Stichworte

zur Lehrveranstaltung Programmieren 2
im zweiten Semester des Studiengangs Medien-Informatik Bachelor
im Wintersemester 2010/11 an der Beuth-Hochschule für Technik Berlin,
von Ulrich Grude.

In dieser Datei finden Sie nach jedem SU (seminaristischen Unterricht, Vorlesung)
- Wiederholungsfragen für den nächsten SU,
- Antworten zu den Wiederholungsfragen des letzten SU und
- Stichworte zum Stoff, der im letzten SU behandelt wurde.

Die Regeln, nach denen Sie in diesem Semester eine Note für das Fach MB2-PR2 erwerben können, stehen am Anfang der Datei AufgabenPR2.pdf.

1. SU, Mo 27.09.10

Organisation:

Begrüßung

Namensschilder

Wie bekommt man eine Note für MB2-PR2?

Die genauen Regeln stehen am Anfang der Datei `AufgabenPR2.pdf`. Hier ein kurzer Überblick über die Regeln:

Etwa alle zwei Wochen sollen Sie eine Aufgabe fertig bearbeitet haben. Dann schreiben wir einen Test. Insgesamt gibt es 9 Tests (einen Test Nr 3 gibt es nicht, die anderen heißen T1 bis T10).
Sie müssen an mindestens 7 der 9 Tests teilnehmen, sonst bekommen Sie keine Note für MB2-PR2.
Mit jedem Test können Sie maximal 15 Punkte bekommen. Die Punkte ihrer sieben besten Tests werden addiert, aus der Summe wird eine (1.0 bis 5.0) Note berechnet. Dieses Note geht mit einem Gewicht von 25 % in Ihre Note für MB2-PR2 ein. Die restlichen 75 % werden durch die Klausur (am Ende des WS10/11) bzw. die Nachklausur (kurz vor dem SS11) ermittelt.

In welcher Woche wir welchen Test schreiben, steht auf meiner Netzseite (unter MB2-PR2). Falls sich kleine zeitliche Verschiebungen ergeben (wenn z.B. ein Test vorgezogen wird :-), werde ich das vorher in der Vorlesung und auf meiner Netzseite bekannt geben.

Der erste Test (über die Aufgabe 1) wird nächste Woche geschrieben.

Hat jemand eine Frage zur Organisation dieser LV?

Jetzt geht es los mit dem Stoff!

Ein Programmierproblem, das wir noch nicht lösen können

Angenommen, wir sollen in einem Programm von 2 Tastaturen Zahlen einlesen und addieren. An jeder Tastatur sitzt ein Benutzer. Wenn ein Benutzer fertig ist, gibt er die Zahl 0 ein. Wenn beide Benutzer 0 eingegeben haben, sollen wir die Summe aller Zahlen ausgeben.

Warum können wir dieses Problem (noch) nicht befriedigend lösen?

Eine schlechte Lösung: S.495, Beispiel-01

Um dieses und ähnliche Probleme elegant und effizient lösen zu können, hat man das Konzept der *Nebenläufigkeit* (engl. concurrency) entwickelt.

Def.: Zwei Programme (oder Programmteile) sind *nebenläufig*, wenn sie *zeitlich unabhängig* voneinander ausgeführt werden.

Statt von *nebenläufiger* Ausführung sprechen manche Leute von *gleichzeitiger* oder von *paralleler* Ausführung. Diese Ausdrucksweisen sollte man verstehen, aber selbst *nicht benutzen*, aus folgenden Gründen:

Die *Gleichzeitigkeit* von zwei Ereignissen ist keine objektive Eigenschaft der Ereignisse, sondern hängt auch (vom Bewegungs-) Zustand des Betrachters ab. Das folgt aus der allgemeinen Relativitätstheorie von Einstein. Deshalb sollte man den Begriff der Gleichzeitigkeit nur zurückhaltend verwenden.

Das Adjektiv "parallel" drückt schon lange aus, dass zwei Dinge (zwei Geraden) "überall den gleichen Abstand haben". Aber welcher Abstand könnte gemeint sein, wenn zwei Programme "parallel zueinander ausgeführt" werden? Der zeitliche Abstand? Aber der kann ja gerade schwanken. Ein räumlicher Abstand? Aber Programme haben keinen räumlichen Abstand. "Parallele Ausführung" ist also eine etwas verkorkste Metapher.

20.1 Prozesse und Fäden (S. 497)

Heute unterscheidet man zwei Arten von nebenläufigen Einheiten:

Prozesse (eines Betriebssystems)
Fäden (engl. threads) eines Programms.

Gemeinsamkeiten:

Prozesse werden *nebenläufig* zueinander ausgeführt.
Fäden werden *nebenläufig* zueinander ausgeführt.

Unterschiede:

Ein Prozess hat einen *eigenen Adressraum*.
Innerhalb *eines Prozesses* können *mehrere Fäden* ablaufen.
Alle Fäden laufen im selben Adressraum wie ihr Prozess.

Besonders wichtige Unterschiede

Ein Prozesswechsel	ist relativ	*teuer*	(bis zu zig-tausend Maschinenbefehle)
Ein Fadenwechsel	ist relativ	*billig*	(ein paar Maschinenbefehle)
Prozesse	können sich	*nur schwer*	gegenseitig stören
Fäden	können sich	*ganz leicht*	gegenseitig stören (Faden-Fehler)

Um einen richtigen Fadenfehler zu produzieren, braucht man nur zwei Fäden, die beide einen gemeinsamen Wertebehälter (z.B. eine gemeinsame Variable) verändern wollen.

Ein konkretes Beispiel für einen Fadenfehler: Wir haben zwei Fäden `F1` und `F2` und eine `int`-Variable namens `otto`, die momentan den Wert `17` hat. Jeder der beiden Fäden will `otto` um `1` erhöhen.

Achtung: Das Erhöhen einer Variablen wie `otto` erfordert (hardwaremäßig) normalerweise 3 Schritte:

Schritt 1: Der momentane Wert von `otto` (z.B. `17`) wird gelesen
Schritt 2: Der neue Wert für `otto` wird berechnet (z.B. `18`)
Schritt 3: Der neue Wert wird in die Variable `otto` geschrieben.

Wenn die Fäden F1 und F2 beide den Schritt 1 ausführen, bevor der andere den Schritt 3 ausführt, tritt ein Fadenfehler auf und `otto` hat am Ende den (falschen) Wert `18` (statt den richtigen Wert `19`).

Alle Fadenfehler laufen nach diesem einfachen Muster ab.

Trotzdem sind Fadenfehler häufig sehr schwer zu entdecken und zu beseitigen, weil sie *nicht reproduzierbar* sind: Viele Ausführungen eines Programms können "gut gehen" und dann erst tritt ein Fadenfehler auf, obwohl man bei allen Ausführungen "alles genau gleich gemacht hat".

Statt von *Fäden* (engl. threads) kann man auch von *sequentiellen Ausführern* sprechen.

Ein einfaches Java-Programm wird von *einem* sequentiellen Ausführer ausgeführt. "Sequentiell" bedeutet hier: Die *Reihenfolge*, in der dieser Ausführer die Befehle des Programms ausführt, ist durch die Sprache Java (und die Eingaben des Programms) eindeutig festgelegt.

Der Programmierer kann einem (sequentiellen) Ausführer befehlen, weitere (sequentielle) Ausführer (oder: Fäden) zu erzeugen und zu starten. Jeder dieser Ausführer führt dann Befehle in einer festgelegten Reihenfolge aus, aber die einzelnen Ausführer arbeiten *nebenläufig* zu einander (oder: zeitlich unabhängig voneinander).

Vergleich: Eine Großküche eines Restaurants. Alle Rezepte stehen (groß, von weitem lesbar) an einer Wand. Viele Köche führen Rezepte aus, einige Köche das selbe Rezept, andere Köche verschiedene Rezepte, je nach den eingehenden Bestellungen der Gäste. Alle Köche arbeiten nebenläufig zueinander (oder: jeder Koch ist ein Faden im Ablauf der gesamten Küche).

Zur Entspannung: Java Entwicklungstendenzen (Mitte 2010)

Die Sprache Java (konkret: Das Markenzeichen Java und eine Reihe damit zusammenhängender Patente) gehören der Firma Sun Microsystems. Diese Firma wurde im April 2009 von der Firma Oracle aufgekauft. Damit gehört auch Java jetzt der Firma Oracle.

Die Firma Sun war bisher mit "ihrer Sprache Java" sehr "behutsam" umgegangen, hatte viele andere Firmen an der Weiterentwicklung beteiligt und niemanden daran gehindert, Java zu benutzen. Oracle hat dagegen (kurz nach dem Erwerb von Sun) eine große Patentklage gegen Google erhoben: Google habe mit seinem Handy-Betriebssystem Android mehrere Java-Patente verletzt.

Über das Ziel, das Oracle mit dieser Klage verfolgt, und die Aussichten dieser Klage gibt es viele verschiedene Spekulationen. Etwas einheitlicher und weiter verbreitet ist die Vermutung, dass diese Klage einen eher verunsichernden und abstoßenden Effekt auf die Entwickler von quelloffener Software haben wird. Das wiederum könnte die Weiterentwicklung von Java beeinflußen.

Die nächste Java-Version (Java 7) war schon lange angekündigt, ihr Erscheinen wurde aber mehrmals verschoben. Seit kurzem kann man von Oracle/Sun einen *Technology Preview* von Java 7 herunterladen. Eine *abgespeckte Version* von Java 7 könnte evtl. Mitte 2011 fertig sein, die *ursprünglich geplante* Version 7 erst Mitte 2012.

Grabo-Programme und Fäden

Ein Java-Programm mit einer grafischen Benutzeroberfläche (Grabo-Programm) wird immer von mindestens *zwei* sequentiellen Ausführern (oder: Fäden) ausgeführt.

Der Hauptfaden (engl. main thread) führt die `main`-Methode aus.
Der Ereignisfaden (engl. event thread) zeichnet die Grabo und reagiert auf Aktionen des Benutzers.

Wie man mit diesen beiden Fäden *Fadenfehler* (engl. race conditions) produzieren kann, wird im Abschnitt 22.5 (S. 555) anhand der Programme `Wettlauf01` und `Wettlauf02` erläutert. Als gemeinsamer Wertebehälter der beiden Fäden wird der Bildschirm verwendet.

20.2 Die Klasse Thread und die Schnittstelle Runnable (S. 500)

Nur kurzer Hinweis. Das Kapitel soll von den StudentInnen zu Hause gelesen werden.

Wiederholungsfragen, 2. SU, Mo 04.10.10

1. Welche Eigenschaft haben sowohl *Prozesse* eines Betriebssystem als auch *Fäden* (engl. threads) eines Programms?

2. Was ist besser an Prozessen als an Fäden?

3. Was ist besser an Fäden als an Prozessen?

4. Wie heißen die beiden Fäden, von denen ein (in Java geschriebenes) Grabo-Programm (engl. GUI program) ausgeführt wird?

5. Was machen diese beiden Fäden (d.h. was macht der eine Faden und was macht der andere Faden)?

6. Was braucht man (mindestens), um einen *Fadenfehler* zu verursachen?

7. Beschreiben Sie kurz ein *konkretes Beispiel*, wie ein Fadenfehler ablaufen kann.

Rückseite der Wiederholungsfragen, 2. SU, Mo 04.10.10

```
// Datei Faden30Tst.java
/* ---------------------------------------------------------------------------
Zwei Faeden (engl. threads) namens O und X versuchen mehrmals, die globale
StringBuilder-Variable text ganz mit 'O's bzw. 'X'en zu fuellen und
auszugeben. Die Ausgaben enthalten manchmal Mischungen aus 'O's und 'X'en.
--------------------------------------------------------------------------- */
import java.util.Random;

class Faden30 extends Thread {
   static StringBuilder text = new StringBuilder("XXXXXXXXXXXXXXXXXXX");

   Random rand = new Random();
   String name;

   public Faden30(String name) {
      super(name);
      this.name = name;
   } // Konstruktor Faden30

   public void run() {
      try {
         for (int i=1; i<=5; i++) {
            for (int j=0; j<text.length(); j++) {   // text fuellen
               text.replace(j, j+1, name);
               Thread.sleep(rand.nextInt(10));
            }
            System.out.println(name + ": " + text); // text ausgeben
            Thread.sleep(rand.nextInt(10));
         }
      } catch (InterruptedException ex) {
         System.out.println("Ausnahme in Methode run: " + ex);
      }
   }
} // class Faden30
// ============================================================================
public class Faden30Tst {

   static public void main(String[] _) {
      System.out.println("Faden30Tst:  Los jetzt!");
      Faden30 fO = new Faden30("O");
      Faden30 fX = new Faden30("X");
      fO.start();
      fX.start();
      System.out.println("Faden30Tst:  Das war's!");
   } // main

} // class Faden30Tst
/* ============================================================================
Drei moegliche Ausgaben des Programms Faden30Tst (nebeneinander):

Faden30Tst:  Los jetzt!   Faden30Tst:  Los jetzt!   Faden30Tst:  Los jetzt!
Faden30Tst:  Das war's!   Faden30Tst:  Das war's!   Faden30Tst:  Das war's!
X: XOOOOOOOOOOOOOOOXXX   O: OXXXXXXXOXXXXOXXXXXX   O: XOOOOOOOOXXXOXOOOOXO
O: XXOOOOOOOOOOOOOOOOO   X: OXXXXXXXOXXXXOXXXXXX   X: OOOOOOOOOXXXOXOOOOXX
X: OOOOOOOOOOXXXOOOOOOX   X: XOOXOOOOOOOOOOOOOXXX   O: XXXXXXXXXXXXXXXOOOOX
O: XXOOOOOOOOXXXOOOOOOO   O: XXOXOOOOOOOOOOOOOOOO   X: OXXXXXXXXXXXXXXOOOOX
X: OOOOOOOOOOOOOOOOOXX   X: OOOOOOOOOOOOOOOOOOOO   X: XXOOOOOOOOOOOXXXXXXX
O: XXXXOOOOOOOOOOOOOOOO   O: XOOOOOOOOOOOOOOOOOO   O: XXXXXXOOOOOOOOOOOOOO
O: OOOOOOOOOOOOOOXXXXXX   X: OOXXXXXOOOOOOOOOOOO   X: OOOOOOOOOOOOOOOXXXXX
X: OOOOOOOOOOOOOOXXXXXX   O: XOXXXXXOOOOOOOOOOOO   O: XXXXOOOOOOOOOOOOOOOO
O: XXXXXXXXXXXXXXXXXXOO   O: OOOOXXXXXXXXXXXXXXOO   X: OOOOOOOOOOOOOOOXXXXX
X: XXXXXXXXXXXXXXXXXXXX   X: OOOOXXXXXXXXXXXXXXXX   O: OOOOOOOOOOOOOOOOOOO
============================================================================ */
```

Wiederholungsfragen, 2. SU, Mo 04.10.10

1. Welche Eigenschaft haben sowohl *Prozesse* eines Betriebssystem als auch *Fäden* (engl. threads) eines Programms?

Sie (Prozesse bzw. Fäden) werden nebenläufig zueinander ausgeführt.

2. Was ist besser an Prozessen als an Fäden?

Jeder Prozess hat einen eigenen Adressraum.

3. Was ist besser an Fäden als an Prozessen?

Ein Fadenwechsel geht viel schneller ("ist viel billiger") als ein Prozesswechsel.

4. Wie heißen die beiden Fäden, von denen ein (in Java geschriebenes) Grabo-Programm (engl. GUI program) ausgeführt wird?

Hauptfaden (engl. main thread)
Ereignisfaden (event thread)

5. Was machen diese beiden Fäden (d.h. was macht der eine Faden und was macht der andere Faden)?

Hauptfaden: Er führt die main-Methode aus.

Ereignisfaden: Er zeichnet die Grabo und reagiert auf Aktionen des Benutzers (z.B. auf Bewegungen der Maus, Mausklicks, Tastendrücke etc.).

6. Was braucht man (mindestens), um einen Fadenfehler zu verursachen?

Zwei Fäden und einen Wertebehälter (dessen Inhalt von beiden Fäden verändert werden soll).

7. Beschreiben Sie kurz ein konkretes Beispiel, wie ein *Fadenfehler* ablaufen kann.

Zwei Fäden F1 und F2 sollen den Wert einer int-Variable namens emil jeweils verdoppeln. Die Variable emil habe anfangs den Wert 5 (und sollte somit am Ende den Wert 20 haben).

F1 liest den Wert (5) von emil.
F2 liest den Wert (5) von emil.
F1 berechnet den neuen Wert (10) und schreibt ihn nach emil.
F2 berechnet den neuen Wert (10) und schreibt ihn nach emil.

Am Ende hat Emil den Wert 10 statt 20.

2. SU, Mo 04.10.10

A. Wiederholung
B. Organisation

10.5 Die Klasse Random und der Zufall (S. 245)

Wann/wozu braucht man beim Programmieren Zufallszahlen?
(z.B. wenn man Glücksspiele programmiert, oder zum automatischen Erzeugen von "großen Mengen" von Testdaten)

S. 245, Beispiel-1: Vereinbarungen von Random-Objekten mit und ohne *Keim* (engl. seed)

S. 246, Beispiel-2: Verschiedene `next`-Funktionen eines `Random`-Objekts

Wenn man die (parameterlose) Funktion `int nextInt()` wiederholt aufruft, bekommt man jeden `int`-Wert *einmal*, bevor man den ersten Wert *ein zweites Mal* bekommt.

Der Unterschied zwischen *reporduzierbaren* und *nicht reproduzierbaren* Zufallszahlen.

Nebenläufigkeit (Fortsetzung und Ende)

Besprechung des Programms `Faden30Tst` auf der Rückseite der Wiederholungsfragen

Wie viele *Klassen* werden hier vereinbart und wie heißen sie? (Zwei, `Faden30` und `Faden30Tst`)

Eine der Klassen ist vor allem ein *Bauplan*, die andere vor allem ein *Modul*. Welche ist was? (`Faden30` ist vor allem ein Bauplan, `Faden30Tst` ist vor allem ein Modul).

Wie viele *Elemente* enthält der Modul `Faden30Tst` und wie heissen sie? (Ein Element, `main`).

Wie viele *Elemente* werden in jedes `Faden30`-Objekt eingebaut und wie heissen sie? (Drei Elemente, `rand`, `name`, `run`).

Wie viele *Elemente* enthält der Modul `Faden30` und wie heissen sie? (Ein Element, `text`).

Wie viele *Module* existieren, wenn die Ausführung des Programms `Faden30Tst` "in vollem Gange" ist und wie heißen diese Module? (Im Wesentlichen sind es 4 Module: Die zwei Klassen `FadenTst` und `Faden30` und die zwei Objekte `f0` und `fX`. Außerdem existieren noch ein paar weitere Objekte: `text`, `rand`, `name` und für jedes im Programm vorkommende `String`-Literal (sieben Stück) ein entsprechendes `String`-Objekt).

Wie viele Ausführer (Fäden, Köche) sind an der Ausführung des Programms `Faden30Tst` beteiligt, wie heissen diese Ausführer und "was führen sie aus" (oder: "Welcher Koch kocht welches Rezept")?
(Drei Ausführer oder Köche, sie heißen `"main"`, `"0"` und `"X"`, siehe Zeilen 40, 41 und 16.
Der Faden namens `"main"` führt die Methode `main` aus.
Der Faden namens `"0"` führt die Methode `f0.run()` aus.
Der Faden namens `"X"` führt die Methode `fX.run()` aus.

Wodurch unterscheiden sich die Methoden `f0.run()` und `fX.run()`? (Die Methode `f0.run` füllt die Variable `text` mit `"0"`s, die Methode `fX.run` füllt die Variable `text` mit `"X"`en).

Anmerkung: Die `Thread.sleep-Befehle` sind in diesem kleinen Demo-Programm eine Art "Vertretung" oder "Ersatz" für die Befehle, die in einem "ernsthaften, großen Programm" nützliche Arbeit leisten und dafür mal etwas mehr oder etwas weniger Zeit verbrauchen würden.

Monitore (in Java: Die synchronized-Anweisung)

Wie kann man verhindern, dass die Fäden `f0` und `fX` sich "gegenseitig stören"?

Indem man die innere `for`-Schleife und den `println`-Befehl (Zeile 23 bis 27) mit der `synchronized`-Anweisung zu einem *Monitor* macht, etwa so:

```
22    for (int i=1; i<=5; i++) {
23       synchronized(text) {
24          for (int j=0; j<text.length(); j++) {   // text fuelle
25             text.replace(j, j+1, name);
26             Thread.sleep(rand.nextInt(10));
27          }
28          System.out.println(name + ": " + text); // text ausgeben
29       }
30       Thread.sleep(rand.nextInt(10));
31    }
```

Der Monitor (oder: synchronized-Block) in Zeile 23 bis 29 hat folgende Eigenschaft:

Wenn er gerade von einem Faden F0 ausgeführt wird und weitere Fäden F1, F2, ... versuchen, ihn auch auszuführen, werden die weiteren Fäden F1, F2, ... "schlafen gelegt". Erst wenn F0 den Monitor fertig ausgeführt hat, wird einer der weiteren Fäden "aufgeweckt" und darf den Monitor (allein) ausführen. Und wenn er fertig ist, wird der "nächste Schläfer vor dem Monitor" aufgeweckt etc., bis kein Faden mehr "vor dem Monitor wartet".

Zur Entspannung:
Was kostet eine Stunde Lehrveranstaltung pro TeilnehmerIn an der Beuth Hochschule?

Wir wissen bereits: Das Studieren an der Beuth-Hochschule kostet ca.
5000 Euro [pro StudentIn und Jahr] (ohne die "privaten Ausgaben" für Wohnen, Essen etc.).

Eine StudentIn hat ca. 30 [Stunden pro Woche] LVs (wobei eine Stunde 45 Minuten hat).
Ein Semester ist ca. 16 Wochen lang (WS: ca. 17 Wochen, SS: ca. 15 Wochen).
Das sind ca. 30 * 16 gleich 480 (≈ 500) [Stunden pro Semester] oder ca. 1000 [Stunden pro Jahr].
Also kostet eine *Stunde* Lehrveranstaltung ca. 5000/1000 gleich 5 [Euro pro TeilnehmerIn] (und ein *Block* kostet demnach ca. 10 [Euro pro TeilnehmerIn]).

Zur Vereinfachung wurde hier davon ausgegangen, dass jede Stunde Lehrveranstaltung gleich teuer ist. Tatsächlich sind Übungen im Allgemeinen teuerer als Vorlesungen und Lehrveranstaltungen in den einzelnen Fachbereichen und Studiengängen unterscheiden sich vermutlich auch.

Rekursion (siehe auch das Papier Rekursion.pdf auf meiner Netzseite)

Was macht der Programmierer wenn er möchte, dass ein bestimmter Befehl (z.B. `pln("*");`) *mehrmals* ausgeführt wird? (Er schreibt den Befehl in den *Rumpf einer Schleife*).

Schleifen bewirken (normalerweise), dass Befehle *mehr als einmal* ausgeführt werden. Aber es gibt noch ein anderes Konstrukt, mit dem man das auch bewirken kann: *Rekursion.*

Anmerkung: Die Idee der Rekursion ist viel älter als die Idee von `for`- und `while`-Schleifen.

Beispiel: Die folgende *rekursive Definition* der Fibonacci-Zahlenfolge steht schon in einem Mathematikbuch, welches vermutlich im Jahre 1202 in Pisa (in Italien) erschien (ca. 200 Jahre bevor Johannes Gutenberg geboren wurde, es gab also nur handgefertigte Abschriften von Büchern):

Sei `fib(0)` gleich `0` und `fib(1)` gleich `1`.
Sei `fib(n+2)` gleich `fib(n) + fib(n+1)`.

Diese Definition ist *rekursiv* ("auf sich selbst zurückgreifend"), weil die Funktion `fib` mit Hilfe der Funktion `fib` definiert wird (`fib(n+2)` wird mit Hilfe von `fib(n)` und `fib(n+1)` definiert).

Berechnen Sie (mit Papier und Bleistift) die Zahlen `fib(2)` bis `fib(10)`.

n	0	1	2	3	4	5	6	7	8	9	10	...
fib(n)	0	1	1	2	3	5	8	13	21	34	55	...

Frage: Wie kann man aus n die Zahl `fib(n)` direkt berechnen, ohne vorher alle Zahlen `fib(0)`, `fib(1)`, `fib(2)`, ..., `fib(n-1)` zu berechnen.

Antwort: Na ganz einfach :-) nach der folgenden Formel:

$$fib(n) = \frac{1}{\sqrt{5}}\left[\left(\frac{(1+\sqrt{5})}{2}\right)^n - \left(\frac{(1-\sqrt{5})}{2}\right)^n\right]$$

Diese *nicht-rekursive* Formel fand der Mathematiker *Abraham de Moivre* im Jahre 1730 (mehr als 500 Jahre nachdem Fibonacci von Pisa die *rekursive* Definition veröffentlicht hatte).

Merkregel für uns: In manchen Fällen ist eine rekursive Definition *viel einfacher* und leichter verständlich als eine entsprechende *nicht-rekursive* Definition.

Anmerkung: Mit Hilfe von Fibonacci-Zahlen kann man unter anderem *Wachstumsprozesse* besonders verständlich beschreiben, z.B. die Vermehrung von Kaninchen.

Wiederholungsfragen, 3. SU, Mo 11.10.10

1. Wozu braucht man (beim Programmieren) Zufallszahlen? Geben Sie zwei Anwendungsgebiete an (ganz kurz, zwei oder drei Worte pro Anwendungsgebiet genügen).

2. Welche beiden Arten von Zufall (konkret: von Zufallszahlen) unterscheidet man allgemein?

3. Wann braucht man die eine Art von Zufall und wann braucht man die andere Art von Zufall?

4. Betrachten Sie die folgenden Befehle:

```
1 Random rand01 = new Random();
2 int n1 = rand01.newInt();
3 int n2 = rand01.newInt(100);
```

Beschreiben Sie (möglichst kurz):

4.1. Mit was für einem Wert wird die Variable `n1` initialisiert?
4.2. Mit was für einem Wert wird die Variable `n2` initialisiert?

5. Die Fibonacci-Funktion `fib`, angewendet auf eine natürliche Zahl `n`, liefert als Ergebnis die `n`-te Fibonacci-Zahl `fib(n)`. Geben Sie die rekursive Definition dieser Funktion `fib` an.

Rückseite der Wiederholungsfragen, 3. SU, Mo 11.10.10, Arbeitsblatt

Angenommen, unser Java-Ausführer wurde von einer hungrigen Maus angeknabbert und funktioniert seitdem nur noch eingeschränkt:

1. Zum Typ `int` gehören nur noch die nicht-negativen ganzen Zahlen 0, 1, , 2, 3, ...
2. Von den `int`-Operationen `+, -, *, /, %, ++, --, ==` funktionieren nur noch die letzten drei (`++, --, ==`). Alle anderen Rechenoperationen funktionieren nicht mehr, aber man kann noch *Funktionen vereinbaren* und *aufrufen*.

Für diesen eingeschränkten Java-Ausführer haben wir als Fingerübung die folgenden beiden Funktionen vereinbart:

```
1  int plus1 (int n) {return ++n;}
2  int minus1(int n) {return --n;}
```

die wie erwartet gut funktionieren.

Damit könne wir eine rekursive *Additionsfunktion* `add` vereinbaren, etwa so:

```
3  int add(int m, int n) {
4     if (n == 0) {
5        return m;                        // m + 0 = m
6     } else {
7        return plus1(add(m, minus1(n))); // m + n = (m + (n-1)) + 1
8     }
9  }
```

Ganz ähnlich (und mit Hilfe dieser `add`-Funktion) kann man auch eine rekursive *Multiplikationsfunktion* `mul` vereinbaren. Es folgt hier eine unvollständige Version einer solchen Vereinbarung:

```
10  int mul(int m, int n) {
11     if (          ) {
12        return
13     } else if (              ) {
14        return
15     } else {
16        return
17     }
18  }
```

Können Sie die fehlenden Ausdrücke (5 Stück, 2 nach den beiden `if`s und 3 nach `return`) ergänzen?

Es folgt das Skelett einer Potenzierungsfunktion:

```
19  int pot(int m, int n) {
20     // Liefert die n-te Potenz von m (d.h. den Wert m hoch n).
21     ...
22  }
```

Können Sie diese Funktion vereinbaren? Natürlich dürfen Sie alle bisher vereinbarten Funktionen (`plus1`, `minus1`, `add`, `mul`) benutzen. Und die Funktion `pot` darf sich natürlich auch selbst aufrufen (wie `add` und `mul`).

Antworten zu den Wiederholungsfragen, 3. SU, Mo 12.10.10

1. Wozu braucht man (beim Programmieren) Zufallszahlen? Geben Sie zwei Anwendungsgebiete an (ganz kurz, zwei oder drei Worte pro Anwendungsgebiet genügen).

Zum Testen, zum Spiele programmieren.

2. Welche beiden Arten von Zufall (konkret: von Zufallszahlen) unterscheidet man allgemein?

Reproduzierbaren Zufall, nicht-reproduzierbaren Zufall.

3. Wann braucht man die eine Art von Zufall und wann braucht man die andere Art von Zufall?

Zum Testen:	Reproduzierbaren Zufall
Zum Spiele programmieren:	Nicht-reproduzierbaren Zufall

4. Betrachten Sie die folgenden Befehle:

```
1 Random rand01 = new Random();
2 int n1 = rand01.newInt();
3 int n2 = rand01.newInt(100);
```

Beschreiben Sie (möglichst kurz):

4.1. Mit was für einem Wert wird die Variable `n1` initialisiert?
4.2. Mit was für einem Wert wird die Variable `n2` initialisiert?

n1 wird mit irgendeinem int-Wert initialisiert.
n2 wird mit einem int-Wert zwischen 0 und 99 initialisiert.

5. Die Fibonacci-Funktion `fib`, angewendet auf eine natürliche Zahl `n`, liefert als Ergebnis die `n`-te Fibonacci-Zahl `fib(n)`. Geben Sie die rekursive Definition dieser Funktion `fib` an.

fib(0) = 0
fib(1) = 1
fib(n) = fib(n-1) + fib(n-2)

Lösungen für die Aufgaben auf dem Arbeitsblatt
(auf der Rückseite der Wiederholungsfragen für den 3. SU, Mo 11.10.10):

Die vorgegebenen Funktionen:

```
 1  int plus1 (int n) {return ++n;}
 2  int minus1(int n) {return --n;}

 3 int add(int m, int n) {
 4     if (n == 0) {
 5        return m;                        // m + 0 = m
 6     } else {
 7        return plus1(add(m, minus1(n))); // m + n = (m + (n-1)) + 1
 8     }
 9  }
```

Die rekursive Multiplikationsfunktion `mul`:

```
10  int mul(int m, int n) {
11     // Liefert das Produkt von m und n:
12     if (n == 0) {
13        return 0;                          // m * 0 = 0
14     } else if (n == 1) {
15        return m;                          // m * 1 = m
16     } else {
17        return add(m, mul(m, minus1(n)));  // m * n = m + (m * (n-1))
18     }
19  }
```

Die rekursive Potenzierungsfunktion `pot`:

```
20  int pot(int m, int n) {
21     // Liefert die n-te Potenz von m ("m hoch n"):
22     if (n == 0) {
23        return 1;                          // m hoch 0 = 1
24     } else {
25        return mul(m, pot(m, minus1(n)));  // m hoch n = m * (m hoch (n-1))
26     }
27  }
```

3. SU, Mo 11.10.10

A, Wiederholung
B. Organisation

Rekursion, Fortsetzung

Beispiel: Noch eine wichtige rekursive Definition:

Ein binärer Baum
ist entweder ein *Leerbaum* oder
er besteht aus einem *Knoten* K, an dem zwei *Bäume* B1 und B2 hängen.

Was der Begriff *Baum* bedeutet, wird hier mit Hilfe des Begriffs *Baum* (genauer: Bäume) definiert.

Beim Programmieren unterscheidet man *rekursive* und *nicht-rekursive* Methoden
(oder: Unterprogramme). Welche Methoden bezeichnet manl als rekursiv? (Solche, die "auf sich selbst zurückgreifen", d.h. die sich selbst aufrufen).

Was versteht man unter der *Fakultät einer natürlichen Zahl n*? (Das Produkt aller natürlichen Zahlen von 1 bis n. Per vernünftiger Definition ist die Fakultät von 0 und die von 1 gleich 1).

Beispiel: Eine rekursive Methode, die die Fakultät einer natürlichen Zahl berechnet

```
1  int fak(int n) {
2     if (n < 2) {
3        return 1;
4     } else {
5        return n * fak(n-1);
6     }
7  }
```

Direkt rekursive Methoden und indirekt rekursive Methoden

Wenn eine Methode `m1` die Methode `m1` aufruft, dann ist m1 *direkt rekursiv*.

Wenn eine Methode `m1` eine Methode `m2` aufruft, und `m2` die Methode `m3` aufruft, und ...
`m16` die Methode m17 aufruft und `m17` die Methode `m1` aufruft, dann ist `m1` *indirekt rekursiv*.

Grundregel für rekursive Methoden:

Der Rumpf einer rekursiven Methode muss immer aus einer *Fallunterscheidung* bestehen
(z.B. aus einer `if`-Anweisung oder einer `switch`-Anweisung), die Folgendes unterscheidet:
nicht-rekursive Fälle (in denen die Methode sich *nicht* rekursiv aufruft) und
rekursive Fälle (in denen die Methode sich rekursiv aufruft)

Begründung:

Wenn die Methode sich in *keinem Fall* rekursiv aufruft ist sie nicht rekursiv.
Wenn die Methode sich in *allen Fällen* rekursiv aufruft, ist sie "endlos-rekursiv".

