Nr.

Aufgabe 5 (Funktionale Programmierung in Haskell, $4 + 2 + 3 + 4 + 2 + 8$ Punkte)

(a) Die Funktionen `f`, `g`, und `h` sind wie folgt definiert:

```
f x = x + 1

g x = [2, 4] ++ x

h x =  \y -> [x] : [[y]]
```

Geben Sie den allgemeinsten Typ von `f`, `g` und `h` an. Gehen Sie hierbei davon aus, dass `2` und `4` den Typ `Int` haben und `+` den Typ `Int -> Int -> Int` hat.

Der allgemeinste Typ von `f` ist `Int -> Int`, der allgemeinste Typ von `g` ist `[Int] -> [Int]` und der allgemeinste Typ von `h` ist `a -> a -> [[a]]`.

(b) Geben Sie für die folgenden Ausdrücke jeweils das Ergebnis der Auswertung an.

(i) `(\x -> x 2) (\x -> x * 3)`

(ii) `map (\x -> x * 2) [2, 3, 4]`

(iii) `filter (\x -> x + 3 == 7) (map (\x -> x * 2) [2, 3, 4])`

(i) `6`

(ii) `[4,6,8]`

(iii) `[4]`

(c) Schreiben Sie eine Funktion `nth :: Int -> [a] -> a`, die zu einer Zahl `x` mit $1 \leq x \leq n$ und einer Liste $[e_1, \ldots, e_n]$ das `x`-te Element e_x liefert. Beispielsweise berechnet `nth 3 ['a','b','c']` das Ergebnis `'c'`. Hierbei dürfen Sie keine in Haskell vordefinierten Funktionen auf Listen außer den Datenkonstruktoren `[]` und `:` verwenden.

```
nth :: Int -> [a] -> a
nth 1     (y:ys) = y
nth (x+1) (y:ys) = nth x ys
```

(d) Eine Tabelle besteht aus Zeilen und Spalten, die Werte beliebigen Typs aufnehmen können. Jede Position in der Tabelle ist durch ihre Zeilen- und Spaltennummer gekennzeichnet. Eine Zeile kann als Liste von Einträgen realisiert werden und eine Tabelle kann dann als Liste von Zeilen dargestellt werden. Insgesamt lassen sich also Tabellen durch Listen von Listen wie folgt realisieren: Die Tabelle

	1	2	3
1	'a'	'b'	'c'
2	'd'	'e'	'f'

Vorname	Name	Matr.-Nr.

wird dann durch `[ ['a','b','c'], ['d','e','f'] ]` repräsentiert. Der Wert `'b'` steht hierbei in der ersten Zeile und der zweiten Spalte (d.h. an der Position `(1,2)`) und `'f'` steht in der zweiten Zeile und der dritten Spalte, d.h. an Position `(2,3)`.

Implementieren Sie in Haskell eine Funktion `summe :: Int -> [[Int]] -> Int`, die die Summe der Einträge einer vorgegebenen Spalte in der Tabelle liefert. Beispielsweise ergibt also `summe 2 [ [10,2,33,14], [4,5,6,8] ]` das Resultat `7`. In dieser Teilaufgabe betrachten wir nur Tabellen zur Speicherung von `Int`-Werten.

```
summe :: Int -> [[Int]] -> Int
summe x []           = 0
summe x (zeile:rest) = nth x zeile + summe x rest
```

(e) Definieren Sie in Haskell eine solche Datenstruktur zur Repräsentation von Tabellen mit Einträgen beliebigen Typs, welche der Beschreibung in Teil (d) entspricht. Durch Ihre Datenstruktur muss sichergestellt sein, dass an jeder Position der Tabelle nur ein Eintrag stehen kann. Hierbei dürfen Sie jedoch die in Haskell vordefinierten Listen *nicht* verwenden.

```
data Tab a = EmptyTab | AddRow (Row a) (Tab a) deriving Show

data Row a = EmptyRow | AddEntry a (Row a) deriving Show
```

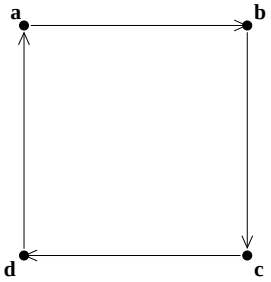
(f) Implementieren Sie in Haskell eine Funktion `suche`, die zu einem Eintrag und einer Tabelle eine Position angibt, an der dieser Eintrag in der Tabelle steht. Positionen sind hierbei wieder Paare von Zahlen `(n,m)`, wobei `n` die Zeilen- und `m` die Spaltennummer ist. Falls die Tabelle den Eintrag nicht enthält, soll `(0,0)` zurückgegeben werden. Verwenden Sie hierbei Ihre Datenstruktur aus Teil (e) und geben Sie auch die Typdeklaration von `suche` an. Sie dürfen davon ausgehen, dass der Vergleichsoperator `(==)` für alle Typen vordefiniert ist.
Hinweis: Implementieren Sie gegebenenfalls geeignete Hilfsfunktionen.

```
suche :: a -> Tab a -> (Int,Int)
suche x t = sucheVon x t 1

  where
       {- "sucheVon x t n" berechnet (n-1+zeile,spalte), falls x in der Tabelle
            t an der Position (zeile,spalte) auftritt und (0,0) sonst. -}
        sucheVon :: a -> Tab a -> Int -> (Int,Int)
        sucheVon x EmptyTab n = (0,0)
        sucheVon x (AddRow zeile rest) n | spalte == 0 = sucheVon x rest (n+1)
                                         | otherwise   = (n,spalte)
                                        where spalte = sucheInZeile x zeile 1

        {- "sucheInZeile x z m" berechnet m-1+spalte, falls x in der Zeile z
           an der Stelle "spalte" auftritt und 0 sonst -}
        sucheInZeile :: a -> Row a -> Int -> Int
        sucheInZeile x EmptyRow m = 0
        sucheInZeile x (AddEntry y rest) m | x == y = m
                                           | otherwise = sucheInZeile x rest (m+1)
```

Vorname	Name	Matr.-Nr.

Aufgabe 6 (Logische Programmierung in Prolog, 1 + 3 + 2 + 5 + 5 Punkte)

(a) Gegeben sei folgender Graph mit den Knoten **a**, **b**, **c**, **d**.



Erstellen Sie ein Prolog-Programm, in dem dieser Graph mit Hilfe eines zweistelligen Prädikats `kante` repräsentiert wird.

```
kante(a,b).
kante(b,c).
kante(c,d).
kante(d,a).
```

(b) Definieren Sie in Prolog ein einstelliges Prädikat `pfad`, wobei `pfad(L)` ausdrückt, dass die Liste `L` einen Pfad in dem Graph repräsentiert.
Ein Pfad in einem Graph `G` ist eine Liste $[V_1, \ldots, V_n]$ von Knoten aus `G` (mit $n \geq 0$), so dass eine Kante von V_i nach V_{i+1} in `G` existiert für alle $i \in \{1, .., n-1\}$. Im Beispiel-Graphen gälte also z.B. `pfad([a,b,c])`.

```
pfad([]).
pfad([_]).
pfad([X,Y|R]) :- kante(X,Y), pfad([Y|R]).
```

Vorname	Name	Matr.-Nr.

(c) Formulieren Sie (basierend auf dem zuvor definierten Prädikat) eine Anfrage, um alle Pfade zu berechnen, die aus mindestens zwei Knoten bestehen und im Knoten `a` beginnen.

```
?- pfad(L), L = [a,_|_].
```

(d) Definieren Sie in Prolog ein zweistelliges Prädikat `min`, wobei `min(X,L)` ausdrückt, dass `X` das Minimum einer Liste `L` ist. Gehen Sie hierbei davon aus, dass `L` eine Liste von Zahlen ist. Es gilt also z.B. `min(2, [5,2,7])`.
Hinweis: Es gibt in Prolog vordefinierte Prädikate `=<` und `>=`.

```
min(X,[X]).
min(X,[Y,Z|L]) :- Y =< Z, min(X,[Y|L]).
min(X,[Y,Z|L]) :- Y >= Z, min(X,[Z|L]).
```

(e) Betrachten Sie folgendes Prolog-Programm:

```
nachfolger(X,succ(X)).
nachfolger(X,Y) :- nachfolger(X,Z), nachfolger(Z,Y).
```

Terminiert das Programm bei der Suche nach den ersten beiden Antworten auf die folgende Anfrage? Geben Sie ggf. die erste bzw. die ersten beiden Antworten an.

```
?- nachfolger(zero, U).
```

Das Programm liefert die beiden ersten Antworten `U = succ(zero)` und `U = succ(succ(zero))`.

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

Diplom-Vorprüfung / Zwischenprüfung Informatik I - Programmierung

6. 3. 2002

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 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 ist.
- 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	14	
Aufgabe 2	11	
Aufgabe 3	17	
Aufgabe 4	18	
Aufgabe 5	21	
Aufgabe 6	15	
Summe	96	
Note	-	

Vorname	Name	Matr.-Nr.

Aufgabe 1 (Programmanalyse, 12 + 2 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 interface Z {
  public void f (Z object);
}

public class X implements Z {
  public int wert1 = 1,
             wert2 = 2;

  public X() {
    wert1 = wert1 + wert2;
  }
  public void f(Z object) {
    object = this;
  }
}

public class Y extends X {
  public int wert2 = 3;

  public Y() {
    wert1 = wert1 + wert2;
  }
  public void f (Z object) {}
}

public class M {
  public static void main(String[] arguments) {
    X x = new X();
    Y y = new Y();
    System.out.println(x.wert1 + ", " + x.wert2);        // AUSGABE:
    System.out.println(y.wert1 + ", " + y.wert2);        // AUSGABE:
    x.f(y);
    System.out.println(y.wert1 + ", " + y.wert2);}       // AUSGABE:
}
```

(b) In der Klasse `Y` wird die Methode `f` nun durch folgenden Code ersetzt. Finden und erklären Sie den Fehler in dieser Methode, der das Compilieren verhindert.

```
protected void f (Y object) {
  Z z = new Y();
  Y y = z;
}
```

Vorname	Name	Matr.-Nr.

Vorname	Name	Matr.-Nr.

Aufgabe 2 (Verifikation, 10 + 1 Punkte)

Der Algorithmus P berechnet die Hälfte einer geraden natürlichen Zahl.

(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: $\mathtt{x} \in \{\, 2 * \mathtt{n} \mid \mathtt{n} \in \mathbb{N} \,\} = \{0, 2, 4, ...\}$
Ausgabe: `res`
Vorbedingung: $\mathtt{x} \geq 0 \wedge 2|\mathtt{x}$ (wobei $2|\mathtt{x}$ für die Aussage "2 teilt x" steht)
Nachbedingung: $2 * \mathtt{res} = \mathtt{x}$

$\langle\, \mathtt{x} \geq 0 \wedge 2|\mathtt{x} \,\rangle$

$\langle\, \mathtt{x} \geq 0 \wedge 2|\mathtt{x} \wedge$ _____ $\,\rangle$

```
z = 0;
```

$\langle\, \mathtt{x} \geq 0 \wedge 2|\mathtt{x} \wedge$ _____ $\,\rangle$

$\langle\, \mathtt{x} \geq 0 \wedge 2|\mathtt{x} \wedge$ _____ $\,\rangle$

```
res = 0;
```

$\langle\, \mathtt{x} \geq 0 \wedge 2|\mathtt{x} \wedge \mathtt{z} = 0 \wedge \mathtt{res} = 0 \,\rangle$

$\langle\, \mathtt{x} \geq 0 \wedge 2|\mathtt{x} \wedge$ _____ $\,\rangle$

