Übungen für MB2-PR2

Übung01: Rekursion

1. Ersetzen Sie in den Methoden der folgenden Klasse die "unsinnigen `return`-Befehle" durch richtige Befehle. Alle Methoden sollen rekursiv arbeiten.

```
 1 class UebReku {
 2    // ------------------------------------------------------------------------
 3    static public int potenzR(final int BASIS, final int EXPONENT) {
 4       // Wirft eine Ausnahme, falls der EXPONENT negativ ist.
 5       // Liefert sonst die Zahl BASIS hoch EXPONENT.
 6       if (EXPONENT<0) throw new Error (
 7          "potenzR: Negativer EXPONENT " + EXPONENT + " ist nicht erlaubt!"
 8       );
 9       return -1; // MUSS ERSETZT WERDEN!
10    } // potenzR
11    // ------------------------------------------------------------------------
12    static public int multR(final int N1, final int N2) {
13       // Liefert das Produkt von N1 und N2.
14       if (N1 <  0) return -multR(-N1, N2); // Pseudo-Rekursion
15       return -1; // MUSS ERSETZT WERDEN!
16    } // multR
17    // ------------------------------------------------------------------------
18    static public int anz10erZiffernR(final int N) {
19       // Wieviele Ziffern braucht man, um N als Zahl im System mit der
20       // Basis 10 darzustellen? Diese Funktion liefert die Antwort.
21       return -1; // MUSS ERSETZT WERDEN!
22    } // anz10erZiffernR
23    // ------------------------------------------------------------------------
24    static public int anzBerZiffernR(final int N, final int B) {
25       // Wieviele Ziffern braucht man, um N als Zahl im System mit der
26       // Basis |B| (Betrag von B) darzustellen? Diese Funktion liefert
27       // die Antwort.
28       return -1; // MUSS ERSETZT WERDEN!
29    } // anzBerZiffernR
30    // ------------------------------------------------------------------------
31    static public int quersumme10erR(final int N) {
32       // Welche Quersumme hat die Zahl N, wenn man sie im System mit der
33       // Basis 10 ("als Dezimalzahl") darstellt? Diese Funktion liefert
34       // die Antwort.
35       return -1; // MUSS ERSETZT WERDEN!
36    } // quersumme10erR
37    // ------------------------------------------------------------------------
38    static public int quersummeBerR(final int N, final int B) {
39       // Welche Quersumme hat die Zahl N, wenn man sie im System mit der
40       // Basis |B| ("als B-er-Zahl") darstellt? Diese Funktion liefert
41       // die Antwort.
42       return -1; // MUSS ERSETZT WERDEN!
43    } // quersummeBerR
44    // ------------------------------------------------------------------------
45    static public String drehRumR(final String S) {
46       // Liefert einen String, der die gleichen Zeichen enthaelt wie S,
47       // aber in umgekehrter Reihenfolge. Beispiel:
48       // drehRumR("ABC") ist gleich "CBA".
49       return "Fehler!"; // MUSS ERSETZT WERDEN!
50    } // drehRumR
51    // ------------------------------------------------------------------------
52    static public boolean istPalindromR(final String S) {
53       // Liefert true genau dann wenn S ein Palindrom ist.