Im Rumpf der `fak`-Funktion (siehe oben):
Wie viele nicht-rekursive Fälle und wo? (einer, in Zeile 3)
Wie viele rekursive Fälle und wo? (einer, in Zeile 5).

Zur Entspannung: Der EPR-Effekt

Einstein mochte die Quantenmechanik nicht. Er dachte sich mehrere Gedankenexperimente aus, die zigen sollten, dass sie fehlerhaft ist oder zumindest "keine vollständige Beschreibung der Natur" liefert. Nils Bohr vertrat die Quantenmechanik und fühlte sich zuständig für ihre Verteidigung. Mehrmals gelang es ihm, in einem Gedankenexperiment von Einstein einen subtilen Fehler zu entdecken und die Argumente von Einstein dadurch zu entkräften. Bei *einem* der Gedankenexperimente gelang ihm das aber nicht.

Einstein mochte die Quantenmechanik nicht, kannte sie aber offenbar so genau, dass er auch einige ihrer entfernten Konsequenzen erkennen konnte. Den Effekt, der heute als *EPR-Effekt* bezeichnet wird (nach Einstein, seinem Physiker-Kollegen Podolski und einem Studenten Rosen) ist eine Konsequenz der Quantenmechanik, die Einstein "so unsinnig und unglaubhaft" vorkam, dass er sie als *Argument gegen die Quantenmechanik* veröffentlichte. Inzwischen hat man diesen merkwürdigen Effekt experimentell nachgewiesen und benutzt ihn zur *abhörsicheren Übertragung von Daten.*

Das Arbeitsblatt (mit `plus1`, `minus1`, `add`, unvollständiger `mul`-Funktion etc.).

Die `add`-Funktion besprechen.
Die `mul`-Funktion vervollständigen lassen.
Die `pot`-Funktion sollte zu Hause gelöst werden.

Falls noch Zeit ist, folgende Aufgabe stellen, bearbeiten lassen, dabei die einzelnen TeilnehmerInnen oder Gruppen individuell ein bisschen unterstützen, evtl. eine Lösung besprechen.

Schreiben Sie eine Methode, die der folgenden Spezifikation entspricht:

```
 1 static void gibAus(int[] ir, int i) {
 2    // Macht nichts, falls i groesser oder gleich ir.length
 3    // oder kleiner als 0 ist.
 4    // Gibt sonst die Komponente ir[i] und alle nachfolgenden
 5    // Komponenten von ir zur Standardausgabe aus, jede Komponente
 6    // auf einer neuen Zeile.
```

Eine Lösung:

```
 1 static void gibAus(int[] ir, int i) {
 2    // Macht nichts, falls i groesser oder gleich ir.length
 3    // oder kleiner als 0 ist.
 4    // Gibt sonst die Komponente ir[i] und alle nachfolgenden
 5    // Komponenten von ir zur Standardausgabe aus, jede Komponente
 6    // auf einer neuen Zeile.
 7
 8    if (n < 0 || ir.length <= n) return;
 9    pln("ir[" + i + "]: " + ir[i]);
10    gibAus(ir, i+1);
11 }
```

Falls immer noch Zeit ist: Was müsste man an dieser Lösung ändern, damit die Komponenten in umgekehrter Reihenfolge ausgegeben werden?

Antwort: Zeile 9 und Zeile 10 vertauschen.

Wiederholungsfragen, 4. SU, Mo 18.10.10

1. Warum ist die folgende Methode *keine* vernünftige rekursive Methode?

```
 1 static int berechneA(int n) {
 2    if (n == 0) {
 3       return berechneA(n-1);
 4    } else {
 5       return berechneA(n+1);
 6    }
 7 }
```

2. Warum ist die folgende Methode *keine* vernünftige rekursive Methode?

```
 8 static int berechneB(int n) {
 9    if (n == 0) {
10       return n - 1;
11    } else {
12       return n + 1;
13    }
14 }
```

3. Warum ist die folgende Methode *keine* vernünftig rekursive Methode?

```
15 static berechneC(int n) {
16    if (n == 0) {
17       return 17;
18    } else {
19       return berechneC(n);
20    }
21 }
```

4. Eine Firma stellt *Roboter* her, die sich *selbst reproduzieren* können. Genauer: 3 solche Roboter brauchen 1 Woche, um einen weiteren Roboter ihresgleichen zu bauen. Die Firma beginnt die Produktion *mit 3 Robotern* und setzt alle ihre Roboter zur Produktion weiterer Roboter ein. Wie viele Roboter hat die Firma nach 1 Woche? Nach 2 Wochen? Nach 3 Wochen? Nach 4 Woche? etc.

Schreiben Sie eine rekursive Funktion, die der folgenden Spezifikation entspricht:

```
22 static long anzRobos(long w) {
23    // Verlaesst sich darauf, dass w gleich oder groesser 0 ist.
24    // Wie viele Roboter hat die Firma nach w Wochen?
25    // Diese Funktion liefert die Antwort.
26    ...
27 }
```

Tip: Beantworten Sie sich die folgenden Fragen:

Frage 1. Für welche Wochenzahl `w` kann man die Anzahl der danach vorhandenen Roboter leicht angeben? Besonders *ganz einfache, triviale* Antworten auf diese Frage können nützlich sein!

Frage 2. Angenommen, die Firma hat schon n Wochen produziert. Jetzt wollen wir herausfinden, wie viele Roboter in *der letzten* dieser n Wochen (in der n-ten Woche) produziert wurden. Welche Zahlen brauchen wir, um diese Anzahl zu berechnen? Wie können wir dieses Zahlen berechnen?

Rückseite der Wiederholungsfragen, 4. SU, Mo 18.10.00, noch ein Arbeitsblatt zur Rekursion)

Betrachten Sie die folgende (mehr oder weniger vernünftige :-) *rekursive Funktion*:

```
1    static String rek37(int n) {
2       if (n % 5 == 0) {
3          return "" + n;
4       } else {
5          return n + " " + rek37(n+1);
6       }
7    } // rek37
```

Aufgabe-01: Führen Sie die Funktion `rek37` mit verschiedenen aktuellen Parametern und mit Hilfe der folgenden *Wertetabelle* aus:

n	**rek37(n)**
...	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	
26	
...	

Aufgabe-2: Betrachten Sie noch einmal die Funktion `rek37`:

Welche Parameter-Werte n erfordern 0 rekursive Aufrufe?
Welche Parameter-Werte n erfordern 1 rekursiven Aufruf?
Welche Parameter-Werte n erfordern 2 rekursive Aufrufe?
Welche Parameter-Werte n erfordern 3 rekursive Aufrufe?
Welche Parameter-Werte n erfordern 4 rekursive Aufrufe?

Aufgabe-3: Die Funktion `rek37` legt eine *Halbordnung* für ihre Parameter (`int`-Werte) fest:
Ein `int`-Wert `n2` ist *kleiner* als ein `int`-Wert `n1` wenn
bei der Auswertung (Ausführung) des Ausdrucks `rek37(n2)` auch
der Ausdruck `rek37(n1)` ausgewertet (ausgeführt) wird.

Beschreiben Sie diese Halbordnung, indem Sie ein paar "Ketten" der Form
`...< n1 < n2 < n3 < ...`
angeben. Wie lang kann man diese Ketten höchstens machen?

Antworten zu den Wiederholungsfragen, 4. SU, Mo 18.10.00

1. Warum ist die folgende Methode keine *vernünftige rekursive Methode*?

```
 1 static int berechneA(int n) {
 2    if (n == 0) {
 3       return berechneA(n-1);
 4    } else {
 5       return berechneA(n+1);
 6    }
 7 }
```

Weil sie sich in jedem Fall rekursiv aufruft (egal ob ihr Parameter n gleich 0 ist oder nicht).

2. Warum ist die folgende Methode keine *vernünftige rekursive Methode*?

```
 8 static int berechneB(int n) {
 9    if (n == 0) {
10       return n - 1;
11    } else {
12       return n + 1;
13    }
14 }
```

Weil sie sich in keinem Fall rekursiv aufruft (egal ob ihr Parameter n gleich 0 ist oder nicht).

3. Warum ist die folgende Methode keine *vernünftig rekursive Methode*?

```
15 static berechneC(int n) {
16    if (n == 0) {
17       return 17;
18    } else {
19       return berechneC(n);
20    }
21 }
```

Weil sie sich im rekursiven Fall (wenn n ungleich 0 ist) mit dem gleichen Parameter n aufruft, mit dem sie selbst aufgerufen wurde (und nicht mit einem kleineren Parameter).

4. Eine Firma stellt *Roboter* her, die sich *selbst reproduzieren* können. Genauer: 3 solche Roboter brauchen 1 Woche, um einen weiteren Roboter ihresgleichen zu bauen. Die Firma beginnt die Produktion *mit 3 Robotern* und setzt alle ihre Roboter zur Produktion weiterer Roboter ein. Wie viele Roboter hat die Firma nach 1 Woche? Nach 2 Wochen? Nach 3 Wochen? Nach 4 Woche? etc.

Es folgt eine rekursive Funktion, die all diese Fragen beantworten kannt:

```
22 static long anzRobos(long w) {
23    // Verlaesst sich darauf, dass w gleich oder groesser 0 ist.
24    // Wie viele Roboter hat die Firma nach w Wochen?
25    // Diese Funktion liefert die Antwort.
26    // "DLW" steht fuer "der letzten Woche", "anz" fuer "Anzahl"
27
28    if (w == 0) return 3;
29    long anzAmAnfangDLW     = anzRobos(w - 1);
30    long anzProduziertInDLW = anzAmAnfangDLW / 3;
31    return anzAmAnfangDLW + anzProduziertInLW;
32 }
```

Lösungen zur Rückseite der Wiederholungsfragen, 4. SU, Mo 18.10.00,

Betrachten Sie die folgende (mehr oder weniger vernünftige :-) *rekursive Funktion*:

```
1    static String rek37(int n) {
2       if (n % 5 == 0) {
3          return "" + n;
4       } else {
5          return n + " " + rek37(n+1);
6       }
7    } // rek37
```

Aufgabe-01: Führen Sie die Funktion `rek37` mit verschiedenen aktuellen Parametern und mit Hilfe der folgenden *Wertetabelle* aus:

n	**rek37(n)**
...	
14	14 15
15	15
16	16 17 18 19 20
17	17 18 19 20
18	18 19 20
19	19 20
20	20
21	21 22 23 24 25
22	22 23 24 25
23	23 24 25
24	23 24
25	25
26	26 27 28 29 30
...	

Aufgabe-2: Betrachten Sie noch einmal die Funktion `rek37`:

Welche Parameter-Werte n erfordern 0 rekursive Aufrufe ..., -5, 0, 5, 10, 15, 20, 25, 30, ...
Welche Parameter-Werte n erfordern 1 rekursiven Aufruf? ..., -6, -1, 4, 9, 14, 19, 24, 29, ...
Welche Parameter-Werte n erfordern 2 rekursive Aufrufe? ..., -7, -2, 3, 8, 13, 18, 23, 28, ...
Welche Parameter-Werte n erfordern 3 rekursive Aufrufe? ..., -8, -3, 2, 7, 12, 17, 22, 27, ...
Welche Parameter-Werte n erfordern 4 rekursive Aufrufe? ..., -9, -4, 1, 6, 11, 16, 21, 26, ...

Aufgabe-3: Eine Halbordnung der `int`-Werte für `rek37`:

```
...
-5 < -6 < -7 < -8 < -9
 0 < -1 < -2 < -3 < -4
 5 <  4 <  3 <  2 <  1
10 <  9 <  8 <  7 <  6
15 < 14 < 13 < 12 < 11
...
```

Bei dieser Halbordnung sind die Zahlen `0` und `5` oder `0` und `4` oder `-5` und `-4` etc. *unvergleichbar*, die Zahlen ... `-5`, `0`, `5`, `10`, `15` ... sind *minimale Elemente* ("keine Zahl ist kleiner") und es gibt kein kleinstes Element (denn für das müßte gelten: "Es ist kleiner als alle anderen Zahlen").

4. SU, Mo 18.10.10

A. Wiederholung
B. Organisation

Rekursive Aufrufe nur mit kleineren Parametern erlaubt

Betrachten Sie das Arbeitsblatt auf der Rückseite der Wiederholungsfragen und lösen Sie (am besten zu zweit) die drei Aufgaben, die dort stehen.

Zusammenfassung der Lösung der Aufgabe-02: Für `rek37` sind die Parameter ..., -9, -4, 1, 6, 11, ... *am größten*, weil sie für jeden dieser Parameter 5 rekursive Aufrufe braucht.

Die Parameter ..., -8, -3, 2, 7, 12, ... sind *etwas kleiner*, weil sie nur 4 rekursive Aufrufe erfordern.
...
Dei Parameter ..., -6, -1, 4, 9, 14, ... sind *noch kleiner*, weil sie nur 1 rekursiven Aufruf erfordern.
Die Parameter ..., -5, 0, 5, 10, 15 , ... sind *am kleinsten*, weil sie 0 rekursive Aufrufe erfordern.

Für die rekursive Funktion `rek37` ist der Parameter 20 also kleiner als 19, weil 20 weniger rekursive Aufrufe erfordert (nämlich 0) als der Parameter 19 (der 1 rekursiven Aufruf erfordert).

Wer das gründlicher und mathematisch genauer verstehen will, sollte sich mit folgenden Begriffen vertraut machen: Halbordnung, minimale Elemente, strikte Halbordnung, fundierte Menge. Diese Begriffe werden aber hier im Fach MB2-PR2 nicht näher behandelt (nur in der Datei `Stichworte.pdf` findet man ein paar Definitionen und Beispiele dazu).

Zwei totale Ordnungen für Strings, die jeder Informatiker kennen sollte

Die lexikografische Ordnung

Nach dieser Ordnung ist ein String um so größer, je weiter hinten er in einem Lexikon stehen müsste.

Beispiele:
`"Andalusien"` ist kleiner als `"Britz"` (obwohl `"Andalusien"` länger ist als `"Britz"`!)
`"Kunstausstellung"` ist kleiner als `"Kunstdünger"` (weil `'a'` kleiner ist als `'d'`).

Die lexikalische Ordnung

Ein längerer String ist immer größer als ein kürzerer String und
für gleich lange Strings gilt die lexikografische Ordnung.

Beispiele:

`"Andalusien"` ist größer als `"Britz"` (weil `"Andalusien"` länger ist als `"Britz"`).
`"Britz"` ist kleiner als `"Bruck"` (weil beide gleich lang sind und `'i'` kleiner ist als `'u'`).

Wie viele Strings liegen bei der *lexikografischen* Ordnung zwischen `"auf"` und `"aus"`?
(*Unendlich* viele, darunter alle, die mit "auf" anfangen, z.B. `"aufa"`, `"aufb"`, ..., `"auffahren"`, ...)

Entsprechendes gilt für viele, aber nicht für alle Paare von Strings.

Wie viele Strings liegen bei der *lexikalischen* Ordnung zwischen `"auf"` und `"aus"`?
(*Endlich* viele, genauer 17 Stück: `"aug"`, `"auh"`, `"aui"`, ..., `"auq"`, `"aur"`).

Entsprechendes gilt für alle Paare von Strings.

Welcher String kommt lexikalisch nach `"ABC"`? (`"ABD"`)

Welcher String kommt lexikografisch nach `"ABC"`? (`"ABC\u0000"`)

Das Literal `'\u0000'` bezeichnet den kleinesten `char`-Wert (den Wert 0 vom Typ char).

Halbgeordnete Mengen, fundierte Mengen, total geordnete Mengen

Beispiele für *total geordnete* Mengen:

Die natürlichen Zahlen mit der Relation `kleiner-gleich`
Die rationalen Zahlen mit der Relation `kleiner-gleich`
Die Menge aller Strings mit der Relation `ist-kürzer-oder-gleich-lang`
Die Menge aller Strings mit der Relation `ist-lexikographisch-kleiner-oder-gleich`
Die Menge aller Strings mit der Relation `ist-lexikalisch-kleiner-oder-gleich`

"total" bedeutet hier:
Nimmt man zwei beliebige Elemente `a` und `b` der betreffenden Menge, dann gilt
entweder `a kleiner-gleich b`
oder `b kleiner-gleich a`
oder beides

Manche Mengen sind bezüglich bestimmter Relationen "ein bisschen geordnet", aber nicht total:

Beispiele für halbgeordnete Mengen:

Die Menge aller Strings mit der Relation `ist-ein-Teilstring-von`
Die natürlichen Zahlen mit der Relation `ist-ein-Teiler-von`
Die Menge der Java-Referenztypen mit der Relation `ist-Untertyp-von`

Bezüglich der Relation `ist-ein-Teilstring-von` sind die Strings `"BC"` und `"ABCDE"` *vergleichbar* und es gilt: `"BC" ist-ein-Teilstring-von "ABCDE"`.

Dagegen sind die Strings `"BC"` und `"DEF"` *unvergleichbar,* weil
weder `"BC" ist-ein-Teilstring-von "DEF"`
noch `"DEF" ist-ein-Teilstring-von "BC"`
gilt.

Ganz entsprechend gilt für Java-Referenz-Typen: `Object` und `String` sind *vergleichbar,* weil `String` ein Untertyp von `Object` ist. Dagegen sind `String` und `Integer` *unvergleichbar,* weil keiner dieser beiden Typen ein Untertyp des anderen ist.

Halbgeordnete Mengen: Eine Menge mit einer zweistelligen Relation `kg` ("kleiner-oder-gleich") ist *halbgeordnet,* wenn für alle Elemente a, b, c gilt:
`kg` ist transitiv: Wenn `a kg b` und `b kg c` dann auch `a kg c`
`kg` ist reflexiv: `a kg a`
`kg` ist antisymmetrisch: (`a kg b` und `b kg a`) gilt nur wenn `a gleich b` ist.

Minimale Elemente: Ein Element m (einer halbgeordneten Menge M) heißt minimal, wenn es in M kein Element n gibt, welches noch kleiner als m ist.

Achtung: Diese Definition sagt *nicht,* dass ein minimales Element kleiner als alle anderen sein muss. Ein minimales Element kan mit anderen Elementen unvergleichbar sein. Nur wenn es vergleichbar ist, muss es kleiner sein.

Fundierte Menge: Eine halbgeordnete Menge heißt fundiert, wenn jede nichtlleere Teilmenge (mindestens) ein minimales Element enthält. Oder: Wenn es keine unendliche Kette von Elementen gibt, von denen jedes kleiner als das vorhergehende ist.

Beispiel: Die *natürlichen Zahlen* (0, 1,. 2. ...) mit der Relation `ist-kleiner-als` ist fundiert (denn zu jeder natürlichen Zahl gibt es nur endlich viele andere natürliche Zahlen, die kleiner sind).

Gegenbeispiel: Die Menge der *ganzen Zahlen* mit der Relation `ist-kleiner-als` (kurz: `<`) ist nicht fundiert, weil es z.B. die unendliche Kette `... -32 < -29 < -26 < -23 < -20 < -17` (diese Kette kann man nach links beliebig weit verlängern).

Zur Entspannung: Die Sütterlin-Schrift und der Mond

In Deutschlang gab es oft langwierige Diskussionen und Streit darüber, welche Schrift benutzt und in den Schulen gelehrt werden sollte. 1911 entwarf Ludwig Sütterlin im Auftrag des preußischen Kultusministeriums zwei Schriften. Eine davon wurde 1915 an preußischen Schulen und ab 1930 in ganz Deutschland eingeführt. 1941 wurde diese Sütterlin-Schrift von Hitler verboten. Ab 1954 wurde sie an einigen Schulen wieder gelehrt, wurde aber von der lateinischen Schreibschrift verdrängt.

Im Gedicht "Der Mond" von Christian Morgenstern (1871-1914, München-Meran) spielen die Großbuchstaben A und Z in Sütterlin ( und ) eine wichtige Rolle:

Als Gott den lieben Mond erschuf / Gab er ihm folgenden Beruf:
Beim Zu- sowohl wie beim Abnehmen / Sich deutschen Lesern zu bequemen
ein  formierend und ein  - / daß keiner groß zu denken hätt.
Befolgend dies ward der Trabant / ein völlig deutscher Gegenstand.

Sammlungen (engl. collections)

Eine kleine Wiederholung aus MB1-PR1:

Angenommen, wir brauchen in einem Programm 500 `double`-Variablen. Welche Vorteile hat es, wenn wir diese Variablen in Form einer *Reihung* vereinbaren, etwa so:

```
double dr = new double[500];
```

statt die Variablen einzeln zu vereinbaren und mit Namen zu versehen, etwa so;

```
double d0, d1, d2, d3, ..., d499;
```

(Die Vereinbarung der Reihung ist viel einfacher und leichter veränderbar. Jede Bearbeitung der 500 einzelnen Variablen erfordert 500 Befehle. Die Reihung kann man mit *Schleifen* bearbeiten).

Nachteil einer Reihung im Vergleich zu einzelnen Variablen? (Man kann den Reihungskomponenten keine *individuellen Namen* geben).

Noch ein Nachteil von Reihungen? Die Länge einer Reihung muss bei ihrer Erzeugung festgelegt werden und kann danach nicht mehr geändert werden ("Reihungen sind aus Beton").

Reihungsobjekte sind "ein bisschen primitiver und spartanischer eingerichtet" als andere Objekte. Ein Reihungsobjekt ist (wie alle anderen Objekte) ein Modul, enthält aber (außer den von der Klasse `Object` geerbten Elementen) nur ein einziges Element: Das Attribut `length` (vom Typ `int`). Die (von `Object` geerbte) Methode `toString` leistet nicht das, was man häufig gern hätte. Viele nützliche Methoden zur Bearbeitung von Reihungen sind nicht in den Reihungsobjekten enthalten, sondern als Klassenmethoden in den Klassen `Array` und `Arrays` vereinbart.

Sammlungen haben Ähnlichkeit mit Reihungen, bieten dem Programmierer aber *mehr Komfort* oder sind *für bestimmte Anwendungen optimiert*. Sammlungsobjekte enthalten (zumindest die wichtigsten) Methoden, die man zu ihrer Bearbeitung braucht.

Zur Erinnerung:

Def.: Eine *Sammlung* ist ein Objekt, in dem man andere Objekte sammeln (d.h. hineintun, darin suchen und wieder entfernen) kann.

Def: Eine Sammlung ist ein `Collection`-Objekt (d.h. ein Objekt einer Klasse, die die Schnittstelle `Collection` implementiert).

Zur Zeit (in Java 6) gibt es in der Java-Standardbibliothek 31 Sammlungsklassen. Warum so viele?

Für einige Anwendungen ist das *Einfügen* von Komponenten besonders wichtig (und sollte besonders schnell gehen), das Suchen und Entfernen von Komponenten ist dagegen nicht so wichtig.

Für andere Anwendungen ist das *Suchen* besonders wichtig. Das Einfügen darf ruhig ein bisschen mehr Zeit kosten, wenn dafür das Suchen schneller geht.

Einige Anwender wollen, dass eine Sammlung *beliebig vergrössert* werden kann. Andere wollen, dass Sammlungen nur bis zu einer bestimmten Größe wachsen können und sich dann gegen weitere Vergrößerungen sträuben (sie z.B. die Klasse `ArrayBlockingQueue`).

Einige Anwender wollen mit *Indizes* auf die Komponenten einer Sammlung zugreifen (ähnlich wie bei Reihungen). Andere wollen nur "oben auf der Sammlung" Komponenten hinzufügen oder entfernen (*Stapel*, stack). Wieder andere wollen Komponenten "vorn" einfügen und "hinten" wieder entfernen (*Schlange*, queue).

Einige für Sammlungen wünschenswerte Eigenschaften *konkurrieren miteinander* oder schließen sich sogar aus: Wenn man möglichst *wenig Speicherplatz* verbrauchen will, muss man meistens bei der *Geschwindigkeit* (des Einfügens und Suchens etc.) Kompromisse machen. Wenn das *Suchen sehr schnell* gehen soll, muss man beim *Einfügen* meistens *mehr Zeit investieren* etc.

Analogie: Es ist kaum möglich, ein billiges, besonders umweltfreundliches, schnelles und bequemes Auto zu bauen. Autobauer suchen nach "guten Kompromißen" zwischen diesen Eigenschaften.

Für viele Anwendungen ist es sehr wichtig, ihre (umfangreichen) Daten in Sammlungen abzulegen, die *genau die richtige Kombination von Eigenschaften* haben. Deshalb gibt es so viele Sammlungsklassen. Und ab und zu muss ein Programmierer für ein spezielle Anwendung noch eine *neue Sammlungsklasse* mit ganz speziellen Eigenschaften entwickeln (meist als Unterklasse einer Standardklasse).

Alle Java-Standard-Sammlungsklassen sind generisch

Man kann den Typ der zu sammelnden Objekte also einschränken. In einer Sammlung des Typs `ArrayList<String>` kann man nur `String`-Objekte sammeln, in einer Sammlung vom Typ `ArrayList<JButton>` nur `JButton`-Objekte. Solche Einschränkungen sind ein Vorteil, bestimmte Flüchtigkeitsfehler werden vom Ausführer schon bei der Übergabe des Programms erkannt, nicht erst später bei der Ausführung (oder gar nicht).

Rohe Typen wie `ArrayList` (ohne `<String>` oder `<Integer>` dahinter) sollte man nur verwenden, wenn es *unbedingt* sein muss (beim Erweitern von "alten Java-Programmen", die mit einer Java-Version vor Version 5 geschrieben wurden).

In einer Sammlung eines rohen Typs wie z.B. `ArrayList` kann man alle möglichen Objekte sammeln!

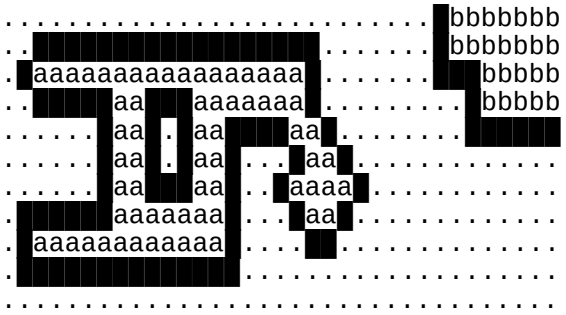
Achtung: Der *rohe Typ* `ArrayList` und der *parametrisierte Typ* `ArrayList<Object>` sind *nicht* gleich!

Der *rohe Typ* `ArrayList` ist ein *Obertyp* der parametrisierten Typen `ArrayList<String>`, `ArrayList<Integer>`, ... etc.

Der *parametriesierte* Typ `ArrayList<Object>` ist *weder Ober- noch Untertyp* der anderen parametrisierten Typen `ArrayList<String>`, `ArrayList<Integer>`, ... etc.

Wiederholungsfrage (heute nur eine), 5. SU, Mo 25.10.10

Angenommen, auf einem PoxelPanel `pop` wurden mit schwarzen Poxeln Figuren gezeichnet, etwa so:



Hier sieht man als Beispiel ein 35x10-PoxelPanel, auf das (mit schwarzen Poxeln) drei Figuren gezeichnet wurden. Die linke, größere Figur enthält eine kleinere, rechteckige Figur. Das Innere der größeren Figur ist mit `a`'s gefüllt.

Das Innere der kleinen rechteckigen Figur gehört *nicht* zum Inneren der umgebenden Figur und ist deshalb nicht mit `a`'s gefüllt.

Die Figur in der rechten oberen Ecke des Panels wird teilweise durch die Ränder des Panels begrenzt. Ihr Inneres ist mit `b`'s gefüllt.

Die Punkte `.` stellen "neutrale" Poxel dar, die nicht zu einer schwarzen Linie und nicht zum Inneren der beiden "gefüllten" Beispielfiguren gehören.

Angenommen, das Innere der Figuren sei *noch nicht* mit `a`'s bzw. `b`'s gefüllt (sondern mit "neutralen" Poxeln). Wenn wir jetzt die Koordinaten von irgendeinem inneren Poxel einer Figur bekommen, wie können wir dann alle inneren Punkte dieser Figur mit einer bestimmten Farbe (und einem bestimmten Zeichen) füllen?

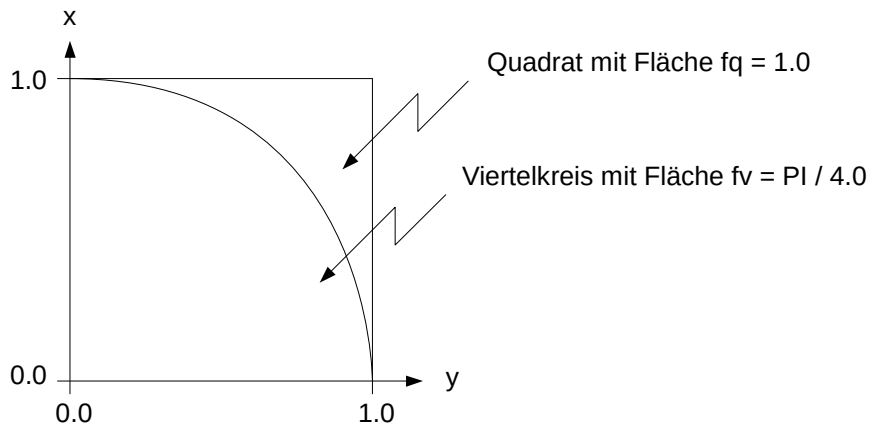
Schreiben Sie eine Methode, die der folgenden Spezifikation entspricht:

```
1    static
2    void fuelle(PoxelPanel pop, int x, int y, Color color1, char char1) {
3       // Schwarze Poxel gelten hier als Begrenzer (einer Flaeche auf pop).
4       // Wenn das aktuellen Poxel (x, y) schwarz ist, passiert nichts.
5       // Sonst werden das Poxel (x, y) und alle mit ihm verbundenen,
6       // nicht-schwarzen Poxel mit der Farbe color1 und dem Zeichen char1
7       // gefuellt (d.h. beschrieben).
8       ...
9    } // fuelle
```

Rückseite der Wiederholungsfrage (heute nur eine), 5. SU, Mo 25.10.10

Zur Entspannung: Die Zahl **π** mit Hilfe von Zufallszahlen berechnen

Für viele Probleme ist es sehr aufwendig, eine exakte Lösung zu berechnen. Für einige dieser Probleme kann man unter Verwendung von *Zufallszahlen* mit viel weniger Aufwand eine *Näherungslösung* berechnen. Beispiel: Die Berechnung der Kreiszahl π mit Hilfe von zufälligen Regentropfen:



Ein Quadrat mit der Kantenlänge 1.0 hat eine Fläche `fq` von ... (wo ist bloß mein Taschenrechner) von `1.0`. Ein Kreis mit dem Radius `1.0` hat eine Fläche von π, ein Viertelkreis davon hat somit eine Fläche `fv` gleich `π/4.0` (siehe obige Skizze). Angenommen, in das Quadrat fallen `tiq` ("Tropfen ins Quadrat") viele Regentropfen und `tiv` viele davon landen im Viertelkrei. Dann gilt:

```
     fv  / fq  = tiv / tiq
Pi / 4.0 / 1.0 = tiv / tiq
Pi             = tiv / tiq * 4.0
```

Die folgenden Befehle berechnen einen Näherungswert für die Zahl `PI`:

```
10    Random rand = new Random(123);   // Keim (seed) 123
11    int    tiq = 10*1000*1000;       // Tropfen die ins Quadrat  fallen
12    double tiv = 0;                  // Tropfen die im Viertelkreis landen
13
14    for (int i=1; i<=tiq; i++) {     // tiq viele Tropfen fallen lassen
15       double x = rand.nextDouble(); // Zufallszahl aus [0.0 .. 1.0)
16       double y = rand.nextDouble(); // Zufallszahl aus [0.0 .. 1.0)
17       if (Math.sqrt(x*x + y*y) <= 1.0) tiv++; // Treffer im Viertelkreis
18    }
19
20    // Aus tiv und tiq einen Naeherungswert Pi fuer pi berechnen und ausgeben:
21    pln("Hier:Pi ist gleich " + (tiv / tiq * 4.0));
22    // Zum Vergleich einen *guten* Naeherungswert fuer PI ausgeben:
23    pln("Math.PI ist gleich " + Math.PI);
```

Ausgabe dieser Befehlsfolge:

```
24    Hier:Pi ist gleich 3.141918
25    Math.PI ist gleich 3.141592653589793
```

Arbeitsblatt für 5. SU, Mo 25.10.10, Vorderseite

Profile verschiedener Sammlungsstrukturen

Wieviele Schritte kosten die Operationen *Einfügen, Suchen in einem positiven Fall, Suchen in einem negativen Fall* und *Entfernen* bei Sammlungen verschiedener Strukturen in einem schlimmsten Fall und im Durchschnitt?

Die Variable n bezeichnet immer die Anzahl der Objekte, die sich bereits in der Sammlung befinden.