```
while (z != x) {
```

$\langle\, \mathtt{x} \geq 0 \wedge 2|\mathtt{x} \wedge \mathtt{z} \neq \mathtt{x} \wedge$ _____ $\,\rangle$

$\langle\, \mathtt{x} \geq 0 \wedge 2|\mathtt{x} \wedge$ _____ $\,\rangle$

```
  res = res + 1;
```

$\langle \mathtt{x} \geq 0 \wedge 2|\mathtt{x} \wedge$ _____ $\,\rangle$

```
  z = z + 2;
```

$\langle\, \mathtt{x} \geq 0 \wedge 2|\mathtt{x} \wedge$ _____ $\,\rangle$

```
}
```

$\langle\, \mathtt{x} \geq 0 \wedge 2|\mathtt{x} \wedge$ _____ $\,\rangle$

$\langle 2 * \mathtt{res} = \mathtt{x} \rangle$

Vorname	Name	Matr.-Nr.

5

(b) Geben Sie (ohne Beweis) eine Variante für die `while`-Schleife an, die belegt, dass die Schleife für $\texttt{x} \geq \texttt{0} \wedge \texttt{2} | \texttt{x}$ terminiert.

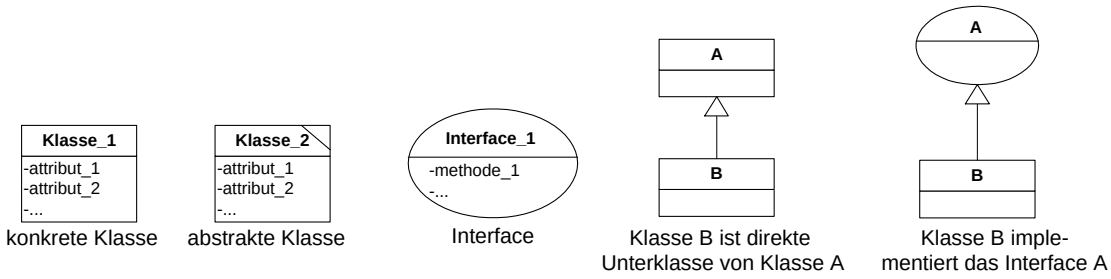
Vorname	Name	Matr.-Nr.

Aufgabe 3 (Datenstrukturen in Java, 6 + 5 + 6 Punkte**)**

Herr Müller möchte seinen Besitz inventarisieren. Dazu beschließt er, ein objektorientiertes Programm in Java zu entwickeln. Jedes seiner Besitzstücke soll eine Inventarnummer haben. Herr Müller besitzt Möbel- und Schmuckstücke. Jedes Möbelstück ist durch seine Art gekennzeichnet (z.B. “Sofa”, “Tisch”, etc.). Nur Schmuckstücke und antike Möbel sind Wertgegenstände, von denen man den Wert durch eine Methode berechnen können soll. Hierzu speichert man bei antiken Möbeln ihr Baujahr und bei Schmuckstücken ihr Gewicht (als ganze Zahl). Bei nicht antiken Möbeln ist es hingegen von Interesse, ob sie kindgerecht sind.

(a) Entwerfen Sie eine geeignete Klassenhierarchie, um den obigen Sachverhalt in Java zu realisieren. Achten Sie dabei darauf, dass gemeinsame Attribute und/oder Methoden in abstrakten Klassen bzw. Interfaces zusammengefasst werden. Verwenden Sie soweit wie möglich Vererbungstechniken.

Notieren Sie Ihren Entwurf grafisch und verwenden Sie dazu die folgende Notation: Für jede Klasse werden ihr Name und die Namen ihrer Attribute angegeben. (Methoden werden bei Klassen nicht mit aufgeführt.) Für jedes Interface werden sein Name und die Namen seiner Methoden angegeben.



Vorname	Name	Matr.-Nr.

(b) Implementieren Sie in Java die Klasse, die die Schmuckstücke realisiert, und die Klasse, die die antiken Möbel realisiert. Ihre Klassen sollen jeweils einen Konstruktor besitzen, der die Attributwerte auf die Parameter des Konstruktors setzt. Außerdem soll jede Ihrer Klassen alle Selektoren sowie eine Methode "`public void drucke()`" besitzen, die die Werte aller Attribute auf dem Bildschirm ausgibt. Es genügt, wenn Sie die Implementierungen der Konstruktoren, Selektoren und der `drucke`-Methode in einer der beiden zu implementierenden Klassen angeben. Gehen Sie davon aus, dass derartige Konstruktoren, Selektoren und `drucke`-Methoden auch in den Oberklassen vorhanden sind.

Außerdem sollen beide Klassen eine Methode besitzen, mit der man den Wert eines Objektes dieser Klasse berechnen kann. Hierzu wird die aktuelle Jahreszahl als Parameter übergeben. Der Wert eines antiken Möbelstücks ist sein hundertfaches Alter. Der Wert eines Schmuckstücks berechnet sich aus seinem Gewicht multipliziert mit der aktuellen Jahreszahl.

Verwenden Sie auch hier wieder soweit wie möglich Vererbungstechniken und Prinzipien der Datenkapselung.

Vorname	Name	Matr.-Nr.

(c) Schreiben Sie eine Methode `teuersterWertgegenstand` in Java, der ein Array von Wertgegenständen und die aktuelle Jahreszahl übergeben wird. Die Methode soll daraus den Gegenstand mit dem höchsten Wert ermitteln und diesen Wertgegenstand zurückliefern.

Vorname	Name	Matr.-Nr.

Aufgabe 4 (Programmierung in Java, 18 Punkte)

In dieser Aufgabe sollen Mengen objektorientiert in Java realisiert werden. Eine Menge soll dabei beliebige Objekte von Unterklassen des Interfaces `Vergleichbar` enthalten. Mengen können beliebig viele Objekte enthalten, sie enthalten jedoch kein Objekt doppelt. Hierzu sind das Interface `Vergleichbar` und die Klasse `Element` vorhanden:

```
public interface Vergleichbar {

  public boolean gleich (Vergleichbar zuvergleichen);
   /* Untersucht zwei Objekte auf Gleichheit */

  public String toString ();
   /* Überführt ein Objekt in einen String zur Ausgabe */
}

public class Element {
  private Vergleichbar wert;
  private Element next;

  public Element (Vergleichbar wert, Element next) {...}
    /* Erzeugt ein neues Element mit den Attributen wert und next */

  public Vergleichbar getWert () {...}
  public void setWert(Vergleichbar wert) {...}
  public Element getNext () {...}
  public void setNext(Element next) {...}
    /* Selektoren für die Attribute */
  }
```

Die Klasse `Menge` hat den folgenden Konstruktor und die folgenden öffentlich sichtbaren Methoden.

- `public Menge ()`
  `/* Erzeugt eine neue leere Menge. */`

- `public boolean enthalten (Vergleichbar wert)`
  `/* Untersucht, ob ein Objekt in der Menge vorkommt. */`

- `public void einfuegen (Vergleichbar wert)`
  `/* Fügt ein Objekt in die Menge ein, falls es noch nicht enthalten ist.*/`

- `public void vereinigungMit (Menge m)`
  `/* Vereinigt die aktuelle Menge mit der Menge m, ohne die Menge m zu verändern.*/`

Implementieren Sie die Klasse `Menge`. Beachten Sie dabei auch die Sichtbarkeiten der Attribute und Methoden, da die Daten gekapselt werden sollen. Die Methode `enthalten` muss dabei *rekursiv* (ohne Verwendung von Schleifen) realisiert werden. Hierbei dürfen Sie Hilfsmethoden definieren, die aber nach außen nicht sichtbar sein sollen. Achten Sie darauf, dass eine Menge keine Objekt doppelt enthält.

Vorname	Name	Matr.-Nr.

10

Vorname	Name	Matr.-Nr.

Aufgabe 5 (Funktionale Programmierung in Haskell, 4 + 2 + 3 + 4 + 2 + 6 Punkte)

(a) Die Funktionen `f`, `g`, und `h` sind wie folgt definiert:

```
f x = 1 : x

g x = x ++ x

h x  = \y -> [x] : [y]
```

Geben Sie den allgemeinsten Typ von `f`, `g` und `h` an. Gehen Sie hierbei davon aus, dass `1` den Typ `Int` hat.

(b) Geben Sie für die folgenden Ausdrücke jeweils das Ergebnis der Auswertung an.

(i) `(\x -> \y -> x + y) ((\x -> x + 1) 1) 1`

(ii) `map (\x -> if x /= "Mond" then "Licht" else "Dunkel") ["Sonne", "Mond", "Stern"]`

(c) Schreiben Sie eine Funktion `expand :: a -> Int -> [a]`, die zu einem Objekt `x` und einer Zahl `n` eine Liste `[x,x,...,x]` der Länge `n` liefert. Beispielsweise berechnet `expand 'B' 3` das Ergebnis `['B','B','B']`.

(d) Um Listen auf kürzere Art und Weise zu repräsentieren, kann man wie folgt vorgehen:

Sei $\mathtt{L} = [\mathtt{e}_1, \mathtt{e}_2, ..., \mathtt{e}_n]$ eine Liste von Elementen vom Typ `a`. Jede nicht-leere Liste $\mathtt{K} = [\mathtt{e}_i, \mathtt{e}_{i+1}, ..., \mathtt{e}_j]$ mit $1 \leq i \leq j \leq n$ heißt dann eine *Teilliste* von `L`. Falls alle Elemente in `K` identisch sind (d.h. $\mathtt{e}_i = \mathtt{e}_{i+1} = \ldots = \mathtt{e}_j$) und die angrenzenden Elemente in `L` aber hiervon verschieden sind (d.h. $\mathtt{e}_{i-1} \neq \mathtt{e}_i$, falls $i > 2$, und $\mathtt{e}_j \neq \mathtt{e}_{j+1}$, falls $j < n - 1$), so nennt man `K` eine *geschlossene Teilliste* für das Element $\mathtt{e}_i$.

Eine Liste `L` kann nun wie folgt komprimiert werden: Jede geschlossene Teilliste der Länge `k` für das Element `x` wird durch ein Paar `(x, k)` repräsentiert.

Vorname	Name	Matr.-Nr.

12

Beispiel: Die Liste `L = ['B','B','B','A','A','C','C','C','C']` kann durch die komprimierte Liste `[('B', 3), ('A', 2), ('C', 4)]` repräsentiert werden.

Implementieren Sie in Haskell eine Funktion `decompress :: [(a,Int)] -> [a]`, die die oben beschriebene Komprimierung wieder rückgängig macht. Beispielsweise ergibt `decompress [('B', 3),('A', 2),('C', 4)]` das Ergebnis `['B','B','B','A','A','C','C','C','C']`.

(e) Definieren Sie in Haskell eine Datenstruktur zur Repräsentation der komprimierten Listen, welche der Beschreibung in Teil (d) entspricht. Hierbei dürfen Sie jedoch die in Haskell vordefinierten Listen *nicht* verwenden.

(f) Implementieren Sie in Haskell eine Funktion `compress`, die die oben beschriebene Komprimierung durchführt, d.h., eine beliebige Liste wird in ein Objekt der Datenstruktur aus Teilaufgabe (e) transformiert. Geben Sie auch die Typdeklaration von `compress` an. Sie dürfen hierbei davon ausgehen, dass der Vergleichsoperator `(==)` für alle Typen vordefiniert ist.
Hinweis: Implementieren Sie gegebenenfalls geeignete Hilfsfunktionen.

Vorname	Name	Matr.-Nr.

Aufgabe 6 (Logische Programmierung in Prolog, 3 + 5 + 2 + 5 Punkte)

(a) Definieren Sie in Prolog ein dreistelliges Prädikat `del`, das ein Vorkommen eines Elements aus einer Liste löscht. Genauer bedeutet `del(L,X,K)`, dass die Liste `K` aus der Liste `L` entsteht, indem darin ein Vorkommen des Elements `X` gelöscht wird. Beispielsweise gilt `del([2,1,5,1],1,[2,5,1])`. Hierbei dürfen Sie keine in Prolog vordefinierten Prädikate verwenden.

(b) Definieren Sie in Prolog ein zweistelliges Prädikat `perm`, wobei `perm(L,K)` ausdrückt, dass die Liste `L` eine Permutation der Liste `K` ist.
Eine Liste ist dabei eine Permutation einer anderen Liste, wenn beide Listen dieselben Elemente (aber ggf. in unterschiedlicher Reihenfolge) enthalten. Beispielsweise hat die Liste `[1,2,2]` die Permutationen `[1,2,2]`, `[2,1,2]` und `[2,2,1]`. Es gilt also z.B. `perm([1,2,2], [2,1,2])`.

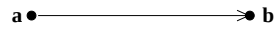
Vorname	Name	Matr.-Nr.

(c) Formulieren Sie (basierend auf dem zuvor definierten Prädikat) eine Anfrage, um alle Permutationen der Liste `[1,2,3]` zu berechnen, die mit `2` beginnen.

(d) Betrachten Sie folgendes Prolog-Programm:

```
kante(a,b).

trans(X, Y) :- kante(X,Y).
trans(X, Y) :- trans(Z,Y), kante(X, Z).
```

Das Faktum `kante(a,b).` definiert dabei folgenden Graph:



Terminiert das Programm bei der Suche nach den ersten beiden Antworten auf die folgende Anfrage? Geben Sie ggf. die erste bzw. die ersten beiden Antworten an.