54       // Beispiele fuer Palindrome: "ANNA", "OTTO", ABCBA".
55       return false; // MUSS ERSETZT WERDEN!
56    } // istPalindromR
57    // ------------------------------------------------------------------------
58    static public int anzSchoeneZeichenR(final String S) {
59       // Liefert die Anzahl der schoenen Zeichen in S. Welche Zeichen
60       // schoen sind, wird durch die Funktion istSchoen festgelegt.
61       return -1; // MUSS ERSETZT WERDEN!
```

```
 62    } // anzSchoeneZeichenR
 63
 64    static public boolean istSchoen(char c) {
 65       // Liefert true, genau dann wenn c eine 10-er-Ziffer ist.
 66       return '0' <= c && c <= '9';
 67    } // istSchoen
 68    // ---------------------------------------------------------------------
 69    // Die Funktion anzGeradeZahlen  hat 1 Parameter und ist nicht-rekursiv.
 70    // Die Funktion anzGeradeZahlenR hat 2 Parameter und ist       rekursiv.
 71
 72    static public int anzGeradeZahlen(int[] ir) {
 73       // Liefert die Anzahl der geraden Zahlen in ir.
 74       return anzGeradeZahlenR(ir, 0);
 75    } // anzGeradeZahlen
 76
 77    static public int anzGeradeZahlenR(int[] ir, int ind) {
 78       // Liefert die Anzahl der geraden Zahlen, die ab dem Index ind
 79       // in ir stehen (ir-Komponenten mit Indizes kleiner als ind werden
 80       // also nicht beachtet).
 81       return -1; // MUSS ERSETZT WERDEN!
 82    } // anzGeradeZahlenR
 83
 84    static public boolean istGerade(int n) {
 85       return n % 2 == 0;
 86    } // istGerade
 87    // ---------------------------------------------------------------------
 88    static int anzFiboR = 0; // Wie oft wird fiboR aufgerufen?
 89
 90    static public long fiboR(final int N) {  // Fibonacci-Zahlen, rekursiv
 91       // Diese Funktion liefert die N-te Fibonacci-Zahl (und zaehlt in der
 92       // Variablen anzFiboR wie oft sie aufgerufen wurde)
 93       // N sollte groesser oder gleich 0 sein. Alle negativen Zahlen
 94       // werden (der Einfachheit halber) wie die Zahl 0 behandelt.
 95
 96       anzFiboR++;
 97
 98       return -1; // MUSS ERSETZT WERDEN!
 99    } // fiboEinfachR
100    // ---------------------------------------------------------------------
101    static public long fiboQ(final int N) {  // Fibonacci-Zahlen, quick
102       // Liefert die N-te Fibonacci-Zahl. Erzeugt dazu eine long-Reihung
103       // fibo der Laenge N+1 und schreibt die i-te-Fibonacci-Zahl in die
104       // Komponente fibo[i].
105       return -1; // MUSS ERSETZT WERDEN!
106    } // fiboQ
107    // ---------------------------------------------------------------------
108    static public long fiboD(final int N) {  // Fibonacci-Zahlen, direkt
109       final double W5 = Math.sqrt(5.0);
110
111       final double A = Math.pow(1.0 + W5, N);
112       final double B = Math.pow(1.0 - W5, N);
113       final double C = Math.pow(   2    , N) * W5;
114
115       return (long) ((A - B) / C);
116    } // fiboD
117    // ---------------------------------------------------------------------
118 } // class UebReku
```