Tabelle mit Schrittzahlen für 6 verschiedene Sammlungstrukturen (6 Profile):

Struktur der Sammlung	Operation	Anz. Schritte in einem schlimmsten Fall	Anz. Schritte im Durchschnitt
Reihung (unsortiert)	Einfügen		
	Suchen pos.		
	Suchen neg.		
	Entfernen		
Reihung (sortiert)	Einfügen		
	Suchen pos.		
	Suchen neg.		
	Entfernen		
verkettete Liste (unsortiert)	Einfügen		
	Suchen pos.		
	Suchen neg.		
	Entfernen		
verkettete Liste (sortiert)	Einfügen		
	Suchen pos.		
	Suchen neg.		
	Entfernen		
binäre Bäume (müssen immer sortiert sein)	Einfügen		
	Suchen pos.		
	Suchen neg.		
	Entfernen		
Hash-Tabellen (müssen immer unsortiert sein)	Einfügen		
	Suchen pos.		
	Suchen neg.		
	Entfernen		

Arbeitsblatt für 5. SU, Mo 25.10.10, Rückseite

Typische Zusammenhänge zwischen
einer Problemgröße n (z.B. der Länge von Reihungen, die sortiert werden sollen) und
der Schrittzahl s, die ein (Sortier-) Algorithmus zum Lösen von Problemen der Größe n benötigt:

Wenn man n so vergrößert	dann vergrößert sich s so	Zeit-komplexität	in Worten
mal 2 mal m	mal 2 mal m	O(n)	Oh von enn
mal 2 mal m	mal 2^2 mal m^2	O(n^2)	Oh von enn quadrat
mal 2 mal m	mal 2^3 mal m^3	O(n^3)	Oh von enn hoch 3
mal 2 mal m	mal 2^k mal m^k	O(n^k)	Oh von enn hoch k
mal 2^1 mal 2^m	plus 1 plus m	O(log n)	Oh von log enn
plus 1 plus m	mal 2^1 mal 2^m	O(2^n)	Oh von zwei hoch enn
mal 2 mal m	mal 1 mal 1	O(1)	Oh von eins

Angenommen, wir wollen die (Zeit-) Komplexität eines Algorithmus ermitteln. Welche der folgenden Befehlsfolgen können wir dabei als *einen Schritt* bewerten (JA) und welche nicht (NEIN)?

Nr	Befehlsfolge	Schritt?
1	Zwei int-Werte vergleichen?	
2	500 int-Werte vergleichen?	
3	Zwei BigInteger-Objekte vergleichen?	
4	Zwei BigInteger-Objekte vergleichen, von denen jedes weniger als 500 Ziffern enthält?	
5	Zwei BigInteger-Objekte addieren?	
6	800 BigInteger-Objekte addieren, von denen jedes weniger als 500 Ziffern enthält?	
7	Zwei String-Objekte vergleichen?	
8	Zwei String-Objekte vergleichen, von denen jedes kürzer als 500 Zeichen ist?	
9	In einer beliebigen Reihung von int-Variablen den größten Wert finden?	
10	In einer Reihung der Länge 500 den größten Wert finden?	

Antwort zu der Wiederholungsfrage, 5. SU, Mo 25.10.10

```
1    // ---------------------------------------------------------------------
2    static
3    void fuelle(PoxelPanel pop, int x, int y, Color color1, char char1) {
4       // Schwarze Poxel gelten hier als Begrenzer (einer Flaeche auf pop).
5       // Wenn das aktuellen Poxel (x, y) schwarz ist, passiert nichts.
6       // Sonst werden das Poxel (x, y) und alle mit ihm verbundenen,
7       // nicht-schwarzen Poxel mit der Farbe color1 und dem Zeichen char1
8       // gefuellt (d.h. beschrieben).
9
10      // Einfache Faelle: Wenn x oder y ungeeignet (zu gross oder zu klein)
11      // ist, nichts machen:
12      if (x < 0 || pop.getAnzPoxX() <= x) return;
13      if (y < 0 || pop.getAnzPoxY() <= y) return;
14
15      // Noch ein einfacher Fall: Wenn das aktuelle Poxel (x, y) schon
16      // gefuellt wurde oder ein Begrenzer ist, nichts machen:
17      Color color2 = pop.getColor(x, y);
18      if (color2!=null && (color2==color1 || color2==Color.BLACK)) return;
19
20      // Sonst: Das aktuelle Poxel (x, y) fuellen ...
21      pop.drawRect(x, y, color1, char1);
22
23      // ... und rekursiv die vier umgebenden Poxel fuellen:
24      fuelle(pop, x  , y+1, color1, char1); // unter        (x, y)
25      fuelle(pop, x  , y-1, color1, char1); // ueber        (x, y)
26      fuelle(pop, x+1, y  , color1, char1); // rechts neben (x, y)
27      fuelle(pop, x-1, y  , color1, char1); // links  neben (x, y)
28   } // fuelle
29   // ---------------------------------------------------------------------
```

5. SU, Mo 25.10.10

A. Wiederholung
B. Organisation: Klausur: Mo 07.02.2011, 8 Uhr, B554

18.2 Die Sammlungsschnittstelle Collection (ohne s) (S. 441)

Diese Schnittstelle ist besonders wichtig und es lohnt sich, sie sehr genau zu lesen. Dabei kommt man auf überraschend viele merk-würdige Fragen und Antworten.

Zur Erinnerung: Seien `K1` und `K2` Klassen und `K2` eine Unterklasse von `K1`: `K1 <--- K2`

Was folgt daraus? Was gilt dann für den Fall, dass man dringend ein `K1`-Objekt braucht
(z.B. als Parameter einer Methode)?

(Man kann anstelle des `K1`-Objekts auch ein `K2`-Objekt angeben)

Also nochmal mit anderen Worten: Jedes K2-Objekt ist auch ein K1-Objekt!

Der Parametertyp der Methode `add` (Zeile 1)? (K, wie "Komponententyp").

```
1 ArrayList<Object>  sam1 = new ArrayList<Object>();
2 ArrayList<String>  sam2 = new ArrayList<String();
3 ArrayList<Integer> sam3 = new ArryList<Integer>();
```

Von welchem Typ ist der Parameter der Methode `sam1.add`? (`Object`)
Von welchem Typ ist der Parameter der Methode `sam2.add`? (`String`)
Von welchem Typ ist der Parameter der Methode `sam3.add`? (`Integer`)

Von welchem Typ ist der Parameter der Methode `sam1.remove`? (`Object`)
Von welchem Typ ist der Parameter der Methode `sam2.remove`? (`Object`)
Von welchem Typ ist der Parameter der Methode `sam3.remove`? (`Object`)

Was bedeutet das? Es ist erlaubt, alle Komponenten einer `String`-Sammlung aus einer `Integer`-Sammlung entfernen zu lassen. Dadurch wird sicherlich "nichts entfernt", weil eine `Integer`-Sammlung sicherlich keine `String`-Objekte enthält, aber befehlen darf man so was. Dadurch werden einige Programme etwas einfacher.

Für `contains` gilt Entsprechendes wie für `remove` (Man darf z.B. fragen: Enthält diese `String`-Sammlung folgende `Integer`-Objekte? Die Antwort wird sicherlich "Nein" (`false`) sein, aber die Frage ist erlaubt).

Vergleich: Wenn Mylord seinen Butler anweist: "James, Entfernen Sie alle grünen Polstersessel aus diesem Raum", dann wird ein guter Butler nicht antworten: "Aber hier gibt es doch gar keine grünen Polstersessel", sonder: "Sehr wohl, Mylord.".

Der Parametertyp der Methode `addAll`: `Collection<? extends K>`

```
4 sam1.addAll(sam2); // Erlaubt,  weil String eine Erweiterung  von Object ist
5 sam2.addAll(sam3); // Verboten, weil Integer keine Erweiterung von String ist
```

Sammlungen *mit was für einem Komponententyp*
darf man der Methode `sam3.removeAll` als Parameter übergeben?
(Sammlungen mit beliebigen Komponententypen)

Ebenso für `containsAll` und `retainsAll`.

Womit darf man die Sammlung `sam3` vergleichen,
wenn man dazu die Methode `sam3.equals` verwendet?
(Mit beliebgen Objekten, nicht nur mit Sammlungen).

Was liefert der Funktionsaufruf `sam3.toArray()`?
(Eine Reihung von `Object`-Variablen, formal: Ein Ergebnis vom Typ `Object[]`)
Was für ein Ergebnis hätten wir uns eigentlich gewünscht?
(Eine Reihung von `Integer`-Variablen, `Integer[]`).

Die zweite `toArray`-Methode hat auch einen Pferdefusss:

```
6 Integer[] ir = new Integer[]{};
7 ... sam3.toArray(ir)...
```

Dieser Funktionsaufruf liefert ein Ergebnis vom richtigen Typ (`Integer[]`),
aber wir mussten einen Parameter (von diesem Typ) angeben.

Auch erlaubt:

```
8 ... sam1.toArray(ir)...
```

Geht gut (und liefert ein `Integer[]`-Ergebnis), wenn `sam1` nur `Integer`-Objekte enthält.
Sonst wird eine `ArrayStoreException` geworfen.

Die opt-Methoden in der Schnittstelle Collection

Alle Methoden der Schnittstelle `Collection`, die eine Sammlung *möglicherweise verändern* (und nicht nur "Informationen über die Sammlung liefern ohne sie zu verändern") sind mit opt (wie optional) gekennzeichnet. Diese Methoden darf man so implementieren, dass sie "nicht richtig" funktionieren, sondern nur eine Ausnahme des Typs `UnsupportedOperationException` werfen.

Keine schöne Lösung, aber dahinter steht ein schwieriges Problem:
Außer *veränderbaren Sammlungen* braucht man häufig auch *unveränderbare Sammlungen*
(z.B. als Parameter für eine Methode, der man "nicht voll vertraut" und der man nicht erlauben will, eine "kostbare Sammlung" zu verändern, z.B. zu zerstören). Wenn man von jedem Sammlungstyp eine veränderbare und eine unveränderbare Variante einführen würde, gäbe es sehr viele Sammlungstypen (sogar deutlich mehr als doppelt so viele wie ohne diese Unterscheidung, weil man dann ja auch
unveränderbare Sammlungen von unveränderbaren Sammlungen und
unveränderbare Sammlungen von veränderbaren Sammlungen und
veränderbare Sammlungen von unveränderbaren Sammlungen und
veränderbare Sammlungen von veränderbaren Sammlungen
unterscheiden müßte.

Eine unveränderbare Sammlung ist jetzt also eine Sammlung, bei der die "verändernden Methoden" eine Ausnahme werfen.

Alle optionalen Methoden (ausser clear) haben den *Rückgabetyp* `boolean`.
Wann liefern sie wohl `true` und wann `false`?
(`true` wenn sie was verändert haben, `false` sonst).

Zur Entspannung: Mit Hilfe von Zufallszahlen pi berechnen
(siehe Rückseite der Wiederholungsfrage)

Algorithmen bewerten und vergleichen, Komplexität eines Algorithmus

Angenommen, wir wollen bestimmte (große) Reihungen sortieren. Dafür gibt es mehrere Algorithmen ("abstrakte Programme, die man unabhängig von der konkreten Sprache oder Notation, in der sie geschrieben sind, verstehen sollte"). Wie können wir diese Algorithmen vergleichen, um den besten (oder zumindest einen besonders guten) herauszufinden?

(Wir können die Algorithmen z.B. in Java programmieren und z.B. im SWE-Labor auf verschiedene große Reihungen anwenden und die zum Sortieren benötigte Zeit messen. Vorteile dieser Vorgehensweise? Nachteile?

Problem: *Abstrakte Algorithmen* haben keinen konkreten Zeit- und Speicherbedarf

(weil dieser Bedarf immer ganz wesentlich auch von dem konkreten Ausführer abhängt, den man verwendet, und nicht nur vom Algorithmus).

Wie kann man Algorithmen trotzdem *bewerten* und miteinander *vergleichen*?

Lösung: Man ermittelt keinen konkreten Zeitbedarf sondern (nur) den
abstrakten *Zusammenhang* zwischen der *Problemgröße* n und
der Anzahl der Rechenschritte s,
die der Algorithmus zur Lösung von Problemen der Größe n benötigt.

Was ist mit "ein Problem der Größe n" gemeint? Ein paar Beispiele:

- Zwei Strings vergleichen? (n ist die Länge der Strings)
- Reihungen sortieren? (n ist die Länge der Reihung)
- Zwei (gleich lange) Zahlen addieren? (n ist die Länge der Zahlen, gemessen in Ziffern oder Bits)

Was ist "ein Schritt" ?

Def.: Ein (Ausführungs-) *Schritt* ist eine *beliebig lange Folge von Befehlen*, von der man annehmen kann, dass ein Ausführer sie immer in (ungefähr) der *gleichen Zeit* ausführen kann. Insbesondere darf die Ausführungszeit eines Schritts *nicht* von der Problemgröße n abhängen.

Die Tabelle auf der Rückseite des Arbeitsblattes (Schritt, JA oder NEIN?) ausfüllen

Wiederholungsfragen, 6. SU, Mo 01.11.10

Betrachten Sie folgende Vereinbarungen:

```
 1    class K1               { ... }
 2    class K11  extends K1  { ... }
 3    class K12  extends K1  { ... }
 4    class K121 extends K12 { ... }
 5    class K122 extends K12 { ... }
 6    ...
 7    class Otto {
 8       ArrayList<K1>  Alk1  = new ArrayList<K1>;  // "Al" wie "ArrayList"
 9       ArrayList<K11> Alk11 = new ArrayList<K11>; // (nicht wie "Alkohol"! :-)
10       ArrayList<K12> Alk12 = new ArrayList<K12>;
11       ...
12    }
```

1. Jede der folgenden Methoden hat genau *einen* Parameter.
Geben Sie an, zu welchem *Typ* dieser Parameter gehört:

Methode	Parameter-Typ	Methode	Parameter-Typ
`Alk1.add`		`Alk1.addAll`	
`Alk1.remove`		`Alk1.removeAll`	
`Alk11.add`		`Alk11.addAll`	
`Alk11.remove`		`Alk11.removeAll`	
`Alk12.add`		`Alk12.addAll`	
`Alk12.remove`		`Alk12.removeAll`	

2. In welchen der drei Sammlungen `Alk1`, `Alk11` und `Alk12` darf man
Objekte des Typs `K121` sammeln?

3. Objekte welcher (der in den Zeilen 1 bis 5 vereinbarten) Typen
darf man in der Sammlung `Alk12` sammeln?

4. Objekte welcher (der in den Zeilen 1 bis 5 vereinbarten) Typen
darf man in der Sammlung `Alk11` sammeln?

Nehmen Sie an, dass (mit der Methode `Alk1.add`)
ein paar `K1`-Objekte in die Sammlung `Alk1` eingefügt wurden.

5. Jetzt wollen Sie eine *Reihung* erzeugen lassen, die alle Objekte enthält, die sich gerade
in der Sammlung `Alk1` befinden. Ergänzen Sie die folgende Variablenvereinbarung entsprechend:

```
13    Object[] ronald =
```

6. Jetzt wollen Sie etwas ähnliches machen wie bei der vorigen Aufgabe, das Ergebnis soll aber (*keine* Reihung von `Object`-Variablen, sondern) eine Reihung von `K1`-Variablen sein. Schreiben Sie dazu einen geeigneten Befehl in die Zeile 14 und eränzen Sie die Variablenvereinbarung in Zeile 15:

```
14
15    K1[] roxana =
```

Rückseite der Wiederholungsfragen, 6. SU, Mo 01.11.10

Eine nützliche Anwendung der Logarithmus-Funktion:

Angenommen, wir wollen eine natürliche Zahl n *als b-er-Zahl* (d.h. als Zahl im Zahlensystem mit der Basis b) darstellen (z.B. als 2-er-Zahl oder als 10-er-Zahl oder als 16-er-Zahl oder als ...).

Wenn man den Logarithmus zur Basis b von n berechnet (und zur nächsten echt größeren Ganzzahl aufrundet) weiß man, wie viele Ziffern man zur Darstellung von n benötigt.

Die folgende Tabelle soll diesen Tatbestand veranschaulichen:

```
 1 ------------------------------
 2        Zahl   Logarithmus Ziff
 3 log10(    1): 0,00000000, 1,00
 4 log10(    2): 0,30103000, 1,00
 5 log10(    9): 0,95424251, 1,00
 6 log10(   10): 1,00000000, 2,00
 7 log10(   11): 1,04139269, 2,00
 8 log10(   99): 1,99563519, 2,00
 9 log10(  100): 2,00000000, 3,00
10 log10(  101): 2,00432137, 3,00
11 log10(  999): 2,99956549, 3,00
12 log10( 1000): 3,00000000, 4,00
13 log10( 1001): 3,00043408, 4,00
14 log10( 9999): 3,99995657, 4,00
15 log10(10000): 4,00000000, 5,00
16 ------------------------------
17        Zahl   Logarithmus Ziff
18 log02(    1): 0,00000000, 1,00
19 log02(    2): 1,00000000, 2,00
20 log02(    3): 1,58496250, 2,00
21 log02(    4): 2,00000000, 3,00
22 log02(    5): 2,32192809, 3,00
23 log02(    7): 2,80735492, 3,00
24 log02(    8): 3,00000000, 4,00
25 log02(    9): 3,16992500, 4,00
26 log02(   15): 3,90689060, 4,00
27 log02(   16): 4,00000000, 5,00
28 log02(   17): 4,08746284, 5,00
29 log02(   31): 4,95419631, 5,00
30 log02(   32): 5,00000000, 6,00
31 ------------------------------
32        Zahl   Logarithmus Ziff
33 log16(    1): 0,00000000, 1,00
34 log16(    2): 0,25000000, 1,00
35 log16(   15): 0,97672265, 1,00
36 log16(   16): 1,00000000, 2,00
37 log16(   17): 1,02186571, 2,00
38 log16(  255): 1,99858836, 2,00
39 log16(  256): 2,00000000, 3,00
40 log16(  257): 2,00140614, 3,00
41 log16( 4095): 2,99991193, 3,00
42 log16( 4096): 3,00000000, 4,00
43 log16( 4097): 3,00008804, 4,00
44 log16(65535): 3,99999450, 4,00
45 log16(65536): 4,00000000, 5,00
46 --------------------------------
```

Dieser Tabelle kann man unter anderem entnehmen:

Im 2-er-System ist 7 (gleich 111_2) die größte Zahl, die man noch mit 3 Ziffern darstellen kann, für die Zahl 8 (gleich 1000_2) braucht man schon 4 Ziffern.

Im 16-er-System ist 15 (gleich F) die größte Zahl, die man noch mit 1 Ziffer darstellen kann, für die Zahl 16 (gleich 10_{16}) braucht man schon 2 Ziffern.

Antworten zu den Wiederholungsfragen, 6. SU, Mo 01.11.10

Betrachten Sie folgende Vereinbarungen:

```
 1    class K1               { ... }
 2    class K11  extends K1  { ... }
 3    class K12  extends K1  { ... }
 4    class K121 extends K12 { ... }
 5    class K122 extends K12 { ... }
 6    ...
 7    class Otto {
 8       ArrayList<K1>  Alk1  = new ArrayList<K1>;  // "Al" wie "ArrayList"
 9       ArrayList<K11> Alk11 = new ArrayList<K11>;
10       ArrayList<K12> Alk12 = new ArrayList<K12>;
11       ...
12    }
```

1. Jede der folgenden Methoden hat genau einen Parameter.
Geben Sie an, zu welchem *Typ* dieser Parameter gehört:

Methode	Parameter-Typ	Methode	Parameter-Typ
`Alk1.add`	**`K1`**	`Alk1.addAll`	**`Collection<? extends K1>`**
`Alk1.remove`	**`Object`**	`Alk1.removeAll`	**`Collection<?>`**
`Alk11.add`	**`K11`**	`Alk11.addAll`	**`Collection<? extends K11>`**
`Alk11.remove`	**`Object`**	`Alk11.removeAll`	**`Collection<?>`**
`Alk12.add`	**`K12`**	`Alk12.addAll`	**`Collection<? extends K12>`**
`Alk12.remove`	**`Object`**	`Alk12.removeAll`	**`Collection<?>`**

2. In welchen der drei Sammlungen `Alk1`, `Alk11` und `Alk12` darf man
Objekte des Typs `K121` sammeln?

In die Sammlungen Alk1 und Alk12.

3. Objekte welcher (der in den Zeilen 1 bis 5 vereinbarten) Typen
darf man in der Sammlung `Alk12` sammeln?

Objekte der Typen K12, K121 und K122.

4. Objekte welcher (der in den Zeilen 1 bis 5 vereinbarten) Typen
darf man in der Sammlung `Alk11` sammeln?

Nur Objekte des Typs K11.

Nehmen Sie an, dass (mit der Methode `Alk1.add`)
ein paar `K1`-Objekte in die Sammlung `Alk1` eingefügt wurden.

5. Jetzt wollen Sie eine *Reihung* erzeugen lassen, die alle Objekte enthält, die sich gerade
in der Sammlung `Alk1` befinden. Ergänzen Sie die folgende Variablenvereinbarung entsprechend:

```
13    Object[] ronald = Alk1.toArray();
```

6. Jetzt wollen Sie etwas ähnliches machen wie bei der vorigen Aufgabe, das Ergebnis soll aber (*keine* Reihung von `Object`-Variablen, sondern) eine Reihung von `K1`-Variablen sein. Schreiben Sie dazu einen geeigneten Befehl in die Zeile 14 und eränzen Sie die Variablenvereinbarung in Zeile 15:

```
14    K1[] rk1    = new K1[Alk1.size()];
15    K1[] roxana = Alk1.toArray(rk1);
```

6. SU, Mo 01.11.10

A. Wiederholung
B. Organisation: Klausur: Mo 07.02.2011, 8 Uhr, Raum B554.

Im vorigen SU haben wir schon mit dem folgenden Kapitel angefangen:

Algorithmen bewerten und vergleichen, Komplexität eines Algorithmus

Angenommen, wir wollen bestimmte (große) Reihungen sortieren. Dafür gibt es mehrere Algorithmen ("abstrakte Programme, die man unabhängig von der konkreten Sprache oder Notation, in der sie geschrieben sind, verstehen sollte"). Wie können wir diese Algorithmen vergleichen, um den besten (oder zumindest einen besonders guten) herauszufinden?

(Wir können die Algorithmen z.B. in Java programmieren und z.B. im SWE-Labor auf verschiedene große Reihungen anwenden und die zum Sortieren benötigte Zeit messen. Vorteile dieser Vorgehensweise? Nachteile?)

Problem: *Abstrakte Algorithmen* haben keinen konkreten Zeit- und Speicherbedarf
(weil dieser Bedarf immer ganz wesentlich auch von der Programmiersprache und dem konkreten Ausführer abhängt, die man verwendet, und nicht nur vom Algorithmus).

Wie kann man Algorithmen trotzdem *bewerten* und miteinander *vergleichen*?

Lösung: Man ermittelt *keinen konkreten Zeitbedarf* sondern (nur) den
abstrakten Zusammenhang zwischen der *Problemgröße* n und
der Anzahl der Rechenschritte s,
die der Algorithmus zur Lösung von Problemen der Größe n benötigt.

Was ist mit "ein Problem der Größe n" gemeint? Ein paar Beispiele:

Zwei Strings vergleichen? (n ist die Länge der Strings)
Reihungen sortieren? (n ist die Länge der Reihung)
Zwei (gleich lange) Zahlen addieren? (n ist die Länge der Zahlen, gemessen in Ziffern oder Bits)

Was ist "ein Schritt" ?

Def.: Ein (Ausführungs-) *Schritt* ist eine *beliebig lange Folge von Befehlen*, von der man annehmen kann, dass ein Ausführer sie immer in (ungefähr) der *gleichen Zeit* ausführen kann. Insbesondere darf die Ausführungszeit eines Schritts *nicht* von der Problemgröße n abhängen.

Die Tabelle auf der Rückseite des Arbeitsblattes (ausgeteilt im vorigen SU, siehe oben S. 28)
(Schritt, JA oder NEIN?) ausfüllen

Haben alle die Schritt (Ja oder Nein?) - Tabelle fertig ausgefüllt?

Bis hierher sind wir im letzten SU gekommen, jetzt machen wir weiter:

Häufig vorkommende Zusammenhänge zwischen einer Problemgröße n und
der Anzahl der Schritte, die zur Lösung eines Problems der Größe n benötigt werden:
Wir sehen uns die Tabelle auf der Rückseite des Arbeitsblatts an (S. 28).

Angenommen, wir haben ein algorithmisches Problem
(z.B.: Reihungen von int-Werten auf eine bestimmte Weise bearbeiten).
Als *Größe* eines Einzelproblems nehmen wir (wie es häufig sinnvoll ist) die *Länge der Reihungen*.

Uns werden zwei Lösungs-Programme P1 und P2 angeboten
P1 realisiert einen O(n)-Algorithmus Alg1 und
P2 realisiert einen $O(n^2)$-Algorithmus Alg2

Was können wir aus den Zeitkomplexitäten der Algorithmen Alg1 und Alg2
über die Programme P1 und P2schließen?

Dass P1 immer schneller ist als P2?

Nein! Denn beim Analysieren von Alg1 wurden vielleicht *viel längere Befehlsfolgen* als *ein* Schritt gewertet als bei Alg2. Es ist also durchaus möglich, dass P1 bei der Bearbeitung von Reihungen der Länge 100 z.B. um den Faktor 500 langsamer ist als P2.

Aber was gilt dann für Probleme der Größe 200, 400, ... ?
Legen Sie eine Tabelle der folgenden Form an und füllen Sie (in Zusammenarbeit mit ihrem Nachbarn) aus. Heute sind Taschenrechner ausnahmsweise erlaubt.

Problemgröße n (Länge der Reihungen)	Zeitbedarf von P1	Zeitbedarf von P2
100	500	1
200	1000	4
400	2000	16
800	4000	64
1600	8000	256
3200	16000	1024
6400	32000	4096
12800	64000	16384
25600	128000	65536
51200	256000	262144

In diesem Beispiel gilt:
Obwohl P1 für Reihungen der Länge 100 etwa *500 mal so schnell* ist wie P2,
ist P2 ab einer Reihungslänge von etwa 50 000 schneller als P1.

Wenn wir meistens Reihungen bearbeiten wollen, die deutlich länger sind als 50 000, sollten wir das O(n)-Programm P1 kaufen. Wenn unsere Reihungen meistens kürzer sind, ist das $O(n^2)$-Programm P2 für uns günstiger.

Allgemein gilt: Es gibt immer eine Problemgröße n, ab der
eine Implementierung eines O(n)-Algorithmus
eine Implementierung einen $O(n^2)$-Algorithmus "überholt".

Jetzt überlegen wir, wie viele Schritte "das Sammeln in unsortierten Reihungen" kostet (und tragen unsere Ergebnisse die Tabelle von S. :

Struktur der Sammlung	Operation	Anz. Schritte in einem schlimmsten Fall	Anz. Schritte im Durchschnitt
Reihung (unsortiert)	Einfügen	1	1
	Suchen pos.	n	n/2
	Suchen neg.	n	n
	Entfernen	Suche + 1	Suchen + 1

Wie schafft man es, eine Komponente mit 1 Schritt zu löschen, wenn man ihren Index

Die Logarithmus-Funktion

Was versteht man unter dem
Logarithmus (zur Basis b) einer Zahl n?

(Die Zahl e für die gilt: b^e ist gleich n.

Zwei (hoffentlich) anschauliche Anwendungen der Logarithmus-Funktionen
(zu verschiedenen Basen) besprechen:

Anwendung 1: Anzahl von Ziffern

Der Logarithmus von n (richtig aufgerundet) gibt
die *Anzahl der Ziffern* an, die man zur Darstellung von n braucht.

Wenn man den 2-er-Logarithmus nimmt: Anzahl der Ziffern im 2-er-System
Wenn man den 3-er-Logarithmus nimmt: Anzahl der Ziffern im 3-er-System
...
Wenn man den 10-er-Logarithmus nimmt: Anzahl der Ziffern im 10-er-System
...

Wir sehen uns die *Tabelle auf der Rückseite der Wiederholungsfragen* an.

Anwendung 2: Wie oft muss man durchsägen?

Angenommen, wir haben einen Baumstamm der n mal so lang ist wie die Tür unseres Ofens. Wie oft müssen wir den Stamm in 2 (gleich lange) Teile zersägen, und dann jedes Teil wieder in 2 Teile zersägen und diese Teile wieder ... bis die Teile kleiner als 1 werden (und somit durch die Ofentür passen)?

Der Logarithmus zur Basis 2 gibt die Antwort.

Welchen Logarithmus müssen wir nehmen, wenn wir den Baumstamm jedesmal nicht in 2 Stücke, sondern in 3, 4, ..., 10, ... zersägen? (den Logarithmus zur Basis 3, 4, ..., 10, ...).

Zur Entspannung: Ein Blatt Papier 50 Mal halbieren und stapeln

Stellen Sie sich vor: Wir haben ein großes Blatt Papier (z.B. eine Blatt der Zeitung "Die Zeit"). Wir reißen oder schneiden das Papier in zwei Hälften und legen die beiden Hälften übereinander. Dann reißen oder schneiden wir diesen kleinen Stapel ebenso in zwei Hälften und legen sie übereinander, dann reißen oder schneiden wir diesen Stapel ebenso in zwei Hälften etc. etc. Insgesamt wiederholen wir diesen Vorgang 50 Mal. Wie hoch ist der Papierstapel am Ende ungefähr?

Tabelle mit 2-er und 10-er-Potenzen, die sich ungefähr entsprechen:

2^{10}	2^{20}	2^{30}	2^{40}	2^{50}	2^{60}
10^3	10^6	10^9	10^{12}	10^{15}	10^{18}
1 Tausend	1 Million	1 Milliarde	1 Billion	1Billiarde	1 Trillion

2^{50} Schichten Zeitungspapier ist gleich
10^{15} Schichten Zeitungspapier (siehe obige Tabelle).

Angenommen, 10 Schichten sind 1 mm dick. Dann gilt:

1 mm	10	Schichten
1 m	10 000	Schichten
1 km	10 000 000	Schichten (d.h. 10^7 Schichten)

10^{15} / 10^7 ist gleich 10^8 gleich 100 Millionen km

Viele algorithmische Probleme kann man lösen,
indem man sie "in Teile zersägt" und die Teilprobleme löst
(indem man sie in noch kleinere Teile zersägt und diese noch kleineren Teilprobleme löst)
(indem man sie in noch kleinere Teile zersägt und diese noch kleineren Teilprobleme löst)
...
bis man Teilprobleme hat, die man leicht ohne weiteres zersägen lösen kann.

Besonders häufig zersägt man Probleme (nicht in 10 oder 5 oder 3 sondern) in 2 Teile. Deshalb vereinbaren wir:

Konvention: Wenn man in der Informatik "Logarithmus" sagt (oder "log" schreibt), ist immer der Logarithmus *zur Basis 2* gemeint. Falls der Logarithmus zu einer anderen Basis gemeint ist, sollte man das ausdrücklich sagen bzw. schreiben.

Die Logarithmen zu verschiedenen Basen sind mathematisch ähnlich "gleichwertig" wie Zahlensysteme mit verschiedenen Basen. Die Logarithmen zu einer Basis b1 hängen mit den Logarithmen zu einer anderen Basis b2 wie folgt zusammen:

$\log_{b2}(n) = \log_{b1}(n) / \log_{b1}(b2)$

Wiederholungsfragen, 7. SU, Mo 08.11.10

1. Wenn man Algorithmen analysiert um ihre Zeitkomplexität herauszufinden, was für Befehlsfolgen darf man dabei als *einen Schritt* betrachten?

2. Bei der Analyse von 4 Algorithmen A1 bis A4 erhalten Sie folgende Tabelle mit Problemgrößen und Schrittzahlen:

Problemgrößen n	10	20	40	80	160
Schrittzahlen A1	2	8	32	128	512
Schrittzahlen A2	500	1000	2000	4000	8000
Schrittzahlen A3	1_000	1_000_000	1_000_000 _000_000	--	--
Schrittzahlen A4	137	138	139	140	141

Geben Sie von jedem der vier Algorithmen an, welche Zeitkomplexität er hat
(natürlich in der Groß-O-Notation, z.B. so: O(n) oder $O(n^2)$ oder O(log n) oder so ähnlich).

3. Sie lesen in einem Buch, dass `log2(17)` ungefähr gleich `4.088` ist.
Was können Sie daraus über die Zahl `17` schließen?

- Dass man zur Darstellung von `2` im `17`-er-System genau `4` Ziffern braucht?
- Dass man zur Darstellung von `17` im `4`-er-System genau `2` Ziffern braucht?
- Dass man zur Darstellung von `17` im `2`-er-System genau `5` Ziffern braucht?
- Dass man zur Darstellung von `17` im `2`-er-System genau `4` Ziffern braucht?

4. Sie wollen einen 16 Fuß langen Baumstamm in Ihrem Holzöfchen verbrennen, dessen Tür nur wenig breiter ist als 1 Fuß. Mit Ihrer Kreissäge können Sie *mit einem Schnitt* beliebig viele Stämme durchsägen (indem Sie die Stämme parallel zueinander quer auf den Sägetisch legen und dann "durchziehen").
Wie viele Schnitte brauchen Sie, um den Baumstamm "öfchengerecht" zu zersägen?

5. Angenommen, Ihr Taschenrechner hat (außer den üblichen Knöpfen für die Grundrechenarten +, -, /, *) nur zwei Knöpfe, mit denen man den Logarithmus zur Basis 10 und den zur Basis e berechnen kann.
Wie können Sie (trotzdem) mit diesem Taschenrechner einen Logarithmus zur Basis 2 berechnen, z.B. `log2(17)`?