```
?- trans(a, U).
```

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

Diplom-Vorprüfung Lösungsvorschlag Informatik I - Programmierung

6. 3. 2002

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 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 ist.
- 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	14	
Aufgabe 2	11	
Aufgabe 3	17	
Aufgabe 4	18	
Aufgabe 5	21	
Aufgabe 6	15	
Summe	96	
Note	-	

Vorname	Name	Matr.-Nr.

Aufgabe 1 (Programmanalyse, 12 + 2 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 interface Z {
  public void f (Z object);
}

public class X implements Z {
  public int wert1 = 1,
             wert2 = 2;

  public X() {
    wert1 = wert1 + wert2;
  }
  public void f(Z object) {
    object = this;
  }
}

public class Y extends X {
  public int wert2 = 3;

  public Y() {
    wert1 = wert1 + wert2;
  }
  public void f (Z object) {}
}

public class M {
  public static void main(String[] arguments) {
    X x = new X();
    Y y = new Y();
    System.out.println(x.wert1 + ", " + x.wert2);        // AUSGABE: 3, 2
    System.out.println(y.wert1 + ", " + y.wert2);        // AUSGABE: 6, 3
    x.f(y);
    System.out.println(y.wert1 + ", " + y.wert2);}       // AUSGABE: 6, 3
}
```

(b) In der Klasse `Y` wird die Methode `f` nun durch folgenden Code ersetzt. Finden und erklären Sie den Fehler in dieser Methode, der das Compilieren verhindert.

```
protected void f (Y object) {
  Z z = new Y();
  Y y = z;
}
```

Vorname	Name	Matr.-Nr.

`Y y = z;` ist falsch, weil `z` vom Typ `Z` ist und daher explizit konvertiert werden muss (d.h., es müsste `Y y = (Y) z;` heißen).

Vorname	Name	Matr.-Nr.

Aufgabe 2 (Verifikation, 10 + 1 Punkte**)**

Der Algorithmus P berechnet die Hälfte einer geraden natürlichen Zahl.

(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: $\mathtt{x} \in \{\, 2 * \mathtt{n} \mid \mathtt{n} \in \mathbb{N} \,\} = \{0, 2, 4, ...\}$
Ausgabe: `res`
Vorbedingung: $\mathtt{x} \geq 0 \wedge 2|\mathtt{x}$ (wobei $2|\mathtt{x}$ für die Aussage "2 teilt x" steht)
Nachbedingung: $2 * \mathtt{res} = \mathtt{x}$

$\langle\, \mathtt{x} \geq 0 \wedge 2|\mathtt{x} \,\rangle$

$\langle\, \mathtt{x} \geq 0 \wedge 2|\mathtt{x} \wedge 0 = 0 \,\rangle$

```
z = 0;
```

$\langle\, \mathtt{x} \geq 0 \wedge 2|\mathtt{x} \wedge \mathtt{z} = 0 \,\rangle$

$\langle\, \mathtt{x} \geq 0 \wedge 2|\mathtt{x} \wedge \mathtt{z} = 0 \wedge 0 = 0 \,\rangle$

```
res = 0;
```

$\langle\, \mathtt{x} \geq 0 \wedge 2|\mathtt{x} \wedge \mathtt{z} = 0 \wedge \mathtt{res} = 0 \,\rangle$

$\langle\, \mathtt{x} \geq 0 \wedge 2|\mathtt{x} \wedge 2 * \mathtt{res} = \mathtt{z} \,\rangle$

```
while (z != x) {
```

$\langle\, \mathtt{x} \geq 0 \wedge 2|\mathtt{x} \wedge \mathtt{z} \neq \mathtt{x} \wedge 2 * \mathtt{res} = \mathtt{z} \,\rangle$

$\langle\, \mathtt{x} \geq 0 \wedge 2|\mathtt{x} \wedge 2 * (\mathtt{res} + 1) = \mathtt{z} + 2 \,\rangle$

```
  res = res + 1;
```

$\langle \mathtt{x} \geq 0 \wedge 2|\mathtt{x} \wedge 2 * \mathtt{res} = \mathtt{z} + 2 \,\rangle$

```
  z = z + 2;
```

$\langle\, \mathtt{x} \geq 0 \wedge 2|\mathtt{x} \wedge 2 * \mathtt{res} = \mathtt{z} \,\rangle$

```
}
```

$\langle\, \mathtt{x} \geq 0 \wedge 2|\mathtt{x} \wedge \mathtt{z} = \mathtt{x} \wedge 2 * \mathtt{res} = \mathtt{z} \,\rangle$

$\langle 2 * \mathtt{res} = \mathtt{x} \rangle$

Vorname	Name	Matr.-Nr.

5

(b) Geben Sie (ohne Beweis) eine Variante für die `while`-Schleife an, die belegt, dass die Schleife für $\texttt{x} \geq \texttt{0} \wedge \texttt{2} | \texttt{x}$ terminiert.

Die Variante ist $x - z$. Da z in jedem Schleifendurchlauf um `2` erhöht wird, verkleinert sich der Wert von $x - z$ mit jedem Schleifendurchlauf.

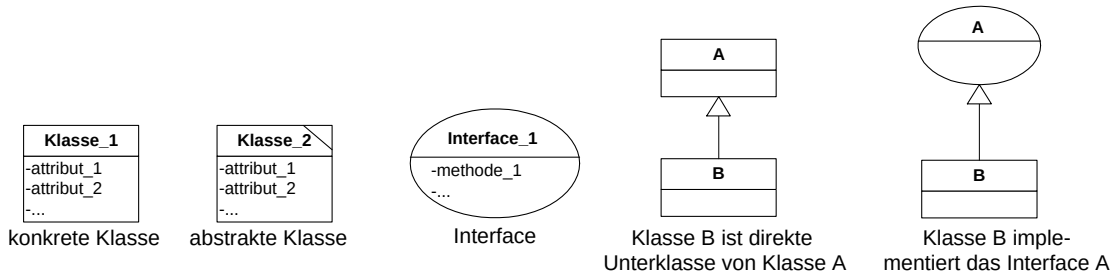
Vorname	Name	Matr.-Nr.

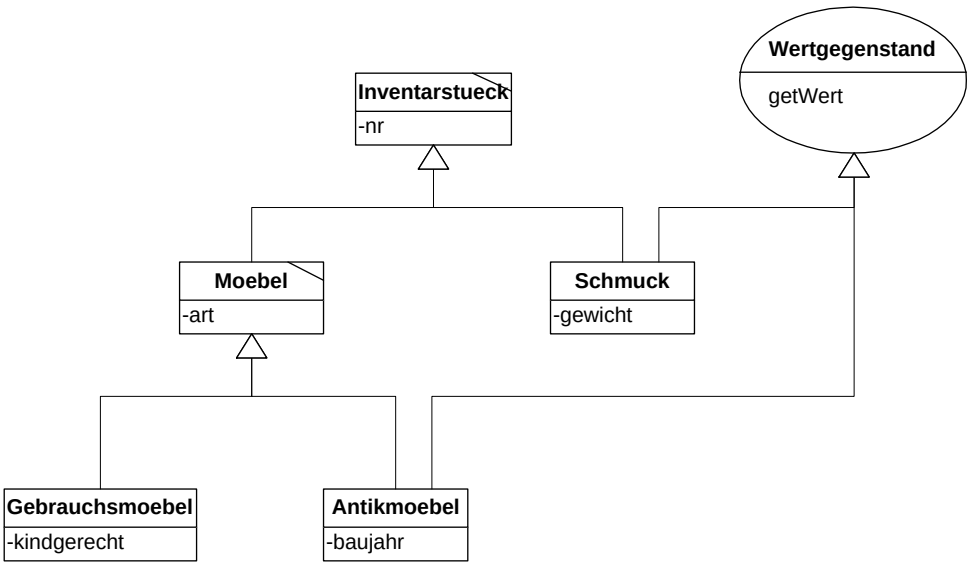
Aufgabe 3 (Datenstrukturen in Java, 6 + 5 + 6 Punkte)

Herr Müller möchte seinen Besitz inventarisieren. Dazu beschließt er, ein objektorientiertes Programm in Java zu entwickeln. Jedes seiner Besitzstücke soll eine Inventarnummer haben. Herr Müller besitzt Möbel- und Schmuckstücke. Jedes Möbelstück ist durch seine Art gekennzeichnet (z.B. “Sofa”, “Tisch”, etc.). Nur Schmuckstücke und antike Möbel sind Wertgegenstände, von denen man den Wert durch eine Methode berechnen können soll. Hierzu speichert man bei antiken Möbeln ihr Baujahr und bei Schmuckstücken ihr Gewicht (als ganze Zahl). Bei nicht antiken Möbeln ist es hingegen von Interesse, ob sie kindgerecht sind.

(a) Entwerfen Sie eine geeignete Klassenhierarchie, um den obigen Sachverhalt in Java zu realisieren. Achten Sie dabei darauf, dass gemeinsame Attribute und/oder Methoden in abstrakten Klassen bzw. Interfaces zusammengefasst werden. Verwenden Sie soweit wie möglich Vererbungstechniken.

Notieren Sie Ihren Entwurf grafisch und verwenden Sie dazu die folgende Notation: Für jede Klasse werden ihr Name und die Namen ihrer Attribute angegeben. (Methoden werden bei Klassen nicht mit aufgeführt.) Für jedes Interface werden sein Name und die Namen seiner Methoden angegeben.





Vorname	Name	Matr.-Nr.

(b) Implementieren Sie in Java die Klasse, die die Schmuckstücke realisiert, und die Klasse, die die antiken Möbel realisiert. Ihre Klassen sollen jeweils einen Konstruktor besitzen, der die Attributwerte auf die Parameter des Konstruktors setzt. Außerdem soll jede Ihrer Klassen alle Selektoren sowie eine Methode "`public void drucke()`" besitzen, die die Werte aller Attribute auf dem Bildschirm ausgibt. Es genügt, wenn Sie die Implementierungen der Konstruktoren, Selektoren und der `drucke`-Methode in einer der beiden zu implementierenden Klassen angeben. Gehen Sie davon aus, dass derartige Konstruktoren, Selektoren und `drucke`-Methoden auch in den Oberklassen vorhanden sind.