Diesen Java-Quelltext (ergänzt um einige Befehle für Testausgaben) finden Sie auch in der Datei `UebReku.java` im Archiv `DateienFuerPr2.zip`.

Übung02: Generische Methoden

Üb02-1. Zwei Beispielprogramme zum Lesen und besprechen:

```
 1 // Datei Vergleichen02.java
 2 /* ---------------------------------------------------------------------
 3 Die generische Methode sortA verlangt, dass ihr Typ-Parameter T
 4 genau die eine Schnittstelle Comparable<T> implementiert.
 5
 6 Die generische Methode sortB verlangt nur, dass ihr Typ-Parameter T
 7 eine der Schnittstellen Comparable<? super T> implementiert.
 8
 9 Die Klasse StudiBht implementiert die Schnittstelle Comparable<StudiBht>.
10 Reihungen mit StudiBht-Komponenten koennen mit sortA und mit sortB
11 bearbeitet werden.
12
13 StudiFb6 ist eine Unterklasse von StudiBht.
14 Reihungen mit StudiFb6-Komponenten koennen mit sortB, aber nicht mit sortA
15 bearbeitet werden.
16 --------------------------------------------------------------------- */
17 import java.util.Arrays;
18
19 class StudiBht implements Comparable<StudiBht> {
20    int mnr; // Matrikel-Nr, einziges Attribut
21    public        StudiBht (int mnr)       {this.mnr = mnr;}
22    public int    compareTo(StudiBht that) {return this.mnr - that.mnr;}
23    public String toString ()              {return "Bht " + mnr;   }
24 }
25
26 class StudiFb6 extends StudiBht {
27    public        StudiFb6(int mnr) {super(mnr);}
28    public String toString()        {return "Fb6 " + mnr;}
29 }
30
31 public class Vergleichen02 {
32    // ------------------------------------------------------------------
33    static public <T extends Comparable<T>>
34    void sortA(T[] tr) {
35       pln("sortA sortiert: " + Arrays.toString(tr));
36    }
37    // ------------------------------------------------------------------
38    static public <T extends Comparable<? super T>>
39    void sortB(T[] tr) {
40       pln("sortB sortiert: " + Arrays.toString(tr));
41    }
42    // ------------------------------------------------------------------
43    static public void main(String[] _) {
44       pln("Vergleichen02: Jetzt geht es los!");
45
46       StudiBht[] r01 = {new StudiBht(11), new StudiBht(12)};
47       StudiBht[] r02 = {new StudiBht(21), new StudiBht(22)};
48       StudiFb6[] r03 = {new StudiFb6(31), new StudiFb6(32)};
49
50       sortA(r01);
51       sortA(r02);
52 //    sortA(r03); // sortA kann nicht auf r03 angewendet werden
53
54       sortB(r01);
55       sortB(r02);
56       sortB(r03);
57    } // main
58    // ------------------------------------------------------------------
59    // Eine Methode mit einem kurzen Namen:
60    static void pln(Object ob) {System.out.println(ob);}
61    // ------------------------------------------------------------------
62 } // class Vergleichen02
63 /* ---------------------------------------------------------------------
```

```
64 Fehlermeldung des Sun-Compilers (wenn Zeile 52 kein Kommentar ist):
65
66 D:\meineDateien\BspJava\Vergleichen02.java:52:
67 <T>sortA(T[]) in Vergleichen02 cannot be applied to (StudiFb6[])
68       sortA(r03); // sortA kann nicht auf r03 angewendet werden
69       ^
70 ---------------------------------------------------------------------- */
```

```
 1 // Datei Vergleichen03.java
 2 /* ---------------------------------------------------------------------
 3 Die generische Methode sortA mit dem Typparmeter <T> erwartet als Parameter
 4 eine Reihunge tr mit Komponenten des Typs T und ein Comparator-Objekt c,
 5 dessen compare-Methode zwei Parameter vom Typ T hat.
 6
 7 Die generische Methode sortB unterscheidet sich nur wenig von sortA:
 8 die Methode c.compare muss zwei Paramter von einem Obertyp von T haben.
 9
10 Das Comparator<StudiBht>-Objekt ctor enthaelt eine compare-Methode mit
11 zwei Parametern des Typs StudiBht.
12
13 Reihungen mit StudiBht-Komponenten kann man (zusammen mit dem Objekt ctor)
14 mit den Methoden sortA und sortB bearbeiten.
15
16 Reihungen mit StudiFb6-Komponenten kann man (zusammen mit dem Objekt ctor)
17 nur mit sortB bearbeiten, aber nicht mit sortA.
18 ---------------------------------------------------------------------- */
19 import java.util.Arrays;
20 import java.util.Comparator;
21
22 class CtorBht implements Comparator<StudiBht> {
23    public int compare(StudiBht s1, StudiBht s2) {
24       return s1.mnr - s2.mnr;
25    }
26 }
27 public class Vergleichen03 {
28    // ---------------------------------------------------------------------
29    static public <T>
30    void sortA(T[] tr, Comparator<T> c) {
31       pln("sortA sortiert: " + Arrays.toString(tr));
32    }
33    // ---------------------------------------------------------------------
34    static public <T>
35    void sortB(T[] tr, Comparator<? super T> c) {
36       pln("sortB sortiert: " + Arrays.toString(tr));
37    }
38    // ---------------------------------------------------------------------
39    static public void main(String[] _) {
40       pln("Vergleichen03: Jetzt geht es los!");
41
42       StudiBht[] r01 = {new StudiBht(11), new StudiBht(12)};
43       StudiBht[] r02 = {new StudiBht(21), new StudiBht(22)};
44       StudiFb6[] r03 = {new StudiFb6(31), new StudiFb6(32)};
45
46       CtorBht ctor = new CtorBht();
47
48       sortA(r01, ctor);
49       sortA(r02, ctor);
50 //    sortA(r03, ctor); // sortA kann nicht auf r03 angewendet werden
51
52       sortB(r01, ctor);
53       sortB(r02, ctor);
54       sortB(r03, ctor);
55    } // main
56    // ---------------------------------------------------------------------
57    ...
58 } // class Vergleichen03
```