Rückseite der Wiederholungsfragen, 7. SU, Mo 08.11.10

Seien `Sam1` und `Sam2` zwei (fast) beliebige *Sammlungstypen.* Im Prinzip kann man dann aus jeder `Sam1`-Sammlung `S1` eine `Sam2`-Sammlung erzeugen, die dieselben Komponenten wie `S1` enthält. An Hand der folgenden Tabelle sollen Sie sich klar machen, was das "fast" bedeutet.

Zur Erinnerung: Es gilt: `Object` ← `Number` ← `Integer`
(d.h. `Number` ist eine Unterklasse von `Object` und `Integer` ist eine Unterklasse von `Number`)

Sam1	Sam2	Kann man aus einer Sam1-Sammlung eine Sam2-Sammlung erzeugen?
`ArrayList<Integer>`	`TreeSet<Integer>`	
`TreeSet<Integer>`	`ArrayList<Integer>`	
`ArrayList<Integer>`	`ArrayList<Number>`	
`ArrayList<Integer>`	`TreeSet<Number>`	
`TreeSet<Object>`	`TreeSet<Integer>`	
`TreeSet<Number>`	`ArrayList<Integer>`	

Anstelle von `ArrayList` und `TreeSet` könnten in dieser Tabelle auch beliebige andere Sammlungstypen stehen wie z.B. `Stack`, `Vector`, `LinkedList`, `PriorityQueue`, `PriorityBlockingQueue`, `CopyOnWriteArrayList`, `CopyOnWriteArraySet`, ... etc.

Jede Sammlungsklasse `Sam<K>` implementiert die Schnittstelle `iterable<K>`! Wozu?

```
 1 class Siggy<K> implements Collection<K>, Iterable<K> { ... }
 2
 3 class IrgendEine {
 4
 5    Siggy sig = new Siggy(...);
 6
 7    void machWas01 {
 8       for (Iterator<K> it=sig.iterator(); it.hasNext(); ) {
 9          K k = sig.next();
10          ... // Die Komponente k kann jetzt bearbeitet werden
11       }
12    }
13
14    void machWas02 {
15       for (K k : sig) {
16          ... // Die Komponente k kann jetzt bearbeitet werden
17       }
18    }
19    ...
20 } // class IrgendEine
```

Iteratoren bzw. `for-each`-Schleifen erlauben es, einen Sammlungstyp wie `Siggy` *auszutauschen*, ohne den verarbeitenden Code zu ändern! Wie man es *nicht* machen sollte:

```
21    ArrayList<String> als = new ArrayList<String>(...);
22    for (int i=0; i<als.size(); i++) {
23       String s = als.get(i);
24       ... // Die Komponente s bearbeiten
```

Jetzt kann man den Typ `ArrayList` nicht durch `TreeSet` ersetzen
(weil `TreeSet`-Objekte keine `get`-Methode haben, siehe Zeile 23).

Antworten zu den Wiederholungsfragen, 7. SU, Mo 08.11.10

1. Wenn man Algorithmen analysiert um ihre Zeitkomplexität herauszufinden, was für Befehlsfolgen darf man dabei als *einen Schritt* betrachten?

Jede Befehlsfolge von der es plausibel ist anzunehmen, dass ein Ausführer sie jedesmal in etwa der gleichen Zeit ausführen kann.

2. Bei der Analyse von 4 Algorithmen A1 bis A4 erhalten Sie folgende Tabelle mit Problemgrößen und Schrittzahlen:

Problemgröße n	10	20	40	80	160
Schrittzahle A1	2	8	32	128	256
Schrittzahle A2	500	1000	2000	4000	8000
Schrittzahle A3	1_000	1_000_000	1_000_000 _000_000	--	--
Schrittzahle A4	137	138	139	140	141

Geben Sie von jedem der vier Algorithmen an, welche Zeitkomplexität er hat
(natürlich in der Groß-O-Notation, z.B. so: O(n) oder $O(n^2)$ oder O(log n) oder $O(2^n)$ oder so ähnlich).

A1: $O(n^2)$, A2: O(n), A3: $O(2^n)$, A4: O(log n)

3. Sie lesen in einem Buch, dass `log2(17)` ungefähr gleich `4.088` ist.
Was können Sie daraus über die Zahl `17` schließen?

- Dass man zur Darstellung von `2` im `17`-er-System genau `4` Ziffern braucht?
- Dass man zur Darstellung von `17` im `4`-er-System genau `2` Ziffern braucht?
- Dass man zur Darstellung von 17 im `2`-er-System genau 5 Ziffern braucht? Richtig
- Dass man zur Darstellung von `17` im `2`-er-System genau `4` Ziffern braucht?

4. Sie wollen einen 16 Fuß langen Baumstamm in Ihrem Holzöfchen verbrennen, dessen Tür nur wenig breiter ist als 1 Fuß. Mit Ihrer Kreissäge können Sie *mit einem Schnitt* beliebig viele Stämme durchsägen (indem Sie die Stämme parallel zueinander quer auf den Sägetisch legen und dann "durchziehen").
Wie viele Schnitte brauchen Sie, um den Baumstamm "öfchengerecht" zu zersägen?

4 Schnitte (denn Sie haben
nach dem 1. Schnitt 2 Stücke der Länge 8 Fuß,
nach dem 2. Schnitt 4 Stücke der Länge 4 Fuß,
nach dem 3. Schnitt 8 Stücke der Länge 2 Fuß und
nach dem 4. Schnitt 16 Stücke der Länge 1 Fuß).

5. Angenommen, Ihr Taschenrechner hat (außer den üblichen Knöpfen für die Grundrechenarten +, -, /, *) nur zwei Knöpfe, mit denen man den Logarithmus zur Basis 10 und den zur Basis e berechnen kann.
Wie können Sie (trotzdem) mit diesem Taschenrechner einen Logarithmus zur Basis 2 berechnen, z.B. `log2(17)`?

Nach der Formel log2(n) ist gleich log10(n) / log10(2)
z.B. log2(17) ist gleich
log10(17) / log10(2) ist ungefähr gleich
1.2304 / 0.3010 ist ungefähr gleich
4.088

Lösung zur Rückseite der Wiederholungsfragen, 7. SU, Mo 08.11.10

Seien `Sam1` und `Sam2` zwei (fast) beliebige *Sammlungstypen*. Im Prinzip kann man dann aus jeder `Sam1`-Sammlung `S1` eine `Sam2`-Sammlung erzeugen, die dieselben Komponenten wie `S1` enthält.
An Hand der folgenden Tabelle sollen Sie sich klar machen, was das "fast" bedeutet.

Zur Erinnerung: Es gilt: `Object` ← `Number` ← `Integer`
(d.h. `Number` ist eine Unterklasse von `Object` und `Integer` ist eine Unterklasse von `Number`)

Sam1	Sam2	Kann man aus einer Sam1-Sammlung eine Sam2-Sammlung erzeugen?
`ArrayList<Integer>`	`TreeSet<Integer>`	ja
`TreeSet<Integer>`	`ArrayList<Integer>`	ja
`ArrayList<Integer>`	`ArrayList<Number>`	ja (weil `Number` ← `Integer`)
`ArrayList<Integer>`	`TreeSet<Number>`	ja (weil `Number` ← `Integer`)
`TreeSet<Object>`	`TreeSet<Integer>`	nein (weil nicht `Integer` ← `Object`)
`TreeSet<Number>`	`ArrayList<Integer>`	nein (weil nicht `Integer` ← `Number`)

Anstelle von `ArrayList` und `TreeSet` könnten in dieser Tabelle auch
beliebige andere Sammlungstypen stehen wie z.B.
`Stack`, `Vector`, `LinkedList`, `PriorityQueue`, `PriorityBlockingQueue`,
`CopyOnWriteArrayList`, `CopyOnWriteArraySet`, ... etc.

7. SU, Mo 08.11.10

A. Wiederholung
B. Organisation

Die Java-Sammlungsklassen sind keine Einzelkämper

Sie bilden ein *System*, weil jede Sammlungsklasse `Sam<K>` folgende *Eigenschaften* hat:

1. Sie implementiert die Schnittstelle `Collection<K>`

2. Sie enthält einen Konstruktor mit einem Parameter des Typs `Collection<? extends K>`

3. Sie implementiert die Schnittstelle `iterable<K>`,
die genau ein Methode enthält: `public Iterator<K> iterator()`.
Dabei ist `Iterator<T>` eine Schnittstelle, die 3 Methoden enthält:
boolean `hasNext()`, `T next()`, `void remove()`.

Zu 1.: Die Schnittstelle `Collection<K>` haben wir ausführlich besprochen. Lesen Sie ab und zu in der Online-Dokumentation dieser Schnittstelle, bis Sie alles mehrmals gelesen und verstanden haben.

Zu 2.: Was bedeutet diese Eigenschaft (die mit dem Konstruktor)?
(Dass man aus jeder Sammlung eines Typs `Sam1` eine Sammlung eines Typs `Sam2` machen)

Dazu jetzt die Tabelle auf der Rückseite der Wiederholungsfragen ausfüllen.

Zu 3. Was bedeutet diese Eigenschaft?
Rückseite der Wiederholungsfragen, untere Hälfte.

Sortierte Reihungen (als Ersatz für Sammlungen)

Für das Problem "Suchen in einer sortierten Reihung" gibt es einen besonders berühmten Lösungs-Algorithmus, der das Problem "durch Zersägen in 2 Teile" löst:

Binäres suchen

An der Tafel mit folgender Tabelle vorführen:

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Kompos	3	15	19	23	25	28	31	33	42	46	51	52	57	59	64
Schritt	4	3	4	2	4	3	4	1	4	3	4	2	4	3	4

Zuerst nur die oberen beiden Zeilen (Index und Kompos der Reihung) ausfüllen. Dann die Zahlen in der untersten Zeile (Schritt) ergänzen (erst die eine 1, dann die beiden 2en, dann die vier 3en etc.)

Die Zahlen in der untersten Zeile (Schritte) geben an, wie viele *Suchschritte* wir brauchen, um beim binären Suchen an dieses Stelle der Reihung zu kommen.

Ein paar Zahlen, die man auswendig kennen sollte (alle Logarithmen zur Basis 2!):

2^{10} = 1 Tausend	log(1 Tausend) = **10**
2^{20} = 1 Million	log(1 Million) = **20**
2^{30} = 1 Milliarde	log(1 Milliarde) = **30**
2^{40} = 1 Billion	log(1 Billion) = **40**

Angenommen, wir haben eine Reihung mit 8 Milliarden Telefon-Nrn darin (etwa für jeden Menschen eine) und wollen von einer bestimmten Nummer feststellen, ob sie schon in der Tabelle steht. Wie viele Suchschritte brauchen wir, wenn die Reihung *sortiert* ist und wir *binär suchen*? (33).

Wie viele Suchschritte würden wir im Durchschnitt brauchen, wenn die Reihung *nicht sortiert* wäre? (etwa 4 Milliarden).

Wir tragen in unsere Tabelle Daten für "sortierte Reihungen" ein:

Struktur der Sammlung	Operation	Anz. Schritte in einem schlimmsten Fall	Anz. Schritte im Durchschnitt
Reihung (sortiert)	Einfügen	Suchen + n	Suchen + n
	Suchen pos.	log(n)	beinahe log(n)
	Suchen neg.	log(n)	log(n)
	Entfernen	Suchen + etwas	Suchen + etwas

Zum Eintrag "beinahe log(n)": In der dreizeiligen Tabelle auf der vorigen Seite kann man erkennen:

Für eine Hälfte aller Komponenten braucht man log(n) Schritte, um sie zu erreichen.
Für ein weiteres Viertel braucht man log(n)-1 Schritte.
Für eine weiteres Achtel braucht man log(n)-2 Schritte.
Für 7/8 aller Komponenten braucht man also log(n)-2 oder mehr Schritte.

Das "etwas" ist also nicht sehr groß (und die Formel dafür ist ein bisschen kompliziert).

Zur Entspannung: Carl Adam Petri (1926 - 2010)

Hielt Nebenläufigkeit für ein besonderes, grundlegendes Phänomen, welches man nicht durch andere Phänomene (z. B. unsere Vorstellung von Zeit) erklären, sondern "axiomatisch einführen" sollte. Er erfand die Petri-Netze, einen Formalismus, mit dem man nebenläufige Systeme präzise beschreiben kann.

Petri war viele Jahre lang Leiter eines eigenen Forschungsinstituts bei der *Gesellschaft für Mathematik und Datenverarbeitung* (GMD) in St. Augustin bei Bonn. Persönlich sehr freundlich. Bei alltäglichen Problemen, die nichts mit seiner Forschung zu tun hatten, eher unbeholfen. Er wurde für den Turing-Preis vorgeschlagen, aber als eine entsprechende Kommission aus Amerika nach Bonn kam, um seine Verdienste zu untersuchen, ließ er sich entschuldigen: Er habe noch Briefe zu schreiben.

Genie und Hilflosigkeit?

Die GMD wurde 2000/2001 in die Fraunhofer-Gesellschaft integriert.

Was ist gut an sortierten Reihungen?

Das Suchen (pos. und neg.) geht sehr schnell!

Was ist schlecht an sortierten Reihungen?

1. Sie sind aus Beton (nicht verlängerbar / verkürzbar)

2. Beim Einfügen und beim Löschen müssen "viele Komponenten verschoben werden".

Beide Probleme kann man mit Sammlungen einer bestimmten Struktur lösen:

Mit verketteten Listen.

Einfach verkettete Listen:
(Fast) jede Komponenten zeigt auf die nachfolgende Komponente
Doppelt verkettete Listen:
(Fast) jede Komponente zeigt auf die nachfolgende und die vorhergehende Komponente.

Die Übung03 (aus der Datei `UebungenPR2.pdf`) verteilen und Bearbeitung beginnen.

Wiederholungsfragen, 8. SU, 15.11.10

Zur Erinnerung: Im vorigen Semester haben wir uns u.a. mit zwei Klassen namens `Person01` und `Person02` befaßt, wobei `Person02` eine Unterklasse von `Person01` war.

1. Angenommen, wir haben ein Sammlung namens `otto` vom Typ `ArrayList<Person01>`. Können wir daraus eine Sammlung namens `emil` vom Typ `TreeSet<Person02>` erzeugen oder nicht? Falls ja, wie müsste die Vereinbarung von `emil` aussehen?

2. Angenommen, wir haben ein Sammlung namens `emil` vom Typ `TreeSet<Person02>`. Können wir daraus eine Sammlung namens `otto` vom Typ `ArrayList<Person01>` erzeugen oder nicht? Falls ja, wie müsste die Vereinbarung von `otto` aussehen?

3. Angenommen die Sammlung `anna` vom Typ `TreeSet<String>` enthält 100 Komponenten, von denen jede 1 MB belegt (`anna` belegt insgesamt also ein klein bisschen mehr als 100 MB). In dieser Situation lassen wir den Befehl

```
1    ArrayList<String> berta = new ArrayList<String>(anna);
```

ausführen. Wie viele MBs belegen die Sammlungen `anna` und `berta` dann zusammengenommen?

4. *Wie viele* Methoden enthält die Schnittstelle `Iterable` und wie *heißen* diese Methoden?

5. *Wie viele* Methoden enthält die Schnittstelle `Iterator` und wie *heißen* diese Methoden?

6. Wie viele Schritte braucht man höchstens, um in einer sortierten Reihung der Länge 8 Tausend in einem negativen Fall zu suchen? Geben Sie die konkrete ("ausgerechnete") Zahl an, keine Formel.

7. Ebenso für eine Reihung der Länge 2 Millionen?

8. Besonders gut ist an sortierten Reihungen, dass das *Suchen* (im positiven und im negativen Fall) sehr schnell geht. Was ist weniger gut an sortierten Reihungen?

9. Welche Zeitkomplexität hat das *Suchen im negativen Fall* in einer sortierten Reihung der Länge n?

Rückseite der Wiederholungsfragen, 8. SU, 15.11.10

Problem: Bei einem Vergleich zweier Zahlen (oder ähnlicher Objekte) `a` und `b` sind *drei* Ergebnisse möglich: `a` ist *kleiner* oder *gleich* oder *größer* als `b`. Die üblichen Vergleichsoperatoren wie <, <=, > etc. unterscheiden aber nur *zwei* mögliche Ergebnisse. Um alle drei möglichen Ergebnisse zu unterscheiden, muss man also *zwei Vergleiche* mit solchen Operatoren durchführen. Beim Vergleich großer Objekte (z.B. Strings, die eine Million `char`-Werte enthalten) kann das ziemlich teuer sein.

Die Schnittstellen `Comparable<T>` und `Comparator<T>` beschreiben je eine Vergleichsmethode für T-Objekte, die mit *einem* Aufruf alle *drei* möglichen Ergebnisse unterscheiden. Diese Methoden liefern als Ergebnis (natürlich keinen `boolean`-Wert sondern) einen `int`-Wert:

```
 2 public interface Comparable<T> {
 3    public int compareTo(T that);
 4    // Liefert eine negative Zahl bzw. 0 bzw. eine positive Zahl,
 5    // wenn this kleiner bzw. groesser bzw. gleich that ist.
 6 }
 7
 8 public interface Comparator<T> {
 9    public int compare(T tob1, T tob2);
10    // Liefert eine negative Zahl bzw. 0 bzw. eine positive Zahl,
11    // wenn tob1 kleiner bzw. groesser bzw. gleich tob2 ist.
12    ...
13    // Eine zweite, sehr selten benoetigte Methode namens equals
14 }
```

Anwendungsbeispiel: Die Klasse `String` implementiert die Schnittstelle `Comparable<String>`. Ein Vergleich zweier `String`-Objekte `s1` und `s2` sieht typischerweise so aus:

```
15    int erg = s1.compareTo(s2);                    // 1 evtl. teurer Vergleich
16    if (erg < 0) {                                 // 1 billiger Vergleich
17       ... // s1 ist lexikografisch kleiner als s2
18    } else if (erg > 0) {                          // 1 billiger Vergleich
19       ... // s1 ist lexikografisch groesser als s2
20    } else {
21       ... // s1 ist gleich s2
22    }
```

Wie kann man Objekte der folgenden Klasse `Mango` *vergleichbar* machen?

```
23 class Mango implements ...                                    {
24    int    saftmenge;
25    char   guete;
26    float  gewicht;
27    ...
```

Möglichkeit 1: Man implementiert in der Klasse `Mango` die Schnittstelle `Comparable<Mango>`.

Ein Vergleich der `Mango`-Objekte `ma` und `mb` sieht dann z.B. so aus:

```
28 int erg = ma.compareTo(mb);
```

Möglichkeit 2: Man implementiert in einer beliebigen Klasse `Otto` die Schnittstelle `Comparator<Mango>` und erzeugt ein Objekt `otto` der Klasse `Otto`.

Ein Vergleich der `Mango`-Objekte `ma` und `mb` sieht dann z.B. so aus:

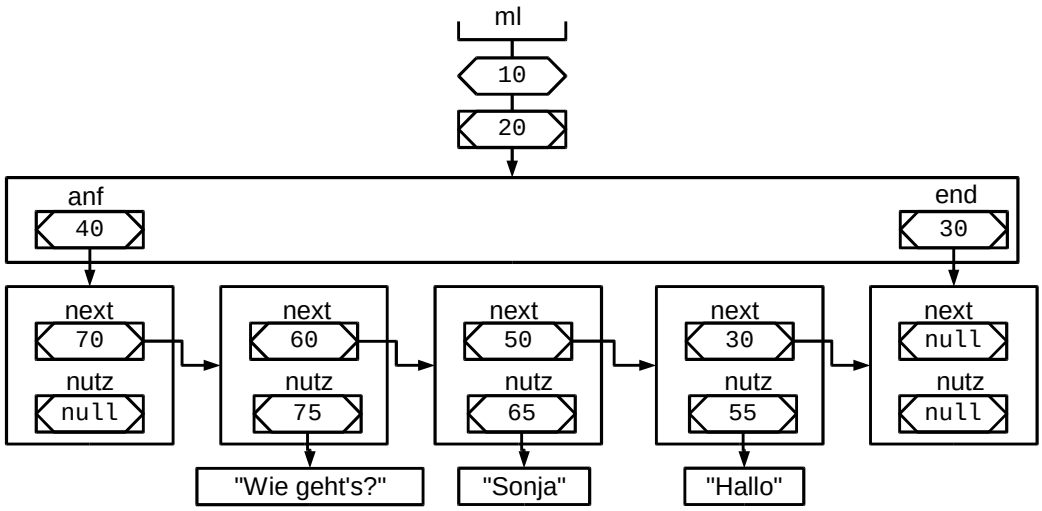
```
29 int erg = otto.compare(ma, mb);
```

Von der Möglichkeit 1 kann man nur *einmal* Gebrauch machen. Die dadurch definierte totale Ordnung (z.B. für `Mango`-Objekte) wird auch als *die natürliche Ordnung* (für `Mango`-Objekte) bezeichnet.

Von der Möglichkeit 2 kann man *beliebig oft* Gebrauch machen. Die dadurch definierten totalen Ordnungen (z.B. für `Mango`-Objekte) bezeichnen wir als *sonstige Ordnungen* (für `Mango`-Objekte).

Arbeitsblatt für 8. SU, Mo 15.11.10 (nur eine Seite, Rückseite ist leer)

Eine mögliche Lösung zur Übung Üb03-2 (die `MeineListe`-Variable `ml` nach dem Einfügen von 3 `String`-Komponenten als Boje dargestellt). Diese Lösung sollte als Grundlage für die Übungen Üb3-3, und Üb3-4 genommen werden, damit wir alle dieselben Referenzwerte haben und unsere Lösungen vergleichen können:



Üb03-3: Führen Sie (auf der Grundlage der hier abgebildeten Bojendarstellung) den folgenden Befehl ("mit Papier und Bleistift") aus:

```
30  Knoten vorKarl = ml.sucheVor("Karl!");
```

Nehmen Sie dabei an, dass `sucheVor` als *öffentliche* Methode vereinbart wurde (nicht als private). Dies ist offenbar eine *Suche in einem negativen Fall.*
Mit welchem (Refernz-) Wert wird `vorKarl` initialisiert?

Üb03-4: Führen Sie ganz entsprechend auch den folgend Befehl aus:

```
31  Knoten vorSonja = ml.sucheVor("Sonja!");
```

Dies ist offenbar eine *Suche in einem positiven Fall.*
Mit welchem (Refernz-) Wert wird `vorSonja` initialisiert?

Üb03-5: Wie könnte der Rumpf der Methode `public boolean istDrin(K nutz)` aussehen?
Tip: Bevor Sie diese Methode programmieren sollten Sie möglichst genau wissen, was die anderen Methoden der Klasse machen (damit Sie nichts noch mal programmieren was es schon fertig gibt).

Üb03-6: Wie könnte der Rumpf der Methode `public boolean entferne(K nutz)` aussehen?

Üb03-7: Wozu ist der Anfangs-Dummy-Knoten gut? Warum hat man ihn eingeführt?

Üb03-8: Wozu ist der End-Dummy-Knoten gut? Warum hat man ihn eingeführt?

Lösungen zum Arbeitsblatt für 8. SU, Mo 15.11.10

Üb03-3: Führen Sie (auf der Grundlage der hier abgebildeten Bojendarstellung) den folgenden Befehl ("mit Papier und Bleistift") aus:

```
32  Knoten vorKarl = ml.sucheVor("Karl!");
```

Mit welchem (Refernz-) Wert wird `vorKarl` initialisiert?

Lös03-3: Die Variable `vorKarl` wird mit dem (Referenz-) Wert **<50>** initialisiert (der auf den Knoten vor dem `end`-Knoten zeigt).

Üb03-4: Führen Sie ganz entsprechend auch den folgend Befehl aus:

```
33  Knoten vorSonja = ml.sucheVor("Sonja!");
```

Mit welchem (Refernz-) Wert wird `vorSonja` initialisiert?

Lös03-4: Die Variable `vorSonja` wird mit dem (Referenz-) Wert **<70>** initialisiert (das ist auch der Wert der Variablen `anf.next`, der auf den ersten "richtigen Knoten" der Liste zeigt).

Üb03-5: Wie könnte der Rumpf der Methode `public boolean istDrin(K nutz)` aussehen? Tip: Bevor Sie diese Methode programmieren sollten Sie möglichst genau wissen, was die anderen Methoden der Klasse machen (damit Sie nichts noch mal programmieren was es schon fertig gibt).

Lös03-5: Der Rumpf der Methode `istDrin` kann etwa so aussehen:

```
 1    public boolean istDrin(K nutz) {
 2       // Liefert true, wenn nutz in dieser Sammlung vorkommt.
 3       Knoten vor = sucheVor(nutz);
 4       return vor.next != end;
 5    } // istDrin
```

Üb03-6: Wie könnte der Rumpf der Methode `public boolean entferne(K nutz)` aussehen?

Lös03-6: Der Rumpf der Methode `entferne` kann etwa so aussehen:

```
 6    public boolean entferne(K nutz) {
 7       // Falls nutz in einem oder mehreren Knoten dieser Sammlung vorkommt,
 8       // wird einer dieser Knoten geloescht und true geliefert. Sonst wird
 9       // false geliefert.
10
11       Knoten vor = sucheVor(nutz);
12       if (vor.next == end) return false; // nutz nicht in dieser Sammlung
13       vor.next = vor.next.next;          // Entferne einen Knoten
14       return true;
15    } // entferne
```

Üb03-7: Wozu ist der Anfangs-Dummy-Knoten gut? Warum hat man ihn eingeführt?

Lös03-7: Der Anfangs-Dummy-Knoten bewirkt, dass auch der 1. richtige Knoten der Liste einen *Vorgänger-Knoten* hat und die Methode `sucheVor` für *jeden* richtigen Knoten (auch für den 1.) einen Vorgänger-Knoten liefern kann. Und weil die `sucheVor`-Methode so "glatt und ohne Ausnahme" funktioniert, kann man mit ihrer Hilfe die Methoden `istDrin` und `entferne` vereinfachen.

Üb03-8: Wozu ist der End-Dummy-Knoten gut? Warum hat man ihn eingeführt?

Lös03-8: Der End-Dummy-Knoten beschleunigt die Methode `sucheVor` (und damit die Methoden `istDrin` und `entferne`). Weil `sucheVor` sich darauf verlassen kann, dass die zu suchenden Nutzdaten in der Liste vorkommen (spätestens im End-Dummy-Knoten), braucht sie in ihrer Schleife nur *eine* Abfrage ("Habe ich die gesuchten Nutzdaten gefunden?") und kann auf eine *zweite* Abfrage ("Bin ich am Ende der Liste?") verzichten.

Antworten zu den Wiederholungsfragen, 8. SU, 15.11.10

Zur Erinnerung: Im vorigen Semester haben wir uns u.a. mit zwei Klassen namens `Person01` und `Person02` befaßt, wobei `Person02` eine Unterklasse von `Person01` war.

1. Angenommen, wir haben ein Sammlung namens `otto` vom Typ `ArrayList<Person01>`. Können wir daraus eine Sammlung namens `emil` vom Typ `TreeSet<Person02>` erzeugen oder nicht? Falls ja, wie müsste die Vereinbarung von `emil` aussehen?

Nein (weil Person01-Objekte keine Person02-Objekte sind)

2. Angenommen, wir haben ein Sammlung namens `emil` vom Typ `TreeSet<Person02>`. Können wir daraus eine Sammlung namens `otto` vom Typ `ArrayList<Person01>` erzeugen oder nicht? Falls ja, wie müsste die Vereinbarung von `otto` aussehen?

```
1    ArrayList<Person01> otto = new ArrayList<Person02>(emil);
```

3. Angenommen die Sammlung `anna` vom Typ `TreeSet<String>` enthält 100 Komponenten, von denen jede 1 MB belegt (`anna` belegt insgesamt also ein klein bisschen mehr als 100 MB). In dieser Situation lassen wir den Befehl

```
1    ArrayList<String> berta = new ArrayList<String>(anna);
```

ausführen. Wie viele MBs belegen die Sammlungen `anna` und `berta` dann zusammengenommen?

Etwas mehr als 100 MB (und nicht etwa 200 MB).

4. *Wie viele* Methoden enthält die Schnittstelle `Iterable` und wie *heißen* diese Methoden?

Eine Methode. Sie heißt iterator.

5. *Wie viele* Methoden enthält die Schnittstelle `Iterator` und wie *heißen* diese Methoden?

Drei Methoden. Sie heißen hasNext, next und remove.

6. Wie viele Schritte braucht man höchstens, um in einer sortierten Reihung der Länge 8 Tausend in einem negativen Fall zu suchen? Geben Sie die konkrete ("ausgerechnete") Zahl an, keine Formel.

13 Schritte.

7. Ebenso für eine Reihung der Länge 2 Millionen?

21 Schritte

8. Besonders gut ist an sortierten Reihungen, dass das *Suchen* (im positiven und im negativen Fall) sehr schnell geht. Was ist weniger gut an sortierten Reihungen?

Dass sie aus Beton sind und dass beim Einfügen viele Komponenten verschoben werden müssen (im Durchschnitt die Hälfte aller Komponenten, um je eine Position).

9. Welche Zeitkomplexität hat das *Suchen im negativen Fall* in einer sortierten Reihung der Länge n?

log(n)

8. SU, Mo 15.11.10

A. Wiederholung
B. Organisation

Wie kann man Objekte "auf kleiner/größer" vergleichbar machen?

Vorgeschichte: Jedes Java-Objekt `ob1` enthält eine Methode namens `equals`, mit der man es mit einem beliebigen Java-Objekt `ob2` vergleichen kann, etwa so:

```
1    boolean erg12 = ob1.equals(ob2);
2    boolean erg21 = ob2.equals(ob1);
```

Dabei dürfen `ob1` und `ob2` zum *selben Typ* oder zu *verschiedenen Typen* gehören, und die Namen `ob1` und `ob2` dürfen *dasselbe* Objekt oder *zwei unterschiedliche* Objekte bezeichnen.

Die Methoden `ob1.equals` und `ob2.equals` sollten so programmiert sein, dass die beiden Ergebnisse `erg12` und `erg21` *gleich* sind. Leider kann diese Bedingung nicht automatisch (d.h. vom Java-Ausführer) geprüft werden. Aber wenn sie *nicht* erfüllt wird (und man z.B. `ob1` und `ob2` in bestimmte Sammlungen einfügt), können daraus böse Fehler entstehen.

Aber was kann bzw. muss man tun, wenn man von einem Objekt `ob1` feststellen will, ob es *größer* (oder kleiner) ist als ein Objekt `ob2`? Um eine Reihung oder Sammlung von Objekten zu sortieren muss der Programmierer Antworten auf solche Fragen festlegen.

Grundregel: In Java kann man mit den Operatoren <, <=, >, >= nur *primitive* Werte vergleichen, aber keine Referenzen oder Zielwerte (d.h. Objekte).

Frage 1: Warum nicht? (Nur 2 Ergebnisse, uneffizient)
Frage 2: Und wie kann man die Sortierreihenfolge von Objekten einer bestimmten Klasse festlegen?

Dazu betrachten wir die Rückseite des Blattes mit den Wiederholungsfragen.

Wichtige Erkenntnis: Vergleichsoperationen mit einem `boolean`-Ergebnis sind (beim Vergleichen von "großen Objekten") *uneffizient*.

In Java (und anderen Sprachen) haben *Vergleichsfunktionen für Objekte* deshalb *nicht* den Ergebnistyp `boolean`, sondern den Ergebnistyp `int`, damit sie nicht nur 2, sondern 3 verschiedene Ergebnisse: (das erste Objekt ist *kleiner*, *gleich* bzw. *größer* als das zweite) unterscheiden können.

Aufgabe A: Ergänzen Sie die Vereinbarung der Klasse `Mango` (siehe Rückseite der WF, ab Zeile 23) nach Möglichkeit 1 (d.h. mit Hilfe der Schnittstelle `Comparable`). Dabei sollen `Mango`-Objekte *aufsteigend* nach ihrem `gewicht` verglichen werden (je größer das `gewicht`, desto größer das `Mango`-Objekt).

Lösung A:

```
 1 class Mango implements Comparable<Mango> {
 2    int    saftmenge;
 3    char   guete;
 4    float  gewicht;
 5
 6    public int compareTo(Mango that) {
 7       // Verlaesst sich darauf, dass Mango-Objekte
 8       // "vernuenftige Gewichte" haben
 9       // (z.B. nicht negativ, kleiner als 1 Milliarde)
10       return (int) (this.gewicht - that.gewicht);
11    }
12    ...
13 }
```

Aufgabe B: Ergänzen Sie die Vereinbarung der Klasse `Mango` (siehe Rückseite der WF, ab Zeile 23) nach Möglichkeit 2 (d.h. mit Hilfe der Schnittstelle `Comparator`). Dabei sollen `Mango`-Objekte *absteigende* nach ihrer `guete` verglichen werden (je kleiner die `guete`, desto größer das `Mango`-Objekt).

Lösung B:

```
14 class Gustav implements Comparator<Mango> {
15    public in compare(Mango a, Mango b) {
16       return b.guete - a.guete;
17    }
18 }
```

Zur Entspannung: Eigenschaften von Qbits (Quanten-Bits)

1. Mit n "normalen Bits" kann man *eine* Zahl (zwischen 0 und 2^n-1) darstellen.
Mit n Qubits kann man gleichzeitig bis zu 2^n Zahlen (zwischen 0 und 2^n-1) darstellen und mit *einer* Operation kann man alle diese Zahlen "gleichzeitig bearbeiten".

2. Wenn man ein Qubit "ansieht und ausliest", bekommt man nur *einen* seiner Werte. Alle anderen Werte gehen dabei unvermeidbar und unwiderruflich verloren.