Außerdem sollen beide Klassen eine Methode besitzen, mit der man den Wert eines Objektes dieser Klasse berechnen kann. Hierzu wird die aktuelle Jahreszahl als Parameter übergeben. Der Wert eines antiken Möbelstücks ist sein hundertfaches Alter. Der Wert eines Schmuckstücks berechnet sich aus seinem Gewicht multipliziert mit der aktuellen Jahreszahl.

Verwenden Sie auch hier wieder soweit wie möglich Vererbungstechniken und Prinzipien der Datenkapselung.

```
public class Schmuck extends Inventarstueck implements Wertgegenstand {

    private int gewicht;

    public Schmuck (int nr, int gewicht) {
        super(nr);
        this.gewicht = gewicht;
    }

    public int getGewicht() {
        return gewicht;
    }

    public void setGewicht(int gewicht) {
        this.gewicht = gewicht;
    }

    public void drucke() {
        super.drucke();
        System.out.print(", " + gewicht);
    }

    public int getWert(int jahreszahl) {
        return gewicht * jahreszahl;
    }
}
```

Vorname	Name	Matr.-Nr.


```
public class Antikmoebel extends Moebel implements Wertgegenstand {

    private int baujahr;

    public Antikmoebel (int nr, String art, int baujahr) {
        super(nr,art);
        this.baujahr = baujahr;
    }

    public int getBaujahr() {
        return baujahr;
    }

    public void setBaujahr(int baujahr) {
        this.baujahr = baujahr;
    }

    public void drucke() {
        super.drucke();
        System.out.print(", " + baujahr);
    }

    public int getWert(int jahreszahl) {
        return 100 * (jahreszahl - baujahr);
    }
}
```

(c) Schreiben Sie eine Methode `teuersterWertgegenstand` in Java, der ein Array von Wertgegenständen und die aktuelle Jahreszahl übergeben wird. Die Methode soll daraus den Gegenstand mit dem höchsten Wert ermitteln und diesen Wertgegenstand zurückliefern.

```
public static Wertgegenstand teuersterWertgegenstand(Wertgegenstand[] w,
                                                     int jahreszahl)     {

    if (w == null || w.length == 0) return null;

    Wertgegenstand maximum = w[0];

    for (int i = 1; i < w.length; i++) {
        if (maximum == null || // kein Fehler wenn dies fehlt
            maximum.getWert(jahreszahl) < w[i].getWert(jahreszahl))
            maximum = w[i];
    }
    return maximum;
}
```

Vorname	Name	Matr.-Nr.

Aufgabe 4 (Programmierung in Java, 18 Punkte)

In dieser Aufgabe sollen Mengen objektorientiert in Java realisiert werden. Eine Menge soll dabei beliebige Objekte von Unterklassen des Interfaces `Vergleichbar` enthalten. Mengen können beliebig viele Objekte enthalten, sie enthalten jedoch kein Objekt doppelt. Hierzu sind das Interface `Vergleichbar` und die Klasse `Element` vorhanden:

```
public interface Vergleichbar {

  public boolean gleich (Vergleichbar zuvergleichen);
   /* Untersucht zwei Objekte auf Gleichheit */

  public String toString ();
   /* Überführt ein Objekt in einen String zur Ausgabe */
}

public class Element {
  private Vergleichbar wert;
  private Element next;

  public Element (Vergleichbar wert, Element next) {...}
    /* Erzeugt ein neues Element mit den Attributen wert und next */

  public Vergleichbar getWert () {...}
  public void setWert(Vergleichbar wert) {...}
  public Element getNext () {...}
  public void setNext(Element next) {...}
    /* Selektoren für die Attribute */
  }
```

Die Klasse `Menge` hat den folgenden Konstruktor und die folgenden öffentlich sichtbaren Methoden.

- `public Menge ()`
  `/* Erzeugt eine neue leere Menge. */`

- `public boolean enthalten (Vergleichbar wert)`
  `/* Untersucht, ob ein Objekt in der Menge vorkommt. */`

- `public void einfuegen (Vergleichbar wert)`
  `/* Fügt ein Objekt in die Menge ein, falls es noch nicht enthalten ist.*/`

- `public void vereinigungMit (Menge m)`
  `/* Vereinigt die aktuelle Menge mit der Menge m, ohne die Menge m zu verändern.*/`

Implementieren Sie die Klasse `Menge`. Beachten Sie dabei auch die Sichtbarkeiten der Attribute und Methoden, da die Daten gekapselt werden sollen. Die Methode `enthalten` muss dabei *rekursiv* (ohne Verwendung von Schleifen) realisiert werden. Hierbei dürfen Sie Hilfsmethoden definieren, die aber nach außen nicht sichtbar sein sollen. Achten Sie darauf, dass eine Menge keine Objekt doppelt enthält.

Vorname	Name	Matr.-Nr.


```
public class Menge {

    private Element kopf;

    public Menge () {
        kopf = null;
    }

    public boolean enthalten (Vergleichbar wert) {
        return enthalten (wert, kopf);
    }

    private static boolean enthalten (Vergleichbar wert, Element kopf) {
        if      (kopf == null)                return false;
        else if (kopf.getWert().gleich(wert)) return true;
        else                                  return enthalten(wert, kopf.getNext());
    }

    public void einfuegen (Vergleichbar wert) {
        if      (kopf == null)        kopf = new Element(wert,null);
        else if (! enthalten (wert))  kopf = new Element(wert,kopf);
    }

    public void vereinigungMit (Menge m) {
        if (m != null) {
           Element element = m.kopf;

           while (element != null) {
                 einfuegen (element.getWert());
                 element = element.getNext();
           }
        }
    }
}
```

Vorname	Name	Matr.-Nr.

Aufgabe 5 (Funktionale Programmierung in Haskell, 4 + 2 + 3 + 4 + 2 + 6 Punkte)

(a) Die Funktionen `f`, `g`, und `h` sind wie folgt definiert:

```
f x = 1 : x

g x = x ++ x

h x  = \y -> [x] : [y]
```

Geben Sie den allgemeinsten Typ von `f`, `g` und `h` an. Gehen Sie hierbei davon aus, dass `1` den Typ `Int` hat.

Der allgemeinste Typ von `f` ist `[Int] -> [Int]`, der allgemeinste Typ von `g` ist `[a] -> [a]` und der allgemeinste Typ von `h` ist `a -> [a] -> [[a]]`.

(b) Geben Sie für die folgenden Ausdrücke jeweils das Ergebnis der Auswertung an.

(i) `(\x -> \y -> x + y) ((\x -> x + 1) 1) 1`

(ii) `map (\x -> if x /= "Mond" then "Licht" else "Dunkel") ["Sonne", "Mond", "Stern"]`

(i) `3`

(ii) `["Licht", "Dunkel", "Licht"]`

(c) Schreiben Sie eine Funktion `expand :: a -> Int -> [a]`, die zu einem Objekt `x` und einer Zahl `n` eine Liste `[x,x,...,x]` der Länge `n` liefert. Beispielsweise berechnet `expand 'B' 3` das Ergebnis `['B','B','B']`.

```
expand :: a -> Int -> [a]
expand x 0     = []
expand x (n+1) = x : expand x n
```

(d) Um Listen auf kürzere Art und Weise zu repräsentieren, kann man wie folgt vorgehen:

Sei $L = [e_1, e_2, ..., e_n]$ eine Liste von Elementen vom Typ `a`. Jede nicht-leere Liste $K = [e_i, e_{i+1}, ..., e_j]$ mit $1 \leq i \leq j \leq n$ heißt dann eine *Teilliste* von `L`. Falls alle Elemente in `K` identisch sind (d.h. $e_i = e_{i+1} = \ldots = e_j$) und die angrenzenden Elemente in `L` aber hiervon verschieden sind (d.h. $e_{i-1} \neq e_i$, falls $i > 2$, und $e_j \neq e_{j+1}$, falls $j < n - 1$), so nennt man `K` eine *geschlossene Teilliste* für das Element e_i.

Eine Liste `L` kann nun wie folgt komprimiert werden: Jede geschlossene Teilliste der Länge `k` für das Element `x` wird durch ein Paar `(x, k)` repräsentiert.

Vorname	Name	Matr.-Nr.

Beispiel: Die Liste `L = ['B','B','B','A','A','C','C','C','C']` kann durch die komprimierte Liste `[('B', 3), ('A', 2), ('C', 4)]` repräsentiert werden.

Implementieren Sie in Haskell eine Funktion `decompress :: [(a,Int)] -> [a]`, die die oben beschriebene Komprimierung wieder rückgängig macht. Beispielsweise ergibt `decompress [('B', 3),('A', 2),('C', 4)]` das Ergebnis `['B','B','B','A','A','C','C','C','C']`.

```
decompress :: [(a,Int)] -> [a]
decompress []          = []
decompress ((x,n) : l) = expand x n ++ decompress l
```

(e) Definieren Sie in Haskell eine Datenstruktur zur Repräsentation der komprimierten Listen, welche der Beschreibung in Teil (d) entspricht. Hierbei dürfen Sie jedoch die in Haskell vordefinierten Listen *nicht* verwenden.

```
data CompList a = Nil | Cons a Int (CompList a) deriving Show
```

(f) Implementieren Sie in Haskell eine Funktion `compress`, die die oben beschriebene Komprimierung durchführt, d.h., eine beliebige Liste wird in ein Objekt der Datenstruktur aus Teilaufgabe (e) transformiert. Geben Sie auch die Typdeklaration von `compress` an. Sie dürfen hierbei davon ausgehen, dass der Vergleichsoperator `(==)` für alle Typen vordefiniert ist.
Hinweis: Implementieren Sie gegebenenfalls geeignete Hilfsfunktionen.

```
compress :: [a] -> CompList a
compress xs = build xs 0
  {- build xs n komprimiert die Liste, die aus xs entsteht, indem vorne
     noch n Kopien des ersten Elements angehaengt werden.              -}
  where build :: [a] -> Int -> (CompList a)
        build []  n = Nil
        build [x] n = Cons x (n+1) Nil
        build (x:y:xs) n | x == y    = build (y:xs) (n+1)
                         | otherwise = Cons x (n+1) (build (y:xs) 0)
```

Vorname	Name	Matr.-Nr.

Aufgabe 6 (Logische Programmierung in Prolog, 3 + 5 + 2 + 5 Punkte)

(a) Definieren Sie in Prolog ein dreistelliges Prädikat `del`, das ein Vorkommen eines Elements aus einer Liste löscht. Genauer bedeutet `del(L,X,K)`, dass die Liste `K` aus der Liste `L` entsteht, indem darin ein Vorkommen des Elements `X` gelöscht wird. Beispielsweise gilt `del([2,1,5,1],1,[2,5,1])`. Hierbei dürfen Sie keine in Prolog vordefinierten Prädikate verwenden.

```
del([X|L], X, L).
del([Y|L], X, [Y|K]) :- del(L, X, K).
```

(b) Definieren Sie in Prolog ein zweistelliges Prädikat `perm`, wobei `perm(L,K)` ausdrückt, dass die Liste `L` eine Permutation der Liste `K` ist.
Eine Liste ist dabei eine Permutation einer anderen Liste, wenn beide Listen dieselben Elemente (aber ggf. in unterschiedlicher Reihenfolge) enthalten. Beispielsweise hat die Liste `[1,2,2]` die Permutationen `[1,2,2]`, `[2,1,2]` und `[2,2,1]`. Es gilt also z.B. `perm([1,2,2], [2,1,2])`.

```
perm([], []).
perm(L, [X|K]) :- del(L, X, R), perm(R, K).
```

Vorname	Name	Matr.-Nr.