Üb02-2. Ein Beispiel-Programm mit Fragen dazu:

```
 1 // Datei GenUebung01.java
 2
 3 class KA  {
 4    int wert;
 5    public     KA       (int wert) {this.wert = wert;}
 6 }
 7
 8 class KAA extends KA implements Comparable<KAB>{
 9    public     KAA      (int wert) {super(wert);}
10    public int compareTo(KAB that) {return -1;}
11 }
12
13 class KAB extends KA implements Comparable<KAB> {
14    public     KAB      (int wert) {super(wert);}
15    public int compareTo(KAB that) {return that.wert - this.wert;}
16 }
17
18 class KABA extends KAB {
19    public     KABA     (int wert) {super(wert);};
20    public int compareTo(KAB that) {return this.wert - that.wert;}
21 }
22
23 class KABB extends KAB {
24    public     KABB     (int wert) {super(wert);};
25    public int compareTo(KAB that) {return that.wert - this.wert;}
26 }
27
28 public class GenUebung01 {
29    // ------------------------------------------------------------------------
30    static <X>                                    void metA(X x) {}
31    static <X extends KAB>                        void metB(X x) {}
32    static <X extends Comparable<X>>              void metC(X x) {}
33    static <X extends Comparable<? extends KAB>> void metD(X x) {}
34    static <X extends Comparable<? super   X>>   void metE(X x) {}
35    // ------------------------------------------------------------------------
36    static public void main(String[] _) {
37       pln("GenUebung01: Jetzt geht es los!");
38
39       KA   ka   = new KA  (10);
40       KAA  kaa  = new KAA (20);
41       KAB  kab  = new KAB (30);
42       KABA kaba = new KABA(40);
43       KABB kabb = new KABB(50);
44
45       metA(ka  );   // Aufruf 01
46       metA(kaa );   // Aufruf 02
47       metA(kab );   // Aufruf 03
48       metA(kaba);   // Aufruf 04
49       metA(kabb);   // Aufruf 05
50
51       metB(ka  );   // Aufruf 06
52       metB(kaa );   // Aufruf 07
53       metB(kab );   // Aufruf 08
54       metB(kaba);   // Aufruf 09
55       metB(kabb);   // Aufruf 10
56
57       metC(ka  );   // Aufruf 11
58       metC(kaa );   // Aufruf 12
59       metC(kab );   // Aufruf 13
60       metC(kaba);   // Aufruf 14
61       metC(kabb);   // Aufruf 15
62
63       metD(ka  );   // Aufruf 16
64       metD(kaa );   // Aufruf 17
65       metD(kab );   // Aufruf 18
```

```
66       metD(kaba);   // Aufruf 19
67       metD(kabb);   // Aufruf 20
68
69       metE(ka  );   // Aufruf 21
70       metE(kaa );   // Aufruf 22
71       metE(kab );   // Aufruf 23
72       metE(kaba);   // Aufruf 24
73       metE(kabb);   // Aufruf 25
74
75       pln("GenUebung01: Das war's erstmal!");
76    } // main
77    // ---------------------------------------------------------------------
78    // Eine Methode mit einem kurzen Namen:
79    static void pln(Object ob) {System.out.println(ob);}
80    // ---------------------------------------------------------------------
81 } // class GenUebung01
```

Üb02-2-1: Wie sieht der Typgraf der Klassen KA, KAA, KAB, KABA, KABB aus? Zeichnen Sie auch den Schnittstellentyp `Comparable<KAB>` ein (ähnlich wie im Buch auf S. 347, Bild 14.4).