3. Es ist nicht möglich, ein Qubit (mit all seinen Werten) zu kopieren. Man kann höchstens *einen* seiner Werte kopieren.

4. Auf Qubits kann man nur *umkehrbare Verknüpfungen* anwenden.

Zur Zeit (2008) erforschen mehrere Tausend Physiker, Informatiker und Ingenieure in mehr als 100 Forschungsgruppen etwa ein Dutzend Möglichkeiten, Qbits zu realisieren (durch ion traps. quantum dots, linear optics, ...).

Eine interessante Einführung in die Quantenmechanik von einer berliner Schülerin:

Silvia Arroyo Camejo: "Skurrile Quantenwelt", Springer 2006, Fischer 2007

Das Arbeitsblatt für 8. SU (siehe oben S. 48) austeilen und bearbeiten lassen.

Wiederholungsfragen, 9. SU, Mo 22.11.10

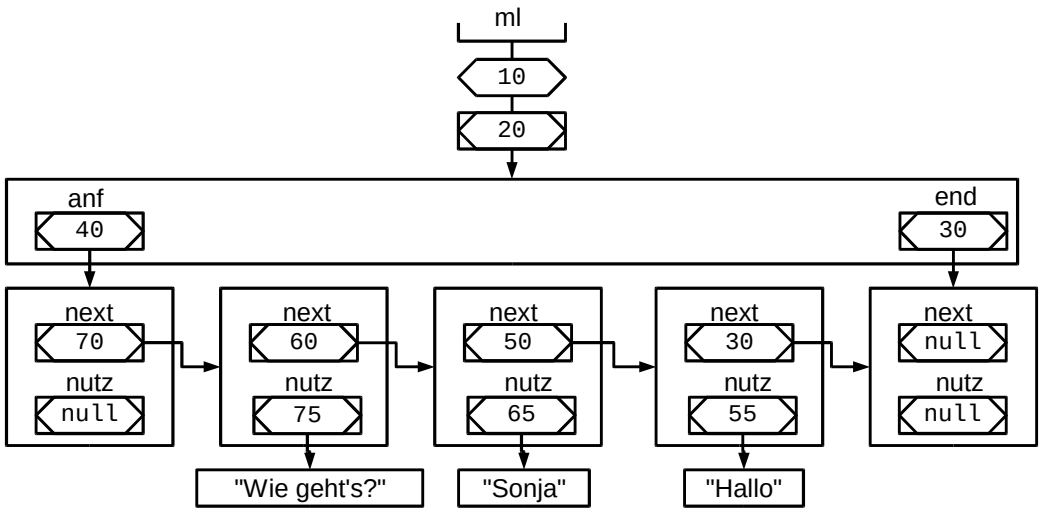
1. Machen Sie Objekte der Klasse `Mango`

```
1 class Mango {
2    int    saftmenge;
3    char   guete;
4    float  gewicht;
5    ...
```

mit Hilfe der Schnittstelle `Comparator` (nicht mit der Schnittstelle `Comparable`!) vergleichbar. Dabei soll ein `Mango`-Objekt `m1` *größer* sein als ein `Mango`-Objekt `m2`, wenn die `safttmenge` von `m1` *größer* ist als die von `m2` (kurz: `Mango`-Ordnung nach `saftmenge`, aufsteigend).

2. Die Klasse `String` implementiert die Schnittstelle `Comparable<String>`. Wie könnte die Methode `compareTo` in der Klasse `String` aussehen? Geben Sie eine mögliche Implementierung an.

3. Betrachten Sie noch einmal ein Objekt `ml` der Klasse `MeineListe`, nachdem drei bestimmte Strings eingefügt wurden:



Angenommen, jetzt wird die folgende Variablenvereinbarung ausgeführt:

```
6    Knoten vorHallo = ml.sucheVor("Hallo");
```

Mit welchem Wert wird die Variable `vorHallo` initialisiert?

Rückseite der Wiederholungsfragen, 9. SU, Mo 22.11.10

Mit DNS-Molekülen kombinatorische Probleme lösen

Die Gene aller Lebewesen bestehen aus DNS (Desoxyribonukleinsäure) Molekülen. Solche Moleküle können große Mengen von Informationen auf sehr kleinem Raum speichern. *Leonard M. Adleman* hat 1994 zuerst gezeigt, dass man mit solchen Molekülen auch bestimmte algorithmische Probleme lösen kann, z.B. das *Problem des Handlungsreisenden* (ein Handlungsreisender will n Städte besuchen und sucht nach einem kürzesten Weg dafür). In ein Reagenzglas passen mehrere Trillionenen DNS-Moleküle, die (unter geeigneten Bedingungen) alle versuchen, sich miteinander zu verbinden. Mit DNS Ligase kann man bestimmte Verbindungen erheblich erleichtern und damit beschleunigen.

Ein DNS-Moleküle besteht aus 2 Ketten von Nukleotiden, von denen es vier Arten gibt: A, G, C, T. Ketten verbinden sich, wenn sich überalle *komplementäre* Nukleotide gegenüberstehen, etwa so:

```
Kette 1: A.C.G.T. ...
         | | | |
Kette 2: T.G.C.A. ...
```

Grundtechnik zur Lösung des Problems des Handlungsreisenden: Jede Stadt wird durch eine einfache (nicht doppelte!) Kette von 20 Nukleotiden dargestellt, z.B. durch

```
Stadt A: TTGACGAATG ATGCTAGAAA (Komplement: AACTGCTTAC TACGATCTTT)
Stadt B: AATCCATGCG AAATTAGCCC (Komplement: TTAGGTACGC TTTAATCGGG)
Stadt C: TATGACCTAG CTAGCATAGC (Komplement: ATACTGGATC GATCGTATCG)
```

Eine Straße von Stadt x nach Stadt y wird ebenfalls durch 20 Nukleotide dargestellt: Die letzten 10 von x und die ersten 10 von y. Eine Straße von `Stadt A` nach `Stadt B` sieht etwa so aus:

```
A-nach-B: ATGCTAGAAA AATCCATGCG
```

Ein solches Molekül kann sich mit dem Komplement von `Stadt B` verbinden wie folgt:

```
        A-nach-B: ATGCTAGAAA AATCCATGCG
                             ||||||||||
Komplement von B:            TTAGGTACGC TTTAATCGGG
```

Dieses Molekül kann sich mit einer Straße `B-nach-C` verbinden etc.

Antworten zu den Wiederholungsfragen, 9. SU, Mo 22.11.10

1. Machen Sie Objekte der Klasse `Mango`

```
 1 class Mango {
 2    int    saftmenge;
 3    char   guete;
 4    float  gewicht;
 5    ...
```

mit Hilfe der Schnittstelle `Comparator` (nicht mit der Schnittstelle `Comparable`!) vergleichbar. Dabei soll ein `Mango`-Objekt `m1` *größer* sein als ein `Mango`-Objekt `m2`, wenn die `safttmenge` von `m1` *größer* ist als die von `m2` (kurz: `Mango`-Ordnung nach `saftmenge`, aufsteigend).

```
 6 class NachSaftmengeAufsteigend implements Comparator<Mango> {
 7    public int compare(Mango m1, Mango m2) {
 8       if (m1.saftmenge < m2.saftmenge) return -1;
 9       if (m1.saftmenge > m2.saftmenge) return +1;
10       return 0;
11    }
12 }
```

2. Die Klasse `String` implementiert die Schnittstelle `Comparable<String>`. Wie könnte die Methode `compareTo` in der Klasse `String` aussehen? Geben Sie eine mögliche Implementierung an.

```
13    public int compareTo(String that) {
14       int min = Math.min(this.length(), that.length());
15       for (int i=0; i<min; i++) {
16          char c1 = this.charAt(i);
17          char c2 = that.charAt(i);
18          if (c1 != c2) return c1 - c2;
19       }
20       return this.length - that.length;
21    }
```

Anmerkung: Die `compareTo`-Methode in der Klasse `String` von Sun sieht ziemlich anders aus, weil sie aus Gründen der Effizienz noch eine Variable `offset` berücksichtigt.

3. Betrachten Sie noch einmal ein Objekt `ml` der Klasse `MeineListe`, nachdem drei bestimmte Strings eingefügt wurden (siehe oben Frage 3.)

Angenommen, jetzt wird die folgende Variablenvereinbarung ausgeführt:

```
22    Knoten vorHallo = ml.sucheVor("Hallo");
```

Mit welchem Wert wird die Variable `vorHallo` initialisiert?

Mit dem Wert <60>. Der Knoten, an dem "Hallo" hängt, hat die Referenz <50>. Die Referenz <60> zeigt auf den davorliegenden Knoten.

9. SU, Mo 22.11.10

A. Wiederholung
B. Organisation

Aufgabe C: (Stimmt überein mit der 1. Wiederholungsfrage) Ergänzen Sie die Vereinbarung der Klasse `Mango` (siehe Rückseite der WF, ab Zeile 23) nach Möglichkeit 2 (d.h. mit Hilfe der Schnittstelle `Comparator`). Dabei sollen `Mango`-Objekte *aufsteigend* nach ihrer `saftmenge` verglichen werden (je größer die `saftmenge`, desto größer das `Mango`-Objekt).

Lösung C:

```
23 class SandraB implements Comparator<Mango> {
24    public int compare(Mango a, Mango b) {
25       return a.saftmenge - b.saftmenge;
26    }
27 }
```

Diese Lösung ist naheliegend, aber *falsch*. Sie liefert in zahlreichen Fällen, in denen bei der Subtraktion ein Überlauf auftritt, falsche Ergebnisse, z.B.

a.saftmenge	b.saftmenge	Ergebnis von compare
+100 Million	-2.1 Milliarden	-2 094 967 296
+ 2 Milliarden	-200 Millionen	-2 094 967 296
-2.1 Milliarden	+100 Millionen	+2 094 967 296
-2.1 Milliarden	+1.2 Milliarden	+994 967 296

Zur Erinnerung: `Integer.MAX_VALUE` ist ungefähr gleich 2.14 Milliarden.

Leider kommt dieser Fehler mehrmals in meinem Buch vor (z.B. auf S. 453 oben). Ich habe einfach nicht an Mangos gedacht, die eine Saftmenge von z.B. -2 Milliarden haben. Ein Programmierer sollte aber insbesondere an "extreme Sonderfälle" denken.

Eine korrekte Lösung steht oben als Antwort auf die 1. Wiederholungsfrage.

Verkettete Listen

Wozu End-Dummyknoten? (Um das Suchen zu beschleunigen. Weil die gesuchten Nutzdaten am End-Dummyknoten hängen, findet man sie bestimmt und braucht pro Schritt nur eine Abfrage (statt 2).

Wozu Anfangs-Dummyknoten? Um das Entfernen des ersten Knotens zu vereinfachen.

Unsere Tabelle mit Schrittzahlen weiter ausfüllen (unsortierte/sortierte Listen)

Struktur der Sammlung	Operation	Anz. Schritte in einem schlimmsten Fall	Anz. Schritte im Durchschnitt
verkettete Liste (unsortiert)	Einfügen	1	1
	Suchen pos.	n	n/2
	Suchen neg.	n	n
	Entfernen	Suchen + 1	Suchen + 1
verkettete Liste (sortiert)	Einfügen	Suchen + 1	Suchen + 1
	Suchen pos.	n	n/2
	Suchen neg.	n	n
	Entfernen	Suchen + 1	Suchen + 1

Kurze Wiederholung und Besprechung der "großen Linie"

Sortierte Reihungen:
gut: Das Suchen geht sehr schnell
schlecht: Beim Einfügen muss man "viel verschieben"

Sortierte Listen:
gut Beim Einfügen muss man "nichts verschieben"
schlecht: Das Suchen dauert sehr lange

Kann man die Vorteile von *sortierten Reihungen* und *sortierten Listen* nicht *kombinieren*? Gesucht:

Datenstruktur X:
gut Das Suchen geht sehr schnell
gut Beim Einfügen muss man "nichts verschieben"

Die Datenstruktur X heißt: Baum (oder etwas spezieller: binärer Baum)

Erinnern Sie sich an den 3. SU (Mo 11.10.10)? :-)

Das Arbeitsblatt (ein Blatt, zwei Seiten) mit der Übung-04 (und der Klasse `MeinBaum`) darauf austeilen und besprechen.

Def.: Binärer Baum

Der Zusammenhang zwischen der Anzahl der Knoten und der Tiefe

Den Begriff "Tiefe eines Baumes" an Hand eines Beispiels erklären.

Tabelle 1:

Tiefe	1	2	3	4	5	...	n
max. Knotenzahl	1	3	7	15	31	...	$2^n - 1$

Tabelle 2:

Knotenzahl	1 Tausend	1 Million	1 Milliarde	1 Billion	1 Billiarde	...	n
min. Tiefe	10	20	30	40	50	...	log(n)

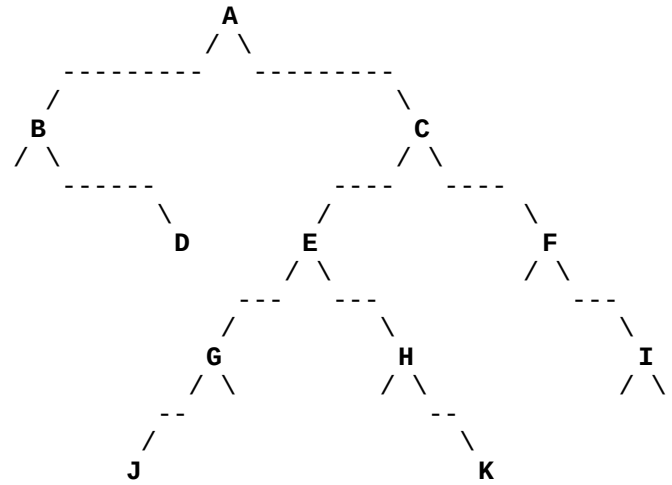
Die Fachbegriffe Mutter-, Tochter-, Wurzel-, Blatt-, innerer Knoten.

Zur Entspannung: Algorithmische Probleme mit DNS-Molekülen lösen (statt mit Computern)
Siehe Rückseite der Blattes mit den Wiederholungsfragen.

Wiederholungsfragen, 10. SU, Mo 29.11.10

1. Was ist besonders gut an sortierten Reihungen?
2. Was ist schlecht an Reihungen und insbesondere an sortierten Reihungen?
3. Was ist gut an verketteten Listen?
4. Was ist schlecht an verketteten Listen?
5. Wie viele Knoten kann ein binärer Baum der Tiefe 15 höchsten enthalten (ungefähr)?
6. Welche Tiefe muss ein binärer Baum, der 64 Millionen Knoten enthält, mindestens haben (genau)?
7. Betrachten Sie den folgenden binären Baum:

Betrachten Sie den folgenden binären Baum:



8.1. Vorgänger (-Knoten) von E?
8.2. Nachfolger (-Knoten) von C?
8.3. Vorgänger (-Knoten) von A?
8.4. Nachfolger (-Knoten) von G?
8.5. Wurzel (-Knoten) des Baumes?
8.6. Wurzel (-Knoten) des linken Unterbaums von A?
8.7. Wurzel (-Knoten) der rechten Unterbaums von G?
8.8. Alle Blätter (oder: Blatt-Knoten)?
8.9. Alle inneren Knoten?

Antworten zu den Wiederholungsfragen, Mo 10. SU, 29.11.10

1. Was ist besonders gut an sortierten Reihungen?

Das Suchen geht schnell.

2. Was ist schlecht an Reihungen und insbesondere an sortierten Reihungen?

Feste Länge ("Beton"). Beim Einfügen und Löschen muss viel verschoben werden.

3. Was ist gut an verketteten Listen?

Beim Einfügen und Löschen muss nichts verschoben werden.

4. Was ist schlecht an verketteten Listen?

Das Suchen ist langsam.

5. Wie viele Knoten kann ein binärer Baum der Tiefe 15 höchsten enthalten (ungefähr)?

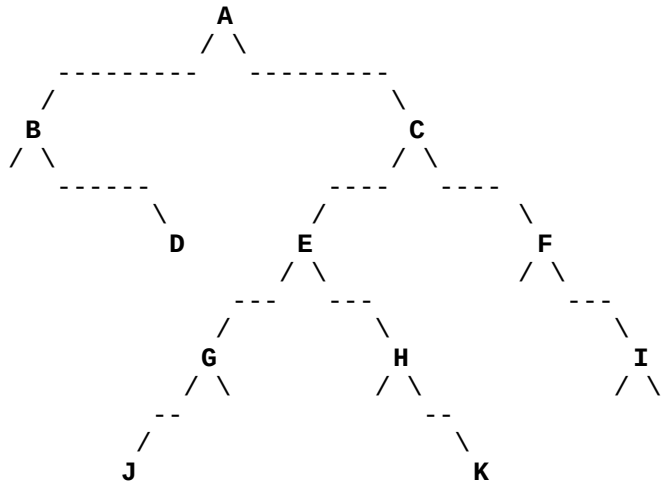
32 Tausend

6. Welche Tiefe muss ein binärer Baum, der 64 Millionen Knoten enthält, mindestens haben (genau)?

Die Tiefe 26

7. Betrachten Sie den folgenden binären Baum:

Betrachten Sie den folgenden binären Baum:



8.1. Vorgänger (-Knoten) von E?	C
8.2. Nachfolger (-Knoten) von C?	E, F
8.3. Vorgänger (-Knoten) von A?	A hat keinen Vorgänger
8.4. Nachfolger (-Knoten) von G?	J
8.5. Wurzel (-Knoten) des Baumes?	A
8.6. Wurzel (-Knoten) des linken Unterbaums von A?	B
8.7. Wurzel (-Knoten) der rechten Unterbaums von G?	G hat einen leeren rechten Unterbaum
8.8. Alle Blätter (oder: Blatt-Knoten)?	D, I, J, K
8.9. Alle inneren Knoten?	B, C, E, F, G, H

10. SU, Mo 29.11.10

A. Wiederholung
B. Organisation

Binäre Bäume, Fortsetzung

Wenn man Bäume als Sammlungen verwendet, müssen sie *sortiert* sein (es gibt keine unsortieren Sammlungen, die als Baum strukturiert sind).

Damit ein Baum sortiert sein kann, müssen seine Knoten *vergleichbar* sein (in Java: Über die Schnittstelle `Comparable` oder `Comparator`).

Def.: Ein Baum ist sortiert wenn ... (siehe Übung-04, wurde im 9. SU ausgeteilt)

Üb04-1: Knoten in einen sortierten Baum einfügen

Anmerkung zum Thema Parameterübergabe:

Wenn man in Java eine Methode aufruft und eine Variable als Parameter angibt, bekommt die Methode immer eine Kopie des *Wertes* der Variablen. Es ist (in Java) nicht möglich, einer Methode eine Kopie der *Referenz* einer Variablen zu übergeben. Deshalb kann eine Methode, der man eine Variable übergibt, den *Wert* der (originalen) Variablen nicht verändern (höchstens die Kopie des Wertes oder den Zielwert der Variablen, falls er vorhanden und veränderbar ist). Berühmtes Beispiel:

In Java kann man keine `swap`-Methode programmieren (die als Parameter zwei Variablen erwartet und deren Werte miteinander *vertauscht*).

In anderen Sprachen (z.B. C++, C#) kann man einer Methode wahlweise den *Wert* oder die *Referenz* einer Variablen übergeben (Parameterübergabe *per Wert* oder *per Referenz*, engl. *pass by value* or *pass by reference*. Die Bezeichnungen "call by value" und "call by reference" sind verbreitet aber schlechtes Englisch, weil Parameter ja gar nicht *aufgerufen* werden (parameters are not called), sondern *übergeben* werden (parameters are passed).

Weil es in Java keine Parameterübergabe *per Referenz* gibt, ist die Sprache deutlich einfacher, aber man kann z.B. binäre Bäume nicht so elegant programmieren wie in anderen Sprachen.

Eine einfache Implementierung von Bäumen ohne Parameterübergabe per Referenz:
Als *Reihung* von Knoten.

Bei dieser Implementierung sind die Knoten nicht durch *Referenzen*, sondern durch ihre *Indizes* verbunden (d.h. man kann aus dem Index eines Knotens ganz einfach die Indizes seiner beiden Nachfolgerknoten und den Index seines Vorgängerknotens berechnen).

Das Beispiel auf dem Blatt Übung-04, S.9 unten besprechen.

Die Methoden `fügeEin` (nicht rekursiv) und `fügeEinR` (rekursiv) besprechen.

Üb04-2: Wozu nicht-rekursive und rekursive Methoden
Üb04-3: Die Methode `istDrinR` programmieren (jeder für sich oder in kleinen Gruppen)
Üb04-4: Die Methode `bearbeitenR` programmieren (ebenso)

Zur Entspannung: Android und iOS, zwei sichere Betriebssysteme

Ältere Betriebssysteme wie Windows und Linux sind grundsätzlich *unsicher*, weil sie in C geschrieben sind und im Prinzip die Installation beliebiger Programme erlauben, auch solcher, die "sehr böse Befehle enthalten". Es ist sehr aufwendig, sich vor solchen Programmen zu schützen.

Android ist teilweise in C geschrieben, erlaubt aber grundsätzlich nur die Installation von Java-Programmen. Java enthält keine "bösen Befehle". Android ist quelloffen (open source).

iOS ist ein Unix-Nachfolger und in C geschrieben. IOS wird vollständig von der Firma Apple kontrolliert. Unter iOS laufen nur Programme, die von Apple "genehmigt" wurden. Apple hat ein starkes Interesse daran, nur "sichere Programme" zu genehmigen.

Wiederholungsfragen, 11. SU, Mo 06.12.10

1. In Java gibt es nur *eine* Art, wie Parameter an Methoden übergeben werden können, nämlich die Übergabe per Wert (engl. pass by value). In vielen anderen Sprachen wie z.B. Pascal, C# und C++ gibt es zusätzlich noch eine zweite Art. Wie heißt die auf Deutsch? Auf Englisch?

2. Betrachten Sie die folgenden Java-Befehle:

```
1    static void pln(Object ob) {
2       System.out.println(ob);
3    }
4
5    static public void main(String[] _) {
6       String str = new String("Hallo!");
7       pln(str);
8       ...
9    }
```

Angenommen, der Ausführer führt die `main`-Methode aus und ist gerade dabei, den Methodenaufruf in Zeile 7 auszuführen. Im Zuge dieser Ausführung ist er dabei, die Zeile 2 auszuführen. Wie sehen in diesem Moment die Variable `str` und der formale Parameter `ob` (als Bojen dargestellt) aus.
Zur Erinnerung: *Formale Parameter* (wie z.B. `ob`) sind auch Variablen, die bei jedem Aufruf der Methode neu erzeugt und mit dem entsprechenden *aktuellen Parameter* initialisiert werden, etwa so:
`Object ob = str;`

3. Wie würden die Bojen aussehen, wenn beim Methodenaufruf in Zeile 7 der aktuelle Parameter `str` *per Referenz* an die Methode `pln` übergeben würde?

4. Wenn man einen binären Baum als Reihung `rei` darstellt, welche Indizes haben dann die beiden *Nachfolger* des Knotens `rei[5]`? Und welchen Index hat der *Vorgänger* von `rei[5]`?

5. Angenommen, Sie haben den Ausführer die folgenden Befehle ausführen lassen (dabei ist `MeinBaum` die Klasse, die in der Übung-04 vereinbart wurde):

```
10    MeinBaum mb = new MeinBaum();
11    mb.fuegeEin("BBB");
12    mb.fuegeEin("CCC");
13    mb.fuegeEin("AAA");
14    mb.bearbeite();
```

In welcher Reihenfolge werden die Komponenten von `mb` durch Ihre Methode `bearbeite` (die Sie im vorigen SU bzw. zu Hause vereinbart haben) ausgegeben?

Rückseite der Wiederholungsfragen, 11. SU, Mo 06.12.10

Worte für große Zahlen (deutsche, klassisch-britische und US-amerikanische)

10 hoch	deutsch	british	USA
3	Tausend	thousand	thousand
6	Million	million	million
9	Milliarde	milliard	billion
12	Billion	billion	trillion
15	Billiarde	thousand billion	quadrillion
18	Trillion	trillion	quintillion
21	Trilliarde	thousand trillion	sextillion
24	Quadrillion	quadrillion	septillion
27	Quadrilliarde	thousand quadrillion	octillion
30	Quintillion	quintillion	nonillion
33	Qunitilliarde	thousand quintillion	decillion
36	Sextillion	sextillion	undecillion
39	Sextilliarde	thousand sextillion	duodecillion
42	Septillion	septillion	tredecillion
45	Septilliarde	thousand septillion	quattuordecillion
48	Oktillion	octillion	quindecillion
51	Oktilliarde	thousand octillion	sexdecillion
54	Nonillion	nonillion	septdecillion
57	Nonilliarde	thousand nonillion	octodecillion
60	Dezillion	decillion	novemdecillion
63	Dezilliarde	thousand decillion	vigintillion
66	Undezillion	undecillion	
69	Undezillarde	thousand undecillion	
72	Duodezillion	duodecillion	
75	Duodezillarde	thousand duodecillion	
78	Tredezillion	tredecillion	
81	Tredezilliarde	thousand tredecillion	
303			centillion
600	Centillion	centillion	

Quelle: Webster's New Encyclopedic Dictionary, Könemann-Verlag Köln, 1994

Siehe auch Wikipedia: Lange und kurze Leiter, échelle longue/courte,
Jacques Peletier du Mans ca. 1550.

Antworten zu den Wiederholungsfragen, 11. SU, Mo 06.12.10

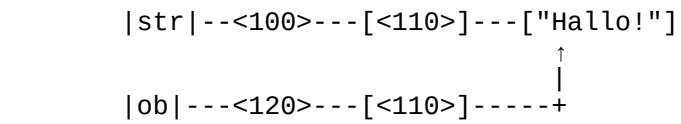
1. In Java gibt es nur *eine* Art, wie Parameter an Methoden übergeben werden können, nämlich die Übergabe per Wert (engl. pass by value). In vielen anderen Sprachen wie z.B. Pascal, C# und C++ gibt es zusätzlich noch eine zweite Art. Wie heißt die auf Deutsch? Auf Englisch?

Deutsch: Parameterübergabe per Referenz. Englisch: Pass by reference.

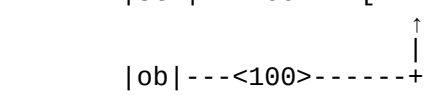
2. Betrachten Sie die folgenden Java-Befehle:

```
1    static void pln(Object ob) {
2       System.out.println(ob);
3    }
4
5    static public void main(String[] _) {
6       String str = new String("Hallo!");
7       pln(str);
8       ...
9    }
```

Angenommen, der Ausführer führt die `main`-Methode aus und ist gerade dabei, den Methodenaufruf in Zeile 7 auszuführen. Im Zuge dieser Ausführung ist er dabei, die Zeile 2 auszuführen. Wie sehen in diesem Moment die Variable `str` und der formale Parameter `ob` (als Bojen dargestellt) aus?
Zur Erinnerung: *Formale Parameter* (wie z.B. `ob`) sind auch Variablen, die bei jedem Aufruf der Methode neu erzeugt und mit dem entsprechenden *aktuellen Parameter* initialisiert werden, etwa so:
`Object ob = str;`



3. Wie würden die Bojen aussehen, wenn beim Methodenaufruf in Zeile 7 der aktuelle Parameter `str` *per Referenz* an die Methode `pln` übergeben würde?



4. Wenn man einen binären Baum als Reihung `rei` darstellt, welche Indizes haben dann die beiden *Nachfolger* des Knotens `rei[5]`? Und welchen Index hat der *Vorgänger* von `rei[5]`?

Nachfolger: 10 und 11. Vorgänger: 2

5. Angenommen, Sie haben den Ausführer die folgenden Befehle ausführen lassen (dabei ist `MeinBaum` die Klasse, die in der Übung-04 vereinbart wurde):

```
10    MeinBaum mb = new MeinBaum();
11    mb.fuegeEin("BBB");
12    mb.fuegeEin("CCC");
13    mb.fuegeEin("AAA");
14    mb.bearbeite();
```

In welcher Reihenfolge werden die Komponenten von `mb` durch Ihre Methode `bearbeite` (die Sie im vorigen SU bzw. zu Hause vereinbart haben) ausgegeben?

Vermutlich in folgender Reihenfolge: "AAA", "BBB", "CCC"
Vielleicht aber auch in einer anderen Reihenfolge (je nachdem, wie Sie die Methode bearbeite programmiert haben).

11. SU, Mo 06.12.10

A. Wiederholung
B. Organisation: Nächstes Mal brauchen wir das Buch (für printf)

Erstaunlicherwese gibt es eine Datenstruktur für Sammlungen, die in vielen Fällen noch schneller ist als Bäume: Hash-Tabellen. Bei einer gut funktionierenden Hash-Tabelle ist der Zeitbedarf der wichtigen Operationen unabhängig von der Größe der Sammlung (d.h. sie haben eine Zeitkomplexität von O(1)).

Das Papier Hash-Tabellen (1 Blatt, 4 Seiten) austeilen und besprechen.

Zur Entspannung: Namen für große Zahlen (siehe Rückseite der Wiederholungsfragen)

In meinem Duden (22. Auflage, aus dem Jahr 2000) habe ich nur Zahlen bis Septillion gefunden, aber schon die Quadrilliarde, Quintilliarde und Sextilliarde fehlten.

Abbildungen

Def.: Eine *Abbildung* (engl. map) ist eine Menge von *Einträgen* (engl. enbtries). Jeder Eintrag besteht aus einem *Schlüssel* und einem *Wert* (a key and a value).

Die Schlüssel müssen eindeutig sein (d.h. ein bestimmter Schlüssel s darf in höchstens einem Eintrag vorkommen). Ein Wert darf in mehreren Einträgen vorkommen.

Beispiel-01: Ein Eintrag besteht aus einem Namen (einer Person) als Schlüssel und einer Telefon-Nr als Wert.

Beispiel-02: Ein Eintrag besteht aus einer Telefon-Nr als Schlüssel und einem Namen als Wert

Frage: Wann ist es problematisch, aus einer Abbildung entsprechend dem Beispiel-01 in eine Abbildung entsprechend dem Beispiel-02 zu erzeugen? (Wenn mehreren Namen dieselbe Telefon-Nr zugeordnet ist).

Def.: In Java ist eine Abbildung ein Objekt einer `Map`-Klasse.

Dabei ist `Map` eine Schnittstelle, die 14 Methoden enthält, darunter die folgenden:

```
1 interface Map<S, W> {
2    W             put(S      schluessel, W wert);
3    W             get(Object schluessel);
4    Set<S>        keySet();
5    Collection<W> values();
6    ...
7 }
```

Dabei ist `Set<K>` eine Erweiterung der Schnittstelle `Collection<K>` um 0 Methoden und die weiche Forderung: Sammlungen eines `Set<K>`-Typs sollten *keine Doppelgänger* erlauben.

Doppelgänger sind zwei Objekte `ob1` und `ob2` für die `ob1.equals(ob2)` gleich `true` ist.

Beispiele für `Map`-Klassen in der Standardbibliothek:

`HashMap`
`ConcurrentHashMap` (ist *fadensicher*, engl. thread safe)
`WeakHashMap` (wenn ein Schlüssel nur noch in einer solchen Abbildung steht, kann er von der Speicherbereinigung zusammen mit dem betreffenden Eintrag gelöscht werden).

Wiederholungsfragen, 12. SU, Mo 13.12.10

1. Was ist eine *Hash-Tabelle*? Geben Sie möglichst die im letzten SU behandelte Kurzdefinition an.

2. Angenommen wir haben eine Hash-Tabelle `ht` und eine Hash-Funktion `hf`. Was liefert einem die Funktion `hf`, wenn man sie aufruft?

3. Die Schnittstelle `Set<K>` erweitert die Schnittstelle `Collection<K>`. Um *wie viele* Methoden wird `Collection<K>` von `Set<K>` erweitert?

4. Welche "*weiche Bedingung*" müssen `Set`-Sammlungen (z.B. `HashSet`, `TreeSet` etc.) einhalten?

5. Eine *Abbildung* besteht aus einer Menge von *Einträgen* (engl. entries). Woraus besteht ein solcher Eintrag?

6. Wie viele *generische Parameter* (Typ-Parameter) hat die Schnittstelle `Map`?

7. Vereinbaren Sie eine Abbildung vom Typ `HashMap`, in der den Namen `"MAX"`, `"MIN"`, und `"NAN"` entsprechende `Double`-Objekte zugeordnet sind.

Hinweis: In der Klasse `Double` gibt es `double`-Klassenkonstanten namens `MAX_VALUE`, `MIN_VALUE` und `NaN`.

Rückseite der Wiederholungsfragen, 12. SU, Mo 13.12.10

Der format- bzw. printf-Befehl (mit variabel vielen Parametern)

1. Aufruf:
```
System.out.printf       (P0, P1, P2, ...);
String s = String.format(P0, P1, P2, ...);
```

`P0` muss ein `String` sein. Der kann *Umwandlungsbefehle* enthalten.