(c) Formulieren Sie (basierend auf dem zuvor definierten Prädikat) eine Anfrage, um alle Permutationen der Liste `[1,2,3]` zu berechnen, die mit `2` beginnen.

```
?- perm([1,2,3], L), L = [2|_].
```

(d) Betrachten Sie folgendes Prolog-Programm:

```
kante(a,b).

trans(X, Y) :- kante(X,Y).
trans(X, Y) :- trans(Z,Y), kante(X, Z).
```

Das Faktum `kante(a,b).` definiert dabei folgenden Graph:

Terminiert das Programm bei der Suche nach den ersten beiden Antworten auf die folgende Anfrage? Geben Sie ggf. die erste bzw. die ersten beiden Antworten an.

```
?- trans(a, U).
```

Die erste Antwort ist `U = b`. Bei der Suche nach der zweiten Antwort terminiert das Programm nicht.

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

Diplom-Vorprüfung / Zwischenprüfung Informatik I - Programmierung

23. 9. 2002

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 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 ist.
- 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	18	
Aufgabe 2	11	
Aufgabe 3	12	
Aufgabe 4	19	
Aufgabe 5	16	
Aufgabe 6	13	
Summe	89	
Note	-	

Vorname	Name	Matr.-Nr.

Aufgabe 1 (Programmanalyse, 12 + 6 Punkte)

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

```
public class A {
  public int wert = 3;

  public static int eq(A x1, A x2) {
    if (x1 == x2) return 1; else return 2;}
  public int eq(A x) {
   if (this == x) return 3; else return 4;}
}

public class B extends A {
  public B(int wert) {
    this.wert = this.wert + wert;}
  public int eq(A x) {
    if (wert == x.wert) return 5; else return 6;}
  public int eq(B y) {
    if (this == y) return 7; else return 8;}
}

public class H {
  public static void main(String[] args) {
    A a1 = new B(5);
    System.out.println(a1.wert);        // AUSGABE:
    B b = (B) a1;
    A a2 = new A();
    System.out.println(a2.wert);        // AUSGABE:
    a2.wert = a1.wert;
    System.out.println(B.eq(a1,a2));    // AUSGABE:
    System.out.println(a2.eq(a1));      // AUSGABE:
    System.out.println(a1.eq(a2));      // AUSGABE:
    System.out.println(b.eq(b));}       // AUSGABE:
}
```

(b) Das Programm wird um eine zusätzliche Klasse `C` ergänzt. Finden und erklären Sie die drei Fehler in dieser Klasse, die das Compilieren verhindern.

```
public class C extends A {
  private static int eq(A x1, A x2) {
    return 1;}

  public double eq (A obj) {
    A x = obj;
    C c = x;
    return c.wert;}
}
```

Vorname	Name	Matr.-Nr.

Vorname	Name	Matr.-Nr.

Aufgabe 2 (Verifikation, 10 + 1 Punkte)

Der Algorithmus P berechnet die Potenz $\mathtt{x}^{\mathtt{y}}$ zweier natürlicher Zahlen $\mathtt{x}$ und $\mathtt{y}$ mit $\mathtt{x} > 0$ und $\mathtt{y} > 0$.

(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: $\mathtt{x}, \mathtt{y} \in \{1, 2, ...\}$
Ausgabe: `res`
Vorbedingung: $\mathtt{x} > 0 \wedge \mathtt{y} > 0$
Nachbedingung: $\mathtt{res} = \mathtt{x}^{\mathtt{y}}$

$\langle\ \mathtt{x} > 0 \wedge \mathtt{y} > 0\ \rangle$

$\langle\ \mathtt{x} > 0 \wedge \mathtt{y} > 0 \wedge$ _____ $\rangle$

```
z = y;
```

$\langle\ \mathtt{x} > 0 \wedge \mathtt{y} > 0 \wedge$ _____ $\rangle$

$\langle\ \mathtt{x} > 0 \wedge \mathtt{y} > 0 \wedge$ _____ $\rangle$

```
res = 1;
```

$\langle\ \mathtt{x} > 0 \wedge \mathtt{y} > 0 \wedge \mathtt{z} = \mathtt{y} \wedge \mathtt{res} = 1\ \rangle$

$\langle\ \mathtt{x} > 0 \wedge \mathtt{y} > 0 \wedge$ _____ $\rangle$

```
while (z != 0) {
```

$\langle\ \mathtt{x} > 0 \wedge \mathtt{y} > 0 \wedge \mathtt{z} \neq 0 \wedge$ _____ $\rangle$

$\langle\ \mathtt{x} > 0 \wedge \mathtt{y} > 0 \wedge$ _____ $\rangle$

```
  res = res * x;
```

$\langle \mathtt{x} > 0 \wedge \mathtt{y} > 0 \wedge$ _____ $\rangle$

```
  z = z - 1;
```

$\langle\ \mathtt{x} > 0 \wedge \mathtt{y} > 0 \wedge$ _____ $\rangle$

```
}
```

$\langle\ \mathtt{x} > 0 \wedge \mathtt{y} > 0 \wedge$ _____ $\rangle$

$\langle\ \mathtt{res} = \mathtt{x}^{\mathtt{y}}\ \rangle$

Vorname	Name	Matr.-Nr.

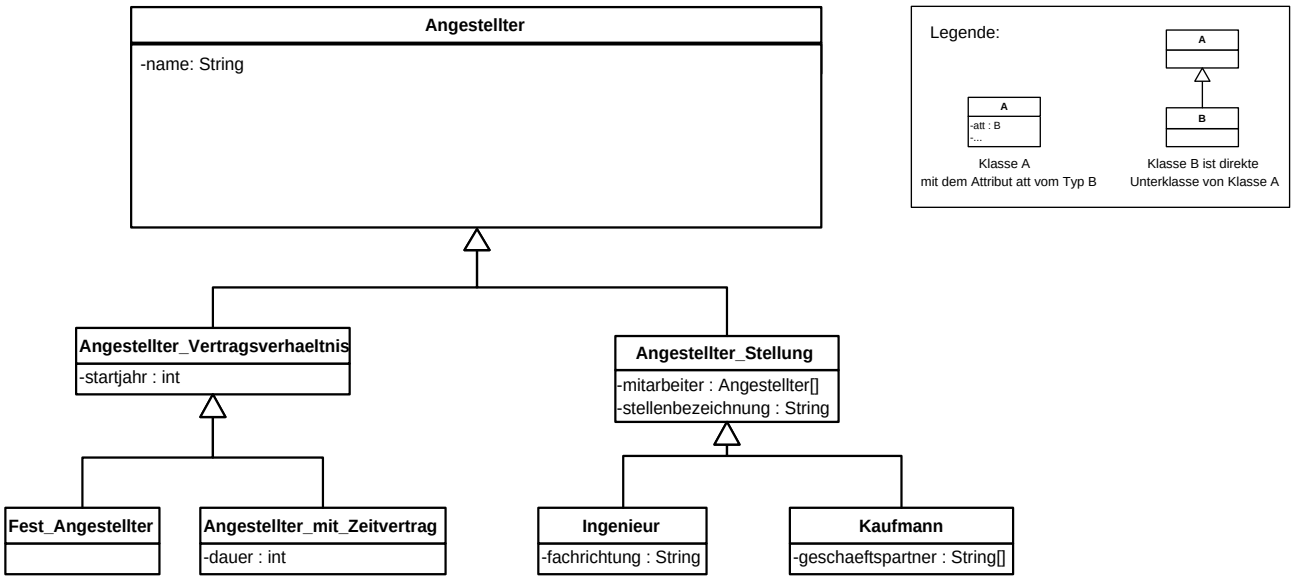
5

(b) Geben Sie (ohne Beweis) eine Variante für die `while`-Schleife an, die belegt, dass die Schleife für $\texttt{x} > 0 \wedge \texttt{y} > 0$ terminiert.

Vorname	Name	Matr.-Nr.

Aufgabe 3 (Datenstrukturen in Java, 2 + 3 + 7 Punkte)

In dieser Aufgabe sollen unterschiedliche Angestelltenverhältnisse modelliert werden. Einen ersten (naiven) Entwurf zeigt die folgende Abbildung:



(a) Erläutern Sie, warum der oben angegebene Entwurf eine schlechte Modellierung der Angestelltenverhältnisse ist.

(b) Verbessern Sie den obigen Entwurf sinnvoll (durch Veränderung im obigen Klassendiagramm).

(c) Die Klasse `Angestellter` soll eine Methode

```
public void drucke(int jahr) {...}
```

enthalten, die den Namen des Angestellten, die Anzahl seiner Mitarbeiter und sein Gehalt im Jahr `jahr` auf dem Bildschirm ausgibt.

Das Gehalt eines Angestellten bestimmt sich nach der Anzahl der Jahre, die er bereits angestellt ist. Im ersten Jahr verdient ein Angestellter 1000 Euro, im zweiten Jahr 2000 Euro, etc. Angestellte werden immer zu Beginn des Jahres eingestellt und bei Zeitverträgen gibt `dauer` die Vertragsdauer in Jahren an.

Implementieren Sie die Klasse `Angestellter` mit der Methode `drucke`. Hierzu dürfen Sie natürlich beliebige Hilfsmethoden (auch in anderen Klassen) schreiben. Geben Sie auch die Implementierungen von denjenigen anderen Klassen an, in denen Sie Hilfsmethoden schreiben. Implementieren Sie keine Konstruktoren. Sie brauchen auch nur diejenigen Selektoren zu implementieren, die Sie für die Methode `drucke` benötigen. Verwenden Sie soweit wie möglich Vererbungstechniken und Prinzipien der Datenkapselung.

Vorname	Name	Matr.-Nr.

7

Vorname	Name	Matr.-Nr.

Aufgabe 4 (Programmierung in Java, $1 + 6 + 3 + 7 + 2$ Punkte**)**

Der Surfverband möchte die Rangliste seiner Surfer elektronisch verwalten. Surfer sind Sportler und für die Verwaltung von Sportlern in Ranglisten existieren bereits folgende Vorgaben für die Implementierung:

```
public abstract class Sportler {
  protected String name;
}

public abstract class Element {
  protected Sportler wert;
  protected Element next;
}

public abstract class Rangliste {
  protected Element kopf;
  public Rangliste (Element kopf) {this.kopf = kopf;}
  public abstract Element sucheSportler(Sportler s);
  public abstract void aktualisiere(Sportler gewinner, Sportler verlierer);
}
```

Da beliebig viele Sportler in die jeweilige Rangliste aufgenommen werden sollen, kann die Länge der Rangliste nicht vorher festgelegt werden. Für die Rangliste sind hier nur diejenigen Methoden aufgeführt, die für das Aktualisieren der Liste aufgrund eines Spielergebnisses benötigt werden. In einem Wettkampf treten immer zwei Sportler gegeneinander an und es gibt immer einen Gewinner und einen Verlierer. Nach jedem Wettkampf wird die Rangliste aktualisiert.

Ziel der Aufgabe ist es, (nicht-abstrakte) Unterklassen `Surfer`, `SurferElement` und `SurferRangliste` von `Sportler`, `Element` und `Rangliste` zu implementieren. Hierbei dürfen die oben angegebenen Implementierungen nicht verändert werden.

Ein Surfer zeichnet sich durch einen Namen und eine Surfernummer aus. Sie dürfen davon ausgehen, dass diese Nummer eindeutig ist und brauchen dies in der Implementierung nicht zu überprüfen.

Beachten Sie auch die Sichtbarkeiten der Attribute und Methoden, da die Daten gekapselt werden sollen. Selbstverständlich dürfen Sie beliebig viele geeignete Hilfsmethoden schreiben. Sie brauchen nur die Selektoren zu implementieren, die Sie benötigen.

Gehen Sie bei der Implementierung der Klasse `SurferRangliste` wie folgt vor. Die beschriebenen Methoden können in späteren Aufgabenteilen jeweils als gegeben vorausgesetzt und verwendet werden.