Üb02-2-2: Welche der Methodenaufrufe `Aufruf 01` bis `Aufruf 25` sind *erlaubt* und welche sind *nicht erlaubt* (weil der Typ des aktuellen Parameters nicht passend ist)?

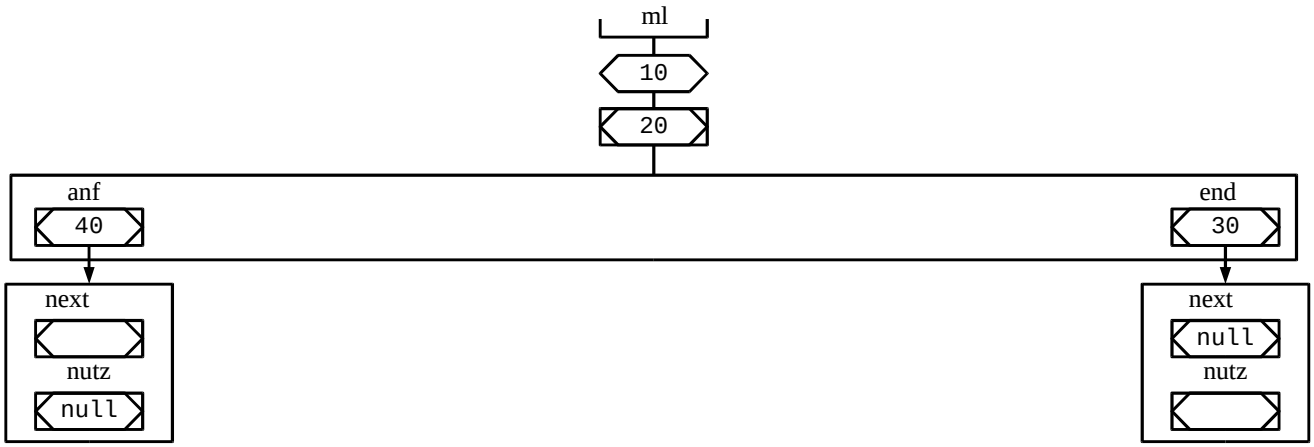
Übung03: Verkettete Listen

```
 1 // Datei MeineListe.java
 2 /* ---------------------------------------------------------------------
 3 Jedes Objekt dieser Klasse ist eine Sammlung mit folgenden Methoden:
 4 fuegeEin, istDrin, entferne und toString.
 5 --------------------------------------------------------------------- */
 6 public class MeineListe<K> {
 7    // ------------------------------------------------------------------
 8    private class Knoten {
 9       Knoten next;
10       K      nutz;
11
12       public Knoten(Knoten next, K nutz) {
13          this.next = next;
14          this.nutz = nutz;
15       } // Konstruktor Knoten
16    } // class Knoten
17    // ------------------------------------------------------------------
18    Knoten end = new Knoten(null, null);
19    Knoten anf = new Knoten(end,  null);
20    // ------------------------------------------------------------------
21    public void fuegeEin(K nutz) {anf.next = new Knoten(anf.next, nutz);}
22    // ------------------------------------------------------------------
23    private Knoten sucheVor(K nutz) {
24       // Haengt die Nutzdaten nutz an den end-Knoten. Liefert den
25       // Vorgaengerknoten eines Knotens, an dem nutzt haengt.
26       end.nutz = nutz;  // Suchbegriff in den Endknoten schreiben
27       Knoten vor = anf;
28       while (true) {
29          if (nutz.equals(vor.next.nutz)) return vor;
30          vor = vor.next;
31       }
32    } // sucheVor
33    // ------------------------------------------------------------------
34    public boolean istDrin(K nutz) {
35       // Liefert true wenn nutz in dieser Sammlung vorkommt, sonst false.
36       ... // DEN FEHLENDEN RUMPF SOLLEN SIE ERGAENZEN!
37    } // istDrin
38    // ------------------------------------------------------------------
39    public boolean entferne(K nutz) {
40       // Wenn nutz an mind. einem Knoten dieser Liste haengt, wird einer
41       // dieser Knoten geloescht und true geliefert.
42       // Sonst wird false geliefert.
43       ... // DEN FEHLENDEN RUMPF SOLLEN SIE ERGAENZEN!
44    } // entferne
45    // ------------------------------------------------------------------
46    public String toString() {
47       // Liefert eine String-Representation dieser Sammlung, z.B.
48       // "[Hallo, Sonja, Wie geht's?]" oder "[10, 20, 30]" oder "[]" etc.
49       Knoten hier = anf.next;
50       if (hier == end) return ("[]");
51       StringBuilder sb = new StringBuilder("[");
52
53       sb.append(hier.nutz);           // 1. Kompo ohne Trennzeichen nach sb
54       while (true) {                  // Alle Kompos ab der 2. bearbeiten
55          hier = hier.next;
56          if (hier == end) break;
57          sb.append(", " + hier.nutz); // Trennzeichen und Kompo nach sb
58       }
59       sb.append("]");
60       return sb.toString();
61    } // toString
62    // ------------------------------------------------------------------
63 } // class MeineListe
```