2. Struktur eines Umwandlungsbefehls: %[Index][Schalter][Breite][.Genauigkeit]UBuchstabe

3. Index: `1$` oder `2$` oder `3$` oder ... (bezeichnet P1 oder P2 oder P3 oder ...)

4. Schalter:

`+`	(Vorzeichen + bzw. -)	
`' '`	(Vorzeichen ' ' bzw. -)	
`(`	(negative Zahlen in runden Klammern)	
`-`	(linksbündig statt rechtsbündig)	// Nur zusammen mit einer Breite
`0`	(zu kurze Operanden mit 0 auffüllen statt mit ' ')	// Nur zusammen mit einer Breite
`,`	(Ziffern vor dem Punkt werden gruppiert)	
`#`	(besondere Behandlung, abhängig vom UBuchstaben)	

5. Breite: z.B. `3` oder `15` oder `37` oder ... (ist immer eine *Mindest*breite, es wird nichts abgeschnitten)

6. Genauigkeit: z.B. `.0` oder `.3` oder `.15` oder ...
(Bei Bruchzahlen: Ziffern nach dem Punkt, sonst: Wie viele Zeichen des Rohergebnisses nehmen?)

7. UBuchstabe: Sei P der umzuwandelnde Parmeter

Für alle Typen:

`b, B`	P gleich `null` bzw. `false` wird zu `"false"`, andere Werte zu `"true"`.
`h, H`	P wird in seinen Hash-Code in hex-Darstellung umgewandelt (z.B. `"FB3868BF"`)
`s, S`	P wird in einen String umgewandelt (falls vorhanden mit `formatTo`, sonst mit `toString`).

Für Zeichen-Typen: `byte`, `Byte`, `char`, `Character`, `short`, `Short`, `int`, `Integer`

`c, C`	P wird in ein Unicode-Zeichen umgewandelt

Für Ganzzahl-Typen: `byte`, `Byte`, `short`, `Short`, `int`, `Integer`, `long`, `Long`, `BigInteger`

`d`	P wird in eine 10-er-Zahl umgewandelt
`o`	P wird in eine 8-er-Zahl umgewandelt
`x, X`	P wird in eine 16-er-Zahl umgewandelt

Für Bruchzahl-Typen: Bruchzahl-Typen: `float`, `Float`, `double`, `Double`, `BigDecimal`

`e, E`	P wird in einen Dezimalbruch-String *mit Exponent* umgewandelt (z.B. `"1.2345e+02"`)
`f`	P wird in einen Dezimalbruch-String *ohne Exponent* umgewandelt (z.B. `"123.45"`)
`g, G`	P zwischen 10^{-3} und 10^{7} wie `f`, sonst wie `e, E`

Für Datum-und-Zeit-Typen: `Calendar`, `long`, `Long`

`t, T` Nach dem `t` bzw. `T` muss einer der Buchstaben
`H, I, k, l, M, S, L, N, p, z, Z, s, Q` (für Zeitangaben) oder
`B, b, H, A, a, C, Y, y, j, m, d, e` (für Datumsangaben) oder
`R, T, r, D, E, F, c` (für Kombinationen Zeit-plus-Datum)
stehen.

Anmerkungen:
1. Der Befehl `System.out.printf` erzeugt einen String und gibt ihn *aus*.
Der Befehl `String.format` erzeugt einen String und gibt ihn *zurück* (mit `return`).
2. Jede Klasse kann die Schnittstelle `Formattable` (mit nur einer Methode `formatTo` darin) implementieren. Dann wird anstelle von `toString` die Methode `formatTo` genommen.
3. Der Schalter # ist nur bei alle Typen, Ganzzahl-Typen, Bruchzahl-Typen erlaubt, die übrigen fünf Schalter (`+ ' ' ( - 0`) sind nur bei Ganzzahl-Typen und Bruchzahl-Typen erlaubt.

Antworten zu den Wiederholungsfragen, 12. SU, Mo 13.12.10

1. Was ist eine Hash-Tabelle? Geben Sie möglichst die im letzten SU behandelte Kurzdefinition an.

Eine Hash-Tabelle ist eine Reihung von Listen.

2. Angenommen wir haben eine Hash-Tabelle `ht` und eine Hash-Funktion `hf`. Was liefert einem die Funktion `hf`, wenn man sie aufruft?

Einen Index für die Reihung ht.

3. Die Schnittstelle `Set<K>` erweitert die Schnittstelle `Collection<K>`. Um wie viele Methoden wird `Collection<K>` von `Set<K>` erweitert?

Um 0 Methoden.

4. Welche "weiche Bedingung" müssen `Set`-Sammlungen (z.B. `HashSet`, `TreeSet` etc.) einhalten?

Sie dürfen keine "Doppelgänger" zulassen (d.h. es darf nicht möglich sein, zwei Objekte a und b einzufügen, für a.equals(b) den Wert true liefert).

5. Eine Abbildung besteht aus einer Menge von Einträgen (engl. entries). Woraus besteht ein solcher Eintrag?

Aus einem Schlüssel (engl. key) und einem Wert (engl. value).

6. Wie viele generische Parameter (Typ-Parameter) hat die Schnittstelle `Map`?

Zwei (den Typ der Schlüssel und den Typ der Werte)

7. Vereinbaren Sie eine Abbildung namens `hm` vom Typ `HashMap`, in der
den Namen `"MAX"`, `"MIN"`, und `"NAN"` entsprechende `Double`-Objekte zugeordnet sind.

Hinweis: In der Klasse `Double` gibt es `double`-Klassenkonstanten namens
`MAX_VALUE`, `MIN_VALUE` und `NaN`.

```
1       HashMap<String, Double> hm = new HashMap<String, Double>();
2       hm.put("MIN", Double.MIN_VALUE);
3       hm.put("MAX", Double.MAX_VALUE);
4       hm.put("NAN", Double.NaN);
```

12. SU, Mo 13.12.10

A. Wiederholung
B. Organisation

Binäre Bäume, Fortsetzung und Schluss

Was ist gut an (sortierten) binären Bäumen? (Das Suchen geht so schnell wie bei sortierten Reihungen, das Einfügen erfordert kein Verschieben vom Komponenten, Bäume sind wie Listen aus Gummi und nicht aus Beton).

Diese sehr positiven Eigenschaften haben Bäume aber nicht immer und automatisch, sondern nur, wenn sie (zumindest ungefähr) *balanciert* sind.

Wege von der Wurzel zu einem Blatt: Unter der Länge eines solchen Weges versteht man die Anzahl der Knoten, an denen man vorbeikommt (wobei der Wurzelknoten nicht gezählt wird).

Def.: Ein binärer Baum ist perfekt *balanciert*, wenn alle Wege von der Wurzel zu einem Blatt fast gleich lang sind (und sich um höchstens 1 unterscheiden).

Def.: Ein binärer Baum ist *ungefähr balanciert*, wenn der längste Weg von der Wurzel zu einem Blatt höchstens *doppelt so lang* ist wie der kürzeste.

Beispiel: Wie viele Suchschritte braucht man im schlimmsten Fall in einem *perfekt balancierten* Baum mit 1 Million Knoten? (20) Und einem *ungefähr balancierten* Baum? (40).

Aufgabe: Fügen Sie die folgenden Knoten in der angegebenen Reihenfolge in einen anfangs leeren (sortierten) binären Baum ein: 17, 30, 40, 35, 39, 38, 37

Welche Tiefe hat der Baum am Ende? (7)

Er ist also *maximal unbalanciert* und nicht besser als eine verkettete Liste.

Es gibt ein ganze Reihe von Verfahren, um Bäume beim Einfügen und Löschen *balanciert* zu halten: *Rot-Schwarz-Bäume* (weil man rote und schwarze Knoten unterscheidet), *AVL-Bäume* (nach den Erfindern Adelson-Welski und Landis benannt), *2-3-4-Bäume* (weil die Knoten 2, 3 oder 4 Nachfolger haben können).

Eine mit Bäumen verwandte Datenstruktur sind die *Skip-Listen* (erfunden 1990 von William Pugh), die ähnliche Eigenschaften wie balancierte Bäume haben, aber manchmal etwas günstiger sein sollen. In Java sind sie implementiert als Klasse `ConcurrentSkipListMap<K,V>`.

Fazit: Bäume sind eine sehr effiziente Datenstruktur, aber nur wenn man beim Einfügen und Löschen ein paar zusätzliche Tricks anwendet.

Zur Entspannung: Noch ein unlösbares Problem

Eine D-Gleichung hat folgende Form:

`Polynom(x1, x2, ..., xn) = 0  // n ∈ {1, 2, 3, ...}`

Eine Lösung für eine solche D-Gleichung besteht aus n *ganzen* Zahlen `(z1, z2, ..., zn)`, auf die die Gleichung zutrifft.
Beispiele: (statt `x1, x2, ...` benutzen wir hier `x, y, ...` als Variablen)

```
 5   x² + 2y²      = 0  // Genau eine Lösung: x=0, y=0
 6   x² - 4        = 0  // Genau zwei Lösungen: x=2 und x=-2
 7   x¹ + 5y¹ - 8  = 0  // unendlich viele Lösungen, z.B. x=3, y=1
 8   x² - y² - z²  = 0  // unendlich viele Lösungen, z.B. x=-5, y=4, z=3
 9   x² + 5        = 0  // keine Lösung (leicht zu sehen)
10   x⁴ - y⁴ - z⁴  = 0  // keine Lösungen (schwer zu beweisen)
```

Satz: Es gibt keinen Algorithmus, der von jeder D-Gleichung korrekt feststellen kann, ob sie lösbar ist (d.h. mindestens eine Lösung hat) oder nicht.

"D-Gleichungen" werden üblicherweise als "diophantische Gleichungen" bezeichnet
(nach dem griechischen Mathematiker Diophant von Alexandrien, der irgendwann zwischen
100 v.Chr. und 350 n.Chr., vermutlich um 250 n.Chr., lebte und 13 Bücher über Arithmetik veröffentlichte, von denen 10 bis heute erhalten sind).

Die Befehle format bzw. printf

Wenn man Daten ausgibt, will man sie vorher häufig noch *formatieren*.

Typische Beispiele:

1. Um Zahlen *stellengerecht untereinander zu schreiben*, möchte man sie unabhängig von ihrer Größe in einer bestimmten *Breite* ausgeben. Zahlen die kürzer sind als diese Breite sollen meistens *rechtsbündig* in "ihrem Raum" erscheinen, z.B. so:

```
   37
 5274
    3
29384
```

Breite: 5 Zeichen

2. Manchmal will man, dass "zu kurze Zahlen" vorn mit Nullen (und nicht mit Blanks) verlängert werden, z.B. so:

```
00037
05274
00003
29384
```

3. Positve und negative Zahlen will man auf eine von vielen Weisen unterscheiden, z.B.

	Positive Zahlen	Negative Zahlen	Kommentar
Kennzeichnung 1	nix	Vorzeichen '-'	minimalistisch, asymmetrisch
Kennzeichnung 2	Vorzeichen ' '	Vorzeichen '-'	plus n und minus n sind gleich lang
Kennzeichnung 3	Vorzeichen '+'	Vorzeichen '-'	symmetrisch
Kennzeichnung 4	nix	in runden Klammern	ein exotisches Behördenformat
Kennzeichnung 5	...	...	...

4. Lange Zahlen werden lesbarer, wenn die Ziffern durch ein *Trennzeichen*
z.B. in *Dreiergruppen* einteilt, z.B. so:

```
         10.000
      1.000.000
100.000.000.000
```

5. Auch Texte will man manchmal unabhängig von ihrer tatsächlichen Länge in einer bestimmten *Breite* ausgeben. Dabei sollen Texte, die kürzer sind als diese Breite, häufig *linksbündig* in "ihrem Raum" erscheinen, z.B. so:

```
String        v1
StringBuilder v2
Integer       v3
```

Breite der Typnamen: 14 Zeichen

5. Für Datums-und-Zeit-Angaben gibt es besonders viele verschiedene Formate:
Nur Datum, nur Uhrzeit, Datum und Uhrzeit, beim Datum das Jahr vorn, das Jahr hinten, den Monat als Zahl, den Monat als ausgeschriebenen Namen (z.B. Dezember), den Monat abgekürzt (z.B. Dez) etc.

6. Einige dieser Formatierungen sind verschieden je nach Land oder Gegend.
Ein Programm internationalisieren (engl. to localize a program) heißt, es so schreiben, dass es sich automatisch den Bräuchen des Ortes anpaßt, an dem es gerade läuft.

Für solche Formatierungen und ähnliche wurden mit Java 5 die folgenden beiden Methoden eingeführt:

`format` (in den Klassen `Formatter`, `String`, `Console`, `PrintStream`, `PrintWriter`) und `printf` (in den Klassen `Console`, `PrintStream`, `PrintWriter`).

Vergleich zu C: Viele C-Programmierer wissen viel über den `printf`-Befehl in C. Der `printf`-Befehl in Java ist ein Versuch, möglichst viel von diesem Wissen *auszunutzen* und *wiederzuverwenden*, d.h. der Java-`printf`-Befehl hat viel Ähnlichkeit mit dem C-`printf`-Befehl. Andererseits gibt es auch erhebliche Unterschiede, vor allem: Der Java-`printf`-Befehl ist zwar nicht statisch typsicher (der Compiler kann nicht alle formalen Fehler finden), aber zumindest dynamisch typsicher (zur Laufzeit löst jeder formale Fehler eine Ausnahme aus). Ausserdem kann der Programmierer ihn erweitern (auf neue Typen ausdehnen). Der C-`printf`-Befehl ist dagegen unsicher und nicht erweiterbar.

Im Buch "C Interfaces and Implementations" von D. R. Hanson, Addison-Wesley 1997, wird gezeigt, wie man selbst in C einen besseren `printf`-Befehl programmieren kann. Der Java-`printf`-Befehl ist noch besser (sicherer), als der beste in C realisierbare Befehl.

Besonderheit: Die Methode `format` kann man mit *variabel vielen* Parametern aufrufen.

Die Vereinbarung der Methode `format` in der Klasse `String` beginnt etwa so:

```
static public String format(String fmt, Object... args) { ... }
```

Die drei Pünktchen `...` nach `Object` sind "Java-Syntax" und drücken aus, dass man nach dem `String`-Parameter beliebig viele (0 oder 1 oder 2 oder ...) `Object`-Parameter angeben darf.
Die drei Pünktchen `...` in den geschweiften Klammern sind keine "Java-Syntax" sondern eine *Auslassung* und bedeuten etwa "Hier kann irgendetwas stehen was gerade nicht wichtig ist".
Statt mehrere `Object`-Parameter darf man auch eine *Reihung* angeben (aber nicht beides gleichzeitig).

Aus der Vereinbarung der Methode `format` folgt: Der erste Param (vom Typ `String`) *muss* sein und heißt `Formatstring`. Die übrigen Parameter (von einem beliebigen Typ) *dürfen* sein und wir bezeichnen sie als *weitere Parameter*.

Ein Beispiel für einen `format`-Befehl:

```
1    int n1 = EM.liesInt();
2    int n2 = EM.liesInt();
3
4    String s = String.format("Betrag: %d Euro und %02d Cent%n", n1, n2);
```

Die Methode `format` wird hier mit 3 Parametern aufgerufen. Der Formatstring enthält *2 Umwandlungsbefehle*: `"%d"` und `"%02d"`. Deshalb mussten noch 2 *weitere Parameter* (`n1` und `n2`) angegeben werden (pro Umwandlungsbefehl ein weiterer Parameter).

Das Ergebnis des `format`-Befehls ist sein Formatstring, nachdem darin jeder Umwandlungsbefehl durch den zugehörigen, formatierten zusätzlichen Parameter ersetzt wurde. Das Ergebnis des Befehls in Zeile 4 kann z.B. so aussehen:

```
Betrag: 123 Euro und 03 Cent
```

Die Rückseite der Wiederholungsfragen ansehen und anhand von Beispielen besprechen.

Weitere Themen:

1. Die Pseudo-Umwandlungsbefehle %% und %n
2. Den Formatstring dynamisch erzeugen, z.B. so:

```
5    int    breite = EM.liesInt();
6    String fmt    = String.format("%%,%dd Euro%n", breite);
7    printf(fmt, 1234567);
```

Wenn der Benutzer z.B. `10` eingibt, wird der String `" 1.234.567 Euro"` ausgegeben.
Wenn der Benutzer z.B. `12` eingibt, wird der String `"   1.234.567 Euro"` ausgegeben.

Den `printf`-Befehl muss man aktiv lernen, z.B. in den Übungen und zu Hause.

Wiederholungsfragen, 13. SU, Mo 20.12.10

1. Wie viele aktuelle Parameter muss man in einem Aufruf der Methode `format` mindestens angeben?

2. Wie wird der erste Parameter in einem `format`-Aufruf gewöhnlich bezeichnet?

3. Welche der folgenden Worte bezeichnen (notwendige oder optionale) Teile eines Umwandlungsbefehls im Formatstring eines format-Befehls?

Länge, Fahrkarte, Schalter, Präzision, Breite, Höhe, Genauigkeit, UZiffer, VZiffer, UBuchstabe, VBuchstabe, Index, Schleife

4. Geben Sie die Zielwerte der folgenden `String`-Variablen an:

```
1
2    String s1 = String.format("Euro %5d", 123);
3    String s2 = String.format("Euro %5d", 1234567);
4    String s3 = String.format("Euro %+5d", -123);
5    String s4 = String.format("Euro %-5d", -123);
6    String s5 = String.format("Euro %(5d", -123);
7    String s6 = String.format("Euro %,d", 1234567);
8    String s7 = String.format("%%%%%%");
```

Rückseite der Wiederholungsfragen, 13. SU, Mo 20.12.10

Warum es unmöglich ist, das Halteproblem für KPS zu lösen

```
 1 /* ------------------------------------------------------------------------
 2 Eine KPS ist eine Klassen-Prozedur mit einem String-Parameter
 3 (a static void-method with one paramter of type String).
 4
 5 Die Funktion loesungDHP ist ein Versuch, das Halte-Problem fuer KPSs
 6 zu loesen. Bevor wir die Funktion voll ausprogrammieren, versuchen wir
 7 erstmal, nur fuer einen bestimmten Aufruf den richtigen Rueckgabewert
 8 herauszufinden, naemlich fuer den Aufruf loesungDHP(gegenTeil, gegenTeil).
 9 Soll die Funktion loesungDHP in diesem Fall das Ergebnis false
10 (Alternative "haelt nicht") oder true (Alternative "haelt") liefern?
11 ------------------------------------------------------------------------ */
12 class Halteproblem {
13    // ---------------------------------------------------------------------
14    static boolean loesungHP(final String PROC, final String PARAM) {
15       // Liefert true, wenn PROC der Quellcode einer KPS ist, die
16       // angewendet auf den String PARAM nach endlich vielen
17       // Ausfuehrungsschritten haelt. Liefert sonst false.
18       // Version 0.1: Noch nicht ganz vollstaendig, es wird erstmal
19       // nur ein einziger Spezialfall behandelt.
20
21       if (PROC.equals(gegenTeil) && PARAM.equals(gegenTeil)) {
22          return false;   // Alternative "haelt nicht"
23 //       return true;    // Alternative "haelt"
24       } else {
25          throw new Error("Dieser Fall ist noch nicht programmiert!");
26       }
27    }
28    // ---------------------------------------------------------------------
29    // Der Quellcode der KPS gegenTeil:
30    static void gegenTeil(String s) {
31       if (loesungHP(s, s)) {
32          while (true) System.out.print('.');
33       } else {
34          return;
35       }
36    }
37    // ---------------------------------------------------------------------
38    // Der Quellcode der KPS gegenTeil als String:
39    static String gegenTeil =
40   "static void gegenTeil(String s) {          " +
41   "   if (loesungHP(s, s)) {                  " +
42   "      while (true) {System.out.print('.'); " +
43   "   } else {                                " +
44   "      return;                              " +
45   "   }                                       " +
46   "}                                          ";
47    // ---------------------------------------------------------------------
48    static public void main(String[] _) {
49       pln("Halteproblem: Jetzt geht es los!");
50       pln("-------------------------------");
51       pln("gegenTeil(gegenTeil) haelt an? " + loesungHP(gegenTeil, gegenTeil));
52       pln("gegenTeil(gegenTeil) wird aufgerufen!");
53       gegenTeil(gegenTeil);
54       pln("gegenTeil(gegenTeil) hat angehalten!");
55       pln("-------------------------------");
56       pln("Halteproblem: Das war's erstmal!");
57    } // main
58    // ---------------------------------------------------------------------
59    // Eine Methode mit einem kurzen Namen:
60    static void pln(Object ob) {System.out.println(ob);}
61    // ---------------------------------------------------------------------
62 } // class Halteproblem
```

Antworten zu den Wiederholungsfragen, 13. SU, Mo 20.12.10

1. Wie viele aktuelle Parameter muss man in einem Aufruf der Methode `format` mindestens angeben?

Einen Parameter

2. Wie wird der erste Parameter in einem `format`-Aufruf gewöhnlich bezeichnet?

Formatstring

3. Welche der folgenden Worte bezeichnen (notwendige oder optionale) Teile eines Umwandlungsbefehls im Formatstring eines format-Befehls?

Länge, Fahrkarte, Schalter, Präzision, Breite, Höhe, Genauigkeit, UZiffer, VZiffer, UBuchstabe, VBuchstabe, Index, Schleife

4. Geben Sie die Zielwerte der folgenden `String`-Variablen an:

```
1                                                       // 123456789012345
2    String s1 = String.format("Euro %5d", 123);        // Euro   123
3    String s2 = String.format("Euro %5d", 1234567);    // Euro 1234567
4    String s3 = String.format("Euro %+5d", -123);      // Euro  -123
5    String s4 = String.format("Euro %-5d", -123);      // Euro -123
6    String s5 = String.format("Euro %(5d", -123);      // Euro (123)
7    String s6 = String.format("Euro %,d", 1234567);    // Euro 1.234.567
8    String s7 = String.format("%%%%%%");               // %%%
9                                                       // 1234567890
```

13. SU, Mo 20.12.10

A. Wiederholung
B. Organisation: Der ursprünglich für Mo 27.12.2010 angekündigte Test T8: Formatierung mit printf wurde auf allgemeinen Wunsch auf Mo 03.01.2011 verlegt. An den Terminen der restlichen Tests hat sich (erstmal) nichts geändert.

Die Befehle format und printf, Ergänzung und Schluss

Mit Indizes arbeiten: Innerhalb eines Umwandlungsbefehls in einem Formatstring bezeichnet
`3$` den dritten weiteren Parameter (`1$`, `2$`, ... entsprechend)
`<` den Parameter, der unmittelbar zuvor umgewandelt wurde

Beispiele:

```
1  printf("A F%s%s%<s, f%1$s%2$s%<s!%n", "latter", "ta");
2  printf("B |%s%s|%1$s%s|%1$s%s|%n", "auf", "geben", "rufen", "sagen");
```

Ausgaben:

```
A Flattertata, flattertata!
B |aufgeben|aufrufen|aufsagen|
```

Um die Befehle `format` und `printf` effizient zu benutzen, muss man eine Menge "fitzeliger Einzelheiten" wissen. Die kann man vermutlich nur "selbst und aktiv" lernen, nicht durch Anhören eines Vortrags.

Abschlußbermerkung:

Die Befehle `format` und `printf` leisten nichts, was man nicht auch mit anderen Java-Befehlen erreichen könnte, aber sie stellen eine viel kompaktere und (nach einiger Gewöhnung) leichter lesbare Notation für die Lösung bestimmter Probleme. Man kann diese Befehle deshalb als eine *Anwendungsspezifische Sprache* (ASS, engl. *domain specific language*, DSL) verstehen.
ASSe sind auf dem Gebiet der Softwareentwicklung zur Zeit ein viel diskutiertes Thema, zu dem es auch schon ein paar Werkzeuge gibt, z.B. das Xtext-Plugin für Eclipse.

Neues Thema: XML

Allgemeines Problem: Viele Programme geben Daten aus, die von anderen Programmen und von Menschen gelesen (und verstanden) werden sollen. Wie soll man solche Daten formatieren?

Konkretes Beispiel: Viele Programme geben Rechnungen aus. Wie kann man es erreichen, dass andere Programme solche Rechnungen einlesen und z.B. herausfinden können:

Von wem wurde die Rechnung ausgestellt?
Wann wurde sie ausgestellt?
Für wen wurde sie ausgestellt?
Wo stehen die Beschreibungen der einzelnen Rechnungsposten?
Wo stehen die Preise der einzelnen Rechnungsposten?
Wo steht die Endsumme?
...

Allgemeiner Lösungsansatz:
Man erzeugt per Programm Dokumente in einer sog. *Auszeichnungssprache*.

Ein solches Dokument enthält nicht nur den Text, um den es geht, sondern auch sog. *Auszeichnungen*, die beschreiben, welches Textstück welche Art von Information enthält. Diese Auszeichnungen sollen es z.B. einem einlesenden Programm ermöglichen, in einem Rechnungsdokument z.B. den Namen des Ausstellers oder die Endsumme zu finden.

Kennt jemand eine (möglichst weit verbreitete) Auszeichnungssprache? (Hoffentlich nennt jemand HTML)

Was für Dokumente kann man in HTML (besonders gut) beschreiben? (Seiten im Internet)

Was für Auszeichnungen gibt es in HTML? (`H1`, ..., `H6`, `TABLE`, `TD`, ...)

Zur Beschreibung von Rechnungen ist HTML kaum geeignet, weil es keine Auszeichnungen gibt, mit denen man den Namen des Ausstellers oder die Endsumme auszeichnen könnte.

Es gibt sehr viele und sehr unterschiedliche Auszeichnungssprachen. Wir wollen uns erstmal ein paar davon kurz ansehen:

Das Papier XML03.pdf austeilen und besprechen

1. CSV (Character Separated Values)

Sehr verbreitet, sehr simpel, sehr fehleranfällig.

2. Curl

Curl ist eine Auszeichnungs-, eine Skript- und eine Programmier-Sprache in einem.
Als Auszeichnungssprache ist sie besonders einfach (engl. lightweight).
Curl ist homoikonisch, d.h. jedes Curl-Programm ist auch ein Wert eines bestimmten Curl-Datentyps.
Curl gehört der japanischen Firma *Sumisho Computer Systems* und wird von der Tochterfirma *Curl* in *Cambridge, Massachusetts* weiterentwickelt und verkauft.

3. YAML (rhymes with camel)

Legt Wert darauf, eine Daten-orientierte Sprache zu sein, und nicht eine Dokumenten-orientierte

Unterschied:

Eine Dokumenten-orientierte Auszeichnungssprache ist für Dokumente gedacht, die (etwa vereinfacht gesagt) aus viel Text und wenig Auszeichnungen bestehen.

Eine Daten-orientierte Auszeichnungssprache ist für "voll strukturierte Daten" gedacht, bei denen freier Text (ohne Auszeichnungen) keine oder eine sehr geringe Rolle spielt.

Zur Entspannung: Eine Demonstration der Unlösbarkeit des Halteproblems

Wiederholungsfragen, 14. SU, Mo 03.01.2011

1. Was geben die folgenden `printf`-Befehle aus?

```
1    printf("A %d %<x %<o A%n", 16);

2    printf("B %3$d %2$d %1$d B%n", 10, 20, 30);

3    printf("C %d %1$x %1$o %d %2$x %2$o C%n", 16, 32);

4    printf("D %d %<x %<o %d %<x %<o D%n", 16, 32);
```

2. Mit welcher *Abkürzung* (3 Großbuchstaben) wird die einfachste Auszeichnungssprache bezeichnet, die wir besprochen haben? Für welche (drei englischen) Worte steht die Abkürzung? Woraus bestehen bei dieser Sprache die *Auszeichnungen*?

3. Wir haben auch (kurz) die Sprache *Curl* besprochen. Woran soll der Name erinnern? Wie sieht eine *Auszeichnung* in dieser Sprache aus?

4. Dann haben wir noch eine Auszeichnungssprache besprochen, deren Name sich auf (den englischen Tiernamen) "camel" reimt. Wie heißt diese Sprache? Auf welchen drei grundlegenden *Datenstrukturen* beruht diese Sprache?

5. Woraus besteht eine *Abbildung* (engl. map, Java-Schnittstelle: `Map`) in Java, C#, Perl, Python, Ruby, ... und in der Auszeichnungssprache Yamel?

Rückseite der Wiederholungsfragen, 14. SU, Mo 03.01.2011

Der Anfangskommentar der Java-Quelldatei Datei `XmlJStd`.java
(zu finden im Archiv `DateienFuerPr2/XML`):

```
 1 // Datei XmlJStd.java
 2 /* ----------------------------------------------------------------------
 3 Dieser Modul enthaelt Methoden zum Bearbeiten von XML-Dateien und
 4 benutzt dazu nur die Standardbibliothek von Java 6.
 5
 6 Die wichtigen Methoden:
 7 lies              Liest eine XML-Datei und erzeugt daraus ein
 8                   entsprechendes DOM-Document-Objekt
 9                   (mit DOMImplementationLS)
10 lies2             Liest eine XML-Datei und erzeugt daraus ein
11                   entsprechendes DOM-Document-Objekt
12                   (mit DocumentBuilderFactory)
13 liesVD            Liest eine XML-Datei und erzeugt daraus ein
14                   entsprechendes DOM-Document-Objekt.
15                   Validiert die XML-Datei dabei gegen ihre DTD.
16 liesVS            Liest eine XML-Datei und erzeugt daraus ein
17                   entsprechendes DOM-Document-Objekt.
18                   Validiert die XML-Datei dabei gegen eine
19                   XML-Schema-Datei.
20 istGueltigVD      Prueft, ob ein DOM-Document-Objekt gueltig ist
21                   gegen seine DTD.
22 istGueltigVS      Prueft, ob ein DOM-Document-Objekt gueltig ist
23                   gegen eine XML-Schema-Datei.
24 schreib           Schreibt ein DOM-Document-Objekt im XML-Format
25                   in eine Datei
26
27 Weniger wichtige Methoden:
28 gibTextDateiAus   Gibt eine beliebige Textdatei zur Standardausgabe aus.
29 gibLesbarAus      Gibt ein DOM-Document-Objekt in einer selbstgestrickten
30                   Form ("als Einrueck-Baum") zur Standardausgabe aus
31 ----------------------------------------------------------------------
32 ...
33 ---------------------------------------------------------------------- */
```

Antworten zu den Wiederholungsfragen, 14. SU, Mo 03.01.2011

1. Was geben die folgenden `printf`-Befehle aus?

```
1    printf("A %d %<x %<o A%n", 16);
     A 16 10 20 A
2    printf("B %3$d %2$d %1$d B%n", 10, 20, 30);
     B 30 20 10 B
3    printf("C %d %1$x %1$o %d %2$x %2$o C%n", 16, 32);
     C 16 10 20 32 20 40 C
4    printf("D %d %<x %<o %d %<x %<o D%n", 16, 32);
     D 16 10 20 32 20 40 D
```

2. Mit welcher *Abkürzung* (3 Großbuchstaben) wird die einfachste Auszeichnungssprache bezeichnet, die wir besprochen haben? Für welche (drei englischen) Worte steht die Abkürzung? Woraus bestehen bei dieser Sprache die *Auszeichnungen*?

CSV, Character Separated Values
Die Auszeichnungen bestehen aus einem beliebigen Zeichen, z.B. einem Komma oder einem Blank oder einem Zeilenwechsel etc.

3. Wir haben auch (kurz) die Sprache *Curl* besprochen. Woran soll der Name erinnern? Wie sieht eine *Auszeichnung* in dieser Sprache aus?

"Curl" soll an "curly brackets" (geschweifte Klammern: { ... }) erinnern. Eine Auszeichnung besteht aus einem geschweiften Klammerpaar und dem ersten Wort darin, etwa so:

{Absatz ... ...}

Die übrigen Worte in den Klammern sind die ausgezeichneten Daten.

4. Dann haben wir noch eine Auszeichnungssprache besprochen, deren Name sich auf (den englischen Tiernamen) "camel" reimt. Wie heißt diese Sprache? Auf welchen drei grundlegenden *Datenstrukturen* beruht diese Sprache?

Yamel. Diese Sprache beruht auf den Datenstrukturen Einzelwert, Liste und Abbildung.

5. Woraus besteht eine *Abbildung* (engl. map, Java-Schnittstelle: `Map`) in Java, C#, Perl, Python, Ruby, ... und in der Auszeichnungssprache Yamel?

Eine Abbildung (engl map) besteht aus Einträgen (engl. entries). Ein Eintrag besteht aus einem Schlüssel (engl key) und einem Wert (engl. value). Innerhalb einer Abbildung darf ein Schlüssel nur in einem Eintrag vorkommen.

14. SU, Mo 03.01.11

A. Wiederholung
B. Organisation

Auszeichnungssprachen, Fortsetzung

4. JASON (Java Script Object Notation)

Ist heute ein wichtiger Bestandteil von JavaFX
(Konkurrent zu Adobe Flash Player/Air und MS Silverlight).

5. HTML
Die traurige Erfolgsgeschichte von HTML:
In den Entwurf hatte sich ein Fehler eingeschlichen ("man braucht nicht alle Klammern schließen")
Deshalb konnte man mit üblichen Werkzeugen keinen HTML-Parser schreiben
Brauserkrieg: Welcher Brauser kann mit HTML-Dateien noch schlunziger umgehen als die anderen?
Der MS InternetExplorer hat gewonnen.
Ergebnis: Es ist sehr schwer, eine HTML-Seite "für alle Brauser" zu entwickeln.

Lichtblick: XHTML ist wie HTML, aber ohne den Entwurfsfehler, und Schlunzigkeit ist verboten.

6. XML
Ist als *Dokumenten-orientierte* Auszeichnungssprache entstanden.
Wir heute vor allem als *Daten-orientierte* Auszeichnungssprache eingesetzt.