(a) Implementieren Sie einen geeigneten Konstruktor

```
public SurferRangliste (Element kopf) {...}
```

in der Klasse `SurferRangliste`.

(b) Implementieren Sie die Methode `sucheSportler`. Beachten Sie: Diese Methode und die von ihr aufgerufenen Methoden dürfen keine Schleifen enthalten, sondern es darf nur *Rekursion* verwendet werden! Hierbei soll die Methode `sucheSportler (Sportler s)` in der Klasse `SurferRangliste` dasjenige Listenelement zurückliefern, welches den übergebenen Surfer `s` enthält. In der gesamten Aufgabe werden zwei Surfer als gleich betrachtet, wenn ihre Surfernummern gleich sind. Wenn `s` kein Surfer ist oder nicht in der Liste enthalten ist, soll `null` zurückgegeben werden.

Gehen Sie in der gesamten Aufgabe davon aus, dass ein Objekt der Klasse `SurferRangliste` nur `Surfer` und keine anderen `Sportler` enthält. Außerdem dürfen Sie generell voraussetzen, dass eine `SurferRangliste` keinen `Surfer` mehrmals enthält.

(c) Implementieren Sie eine Hilfsmethode

```
private boolean dahinterPlatziert(Surfer a, Surfer b) {...}
```

in der Klasse `SurferRangliste`, die prüft, ob der Surfer `a` in der Rangliste hinter Surfer `b` liegt. Setzen Sie dabei voraus, dass beide Surfer in der Rangliste sind.

(d) Die Methode `aktualisiere (Sportler gewinner, Sportler verlierer)` in der Klasse `SurferRangliste` soll die Rangliste wie folgt aktualisieren: Falls der Surfer `gewinner` bisher hinter dem Surfer `verlierer` in der Rangliste platziert war, wird nun der Surfer `gewinner` in der Rangliste direkt vor dem Surfer `verlierer` platziert. Wenn der Surfer `gewinner` ohnehin schon vor dem Surfer `verlierer` platziert war, so soll sich die Rangliste nicht ändern. Wenn mindestens einer der übergebenen Sportler `gewinner` bzw. `verlierer` nicht in der Rangliste oder kein Surfer ist, so soll sich die Rangliste nicht ändern.

Sie können dabei auf die Methoden `sucheSportler`, `dahinterPlatziert` und eine weitere Methode

```
public SurferElement sucheVorgaenger(Surfer s) {...}
```

in der Klasse `SurferRangliste` zurückgreifen, wobei Sie die Methode `sucheVorgaenger` *nicht* implementieren müssen. Sie gibt das Listenelement zurück, welches vor dem übergebenen Surfer `s` in der Rangliste steht. Wenn es kein solches Listenelement gibt, wird `null` zurückgegeben.

(e) Vergessen Sie nicht, die Klassen `Surfer` und `SurferElement` zu implementieren. Dabei brauchen Sie nur die Methoden (und Konstruktoren) zu schreiben, die Sie in den anderen Aufgabenteilen benutzen.

Vorname	Name	Matr.-Nr.

10

Vorname	Name	Matr.-Nr.

Vorname	Name	Matr.-Nr.

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

(a) Die Funktionen `f`, `g`, und `h` sind wie folgt definiert:

```
f x = x : [x]

g x y = g y x

h x  = \y -> x ++ (y : x)
```

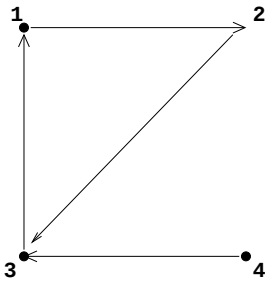
Geben Sie den allgemeinsten Typ von `f`, `g` und `h` an.

(b) Geben Sie für die folgenden Ausdrücke jeweils das Ergebnis der Auswertung an.

(i) `(\x -> \y -> map x y) (\x -> x + 2) [10, 3, 2]`

(ii) `filter (\x -> x < 7) (filter (\x -> x > 2) [10, 3, 2, 0, 5, 3])`

(c) Graphen mit Knoten eines Typs `a` können als Liste des Typs `[(a,a)]` repräsentiert werden. Hierbei wird jede Kante des Graphens als Paar in der Liste gespeichert. Der folgende Graph



könnte also zum Beispiel durch die Liste `[ (1,2), (2,3), (3,1), (4,3) ]` repräsentiert werden.

Definieren Sie eine Funktion `delete :: a -> a -> [(a,a)] -> [(a,a)]`, wobei `delete(x,y,k)` *alle* Vorkommen der Kante von `x` nach `y` aus der Liste `k` löscht, die den Graphen repräsentiert. Wenn `k` die obige Liste ist, so ergibt `delete (2,3,k)` also das Ergebnis `[ (1,2), (3,1), (4,3)]`. Hierbei dürfen Sie keine in Haskell vordefinierten Funktionen auf Listen außer den Datenkonstruktoren `[]` und `:` verwenden. Sie dürfen davon ausgehen, dass der Vergleichsoperator `(==)` für alle Typen vordefiniert ist.

Vorname	Name	Matr.-Nr.

(d) Definieren Sie in Haskell eine Datenstruktur zur Repräsentation von Graphen, welche der Beschreibung in Teil (c) entspricht. Hierbei dürfen Sie jedoch die in Haskell vordefinierten Listen *nicht* verwenden.

(e) Implementieren Sie in Haskell eine Funktion `path`, die zwei Knoten `x`, `y` und einen Graph `g` als Eingabe bekommt. Dann soll `path x y g` genau dann `true` liefern, wenn es im Graph `g` einen Pfad von `x` nach `y` gibt oder wenn `x = y` ist. (Ein Pfad in einem Graphen `g` ist eine Liste $[v_1, \ldots, v_n]$ von Knoten aus `g` (mit $n \geq 0$), so dass für alle $i \in \{1, \ldots, n-1\}$ eine Kante von v_i nach v_{i+1} in `g` existiert. Im Beispielgraphen gibt es also einen Pfad von `1` nach `3`, aber keinen Pfad von `1` nach `4`.) Verwenden Sie hierbei Ihre Datenstruktur aus Teil (d) und geben Sie auch die Typdeklaration von `path` an. Sie dürfen wieder davon ausgehen, dass der Vergleichsoperator `(==)` für alle Typen vordefiniert ist.

Vorname	Name	Matr.-Nr.

Aufgabe 6 (Logische Programmierung in Prolog, $3 + 5 + 2 + 3$ Punkte)

(a) Definieren Sie in Prolog ein dreistelliges Prädikat `last`, welches das letzte Element von einer Liste abtrennt. Genauer bedeutet `last(X,K,L)`, dass `X` das letzte Element der Liste `L` ist und dass `K` die Liste `L` ohne ihr letztes Element ist. Beispielsweise gilt `last(3, [1,2], [1,2,3])`. Hierbei dürfen Sie keine in Prolog vordefinierten Prädikate verwenden.

(b) Definieren Sie ein einstelliges Prädikat `palindrom`, welches ausdrückt, dass eine Liste ein Palindrom ist.
Eine Liste $[\mathtt{A}_1, ..., \mathtt{A}_\mathtt{n}]$ ist ein Palindrom gdw. $A_j = A_{n-j+1}$ für alle $\mathtt{j} \in \{\mathtt{1}, ..., \mathtt{n}\}$. Palindrome sind z.B. `[a,b,a]`, `[a,b,b,a]`, `[a,b,c,b,a]`.

Vorname	Name	Matr.-Nr.

15

(c) Formulieren Sie (basierend auf dem zuvor definierten Prädikat) eine Anfrage, um alle Palindrom-Listen zu berechnen, bei denen die ersten beiden Elemente übereinstimmen.

(d) Betrachten Sie folgendes Prolog-Programm:

```
nachfolger(X,succ(X)).
nachfolger(X,Y) :- nachfolger(X,succ(Y)).
```

Terminiert das Programm bei der Suche nach den ersten drei Antworten auf die folgende Anfrage? Geben Sie ggf. die erste bzw. die ersten beiden bzw. die ersten drei Antworten an.

```
?- nachfolger(zero,U).
```

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

Diplom-Vorprüfung Lösungsvorschlag Informatik I - Programmierung

23. 9. 2002

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 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 ist.
- 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	18	
Aufgabe 2	11	
Aufgabe 3	12	
Aufgabe 4	19	
Aufgabe 5	16	
Aufgabe 6	13	
Summe	89	
Note	-	

Vorname	Name	Matr.-Nr.

Aufgabe 1 (Programmanalyse, 12 + 6 Punkte)

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

```
public class A {
  public int wert = 3;

  public static int eq(A x1, A x2) {
    if (x1 == x2) return 1; else return 2;}
  public int eq(A x) {
   if (this == x) return 3; else return 4;}
}

public class B extends A {
  public B(int wert) {
    this.wert = this.wert + wert;}
  public int eq(A x) {
    if (wert == x.wert) return 5; else return 6;}
  public int eq(B y) {
    if (this == y) return 7; else return 8;}
}

public class H {
  public static void main(String[] args) {
    A a1 = new B(5);
    System.out.println(a1.wert);        // AUSGABE: 8
    B b = (B) a1;
    A a2 = new A();
    System.out.println(a2.wert);        // AUSGABE: 3
    a2.wert = a1.wert;
    System.out.println(B.eq(a1,a2));    // AUSGABE: 2
    System.out.println(a2.eq(a1));      // AUSGABE: 4
    System.out.println(a1.eq(a2));      // AUSGABE: 5
    System.out.println(b.eq(b));}       // AUSGABE: 7
}
```

(b) Das Programm wird um eine zusätzliche Klasse `C` ergänzt. Finden und erklären Sie die drei Fehler in dieser Klasse, die das Compilieren verhindern.

```
public class C extends A {
  private static int eq(A x1, A x2) {
    return 1;}

  public double eq (A obj) {
    A x = obj;
    C c = x;
    return c.wert;}
}
```

Vorname	Name	Matr.-Nr.

- `private static int eq(A x1, A x2) ...` geht nicht, da diese `public`-Methode nicht durch eine Methode mit geringerer Sichtbarkeit überschrieben werden darf. Man müsste `private` durch `public` ersetzen.
- `public double eq (A obj) ...` geht nicht, denn beim Überladen darf nicht bloß der Rückgabetyp verschieden sein. Der Typ `double` müsste durch `int` ersetzt werden.
- `C c = x;` geht nicht, da explizit konvertiert werden muss (d.h., man müsste es z.B. durch `C c = (C) x;` ersetzen).

Vorname	Name	Matr.-Nr.

Aufgabe 2 (Verifikation, 10 + 1 Punkte)

Der Algorithmus P berechnet die Potenz $\mathtt{x}^{\mathtt{y}}$ zweier natürlicher Zahlen $\mathtt{x}$ und $\mathtt{y}$ mit $\mathtt{x} > 0$ und $\mathtt{y} > 0$.