Betrachten Sie die folgenden Befehle:

```
1  MeineListe<String> ml = new MeineListe<String>();
2  ml.fuegeEin("Hallo");
3  ml.fuegeEin("Sonja!");
4  ml.fuegeEin("Wie geht's?");
```

Nach Ausführung von Zeile 1 sieht die Variable `ml` in vereinfachter Bojendarstellung etwa so aus:



Üb03-0: Tragen Sie *mit Bleistift (!)* die Anfangswerte der Variablen `anf.next` und `end.nutz` ein.

Üb03-1: Wie viele Knoten enthält eine leere Sammlung des Typs `MeineListe<String>`?

Üb03-2: Wie sieht die Bojendarstellung von `ml` aus, nachdem die Zeile 4 fertig ausgeführt ist?

Üb03-3: Führen Sie (auf der Grundlage der obigen, von Ihnen erweiterten Bojendarstellung) den folgenden Befehl ("mit Papier und Bleistift") aus:

```
5  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:

```
6  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?

Übung-04: Binäre Bäume

Def.: Ein binärer Baum
ist entweder ein *leerer binärer Baum* oder
er besteht aus einem *Knoten* K, an dem zwei *binäre Bäume*, hängen,
die man auch als *linken Unterbaum* und *rechten Unterbaum* von K bezeichnet.

Ein paar Fachbegriffe: Sei K irgendein Knoten eines Baumes B.

Mutter	(-Knoten) von K:	Der Knoten, an dem K dranhängt.
Tochter	(-Knoten) von K:	Die (maximal 2) Knoten, die an K dranhängen
Wurzel	(-Knoten) von B:	Der oberste Knoten von B (der einzige der keine Mutter hat)
Blatt	(-Knoten) von B:	Ein Knoten von B, an dem 2 leere Bäume hängen
Innerer Knoten	von B:	Ein Knoten von B, der eine Mutter und mindestens eine Tochter hat.

Wenn man Bäume als *Sammlungen* verwendet (und nicht etwa als Struktur eines Dateisystems oder als Struktur einer XML-Datei oder zum Heizen oder ...), müssen sie *sortiert* sein. Damit ein Baum sortiert sein kann, müssen seine Knoten *vergleichbar* sein (in Java: Über die Schnittstelle `Comparable` oder die Schnittstelle `Comparator`).

Def.: Ein binärer Baum ist *sortiert*, wenn für jeden Knoten K gilt:
Alle Knoten im linken Unterbaum von K sind kleiner als K und
alle Knoten im rechten Unterbaum von K sind größer als K.