Erhebliche Nachteile:

1. Ist für Menschen ähnlich leicht/schwer zu lesen, wie früher sog. Hex-Dumps (schon nach wenigen Jahren Übung konnte man die einigermaßen flüssig lesen).

2. XML ist so kompliziert, dass man grundlegende Werkzeuge dafür erst nach einer Einarbeitung von vielen Monaten oder einigen Jahren programmieren kann.

Andererseits: Einige Menschen haben sich ein paar Jahre eingearbeitet und die grundlegenden Werkzeuge geschrieben. Wir brauchen die nur noch anzuwenden.

Der wichtigste Vorteil von XML: XML ist *sehr weit verbreitet.*

Genau genommen ist XML keine Auszeichnungssprache, sondern eine Metasprache, in der man Auszeichnungssprachen definieren kann. Die in XML definierten Auszeichnungssprachen werden als *Anwendungen* (engl. applications) bezeichnet.

Beispiele für XML-Anwendungen:

XHTML	("vernünftiges" HTML)
RSS	(Really Simple Syndication, für "Artikel im Internet")
SOAP	(Simple Object Access Protocol, mit XML-Dokumenten als Nachrichten)
DocBook	(für technische Dokumentationen und andere Bücher)
...	(es gibt hunderte solcher Anwendungen)

Wohlgeformte und gültige Dokumente
(engl. well-formed and valid documents)

Verschiedene Typ-Beschreibungssprachen

- Dokumenten Typ-Definitionen (DTDs)
- XML-Schemas
- RELAX NG (REgular LAnguage for XML, Next Generation)
- Schematron

Empfehlung: Lesen Sie die Papiere XML01.pdf und XML02.pdf oder
das Buche XML in a Nutshell von Elliotte Rusty Harold und W. Scott Means, O'Reilly

Damit sollte es dann ziemlich leicht sein, die Aufgabe 9: XML-Dokumente erstellen ... zu lösen.

Interessante XML-Editoren:

Serna (siehe www.syntext.com)
XML Notepad 2007 von Microsoft

DOM und JDOM

Das *Document Object Model* (DOM) ist eine Schnittstelle die festlegt, dass ein XML-Dokument (hauptsächlich) aus einem *Baum* von *Elementen* besteht und mit welchen Unterprogrammen (einer beliebigen Programmiersprache) man auf diese Elemente zugreifen und sie verändern kann.

DOM ist ein W3C-Standard (W3C: World Wide Web Consortium). Dieser Standard ist
plattformunabhängig und
sprachunanbhängig und
enthält nur 2 *Sprachbindungen* (language bindings): Für `Java` und für ECMAScript (früher: JavaScript).

Schwachstelle: Der Standard ist wirklich völlig sprachunabhängig und und könnte im Prinzip auch in Fortran oder C implementiert werden. D.h. er macht keinen Gebrauch von OO-Konzepten (Klasse, Objekt, Vererbung, ...). Deshalb fühlt er sich in einer OO-Sprache fremd an.

JDOM ist eine alternative Java-Schnittstelle, die Ähnliches leistet wie DOM, aber viel einfacher und natürlicher.

ECMA:
1970: European Computer Manufacturers Association
1994: Ecma International - European association for standardizing information and communication systems

Anmerkungen zu einigen Dateien im Verzeichnis `Xml` im Archiv `DateienFuerPR2.zip`:

Datei	Inhalt
`XmlJStd.java`	Typische Anwendungen einer Java-Implementierung der DOM-Schnittstelle
`XmlJDom.java`	Typische Anwendungen von JDOM
`CreateDOM10.java`	Wie erzeugt man in einem Java-Programm ein DOM-`Document`-Objekt und wie kann man es prüfen lassen?
`CreateJdom10.java`	Wie erzeugt man in einem Java-Programm ein JDOM-`Document`-Objekt und wie kann man es prüfen lassen?

Zur Entspannung: Die Programmiersprache Ada

Wurde 1974-1980 im Auftrag des amerikanischen DOD von einer europäischen Firma (Honeywell-Bull) mit internationalem Mitarbeiterstab entwickelt. ISO-Standard: 1983, 1995 und 2005.

Obwohl älter als Java, ist Ada in einigen Einzelheiten "moderner": Einfachere Syntax "ohne von C geerbte Probleme", sichere Ganzzahlarithmetik etc. Heute darf z. B. die Software für Flugzeuge und ähnliche "Sicherheitssoftware" in aller Regel nur in Ada geschrieben werden.

Ada hat sich viel weniger verbreitet, als 1980 viele annahmen. Warum?

Hinterher ist man schlauer. Ein wichtiger "technischer Fehler" beim Entwurf: Ada nimmt Rücksicht auf unterschiedliche Prozessorstrukturen (z. B. wird die Länge eines `int`-Wertes von der Sprache nicht festgelegt, jeder Ada-Ausführer darf sie so festlegen, wie es ihm passt). Deshalb ist es möglich, dass ein Ada-Programm auf *einer* Maschine ("mit langen `int`-Werten") funktioniert, aber auf einer anderen Maschine ("mit kurzen `int`-Werten") mit einer Ausnahme abbricht. *Ada*-Programme sind also in einem wichtigen Sinne Hardware-abhängig. Die Entwickler von *Java* haben dagegen unter anderem festgelegt, dass ein `int`-Wert genau 32 Bit lang sein muss (ob das einer bestimmten Hardware nun passt oder nicht). Das hat sich als sehr großer Vorteil herausgestellt.

Wiederholungsfragen, 15. SU, Mo 10.01.11

1. Unter den Syntax-Regeln der Auszeichnungssprache HTML gibt es eine, die sehr nachteilige Folgen hatte. Was erlaubt diese Regel dem Schreiber eines HTML-Dokuments?

2. Was geschah beim sog. Browser-Krieg (zwischen Netscape's Navigator und Microsoft's Internet Explorer, Ende der 1990-er Jahre) in Bezug auf HTML-Dateien?

3. Die Auszeichnungssprachen HTML und XHTML stimmen weitgehend überein. Wodurch unterscheiden sie sich?

4. Das *Document Object Model* (DOM) ist eine *plattformunabhängige* und *sprachunabhängige* Programmierschnittstelle. Was kann man mit den Unterprogrammen, die darin beschrieben werden, machen?

5. Die Schnittstelle JDOM leistet ganz Ähnliches wie DOM. Was ist der wichtigste Unterschied zwischen JDOM und DOM? (Tip: Der Unterschied hat mit dem ersten Buchstaben von "JDOM" zu tun)

6. Wie heißen die beiden wichtigsten Sprachen, in denen man *Typen* von XML-Dateien beschreiben kann (z.B. Dateitypen wie Rechnung, Formel, Internetseite etc.)?

Antworten zu den Wiederholungsfragen, 15. SU, Mo 10.01.11

1. Unter den Syntax-Regeln der Auszeichnungssprache HTML gibt es eine, die sehr nachteilige Folgen hatte. Was erlaubt diese Regel dem Schreiber eines HTML-Dokuments?

Sie erlaubt es, eine früher geöffnete Klammer zu schließen, ohne die später geöffneten Klammern zu schließen (die gelten dann als "implizit geschlossen").

2. Was geschah beim sog. Browser-Krieg (zwischen Netscape's Navigator und Microsoft's Internet Explorer, Ende der 1990-er Jahre) in Bezug auf HTML-Dateien?

Die Browser vermieden (und vermeiden bis heute) eine strenge Prüfung von HTML-Dateien, sondern wetteifern darin, möglichst viele HTML-Syntaxfehler "besonders wohlwollend zu behandeln".

3. Die Auszeichnungssprachen HTML und XHTML stimmen weitgehend überein. Wodurch unterscheiden sie sich?

Man muss alle geöffneten Klammern explizit schließen.

4. Das *Document Object Model* (DOM) ist eine plattformunabhängige und sprachunabhängige Programmierschnittstelle. Was kann man mit den Unterprogrammen, die darin beschrieben werden, machen?

Per Programm XML-Dokumente erzeugen und auf die einzelnen Teile solcher Dokumente zugreifen.

5. Die Schnittstelle JDOM leistet ganz Ähnliches wie DOM. Was ist der wichtigste Unterschied zwischen JDOM und DOM? (Tip: Der Unterschied hat mit dem ersten Buchstaben von "JDOM" zu tun)

DOM ist sprachunabhängig und nützt die speziellen Möglichkeiten objektorientierter Sprachen nicht aus. JDOM ist eine Java-Schnittstelle (und nützt spezielle Möglichkeiten von Java aus).

6. Wie heißen die beiden wichtigsten Sprachen, in denen man *Typen* von XML-Dateien beschreiben kann (z.B. Dateitypen wie Rechnung, Formel, Internetseite etc.)?

Document Type Definition (DTD) und XML Schema

15. SU, Mo 10.01.11

A. Wiederholung
B. Organisation

Meine persönliche Einschätzung von XML:

1. Die Entwickler von SGML und später die von XML haben sich große Verdienste erworben, weil sie den Nutzen und die Notwendigkeit von Auszeichnungssprachen erkannt und entsprechende Sprachen entwickelt haben. Allerdings wurde SGML für die Auszeichnung von *wenig strukturierten Texten* (Dokumentationen von Behörden, dem Militär, von Firmen etc.) entwickelt, die aus viel *freiem Text* bestehen, in dem man ab und zu ein Worrt als ein Stichwort (für ein Stichwortverzeichnis), eine Zeile als Überschrift oder einen Abschnitt als Inhaltsverzeichnis auszeichnen möchte.

2. SGML war viel, viel zu kompliziert (was bei einem ersten Versuch, eine ganz neue *Art* von Sprache zu entwickeln, nicht verwundern sollte).

3. XML ist eine Untermenge von SGML und wird für *zwei* unterschiedliche Aufgaben eingesetzt:
3.1. Zum Auszeichnen relativ schwach strukturierter Texte (siehe z.B. DocBook).
3.2. Zum Serialisieren von (stark strukturierten) Programm-Ausgabedaten.

Die unterschiedlichen Anforderungen, die daraus entstehen, haben dazu beigetragen, dass XML für beide Aufgabengebiete nicht "maßgeschneidert" und zu kompliziert ist.

JSON und YAML sind sehr gut zum Serialisieren von Programm-Ausgabedaten geeignet, aber nicht zum Auszeichnen schwach strukturierter Texte.

4. Negative Eigenschaften von XML:

4.1. XML ist sehr kompliziert. Eine vollständige Beschreibung füllt ein dickes Buch. XML vollständig zu lernen dauert Monate oder Jahre, nicht Stunden oder Tage.

4.2. XML beruht nicht auf den Grundkonzepten verbreiteter Programmiersprachen (*Typen* und *Variablen*), sondern auf dem ganz anderen Konzept eines *Tags*. Tags haben mit Typen und Variablen zu tun, aber auf eine ziemlich komplizierte und schwer zu durchschauende Weise.

4.2. In XML fehlen die Grundtypen, die es in vielen Programmiersprachen gibt (`int`, `float`, `char`, `String`). In XML kann man nicht ausdrücken, dass eine bestimmte Information durch eine Ganzzahl (oder eine Bruchzahl) dargestellt werden soll.

4.3. XML ist von Menschen nur in einfachen Fällen lesbar und in vielen praktischen Fällen nicht lesbar.

4.4. Es gibt zu viele Schema-Sprachen, d.h. Sprachen, mit denen Typen von XML-Dokumenten beschrieben werden:

DTD	(Document Type Definition, gehört zu XML, ist aber ziemlich schwach)
DSD	(Document Structure Description, BRICS in Aarhus und AT&T)
XML-Schemen	(Empfehlung des W3C [World Wide Web Consortium])
Relax	(Regular Language Description for XML, ISO/IEC Technischer Bericht 22250-1)
Relax-NG	(Relax New Generation, ISO/IEC 19757-2),
Schematron	(ISO/IEC-Standard 19757-3:2006, eine neue Version ist unterwegs)
TREX	(Tree Regular Expressions for XML)

Einige dieser Schema-Sprachen sind selbst XML-Sprachen (DSD, XML-Schema, Relax, Relax-NG, Schematron), und ihre Schemen sind entsprechen schwer lesbar und voller störender Redundanz.

Andere Schema-Sprachen bieten eine kompaktere, lesbarere Schreibweise (DTD, Relax-NG).

4. Eine gleichzeitig positive und negative Eigenschaft von XML:
Viele Menschen (darunter auch viele Manager) haben beschlossen, XML zu verwenden.

5. Möglicherweise konnten viele Manager nur deshalb vom Nutzen von XML überzeugt werden, weil XML sehr kompliziert ist. Diese Manager hätte man möglicherweise nicht vom Nutzen einer ganz einfachen (aber ähnlich nützlichen) Sprache überzeugen können.

6. Ich hoffe, dass für das Serialisieren von Programm-Ausgabedaten XML in den nächsten Jahren zunehmend durch einfachere, leichter beschreibbare, leichter lernbare, leichter handhabbare aber ausdrucksstärkere Auszeichnungssprachen ersetzt wird.

Parser

Ein Parser ist ein Programm, welches von Text-Dokumenten feststellen kann, ob sie eine bestimmte Struktur haben oder nicht (z.B. die Struktur einer Java-Klasse oder die Struktur einer C-Datei oder die Struktur einer Zahnarztrechnung oder ...).

Für jede Struktur braucht man einen entsprechenden Parser.

Viele Parser machen noch mehr: Wenn sie feststellen, dass das Text-Dokument die betreffende Struktur hat, erzeugen sie aus dem Text-Dokument eine programminterne Datenstruktur (einen sogenannten Syntaxbaum), bei der man leicht auf die einzelnen Teile zugreifen kann.

Für Text-Dokumente, die mit einer Auszeichnungssprache ausgezeichnet wurden, kann man einen *universellen Parser* schreiben. Ein universeller XML-Parser kann jedes Text-Dokument daraufhin prüfen, ob es *wohlgeformt* ist ("die allgemeinen Regeln von XML einhält") und kann daraus einen programminternen Syntaxbaum erzeugen (z.B. ein DOM-Document-Objekt, oder ein JDOM-Document-Objekt, ...).

Es gibt auch universelle *validierende* XML-Parser, denen man
ein Text-Dokument TEXT und
eine Typ-Defininition TYP
übergeben kann, und die dann prüfen, ob der TEXT relativ zu TYP *gültig* (engl. valid) ist. Einige dieser Parser erzeugen dann auch einen Syntaxbaum.

Ein SAX-Parser ist kein Parser, sondern nur ein Teil eines Parsers (den der Programmierer dann zu einem richtigen Parser ausbauen kann)

SAX: Simple API für XML
API: Application Programming Interface

Ein DOM-Parser ist ein Parser, der ein Text-Dokument prüfen und daraus ein DOM-Document-Object erzeugen kann.

Zur Entspannung: Was ist eine Funktion?

Mathematiker definieren eine Funktion üblicherweise als eine *Menge von* (Argument-Wert-) *Paaren* (oder als eine *linkstotale* und *rechtseindeutige Relation*). Eine solche Funktion kann man *nicht* auf sich selbst anwenden (da eine Menge sich nicht selbst als Element enthalten kann).

Informatiker stellen sich unter einer Funktion meist einen bestimmten Programmteil ("mit Parametern und einem Ergebnistyp") oder (etwas abstrakter) eine *Anwendungsvorschrift* vor. Solche Funktionen kann man auf bestimmte aktuelle Parameter anwenden, und einige solche Funktionen kann man auch auf sich selbst anwenden: Z. B. kann ein Klammerprüfprogramm prüfen, ob in seinem Quelltext alle Klammern korrekt geschlossen werden, bestimmte Compiler (die in ihrer eigenen Quellsprache geschrieben sind) kann man mit sich selbst compilieren und einen Editor mit sich selbst editieren etc.

21 Reflexion (S. 515)

Was für Klassen würde man vereinbaren für ein *Beuth-Hochschule-Verwaltungsprogramm*?
(Person, Unterklassen Studi, Prof, ..., Raum, Lehrveranstaltung, ...)

Was für Klassen würde man vereinbaren für ein *Java-Programme-Bearbeitungsprogramm*?
(Klasse, Attribut, Methode, Konstruktor, ...)

Ein Java-Programm kann, während es ausgeführt wird, sozusagen "in einen Spiegel schauen und sich selbst betrachten und untersuchen". Mit den Befehlen der *Java-Reflexionsschnittstelle* kann ein Programm z.B. feststellen, wie viele Methoden eine bestimmte Klasse enthält, wie die Methoden heißen und welche Parameter die Methoden haben. Außerdem kann es diese Methoden aufrufen, obwohl der Programmierer beim Schreiben des Programms den Namen der Methoden (und ihre Parameter etc.) noch gar nicht kannte.

Die Java-Reflexionsschnittstelle ist besonders nützlich um Programme zu schreiben, die Java-Programme bearbeiten sollen. Beispiele für solche Programme:

Ein Klassen-Browser. *Eingabe*: Der (volle) Name einer Klasse. *Ausgabe*: Eine genaue Beschreibung aller Elemente und Konstruktoren der Klasse (siehe `KlassenBrowser` im Archiv DateienFuerPR2.zip).

Ein Dokumentations-Programm: *Eingabe*: Der Name einer Klasse. *Ausgabe*: Eine gut strukturierte Dokumentation aller Elemente und Konstruktoren der Klasse. Voraussetzung: Der Programmierer hat die einzelnen Elemente und Konstruktoren vernünftig kommentiert. Ein solches (kostenloses) Programm namens `javadoc` gehört zu Java-Distribution der Firma Sun.

Eine Funktion, die Objekte beliebiger Typen erzeugen kann: *Parameter*: Der Name einer Klasse (oder eines Reihungstyps). *Ergebnis*: Ein Objekt dieser Klasse (bzw. dieses Reihungstyps), bei dem alle Attribute zufällig gewählte Werte haben. Kann für Maßentests gut sein, wo man z.B. viele Objekte eines bestimmten Typs benötigt, es aber nicht wichtig ist, wie die einzelnen Objekte aussehen.

Zur Java-Reflexionsschnittstelle gehören etwa 20 Klassen und Schnittstellen. Damit kann man u.a. Programme schreiben, die Probleme der folgenden Art lösen:

Problem 1: Den Namen einer beliebigen Klasse einlesen und die Namen aller Elemente dieser Klasse ausgeben.

Problem 2: Den Namen einer beliebigen Klasse K und den Namen einer (in K vereinbarten) Methode einlesen und die Methode aufrufen.

Problem 3: Ein Objekt einer beliebigen Klasse einlesen (aus einer Datei oder vom Internet) und die Werte all seiner Attribute ausgeben.

Wenn man ein objektorientiertes Programm zur Verwaltung der BHT entwickeln will, muss man wahrscheinlich Klassen namens `StudentIn`, `ProfessorIn`, `Raum`, `Vorlesung` etc. programmieren.

Wenn man ein Programm zum Analysieren und bearbeiten von Java-Programmen entwickeln will, muss man wahrscheinlich Klassen namens `Klasse`, `Methode`, `Attribut`, `Konstruktor` etc. programmieren. Genau solche Klassen gehören zur Java-Reflexionsschnittstelle.

Bisher haben wir *Klassen* und *Objekte* sorgfältig unterschieden: Eine Klasse ist (ein Modul und) ein Bauplan für Objekte, und damit etwas *ganz anderes* als ein Objekt.

Die Wahrheit ist subtiler. Jetzt (nach fast 2 Semestern) sind Sie (hoffentlich) reif dafür:

Klassen werden (während der Ausführung eines Java-Programms) als *Objekte* der Klasse `Class` dargestellt. Ein solches `Class`-Objekt enthält alle Informationen über die betreffende Klasse. Man sagt auch: Das `Class`-Objekt *reflektiert* die betreffende Klasse.

`Class`-Objekte sind sozusagen "Der Eingang in die Welt der Reflexion".

3 Möglichkeiten, sich ein bestimmtes Class-Objekt zu besorgen

S. 517, Beispiel-01

Die ersten beiden Möglichkeiten (`String.class` und `strob.getClass()`) "gehen immer gut", d.h. sie werfen *nie eine Ausnahme*. Die dritte Möglichkeit (`Class.forName("...")`) "kann schief gehen" und wirft dann eine Ausnahme des Typs `ClassNotFoundException`.

Wiederholungsfragen, 16. SU, Mo 17.01.11

1. Skizzieren Sie zwei sehr allgemeine, aber unterschiedliche Gebiete, auf denen man XML verwendet. Oder: Welche unterschiedlichen Arten von Text-Dokumenten zeichnet man mit XML aus?

2. Was leistet ein Parser? Nennen Sie zwei Teilaufgaben, die viele Parser erledigen (einige erledigen nur eine der Teilaufgaben).

3. Was leistet ein DOM-Parser?

4. Was leistet ein Jdom-Parser?

5. Angenommen, Sie schreiben ein Programm und wollen sich jetzt das Class-Objekt einer bestimmten Klasse besorgen. Wie machen Sie das

5.1. wenn Sie den vollen Namen der Klasse schon beim Schreiben des Programms wissen (z.B. den Namen `java.util.JButton`)?

5.2. wenn Sie ein Objekt der betreffenden Variablen haben (z.B. eines namens `meinObjekt`)?

5.3. wenn Sie den vollen Namen der Klasse als Zielwert einer `String`-Variablen haben (z.B. einer `String`-Variablen namens `klassenName`)?

6. Angenommen, die Variable `kob` zeigt auf ein `Class`-Objekt.
Was liefert dann der Funktionsaufruf `kob.getDeclaredFields()` ?

7. Ebenso für den Funktionsaufruf `kob.getFields()` ?

Rückseite der Wiederholungsfragen, 16. SU, Mo 17.01.11

Die Wahrheit: Klassen sind auch Objekte (der Klasse Class)

Bisher haben wir sorgfältig zwischen einer Klasse (einem Modul und Bauplan für Module) und einem Objekt (einem Modul, der nach einem Bauplan gebaut wurde) unterschieden. Tatsächlich werden Klassen überlicherweise auch als *Objekte* einer speziellen Klasse namens `Class` realisiert.

Sei `CL` eine Klasse, von der wir ein paar Objekte `ob1`, `ob2`, ... erzeugt haben, etwa so:

```
CL ob1 = new CL(...);
CL ob2 = new CL(...);
```

Bisher haben wir die Klasse `CL` und ihre Objekte `ob1`, `ob2`, ... etwa so beschrieben:

```
+-------------------------------+ +-------------------------+ +-------------------------+
| Modul CL                      | | CL-Objekt ob1           | | CL-Objekt ob2           |
|                               | |                         | |                         |
| kAtt1                         | | oAtt1                   | | oAtt1                   |
| kAtt2                         | | oAtt2                   | | oAtt2                   |
| ...                           | | ...                     | | ...                     |
| kMet1(T1 p1, T2 p2, ...)      | | oMet1(T1 p1, T2 p2, ...)| | oMet1(T1 p1, T2 p2, ...)|
| kMet2(T1 p1, T2 p2, ...)      | | oMet2(T1 p1, T2 p2, ...)| | oMet2(T1 p1, T2 p2, ...)|
| ...                           | | ...                     | | ...                     |
+-------------------------------+ +-------------------------+ +-------------------------+
```

Abkürzungen:
`kAtt`: Klassen-Attribut **`oAtt`**: Objekt-Attribut
`kMet`: Klassen-Methode **`oMet`**: Objekt-Methode

Implementiert wird das Ganze aber so, dass Objekte nur ihre *Attribute* enthalten, die Objekt-*Methoden* aber nicht in jedem Objekt, sondern auch nur *einmal* im Modul `CL` gespeichert werden, etwa so:

```
+-------------------------------+ +-------------------------+ +-------------------------+
| Modul CL                      | | CL-Objekt ob1           | | CL-Objekt ob2           |
|                               | |                         | |                         |
| kAtt1                         | | oAtt1                   | | oAtt1                   |
| kAtt2                         | | oAtt2                   | | oAtt2                   |
| ...                           | | ...                     | | ...                     |
| kMet1(CL p0, T1 p1, T2 p2, ...)| +-------------------------+ +-------------------------+
| kMet2(CL p0, T1 p1, T2 p2, ...)|
| ...                           |
| oMet1(CL p0, T1 p1, T2 p2, ...)|
| oMet2(CL p0, T1 p1, T2 p2, ...)|
| ...                           |
+-------------------------------+
```

Wenn es viele Objekte derselben Klasse `CL` gibt, spart man dadurch viel Speicherplatz.

Allerdings galt ja:
Die Objektmethoden im Objekt `ob1` haben auf die Attribute in `ob1` zugegriffen und
die Objektmethoden im Objekt `ob2` haben auf die Attribute in `ob2` zugegriffen etc.

Damit das auch weiterhin richtig funktioniert, ändert der Ausführer jedes Quellprogramm wie folgt:

1. Er versieht jede Objektmethode mit einen *zusätzlichen Parameter* `p0` vom Typ `CL`.
2. Er wandelt jeden *Aufruf einer Objekt-Methode* wie folgt um:

```
Alter Aufruf: ... ob.oMet1(ap1, ap2, ...) ...
                    \       |    |
                     \       \    \
                      \       |    |
                       \      |    |
Neuer Aufruf: ... oMet1(ob, ap1, ap2, ...) ...
```

Und damit er alle Methoden möglichst ähnlich behandeln kann, versieht er auch jede Klassen-Methode mit einem zusätzlichen Parameter `p0` vom Typ `CL`. Klassen-Methoden greifen allerdings nie auf `p0` zu, und in einem Aufruf der Methode wird als aktueller Parameter für `p0` immer `null` übergeben.

Eine Übungsaufgabe (für den 16. SU, Mo 17.01.11)

Fragen zum Abschnitt 21.2 (S. 520 - 523) im Buch

Die folgenden Fragen beziehen sich auf die Methoden `main`, `pruefeUndRufeAuf` und `rufeAuf` auf den Seiten 521-522 des Buches. Diskutieren Sie die Fragen in kleinen Gruppen (zu zweit oder zu dritt) und schreiben Sie die Antworten lesbar auf. Wer im Buch die Erläuterungen zu den Methoden liest, wird nicht als Warmduscher oder Schummler beschimpft (versprochen!).

1. Fragen zur Methode main (Zeile 1 bis 10)

1.1. Warum muss in Zeile 2 `throws ClassNotFoundException` stehen? Welcher Befehl in dieser Methode könnte eine solche Ausnahme werfen?

1.2. Von welchem Typ ist die Variable `rm`?

1.3. Was liefert der Funktionsaufruf `kob.getDeclaredMethods()` als Ergebnis?

1.4. Was macht die `for-each`-Schleife in Zeile 9?

2. Fragen zur Methode pruefeUndRufeAuf

2.1. Was erwartet diese Methode als Parameter? Einen `int`-Wert? Nein! Ein `String`-Objekt? Nein! Sondern?

2.2. Von welchem Typ ist die Variable `rt`?

2.3. Was liefert der Funktionsaufruf `m.getParameterTypes()` (in Zeile 12) als Ergebnis?

2.4. Was liefert der Funktionsaufruf `m.getName()` wohl als Ergebnis?

2.5. Was bezeichnet der Name `Double.TYPE`?

2.6. Übersetzen Sie die Bedingung der `if`-Anweisung in Zeile 15 in verständliches Deutsch. Welche Eigenschaften der Methode `m` prüft diese Bedingung?

3. Fragen zur Methode rufeAuf

3.1. Was erwartet diese Methode als Parameter?

3.2. Womit wird die Variable `metName` initialisiert? Mit dem Namen des Programmierers? Nein! Sondern?

3.3. Was bewirkt der Methodenaufruf `m.invoke(null, dopa)` (in Zeile 24)?

3.4. Was liefert der Methodenaufruf `m.invoke(null, dopa)`, wenn das Method-Objekt `m` eine *Funktion* reflektiert?

3.5. Was liefert der Methodenaufruf `m.invoke(null, dopa)`, wenn das Method-Objekt `m` eine *Prozedur* reflektiert?

3.6. Wann wirft der Methodenaufruf `m.invoke(null, dopa)` eine Ausnahme und von welchem Typ ist diese Ausnahme?

3.7. Was liefert der Funktionsaufruf `m.getDeclaringClass()` als Ergebnis?

3.8. Was liefert der Funktionsaufruf `m.getDeclaringClass().newInstance()` als Ergebnis?

3.9. Was bewirkt der Methodenaufruf `m.invoke(ob, dopa)` (in Zeile 33)?

3.10. Was liefert der Methodenaufruf `m.invoke(ob, dopa)` wenn das Method-Objekt `m` eine *Funktion* reflektiert?

3.11. Was liefert der Methodenaufruf `m.invoke(ob, dopa)` wenn das Method- Objekt `m` eine *Prozedur* reflektiert?

Antworten zu den Wiederholungsfragen, 16. SU, Mo 17.01.11

1. Skizzieren Sie zwei sehr allgemeine, aber unterschiedliche Gebiete, auf denen man XML verwendet. Oder: Welche unterschiedlichen Arten von Text-Dokumenten zeichnet man mit XML aus?

1.1. Auszeichnung von schwach strukturierten Texten (z.B. Dokumentationen oder Bücher oder ...)
1.2. Auszeichnung von stark strukturierten Texten (Ausgaben eines Programms, die von anderen Programmen weiterverarbeitet werden sollen)

2. Was leistet ein Parser? Nennen Sie zwei Teilaufgaben, die viele Parser erledigen (einige erledigen nur eine der Teilaufgaben).

2.1. Er prüft von einem beliebigen Textdokument, ob es eine bestimmte Struktur hat.
2.2. Er erzeugt aus dem Textdokument einen Syntaxbaum (eine programminterne Datenstruktur, bei der man leicht auf die einzelnen Teile zugreifen kann).

3. Was leistet ein DOM-Parser?

Er prüft von einem beliebigen Textdokument, ob es die Struktur einer XML-Datei hat und erzeugt daraus ein DOM-Document-Objekt.

4. Was leistet ein Jdom-Parser?

Er prüft von einem beliebigen Textdokument, ob es die Struktur einer XML-Datei hat und erzeugt daraus ein Jdom-Document-Objekt.

5. Angenommen, Sie schreiben ein Programm und wollen sich jetzt das Class-Objekt einer bestimmten Klasse besorgen. Wie machen Sie das

5.1. wenn Sie den vollen Namen der Klasse schon beim Schreiben des Programms wissen (z.B. den Namen `java.util.JButton`)?

```
Class<?> kob = java.util.JButton.class;
```

5.2. wenn Sie ein Objekt der betreffenden Variablen haben (z.B. eines namens `meinObjekt`)?

```
Class<?> kob = meinObjekt.getClass();
```

5.3. wenn Sie den vollen Namen der Klasse als Zielwert einer `String`-Variablen haben (z.B. einer `String`-Variablen namens `klassenName`)?

```
Class<?> kob = Class.forName(klassenName);
```

6. Angenommen, die Variable `kob` zeigt auf ein `Class`-Objekt.
Was liefert dann der Funktionsaufruf `kob.getDeclaredFields()` ?

Eine Reihung, die für jedes in der Klasse kob vereinbarte Attribut ein entsprechendes Field-Objekt enthält (von kob geerbte Attribute sind also ausgeschlossen).

7. Ebenso für den Funktionsaufruf `kob.getFields()` ?

Eine Reihung, die für jedes öffentliche (public) Attribut der Klasse kob ein entsprechendes Field-Objekt enthält (egal ob das Attribut in kob vereinbart wurde oder von kob geerbt wurde).

Antworten zur Übungsaufgabe "Fragen zum Abschnitt 21.2 (S. 520 - 523) im Buch"

1. Fragen zur Methode main (Zeile 1 bis 10)

1.1. Warum muss in Zeile 2 `throws ClassNotFoundException` stehen? Welcher Befehl in dieser Methode könnte eine solche Ausnahme werfen? (Der Befehle Class.forName(...) in Zeile 6)
1.2. Von welchem Typ ist die Variable `rm`? (Vom Typ Reihung von Method)
1.3. Was liefert der Funktionsaufruf `kob.getDeclaredMethods()` als Ergebnis? (Eine Reihung von Method-Objekten. Jedes Objekt beschreibt eine in der Klasse kob vereinbarte Methode).
1.4. Was macht die `for-each`-Schleife in Zeile 9? (Sie ruft mehrmals die Methode pruefeUndRufeAuf auf und übergibt ihr eines der Method-Objekte m aus der Reihung rm).

2. Fragen zur Methode pruefeUndRufeAuf

2.1. Was erwartet diese Methode als Parameter? Einen `int`-Wert? Nein! Ein `String`-Objekt? Nein! Sondern? (Ein Method-Objekt)
2.2. Von welchem Typ ist die Variable `rt`? (Vom Typ Reihung von Class)

2.3. Was liefert der Funktionsaufruf `m.getParameterTypes()` (in Zeile 12) als Ergebnis? (Eine Reihung von Class-Objekten. Jedes Objekt beschreibt den Typ eines Parameters der Methode m)
2.4. Was liefert der Funktionsaufruf `m.getName()` wohl als Ergebnis? (Den Namen der Methode m)
2.5. Was bezeichnet der Name `Double.TYPE`? (Das Class-Objekt, welches den primitiven Typ double reflektiert)
2.6. Übersetzen Sie die Bedingung der `if`-Anweisung in Zeile 15 in verständliches Deutsch. Welche Eigenschaften der Methode `m` prüft diese Bedingung? (Wenn m genau 1 Parameter hat und dieser Parameter vom Typ double ist, dann ...)