(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: $\mathtt{x}, \mathtt{y} \in \{1, 2, ...\}$
Ausgabe: `res`
Vorbedingung: $\mathtt{x} > 0 \wedge \mathtt{y} > 0$
Nachbedingung: $\mathtt{res} = \mathtt{x}^{\mathtt{y}}$

$\langle\ \mathtt{x} > 0 \wedge \mathtt{y} > 0\ \rangle$

$\langle\ \mathtt{x} > 0 \wedge \mathtt{y} > 0 \wedge \mathtt{y} = \mathtt{y}\ \rangle$

```
z = y;
```

$\langle\ \mathtt{x} > 0 \wedge \mathtt{y} > 0 \wedge \mathtt{z} = \mathtt{y}\ \rangle$

$\langle\ \mathtt{x} > 0 \wedge \mathtt{y} > 0 \wedge \mathtt{z} = \mathtt{y} \wedge 1 = 1\ \rangle$

```
res = 1;
```

$\langle\ \mathtt{x} > 0 \wedge \mathtt{y} > 0 \wedge \mathtt{z} = \mathtt{y} \wedge \mathtt{res} = 1\ \rangle$

$\langle\ \mathtt{x} > 0 \wedge \mathtt{y} > 0 \wedge \mathtt{res} = \mathtt{x}^{\mathtt{y}-\mathtt{z}}\ \rangle$

```
while (z != 0) {
```

$\langle\ \mathtt{x} > 0 \wedge \mathtt{y} > 0 \wedge \mathtt{z} \neq 0 \wedge \mathtt{res} = \mathtt{x}^{\mathtt{y}-\mathtt{z}}\ \rangle$

$\langle\ \mathtt{x} > 0 \wedge \mathtt{y} > 0 \wedge \mathtt{res} * \mathtt{x} = \mathtt{x}^{\mathtt{y}-(\mathtt{z}-1)}\ \rangle$

```
  res = res * x;
```

$\langle \mathtt{x} > 0 \wedge \mathtt{y} > 0 \wedge \mathtt{res} = \mathtt{x}^{\mathtt{y}-(\mathtt{z}-1)}\ \rangle$

```
  z = z - 1;
```

$\langle\ \mathtt{x} > 0 \wedge \mathtt{y} > 0 \wedge \mathtt{res} = \mathtt{x}^{\mathtt{y}-\mathtt{z}}\ \rangle$

```
}
```

$\langle\ \mathtt{x} > 0 \wedge \mathtt{y} > 0 \wedge \mathtt{z} = 0 \wedge \mathtt{res} = \mathtt{x}^{\mathtt{y}-\mathtt{z}}\ \rangle$

$\langle\ \mathtt{res} = \mathtt{x}^{\mathtt{y}}\ \rangle$

Vorname	Name	Matr.-Nr.

5

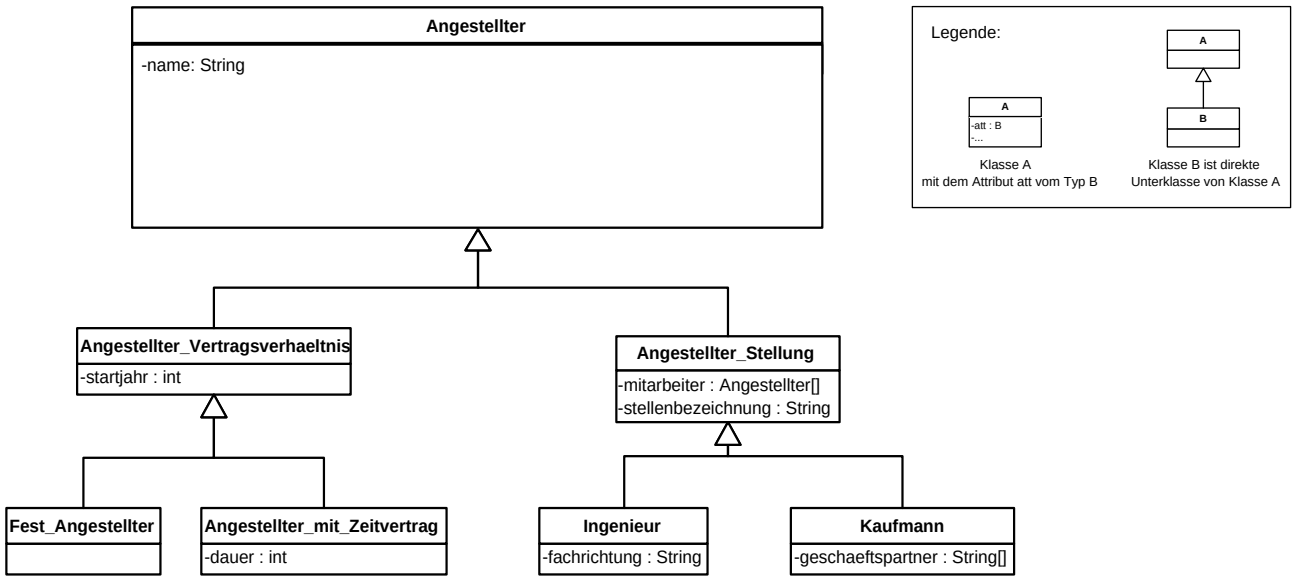
(b) Geben Sie (ohne Beweis) eine Variante für die `while`-Schleife an, die belegt, dass die Schleife für $\texttt{x} > 0 \wedge \texttt{y} > 0$ terminiert.

Die Variante ist z, da z in jedem Schleifendurchlauf um `1` verkleinert wird.

Vorname	Name	Matr.-Nr.

Aufgabe 3 (Datenstrukturen in Java, 2 + 3 + 7 Punkte)

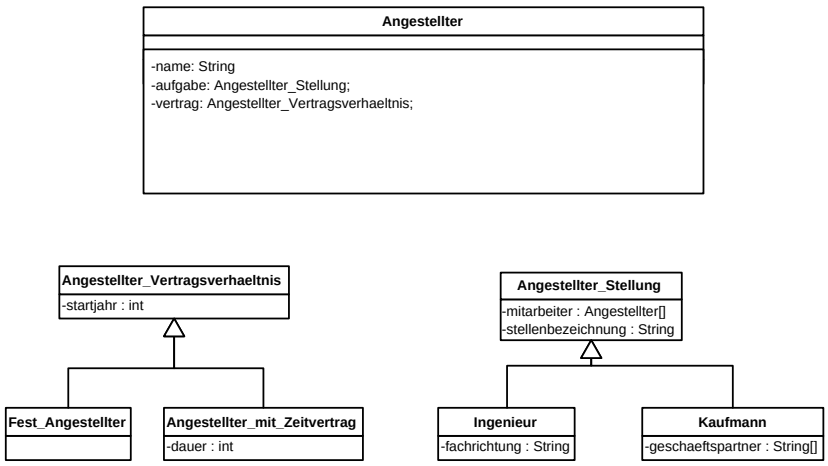
In dieser Aufgabe sollen unterschiedliche Angestelltenverhältnisse modelliert werden. Einen ersten (naiven) Entwurf zeigt die folgende Abbildung:



(a) Erläutern Sie, warum der oben angegebene Entwurf eine schlechte Modellierung der Angestelltenverhältnisse ist.

Das Problem ist, dass `Angestellter_Vertragsverhaeltnis` und `Angestellter_Stellung` als Unterklassen von `Angestellter` modelliert wurden. Auf diese Weise kann man z.B. nicht ausdrücken, dass ein Ingenieur einen Zeitvertrag hat.

(b) Verbessern Sie den obigen Entwurf sinnvoll (durch Veränderung im obigen Klassendiagramm).



Alternativ reicht es auch, nur eine der beiden Klassen `Angestellter_Vertragsverhaeltnis` oder `Angestellter_Stellung` in ein Attribut anstelle einer Unterklasse umzuwandeln.

(c) Die Klasse `Angestellter` soll eine Methode

```
public void drucke(int jahr) {...}
```

enthalten, die den Namen des Angestellten, die Anzahl seiner Mitarbeiter und sein Gehalt im Jahr

Vorname	Name	Matr.-Nr.

`jahr` auf dem Bildschirm ausgibt.

Das Gehalt eines Angestellten bestimmt sich nach der Anzahl der Jahre, die er bereits angestellt ist. Im ersten Jahr verdient ein Angestellter 1000 Euro, im zweiten Jahr 2000 Euro, etc. Angestellte werden immer zu Beginn des Jahres eingestellt und bei Zeitverträgen gibt `dauer` die Vertragsdauer in Jahren an.

Implementieren Sie die Klasse `Angestellter` mit der Methode `drucke`. Hierzu dürfen Sie natürlich beliebige Hilfsmethoden (auch in anderen Klassen) schreiben. Geben Sie auch die Implementierungen von denjenigen anderen Klassen an, in denen Sie Hilfsmethoden schreiben. Implementieren Sie keine Konstruktoren. Sie brauchen auch nur diejenigen Selektoren zu implementieren, die Sie für die Methode `drucke` benötigen. Verwenden Sie soweit wie möglich Vererbungstechniken und Prinzipien der Datenkapselung.

```
public class Angestellter {
    private String name;
    private Angestellter_Stellung aufgabe;
    private Angestellter_Vertragsverhaeltnis vertrag;

    public void drucke(int jahr) {
        System.out.println(name + " hat " + aufgabe.anzahlMitarbeiter() +
                    " Mitarbeiter und verdient " + vertrag.gehalt(jahr));
    }
}
public class Angestellter_Stellung {
    private Angestellter[] mitarbeiter;

    public int anzahlMitarbeiter() {
        if   (mitarbeiter != null)
             return mitarbeiter.length;
        else return 0;
    }
}
public class Angestellter_Vertragsverhaeltnis {
    protected int startjahr;

    public int gehalt(int jahr) {
        int beschaeftigungsdauer = jahr - startjahr + 1;
        if   (beschaeftigungsdauer > 0)
             return beschaeftigungsdauer * 1000;
        else return 0;
    }
}
public class Zeitvertrag extends Angestellter_Vertragsverhaeltnis {
        private int dauer;

        public int gehalt(int jahr) {
            if   (startjahr + dauer - 1 < jahr)
                 return 0;
            else return super.gehalt(jahr);
        }
}
```

Vorname	Name	Matr.-Nr.

Aufgabe 4 (Programmierung in Java, $1 + 6 + 3 + 7 + 2$ Punkte)

Der Surfverband möchte die Rangliste seiner Surfer elektronisch verwalten. Surfer sind Sportler und für die Verwaltung von Sportlern in Ranglisten existieren bereits folgende Vorgaben für die Implementierung:

```
public abstract class Sportler {
  protected String name;
}

public abstract class Element {
  protected Sportler wert;
  protected Element next;
}

public abstract class Rangliste {
  protected Element kopf;
  public Rangliste (Element kopf) {this.kopf = kopf;}
  public abstract Element sucheSportler(Sportler s);
  public abstract void aktualisiere(Sportler gewinner, Sportler verlierer);
}
```

Da beliebig viele Sportler in die jeweilige Rangliste aufgenommen werden sollen, kann die Länge der Rangliste nicht vorher festgelegt werden. Für die Rangliste sind hier nur diejenigen Methoden aufgeführt, die für das Aktualisieren der Liste aufgrund eines Spielergebnisses benötigt werden. In einem Wettkampf treten immer zwei Sportler gegeneinander an und es gibt immer einen Gewinner und einen Verlierer. Nach jedem Wettkampf wird die Rangliste aktualisiert.

Ziel der Aufgabe ist es, (nicht-abstrakte) Unterklassen `Surfer`, `SurferElement` und `SurferRangliste` von `Sportler`, `Element` und `Rangliste` zu implementieren. Hierbei dürfen die oben angegebenen Implementierungen nicht verändert werden.

Ein Surfer zeichnet sich durch einen Namen und eine Surfernummer aus. Sie dürfen davon ausgehen, dass diese Nummer eindeutig ist und brauchen dies in der Implementierung nicht zu überprüfen.

Beachten Sie auch die Sichtbarkeiten der Attribute und Methoden, da die Daten gekapselt werden sollen. Selbstverständlich dürfen Sie beliebig viele geeignete Hilfsmethoden schreiben. Sie brauchen nur die Selektoren zu implementieren, die Sie benötigen.

Gehen Sie bei der Implementierung der Klasse `SurferRangliste` wie folgt vor. Die beschriebenen Methoden können in späteren Aufgabenteilen jeweils als gegeben vorausgesetzt und verwendet werden.

(a) Implementieren Sie einen geeigneten Konstruktor

```
public SurferRangliste (Element kopf) {...}
```

in der Klasse `SurferRangliste`.