Eine besonders simple Implementierung von Bäumen beruht auf der Idee, die Knoten in einer Reihung `rei` zu speichern. Die Komponente `rei[0]` wird nicht benutzt.
Die Komponente `rei[1]` ist die Wurzel des Baumes.
Sei `rei[i]` irgendein Knoten des Baumes. Dann gilt:

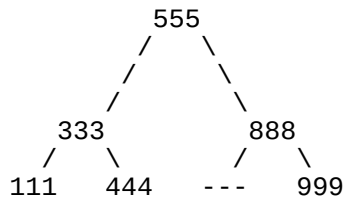
Der linke Unterbaum von `rei[i]` beginnt bei `rei[2*i]`.
Der rechte Unterbaum von `rei[i]` beginnt bei `rei[2*i+1]`
Die Mutter von `rei[i]` ist `rei[i/2]` (nur wenn `i/2` ungleich 0 ist, d.h. wenn `i>1` ist).

Beispiel:

Ein Baum als *Reihung* `rei` dargestellt:

Index i	0	1	2	3	4	5	6	7
rei[i]	unbenutzt	555	3333	888	111	444	---	999

Der gleiche Baum als Baum dargestellt:



Der Knoten 888 hat einen linken Unterbaum der leer ist (`---`) und einen rechten Unterbaum der mit dem Knoten 999 beginnt. Die leeren Unterbäume der Blätter 111, 444, 999 sind hier *nicht* eingezeichnet (Grund: Faulheit :-).

Üb04-1: Fügen Sie (mit Papier und Bleistift) die folgenden Knoten in der angegebenen Reihenfolge in einen (anfangs leeren) sortierten binären Baum ein: 53, 78, 75, 27, 40, 55, 13, 89, 79
Dabei sollen Sie den Baum wie einen *Baum* darstellen (nicht als Reihung!).

Eine besonders simple Implementierung einer Klasse von binären Bäumen:

```
 1 // Datei MeinBaum.java
 2 /* ---------------------------------------------------------------------
 3 Jedes Objekt dieser Klasse ist eine Sammlung, die als binaerer Baum
 4 strukturiert ist. Die Knoten des Baums stehen in der Reihung rei. Es gilt:
 5 rei[2*i]   ist der Wurzelknoten des linken  Unterbaums des Knotens rei[i]
 6 rei[2*i+1] ist der Wurzelkenten des rechten Unterbaums des Knotens rei[i]
 7 rei[i/2]   ist der Mutterknoten                       des Knotens rei[i]
 8 --------------------------------------------------------------------- */
 9 class MeinBaum {
10    // ---------------------------------------------------------------
11    String[] rei = new String[32];
12    // ---------------------------------------------------------------
13    static int nachLinks (int n) {return 2*n;}
14    static int nachRechts(int n) {return 2*n+1;}
15    // ---------------------------------------------------------------
16    public boolean fuegeEin(String nutz) {return fuegeEinR(nutz, 1);}
17    // Fuegt nutz in die Sammlung ein. Liefert true, wennn das gelingt
18    // oder wenn nutz sich schon in der Sammlung befand.
19
20    private boolean fuegeEinR(String nutz, int hier) {
21       if (rei.length <= hier) return false;
22       if (rei[hier]==null) {
23          rei[hier] = nutz;
24          return true;
25       }
26
27       int vErg = nutz.compareTo(rei[hier]);
28       if (vErg <  0) return fuegeEinR(nutz, nachLinks (hier));
29       if (vErg >  0) return fuegeEinR(nutz, nachRechts(hier));
30       return true;
31    } // fuegeEinR
32    // ---------------------------------------------------------------
33    public boolean istDrin(String nutz) {return istDrinR(nutz, 1);}
34    // Liefert true wennn nutz sich im Baum befindet.
35
36    public boolean istDrinR(String nutz, int hier) {
37       ... // MUSS ERGAENZT WERDEN
38    } // istDrinR
39    // ---------------------------------------------------------------
40    public void bearbeite() {bearbeiteR(1); printf("%n");}
41    // Bearbeitet alle Komponenten dieser Sammlung in aufsteigender
42    // Reihenfolge. Beispiel fuer Bearbeitung: Mit printf ausgeben.
43
44    public void bearbeiteR(int hier) {
45       ... // MUSS ERGAENZT WERDEN
46    } // bearbeiteR
47    // ---------------------------------------------------------------
48    ...
49    // ---------------------------------------------------------------
50    // Eine Methode mit einem kurzen Namen:
51    static void printf(String f, Object... v) {System.out.printf(f, v);}
52    // ---------------------------------------------------------------
53 } // class MeinBaum
```

Üb04-2: Jede der nicht-rekursiven Methoden `fuegeEin`, `istDrin` und `bearbeite` gibt es eine rekursive "Partner-Methode". Wodurch unterscheiden sich eine Methode und ihr Partner?