3. Fragen zur Methode rufeAuf

3.1. Was erwartet diese Methode als Parameter? (Ein Method-Objekt)
3.2. Womit wird die Variable `metName` initialisiert? Mit dem Namen des Programmierers? Nein! Sondern? (Mit dem Namen der Methode m)
3.3. Was bewirkt der Methodenaufruf `m.invoke(null, dopa)` (in Zeile 24)? (Die Methode m wird als Klassenmethode mit dem Parameter dopa aufgerufen)

3.4. Was liefert der Methodenaufruf `m.invoke(null, dopa)`, wenn das Method-Objekt `m` eine *Funktion* reflektiert? (Das Ergebnis der Funktion m)

3.5. Was liefert der Methodenaufruf `m.invoke(null, dopa)`, wenn das Method-Objekt `m` eine *Prozedur* reflektiert? (Den Wert null)
3.6. Wann wirft der Methodenaufruf `m.invoke(null, dopa)` eine Ausnahme und von welchem Typ ist diese Ausnahme? (Wenn m keine Klassenmethode [sondern eine Objektmethode] ist, wird eine Ausnahme des Typs NullPointerException geworfen)

3.7. Was liefert der Funktionsaufruf `m.getDeclaringClass()` als Ergebnis? (Die Heimatklasse von m. D.h. das Class-Objekt, welches die Klasse reflektiert, in der die Methode m vereinbart wurde)
3.8. Was liefert der Funktionsaufruf `m.getDeclaringClass().newInstance()` als Ergebnis? (Ein Objekt der Heimatklasse von m)
3.9. Was bewirkt der Methodenaufruf `m.invoke(ob, dopa)` (in Zeile 33)? (Die Methode m wird als Objektmethode mit dem Parameter dopa aufgerufen und greift auf die Attribute des Objekts ob zu)
3.10. Was liefert der Methodenaufruf `m.invoke(ob, dopa)` wenn das Method-Objekt `m` eine *Funktion* reflektiert? (Das Ergebnis der Funktion m)

3.11. Was liefert der Methodenaufruf `m.invoke(ob, dopa)` wenn das Method- Objekt `m` eine *Prozedur* reflektiert? (Den Wert null)

16. SU, Mo 17.01.11

A. Wiederholung
B. Organisation

Was man als reflektiver Programmierer über Methoden wissen sollte

Dazu betrachten wir die Rückseite der Wiederholungsfragen.

Reflektiver Zugriff auf nicht-öffentliche Elemente

Im Prinzip kann man per Reflektion auf *alle* Elemente und Konstruktoren einer Klasse zugreifen. Ob man auf nicht-öffentliche Elemente oder Konstruktoren zugreifen darf, hängt von zwei Voraussetzungen ab, die hier kurz erläutert werden:

Die Klassen `Field`, `Method` und `Constructor` haben eine gemeinsame direkte Oberklasse:

```
AccessibleObject
        |
        +--- Field
        |
        +--- Method
        |
        +--- Constructor
```

Die Klasse `AccesibleObject` enthält ein
paketweit erreichbares Objektattribut namens `override` vom Typ `boolean` und eine öffentliche Objektmethode `void setAccessible(boolean b)`,
mit der man `override` auf `true` (oder `false`) setzen kann.

Auf ein nicht-öffentliches Element oder einen nicht-öffentlichen Konstruktor darf man nur zugreifen, wenn sein `override`-Attribut den Wert `true` hat.

Das `override`-Attribut hat den Anfangswert `false`. Ob man es mit `setAccessible` auf `true` setzen darf, hängt davon ab, ob das betreffende Programm unter Aufsicht eines `SecurityManager`-Objekts läuft:
Wenn nein, dann darf man `override` auf true setzen.
Wenn ja, dann kommt es darauf an, ob der `SecurityManager` einem erlaubt, `override` zu ändern.

In einem Programm kann man etwa wie folgt feststellen, ob ein `SecurityManager` die Ausführung überwacht:

```
SecurityManager sm = System.getSecurityManager();
```

Wenn `sm` danach den Wert `null` hat, gibt es *keinen* `SecurityManager`.

Wenn ein `SecurityManager` die Programmausführung überwacht, dann *verbietet* er einem alles, was in der Datei namens `java.policy` nicht ausdrücklich erlaubt ist.

Ein Beispiel für eine policy-Datei finden Sie im Archiv `DateienFuerPR2.zip`.

Ein SecurityManager-Objekt erzeugen und aktivieren kann man etwa so:

```
5    SecurityManager sm = new SecurityManager();
6    System.setSecurityManager(sm);
```

Zur Entspannung: Was ist schrecklich an Grabos (engl. GUIs)?

1. Es gibt keine einfache, klare Sprache, in der man die *Bedienung* einer Grabo beschreiben könnte.

Anleitungen wie "Öffnen Sie Menü A, wählen sie Punkt B, klicken Sie auf Knopf C, geben Sie "ABC" in das Fensterchen neben "Eingabe" ein, ..." sind schwer auszuführen und erklären einem nichts über die Struktur der Grabo, mit der man gerade arbeitet.

Screenshots sind häufig sehr umfangreich, selbst für einfache Funktionen muss man mehrere Seiten davon anlegen/lesen. Ein einzelner Screenshot enthält häufig sehr viele Details, von denen nur ganz wenige relevant sind. Man lernt vor allem die Farbe bestimmter Knöpfe und die Positionen gewisser Fenster, die nichts mit dem zu lösenden Problem zu tun haben und in der nächsten Version der Grabo anders sein können.

2. Es gibt keine Möglichkeit, komplizierte Bedienschritte zu automatisieren (z.B. durch ein Skript). Heutige Makro-Recorder leisten viel zu wenig.

3. Grabos lassen dem Programmierer sehr viel Wahlmöglichkeiten und es gibt keine Regeln, wie er wählen sollte. Deshalb kann ein Benutzer kaum vorhersehen, welches Problem durch welche Art von Grabo-Objekt gelöst wurde (durch einen Knopf? einen Menüpunkt? ein Eingabefeld? eine ComboBox? ...)

4. Viele Grabos enthalten außerdem elementare Fehler, z.B.:

- Bestimmte Fenster sind zu klein und können nicht vergrößert werden (obwohl man sich gerade einen sehr großen Bildschirm gekauft hat)
- Texteingaben müßen in einem schwer handhabbaren Format erfolgen und werden nicht sofort geprüft (z.B. die Werte von Umgebungsvariablen bei Windows und Linux)
- Mamchmal muss man einen bestimmten Arbeitsgang für viele Objekte wiederholen, statt dass man alle Objekte auswählt und dann den Arbeitsgang *einmal* durchführt.

Class-Objekte, die die primitiven Typen repräsentieren

S. 520, Beispiel-04: Das Attribut TYPE in den Hüllklassen

Class-Objekte, die Reihungstypen repräsentieren

S. 519, Beispiel-02: mit `.class` und mit `getClass`

S. 519, Beispiel-03: mit `Class.forName`

Wiederholungsfragen, 17. SU, Mo 24.01.11

1. Angenommen, ein Java-Quellprogramm enthält den folgenden Aufruf einer *Objektmethode*:

```
1    ... s.charAt(17) ...
```

In was für einen Methodenaufruf wird dieser Aufruf vom Java-Ausführer umgewandelt (damit er nur *ein* Exemplar der Objekt-Methode braucht, statt in jedem Objekt eine Kopie)?

2. Ebenso für den folgenden Aufruf einer *Klassenmethode*:

```
2    ... Math.sin(2.5) ...
```

3. Angenommen, `f07` ist ein `Field`-Objekt, welches ein *privates* Attribut repräsentiert. Sie wollen (natürlich reflektiv) auf den Wert dieses Attributs zugreifen. Welchen Methodenaufruf müssen Sie vorher ausführen lassen?

4. Die Variable `kobA` soll auf das `Class`-Objekt zeigen, welches den primitiven Typ `char` repräsentiert. Ergänzen Sie die folgende Variablen-Vereinbarung entsprechend:

```
3    Class<?> kobA =
```

5. Die Variable `kobB` soll auf das `Class`-Objekt zeigen, welches den Reihungstyp `char[]` repräsentiert. Ergänzen Sie die folgende Variablen-Vereinbarung durch einen entsprechenden Aufruf der Methode `Class.forName`:

```
4    Class<?> kobB =
```

6. Die Variable `kobC` soll auf das `Class`-Objekt zeigen, welches den Reihungstyp `char[]` repräsentiert. Ergänzen Sie die folgende Variablen-Vereinbarung, *ohne* die Methode `Class.forName` aufzurufen:

```
5    Class<?> kobC =
```

7. Die Variable `kobD` soll auf das `Class`-Objekt zeigen, welches den Reihungstyp `Integer[][][]` repräsentiert. Ergänzen Sie die folgende Variablen-Vereinbarung durch einen entsprechenden Aufruf der Methode `Class.forName`:

```
6    Class<?> kobD =
```

8. Die Variable `kobE` soll auf das `Class`-Objekt zeigen, welches den Reihungstyp `Integer[][][]` repräsentiert. Ergänzen Sie die folgende Variablen-Vereinbarung, *ohne* die Methode `Class.forName` aufzurufen:

```
7    Class<?> kobE =
```

Rückseite der Wiederholungsfragen, 17. SU, Mo 24.01.11, Anmerkungen (engl. annotations)

Beispiel-01: Drei Anmerkungstypen werden vereinbart

```
 1  public @interface AutorArno {}
 2
 3  public @interface Autor {
 4     String value() default "Noch unklar";
 5  }
 6
 7  public @interface Hersteller {
 8     String firma() default "ABC GmbH";
 9     int    jahr()  default 2003;
10  }
```

Beispiel-02: Eine Klasse und einige ihrer Elemente mit verschiedenen Anmerkungen versehen

```
11  @AutorArno
12  @Autor("Carl")
13  @Hersteller(jahr=2005, firma="Meyer&Sohn")
14  class AnmerkA {
15
16     @Autor      // Standardbelegung "Noch unklar"
17     @AutorArno
18     static public void druckeA() {pln("Methode druckeA!");}
19
20     static public void druckeB() {pln("Methode druckeB!");}
21
22     @Hersteller(firma="Schulz&Tochter")// Standardjahr 2003
23     String text1 = "Attribut text1";
24
25     @Autor(value="Carl")
26     @Hersteller(jahr=2002)             // Standardfirma "ABC GmbH"
27     String text2 = "Attribut text2";
28
29     ...
30  } // class AnmerkA
```

Beispiel-04: Anmerkungstypen mit eingeschränktem Ziel

```
31  @Target   (ElementType.TYPE)
32  @interface AutorT {
33     String value();
34  }
35
36  @Target   ({ElementType.METHOD, ElementType.FIELD})
37  @interface AutorMA {
38     String value();
39  }
```

Mögliche Ziele: ANNOTATION_TYPE, CONSTRUCTOR, FIELD, LOCAL_VARIABLE, METHOD, PACKAGE, PARAMETER und TYPE.

Beispiel-05: Drei Anmerkungstypen, deren Instanzen „verschieden weit mitgenommen“ werden

```
40  @Retention(RetentionPolicy.SOURCE)
41  @interface AutorS {
42     String value();
43  }
44
45  @Retention(RetentionPolicy.CLASS)
46  @interface AutorC {
47     String value();
48  }
49
50  @Retention(RetentionPolicy.RUNTIME)
51  @interface AutorR {
52     String value();
53  }
```

Antworten zu den Wiederholungsfragen, 17. SU, Mo 24.01.11

1. Angenommen, ein Java-Quellprogramm enthält den folgenden Aufruf einer *Objektmethode*:

```
1    ... s.charAt(17) ...
```

In was für einen Methodenaufruf wird dieser Aufruf vom Java-Ausführer umgewandelt (damit er nur *ein* Exemplar der Objekt-Methode braucht, statt in jedem Objekt eine Kopie)?

... charAt(s, 17) ...

2. Ebenso für den folgenden Aufruf einer *Klassenmethode*:

```
2    ... Math.sin(2.5) ...
```

... sin(null, 2.5) ...

3. Angenommen, `f07` ist ein `Field`-Objekt, welches ein privates Attribut repräsentiert. Sie wollen (natürlich reflektiv) auf den Wert dieses Attributs zugreifen. Welchen Methodenaufruf müssen Sie vorher ausführen lassen?

f07.setAccessible(true);

4. Die Variable `kobA` soll auf das `Class`-Objekt zeigen, welches den primitiven Typ `char` repräsentiert. Ergänzen Sie die folgende Variablen-Vereinbarung entsprechend:

```
3    Class<?> kobA = Character.TYPE;
```

5. Die Variable `kobB` soll auf das `Class`-Objekt zeigen, welches den Reihungstyp `char[]` repräsentiert. Ergänzen Sie die folgende Variablen-Vereinbarung durch einen entsprechenden Aufruf der Methode `Class.forName`:

```
4    Class<?> kobB = Class.forName("[C");
```

6. Die Variable `kobC` soll auf das `Class`-Objekt zeigen, welches den Reihungstyp `char[]` repräsentiert. Ergänzen Sie die folgende Variablen-Vereinbarung, *ohne* die Methode `Class.forName` aufzurufen:

```
5    Class<?> kobC = char[].class;
```

7. Die Variable `kobD` soll auf das `Class`-Objekt zeigen, welches den Reihungstyp `Integer[][][]` repräsentiert. Ergänzen Sie die folgende Variablen-Vereinbarung durch einen entsprechenden Aufruf der Methode `Class.forName`:

```
6    Class<?> kobD = Class.forName("[[[Ljava.lang.Integer;");
```

8. Die Variable `kobE` soll auf das `Class`-Objekt zeigen, welches den Reihungstyp `Integer[][][]` repräsentiert. Ergänzen Sie die folgende Variablen-Vereinbarung, *ohne* die Methode `Class.forName` aufzurufen:

```
7    Class<?> kobE = Integer[][][].class;
```

17. SU, Mo 24.01.11

A. Wiederholung
B. Organisation: Der nächste SU (am Mo 31.01.11) ist der letzte vor der Klausur. Wenn Sie Fragen zum Stoff der LV MB2-PR2 haben, sollten Sie die möglichst beim nächsten SU stellen (und nicht erst nach der Klausur :-).

Warum ist die Klasse Class generisch?

Zur Erinnerung: Die Klasse `ArrayList` ist *generisch*, mit *einem* Typ-Parameter: `ArrayList<K>`. Somit definiert die *eine* Klasse `ArrayList` die (unendlich) *vielen* parametrisierten Typen

```
ArrayList<String>,
ArrayList<Integer>,
ArrayList<ArrayList<String>>, ...
...
```

und zusätzlich den *rohen* (nicht-parametriesierten) Typ `ArrayList`. Den rohen Typ sollte man möglichst meiden (wie in einer Kneipe :-).

Ähnlich wie `ArrayList` ist auch die Klasse `Class` generisch mit *einem* Typ-Parameter: `Class<T>` Das `Class`-Objekt, welches die Klasse `String` repräsentiert, ist nicht nur vom rohen Typ `Class`, sondern genauer vom Typ `Class<String>`. Für alle anderen Typen gilt Entsprechendes.

Man beachte: Vom parametrisierten Typ `Class<String>` gibt es nie mehr als ein einziges Objekt, nämlich das, welches die Klasse `String` repräsentiert.

Die Klasse `Class` ist generisch, damit sie Funktionen mit dem Rückgabetyp `T` enthalten kann, nämlich: `T cast(Object ob)` und `T newInstance()`.

Als `Class` noch nicht generisch war (vor Java 5), gab es die Methode `cast` noch nicht und `newInstance` hatte den Rückgabetyp `Object`.

Die Methode `newInstance()` (mit 0 Parametern) wirft eine `InstantiationException`, wenn der betreffende Typ keine Klasse mit einem Standardkonstruktor ist.

Wie erzeugt man reflektiv eine Reihung? Die Klasse `java.lang.reflect.Array` (nicht mit `java.utils.Array`**`s`** zu verwechseln) enthält dazu die Methode

```
static public Object newInstance(Class<?> kt, int... stufe);
```

Dabei ist `kt` der Komponententyp und der `int`-Wert `stufe[i]` gibt an, wie lang die Reihung auf dieser Stufe ist. Beispiel:

```
... Array.newInstance(Integer.TYPE, 3, 5) ...
```

erzeugt eine zweistufige Reihung von 3 mal 5 gleich 15 `int`-Variablen.

Zur Entspannung: Woran erkennt man richtige Informatiker?

Informatiker automatisieren Arbeitsvorgänge. Richtige Informatiker erkennt man daran, dass sie stets versuchen, auch ihre *eigene* Arbeit zu rationalisieren und zu automatisieren.

Typische Hilfsmittel dazu: Für häufige Vorgänge am Computer spezielle Tastenkombinationen einrichten, Makros definieren (z. B. in Editoren), Skripte in verschiedenen Skript-Sprachen schreiben (DOS-bat-Dateien, PowerShell-Skripte, bash-Dateien, Perl-, PHP-, Tcl-Skripte etc.). Spezielle Programme wie `make` und `ant` verwenden.

Wenn man einen nicht-trivialen Arbeitsvorgang mehr als zwei Mal ausgeführt hat, sollte man überlegen, ob man ihn nicht (mit einer Tastenkombination oder einem Skript etc.) vereinfachen oder ganz automatisieren kann.

Ein Nachteil von Grabos (grafischen Benutzeroberflächen): Man kann komplizierte Arbeitsvorgänge nicht automatisieren (die üblichen "Makrorecorder" funktionieren nicht richtig).

Der Befehl

```
... Array.newInstance(String.class, 10, 10, 10, 10, 10) ...
```

erzeugt eine 5-stufige Reihung mit 10 mal 10 mal 10 mal 10 mal 10 gleich
100 Tausend `String`-Variablen.

14.6 Anmerkungen (engl. annotations)

Mit der Version 5 von Java wurden (nicht nur generische Typen, sondern) auch *Anmerkungen* eingeführt. Eine Anmerkung ist eine Art *formaler Kommentar*, den man in Form eines Objekts mit einem bestimmten Programmteil (einer Klasse, einer Methode, einem Attribut etc.) verbinden kann.

Werkzeuge, die Java-Programme bearbeiten (z.B. Compiler, Editoren, Prüfprogramme, ...) können die Anmerkungen erkennen und darauf reagieren.
Beispiel: Die Anmerkung `@Override` vor einer Objektmethode.

Welche Programmteile kann man mit Anmerkungen versehen:

Anmerkungstypen, Konstruktoren, Attribute, lokale Variablen von Methoden und Konstruktoren, Methoden, Pakete, Parameter von Methoden und Konstruktoren und alle Typen
(Klassen, Schnittstellen, Aufzählungstypen und die bereits erwähnten Anmerkungstypen)

Anmerkungstypen

Bevor man Programmteile mit Anmerkungen versehen kann, muss man sich geeignete *Anmerkungstypen* (engl. annotation types) verschaffen. Ein Anmerkungstyp ist ein Untertyp des Typs `java.lang.Annotation` (der selbst aber *kein* Anmerkungstyp ist).

In Java 6 gib es etwa 70 vordefinierte Anmerkungstypen (in den Paketen `java.lang.annotation` und `javax.annotation`). Der Programmierer kann eigene Anmerkungstypen vereinbaren, aber nicht, indem er den Typ `Annotation` erweitert, sondern mit einer speziellen Syntax.

Rückseite der Wiederholungsfragen,
Beispiel-01: Drei Anmerkungstypen

`AutorArno` ist ganz einfach, `Autor` ist einfach und `Hersteller` ist ein "normaler" Anmerkungstyp

Hat man geeignete Anmerkungstypen, kann man Programmteile mit Anmerkungen versehen.

Beispiel-02: Eine Klasse, deren Teile mit Anmerkungen versehen sind

Mit welchen Anmerkungen wurde die Klasse `AnmerkA` versehen?

Die Methode `druckeA`? Die Methode `druckeB`? Das Attribut `text1`? Das Attribut `text2`?

Die Attribute einer Anmerkung

Eine Anmerkung ist ein Objekt mit privaten Attributen (z.B. `value`, `firma`, `jahr` etc.) und öffentlichen getter-Methoden (`value()`, `firma()`, `jahr()` etc.).

Eigenschaften von Anmerkungen werden mit Anmerkungen festgelegt

Beispiel-04, die Anmerkungseigenschaft `Target`

Beispiel-05, die Anmerkungseigenschaft `Retention`

Wiederholungsfragen, 18. SU, Mo 31.01.11

1. Nennen Sie ein mind 3 Bestandteile eines Java-Programms, die man mit *Anmerkungen* (oder: Annotationen, engl. annotations) versehen kann.

Betrachten Sie die folgenden Vereinbarungen von drei Anmerkungstypen:

```
 1  public @interface Copyleft {}
 2
 3  public @interface Schwierigkeitsgrad {
 4     int value();
 5  }
 6
 7  public @interface Kontakt {
 8     String email() default "fb06@bueth-hochschule.de";
 9     int    fon();
10  }
```

2. Vereinbaren Sie ein privates Objekt-Attribut namens `roman01` vom Typ `StringBuilder` mit dem Anfangswert `null` und versehen Sie dieses Attribut mit einer Anmerkung des Typs `Copyleft`.

3. Vereinbaren Sie eine öffentliche Klassen-Methode namens `istPrim` mit dem Rückgabetyp `boolean` und einem `int`-Parameter und versehen Sie diese Methode mit einer Anmerkung des Typs `Schwierigkeitsgrad`. Es genügt, wenn Sie den Rumpf der Methode durch eine *Auslassung* `...` andeuten.

4. Vereinbaren Sie eine öffentliche Klasse namens `Fehler17` als Erweiterung der Klasse `Error` und mit leerem Rumpf `{}`. Versehen Sie diese Klasse mit einer Anmerkung des Typs `Kontakt`.

5. Angenommen, Sie vereinbarben einen Anmerkungstyp namens `MeinKommentar` und versehen diesen Typ mit der folgenden Anmerkung:

```
@Target (ElementType.FIELD, ElementType.LOCAL_VARIABEL)
public @interface MeinKommentar { ... }
```

Was bewirkt die Anmerkung des Typs `Target`?

6. Angenommen, Sie vereinbarben einen Anmerkungstyp namens `OttosKommentar` und versehen diesen Typ mit der folgenden Anmerkung:

```
@Retention (RetentionPolicy.RUNTIME)
public @interface OttosKommentar { ... }
```

Was bewirkt die Anmerkung des Typs `Retention`?

Rückseite der Wiederholungsfragen, 18. SU, Mo 31.01.11

Wahrscheinlichkeiten, dass zwei Personen am selben Tag Geburtstag haben

Wie groß ist die Wahrscheinlichkeit, dass von N zufällig gewählten Personen (z. B. den Gästen einer Party) mindestens zwei am gleichen Tag im Jahr (z. B. am 7. März oder am 23. August etc.) Geburtstag haben? Die Wahrscheinlichkeit ist wesentlich höher, als viele Menschen "gefühlsmäßig vermuten".

Anzahl Gäste auf einer Party	Wahrscheinlichkeit, dass mindestens 2 am gleichen Tag im Jahr Geburtstag haben
1	0,000
2	0,003
3	0,008
4	0,016
5	0,027
6	0,040
7	0,056
8	0,074
9	0,095
10	0,117
11	0,141
12	0,167
13	0,194
14	0,223
15	0,253
16	0,284
17	0,315
18	0,347
19	0,379
20	0,411
21	0,444
22	0,476
23	0,507
24	0,538
25	0,569
26	0,598
27	0,627
28	0,654
29	0,681
30	0,706
31	0,730
32	0,753
33	0,775
34	0,795
35	0,814
36	0,832
37	0,849
38	0,864
39	0,878
40	0,891

Anzahl Gäste auf einer Party	Wahrscheinlichkeit, dass mindestens 2 am gleichen Tag im Jahr Geburtstag haben
41	0,903
42	0,914
43	0,924
44	0,933
45	0,941
46	0,948
47	0,955
48	0,961
49	0,966
50	0,970
51	0,974
52	0,978
53	0,981
54	0,984
55	0,986
56	0,988
57	0,990
58	0,992
59	0,993
60	0,994
61	0,995
62	0,996
63	0,997
64	0,997
65	0,998
66	0,998
67	0,998
68	0,999
69	0,999
70	0,999
71	0,999
72	0,999
73	1,000
74	1,000
75	1,000
76	1,000
77	1,000
78	1,000
79	1,000
80	1,000

Antworten zu den Wiederholungsfragen, 18. SU, Mo 31.01.11

1. Nennen Sie ein mind 3 Bestandteile eines Java-Programms, die man mit *Anmerkungen* (oder: Annotationen, engl. annotations) versehen kann.

Anmerkungstypen, Konstruktoren, Attribute, lokale Variablen von Methoden und Konstruktoren, Methoden, Pakete, Parameter von Methoden und Konstruktoren und alle Typen (Klassen, Schnittstellen, Aufzählungstypen und die bereits erwähnten Anmerkungstypen)

Betrachten Sie die folgenden Vereinbarungen von drei Anmerkungstypen:

```
 1  public @interface Copyleft {}
 2
 3  public @interface Schwierigkeitsgrad {
 4     int value();
 5  }
 6
 7  public @interface Kontakt {
 8     String email() default "fb06@bueth-hochschule.de";
 9     int    fon();
10  }
```

2. Vereinbaren Sie ein privates Objekt-Attribut namens `roman01` vom Typ `StringBuilder` mit dem Anfangswert `null` und versehen Sie dieses Attribut mit einer Anmerkung des Typs `Copyleft`.

```
11  @Copyleft
12  private StringBuilder roman01 = null;
```

3. Vereinbaren Sie eine öffentliche Klassen-Methode namens `istPrim` mit dem Rückgabetyp `boolean` und einem `int`-Parameter und versehen Sie diese Methode mit einer Anmerkung des Typs `Schwierigkeitsgrad`. Es genügt, wenn Sie den Rumpf der Methode durch eine *Auslassung* `...` andeuten.

```
13  @Schwierigkeitsgrad(7) // oder: @Schwierigkeitsgrad(value=7)
14  static public boolean istPrim(int n) { ... }
```

4. Vereinbaren Sie eine öffentliche Klasse namens `Fehler17` als Erweiterung der Klasse `Error` und mit leerem Rumpf `{}`. Versehen Sie diese Klasse mit einer Anmerkung des Typs `Kontakt`.

```
15  @Kontakt(email="info@gmx.net", fon=8835566)
16  public class Fehler17 extends Error {}
```

5. Angenommen, Sie vereinbarben einen Anmerkungstyp namens `MeinKommentar` und versehen diesen Typ mit der folgenden Anmerkung:

```
@Target (ElementType.FIELD, ElementType.LOCAL_VARIABEL)
public @interface MeinKommentar { ... }
```

Was bewirkt die Anmerkung des Typs `Target`?

Dass nur Attribute und lokale Variablen (Variablen die in einer Methode oder in einem Konstruktor vereibart werden) mit Anmerkungen des Typs MeiKommentar versehen werden dürfen.

6. Angenommen, Sie vereinbarben einen Anmerkungstyp namens `OttosKommentar` und versehen diesen Typ mit der folgenden Anmerkung:

```
@Retention (RetentionPolicy.RUNTIME)
public @interface OttosKommentar { ... }
```

Was bewirkt die Anmerkung des Typs Retention?

Dass Anmerkungen des Typs OttosKommentar (aus der Quelldatei) in die .class-Datei übernommen werden und auch während der Ausführung des betreffenden Programms zur Verfügung stehen.

18. SU, Mo 31.01.11

A. Wiederholung
B. Organisation: Wer nicht ganz sicher ist, ob er zur Klausur zugelassen ist oder nicht, kann nach dieser Vorlesung oder in der Übung im zweiten Block in meiner Liste nachsehen.

Zur Organisation der Klausur

Als Unterlagen darf jeder mitbringen: 5 Blätter, max. DIN A 4, Beschriftung beliebig

Bringen Sie zusätzlich ausreichend viele (z.B. 20) leere Blätter Papier mit (für ihre Lösungen).

Sehr gut geeignet ist kariertes, ganz weißes, "radierfestes" Papier, *kein graues Umweltpapier*!

Sie können schon zu Hause einige der leeren Blätter (z.B. rechts oben) mit ihrem Nachnamen kennzeichnen (weitere Angaben wie Vorname, Kragenweite, Matrikelnr, Schuhgröße etc. sind erlaubt, aber nicht nötig).

Am Ende der Klausur sollte jedes Blatt, das sie abgeben, mit ihrem *Nachnamen* gekennzeichnet sein.

Sonderregel für Teilnehmer namens Schmidt oder Schäfer:
Sie müssen ihre Bläter mit Vornamen und Nachnamen kennzeichnen!

Während der Klausur:

Schreiben Sie zuerst ihren Vor- und Nachnamen auf das ausgeteilte Blatt mit den Klausuraufgaben und machen Sie hinter "Diese Klausur ist mein letzter Prüfungsversuch!" an der richtigen Stelle (bei Ja bzw. Nein) ein Kreuzchen.

Schreiben Sie jede Lösung einer Aufgabe auf ein neues Blatt (nicht mehrere Lösungen auf dasselbe Blatt).

Beschreiben Sie von ihren Blättern nur die Vorderseiten und lassen sie die Rückseiten leer.

Es gibt 6 Aufgaben. Normalerweise sollte jede Lösung einer Aufgabe auf eine Seite passen. Aber wenn sie mehr Platz benötigen, dann schreiben Sie die Lösung auf die Vorderseiten von mehreren Blättern.

Ziel: Es soll möglich sein, ihre Lösungen so auf einen Tisch zu legen, dass man sie alle und vollständig sehen kann, ohne Blätter rumdrehen zu müssen.

Tip: Überfliegen Sie erstmal alle Aufgaben, damit Sie ein Gefühl dafür bekommen, welche sie leichter und welche sie schwerer finden. Lösen Sie dann die Aufgaben in aufsteigender Reihenfolge ihrer Schwierigkeit (erst die einfachste, dann die zweiteinfachste etc.).

Bevor sie daran gehen, eine bestimmte Aufgabe zu lösen, sollten sie den Text der Aufgabe vollständig durchlesen (bis zum letzten Punkt!). Möglicherweise müssen sie bestimmte Stellen des Textes mehrmals lesen, um sie zu verstehen.

Prüfen Sie dann, ob sie die Aufgabe unzweifelhaft verstanden haben. Wenn sie unsicher sind, wie die Aufgabe "gemeint ist" oder "verstanden werden sollte", dann heben sie ihren linken Arm. Ich komme dann zu ihnen und sie können dann Fragen der Art "Ist das hier so gemeint?" stellen.

Insgesamt haben sie 90 Minuten für 6 Augaben. Das sind etwa 15 Minuten pro Aufgabe.

Rückgabe der Klausur: Fr 11.02.2011, 8.30 Uhr im SWE-Labor

Gibt es Fragen zum Stoff dieses Semester?

Ein paar Bemerkungen zur Klausur:

6 Aufgaben. Insgesamt 100 Punkte.

Aufgabe 1 (25 Punkte): Besteht aus *zwei Teilaufgaben* der Form:
"Schreiben Sie eine Methode entsprechend der folgenden Spezifikation: ..."

Aufgabe 2 (15 Punkte)
"Schreiben Sie eine Methode entsprechend der folgenden Spezifikation: ..."

Für die übrigen 4 Aufgaben gibt es je 15 Punkte.

Was bedeutet "verlässt sich darauf"?

Angenommen, Sie sollen eine Methode mit einem `int`-Parameter namens `n` schreiben und im Anfangskommentar steht:

```
"Verlaesst sich darauf, dass n groesser als 0 ist."
```

(Dass die Methode *nicht* prüft, ob n größer als 0 ist, sondern sich darauf verläßt, dass die *Aufrufer* dafür sorgen).

Grundsätzlich gilt: Sie dürfen alle nützlichen import-Befehle *voraussetzen* und alle Standardklassen mit ihren *einfachen Namen* bezeichnen (z.B. `ArrayList` oder `Arrays` etc. statt `java.util.ArrayList` oder `java.util.Arrays` etc.)

Es schadet wahrscheinlich nicht, mit folgenden Stoffgebieten gut vertraut zu sein:

Wiederholungsfragen

Rekursion

Grundlagen von XML

Generische Klassen und generische Funktionen

Die Sammlungsstrukturen *Verkettete Liste* und *Baum*

insbesondere die konkreten Codebeispiele `MeineListe<K>` und `MeinBaum`
(in der Datei `UebungenPR2.pdf` auf meiner Netzseite)

printf

Reflektion

Zur Entspannung: Haben von den Anwesenden zufällig zwei am selben Tag Geburtstag?

Ein Jahr hat rund 360 Tage. Dann ist die Wahrscheinlichkeit, dass von 60 Personen zwei am selben Tag Geburtstag haben, ungefähr 1/6, als etwa 16% oder 0,16. Oder etwa nicht?

Tatsächlich gilt: Obige Rechnung ist völlig falsch. Schon bei 23 Personen ist die Wahrscheinlichkeit eines Doppelgeburtstags etwa 50% und bei 60 Personen ist sie etwas größer als 99%. Siehe dazu auch die Tabelle Wahrscheinlichkeiten, dass von N Personen zwei am selben Tag im Jahr Geburtstag haben.

Mit diesem Beispiel demonstrieren Psychologen und Statistiker gern, dass sich "unser Gefühl" beim Abschätzen von Wahrscheinlichkeit in einigen Fällen erstaunlich weit von der mathematischen Realität entfernt.

Siehe auch: http://quiz.worldofuncertainty.org
Da kann man sein "statistisches Gefühl" trainieren.