(b) Implementieren Sie die Methode `sucheSportler`. Beachten Sie: Diese Methode und die von ihr aufgerufenen Methoden dürfen keine Schleifen enthalten, sondern es darf nur *Rekursion* verwendet werden! Hierbei soll die Methode `sucheSportler (Sportler s)` in der Klasse `SurferRangliste` dasjenige Listenelement zurückliefern, welches den übergebenen Surfer `s` enthält. In der gesamten Aufgabe werden zwei Surfer als gleich betrachtet, wenn ihre Surfernummern gleich sind. Wenn `s` kein Surfer ist oder nicht in der Liste enthalten ist, soll `null` zurückgegeben werden.

Gehen Sie in der gesamten Aufgabe davon aus, dass ein Objekt der Klasse `SurferRangliste` nur `Surfer` und keine anderen `Sportler` enthält. Außerdem dürfen Sie generell voraussetzen, dass eine `SurferRangliste` keinen `Surfer` mehrmals enthält.

Vorname	Name	Matr.-Nr.

(c) Implementieren Sie eine Hilfsmethode

```
private boolean dahinterPlatziert(Surfer a, Surfer b) {...}
```

in der Klasse `SurferRangliste`, die prüft, ob der Surfer `a` in der Rangliste hinter Surfer `b` liegt. Setzen Sie dabei voraus, dass beide Surfer in der Rangliste sind.

(d) Die Methode `aktualisiere (Sportler gewinner, Sportler verlierer)` in der Klasse `SurferRangliste` soll die Rangliste wie folgt aktualisieren: Falls der Surfer `gewinner` bisher hinter dem Surfer `verlierer` in der Rangliste platziert war, wird nun der Surfer `gewinner` in der Rangliste direkt vor dem Surfer `verlierer` platziert. Wenn der Surfer `gewinner` ohnehin schon vor dem Surfer `verlierer` platziert war, so soll sich die Rangliste nicht ändern. Wenn mindestens einer der übergebenen Sportler `gewinner` bzw. `verlierer` nicht in der Rangliste oder kein Surfer ist, so soll sich die Rangliste nicht ändern.

Sie können dabei auf die Methoden `sucheSportler`, `dahinterPlatziert` und eine weitere Methode

```
public SurferElement sucheVorgaenger(Surfer s) {...}
```

in der Klasse `SurferRangliste` zurückgreifen, wobei Sie die Methode `sucheVorgaenger` *nicht* implementieren müssen. Sie gibt das Listenelement zurück, welches vor dem übergebenen Surfer `s` in der Rangliste steht. Wenn es kein solches Listenelement gibt, wird `null` zurückgegeben.

(e) Vergessen Sie nicht, die Klassen `Surfer` und `SurferElement` zu implementieren. Dabei brauchen Sie nur die Methoden (und Konstruktoren) zu schreiben, die Sie in den anderen Aufgabenteilen benutzen.

```
public class Surfer extends Sportler {

    private int surfernummer;

    public boolean equals(Sportler s) {

        if   (s instanceof Surfer)
             return (surfernummer == ((Surfer)s).surfernummer);
        else return false;
    }

}
```

Vorname	Name	Matr.-Nr.


```
public class SurferElement extends Element {

    public SurferElement getNext() {

        if   (next instanceof SurferElement)
             return (SurferElement) next;
        else return null;
    }

    public void setNext(SurferElement next) {

        this.next = next;
    }

    public SurferElement sucheSurfer(Surfer s) {

        if      (s.equals(wert))      return this;
        else if (getNext() == null)   return null;
        else                          return getNext().sucheSurfer(s);
    }

}



public class SurferRangliste extends Rangliste {

    public SurferRangliste (Element kopf) {

        super(kopf);
    }


    public Element sucheSportler(Sportler s) {

        SurferElement skopf = (SurferElement) kopf;

        if   (s instanceof Surfer && skopf != null)
             return skopf.sucheSurfer((Surfer) s);

        else {
             System.out.println("Liste leer oder kein Surfer");
             return null;
        }
    }


    private boolean dahinterPlatziert(Surfer a, Surfer b) {

        SurferElement aElement = (SurferElement) sucheSportler(a);
        return (aElement.sucheSurfer(b) == null);
    }
```

Vorname	Name	Matr.-Nr.


```
    public void aktualisiere(Sportler gewinner, Sportler verlierer) {

        Surfer        gewinnerSurfer, verliererSurfer;

        SurferElement gewinnerElement, verliererElement,
                      vorGewinnerElement, vorVerliererElement;

        if (gewinner != null && verlierer != null &&
            gewinner instanceof Surfer && verlierer instanceof Surfer) {

            gewinnerSurfer   = (Surfer) gewinner;
            verliererSurfer  = (Surfer) verlierer;
            gewinnerElement  = (SurferElement) sucheSportler(gewinner);
            verliererElement = (SurferElement) sucheSportler(verlierer);

            if (gewinnerElement == null || verliererElement == null)
                System.out.println("Mindestens ein Surfer nicht in Rangliste");

            else {

                if   (dahinterPlatziert (verliererSurfer, gewinnerSurfer))
                     System.out.println("Keine Veraenderung in Rangliste");

                else {

                     vorGewinnerElement = sucheVorgaenger(gewinnerSurfer);
                     vorVerliererElement = sucheVorgaenger(verliererSurfer);

                     vorGewinnerElement.setNext(gewinnerElement.getNext());
                     gewinnerElement.setNext(verliererElement);

                     if   (vorVerliererElement == null)
                          kopf = gewinnerElement;
                     else vorVerliererElement.setNext(gewinnerElement);
                }
            }
        }

        else System.out.println("Fehlerhafte Eingabe");
    }
}
```

Vorname	Name	Matr.-Nr.

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

(a) Die Funktionen `f`, `g`, und `h` sind wie folgt definiert:

```
f x = x : [x]

g x y = g y x

h x  = \y -> x ++ (y : x)
```

Geben Sie den allgemeinsten Typ von `f`, `g` und `h` an.

Der allgemeinste Typ von `f` ist `a -> [a]`, der allgemeinste Typ von `g` ist `a -> a -> b` und der allgemeinste Typ von `h` ist `[a] -> a -> [a]`.

(b) Geben Sie für die folgenden Ausdrücke jeweils das Ergebnis der Auswertung an.

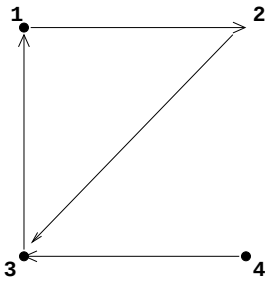
(i) `(\x -> \y -> map x y) (\x -> x + 2) [10, 3, 2]`

(ii) `filter (\x -> x < 7) (filter (\x -> x > 2) [10, 3, 2, 0, 5, 3])`

(i) `[12,5,4]`

(ii) `[3,5,3]`

(c) Graphen mit Knoten eines Typs `a` können als Liste des Typs `[(a,a)]` repräsentiert werden. Hierbei wird jede Kante des Graphens als Paar in der Liste gespeichert. Der folgende Graph



könnte also zum Beispiel durch die Liste `[ (1,2), (2,3), (3,1), (4,3) ]` repräsentiert werden.

Definieren Sie eine Funktion `delete :: a -> a -> [(a,a)] -> [(a,a)]`, wobei `delete(x,y,k)` *alle* Vorkommen der Kante von `x` nach `y` aus der Liste `k` löscht, die den Graphen repräsentiert. Wenn `k` die obige Liste ist, so ergibt `delete (2,3,k)` also das Ergebnis `[ (1,2), (3,1), (4,3)]`. Hierbei dürfen Sie keine in Haskell vordefinierten Funktionen auf Listen außer den Datenkonstruktoren `[]` und `:` verwenden. Sie dürfen davon ausgehen, dass der Vergleichsoperator `(==)` für alle Typen vordefiniert ist.

Vorname	Name	Matr.-Nr.


```
delete :: a -> a -> [(a,a)] -> [(a,a)]
delete x y [] = []
delete x y (p:xs) | (x,y) == p = delete x y xs
                  | otherwise  = p : delete x y xs
```

(d) Definieren Sie in Haskell eine Datenstruktur zur Repräsentation von Graphen, welche der Beschreibung in Teil (c) entspricht. Hierbei dürfen Sie jedoch die in Haskell vordefinierten Listen *nicht* verwenden.

```
data Graph a = Empty | Edge a a (Graph a) deriving Show
```

(e) Implementieren Sie in Haskell eine Funktion `path`, die zwei Knoten `x`, `y` und einen Graph `g` als Eingabe bekommt. Dann soll `path x y g` genau dann `true` liefern, wenn es im Graph `g` einen Pfad von `x` nach `y` gibt oder wenn `x = y` ist. (Ein Pfad in einem Graphen `g` ist eine Liste $[v_1, \ldots, v_n]$ von Knoten aus `g` (mit $n \geq 0$), so dass für alle $i \in \{1, \ldots, n-1\}$ eine Kante von v_i nach v_{i+1} in `g` existiert. Im Beispielgraphen gibt es also einen Pfad von `1` nach `3`, aber keinen Pfad von `1` nach `4`.) Verwenden Sie hierbei Ihre Datenstruktur aus Teil (d) und geben Sie auch die Typdeklaration von `path` an. Sie dürfen wieder davon ausgehen, dass der Vergleichsoperator `(==)` für alle Typen vordefiniert ist.

```
path :: a -> a -> Graph a -> Bool
path x y Empty | x == y    = True
               | otherwise = False
path x y (Edge u v rest) = path x y rest ||
                           (path x u rest) && (path v y rest)
```

Vorname	Name	Matr.-Nr.

Aufgabe 6 (Logische Programmierung in Prolog, $3 + 5 + 2 + 3$ Punkte)

(a) Definieren Sie in Prolog ein dreistelliges Prädikat `last`, welches das letzte Element von einer Liste abtrennt. Genauer bedeutet `last(X,K,L)`, dass `X` das letzte Element der Liste `L` ist und dass `K` die Liste `L` ohne ihr letztes Element ist. Beispielsweise gilt `last(3, [1,2], [1,2,3])`. Hierbei dürfen Sie keine in Prolog vordefinierten Prädikate verwenden.

```
last(X, [],    [X]  ).
last(X, [Y|K], [Y|L]) :- last(X, K, L).
```

(b) Definieren Sie ein einstelliges Prädikat `palindrom`, welches ausdrückt, dass eine Liste ein Palindrom ist.
Eine Liste $[A_1, ..., A_n]$ ist ein Palindrom gdw. $A_j = A_{n-j+1}$ für alle $j \in \{1, ..., n\}$. Palindrome sind z.B. `[a,b,a]`, `[a,b,b,a]`, `[a,b,c,b,a]`.

```
palindrom([]).
palindrom([X]).
palindrom([X|L]) :- last(X, K, L), palindrom(K).
```

Vorname	Name	Matr.-Nr.

(c) Formulieren Sie (basierend auf dem zuvor definierten Prädikat) eine Anfrage, um alle Palindrom-Listen zu berechnen, bei denen die ersten beiden Elemente übereinstimmen.

```
?- palindrom(L), L = [X,X|_].
```

(d) Betrachten Sie folgendes Prolog-Programm:

```
nachfolger(X,succ(X)).
nachfolger(X,Y) :- nachfolger(X,succ(Y)).
```

Terminiert das Programm bei der Suche nach den ersten drei Antworten auf die folgende Anfrage? Geben Sie ggf. die erste bzw. die ersten beiden bzw. die ersten drei Antworten an.

```
?- nachfolger(zero,U).
```

Die erste Antwort ist `U = succ(zero)`, die zweite ist `U = zero` und bei der Suche nach der dritten Antwort terminiert das Programm nicht.