Üb04-3: Ergänzen Sie den Rumpf der rekursiven Methode `istDrinR`.

Üb04-4: Ergänzen Sie den Rumpf der rekursiven Methode `bearbeiteR`.

Übung05: Der printf-Befehl

In jeder der Übungen Üb05-1 bis Üb05-5 sind folgende Größen vorgegeben:

Ein paar Variablen (z.B. `posL` und `negL`) und
eine Reihe von `printf`-Befehlen mit Variablen als Formatstrings (z.B. `g01a`, `g01b`, ...).
Nach jedem `printf`-Befehl ist seine Soll-Ausgabe als Kommentar angegeben.
Wie müssen die Formatstrings aussehen, damit diese Soll-Ausgaben produziert werden?

Üb05-1:

```
 1       // Ein paar Ganzzahlen:
 2       long posL = +123456789012L;
 3       long negL = -123456789012L;
 4
 5       // Die printf-Befehle (und ihre Soll-Ausgabe als Kommentar):
 6                           //       1   5   10   15   20   25   30
 7                           //       |   |    |    |    |    |    |
 8       printf(g01a, posL); // g01a: 123456789012
 9       printf(g01a, negL); // g01a: -123456789012
10       printf(g01b, posL); // g01b:  123456789012
11       printf(g01b, negL); // g01b: -123456789012
12       printf(g01c, posL); // g01c:         123456789012
13       printf(g01c, negL); // g01c:        -123456789012
14       printf(g01d, posL); // g01d: 00000000123456789012
15       printf(g01d, negL); // g01d: -000000123456789012
16       printf(g01e, posL); // g01e:       +123456789012
17       printf(g01e, negL); // g01e:        -123456789012
18       printf(g01f, posL); // g01f:         123456789012
19       printf(g01f, negL); // g01f:       (123456789012)
20       printf(g01g, posL); // g01g:      123.456.789.012
21       printf(g01g, negL); // g01g:     -123.456.789.012
22       printf(g01h, posL); // g01h:     +123.456.789.012
23       printf(g01h, negL); // g01h:     -123.456.789.012
24       printf(g01i, posL); // g01i: 00000123.456.789.012
25       printf(g01i, negL); // g01i: -0000123.456.789.012
```

Vereinbaren Sie die Formatstrings `g01a` bis `g01i`.

Üb05-2:

```
 1       // Ein paar Ganzzahlen:
 2       int posI = +167;
 3       int negI = -1;
 4
 5       // Die printf-Befehle (und ihre Soll-Ausgabe als Kommentar):
 6                           //       1   5   10   15   20   25   30
 7                           //       |   |    |    |    |    |    |
 8       printf(g02a, posI); // g02a: a7
 9       printf(g02a, negI); // g02a: ffffffff
10       printf(g02b, posI); // g02b: A7
11       printf(g02b, negI); // g02b: FFFFFFFF
12       printf(g02c, posI); // g02c: 0xa7
13       printf(g02c, negI); // g02c: 0xffffffff
14       printf(g02d, posI); // g02d: 0XA7
15       printf(g02d, negI); // g02d: 0XFFFFFFFF
16       printf(g02e, posI); // g02d: 0x000000a7
17       printf(g02e, negI); // g02d: 0xffffffff
```

Vereinbaren Sie die Formatstrings `g02a` bis `g02e`.

Üb05-3:

```
 1       // Ein paar Bruchzahlen:
 2       double posD = +12345.678;
 3       double negD = -12345.678;
 4
 5       // Die printf-Befehle (und ihre Soll-Ausgabe als Kommentar):
 6                              //       1   5   10   15   20   25   30
 7                              //       |   |    |    |    |    |    |
 8       printf(b01a, posD); // b01a: 12345,678000
 9       printf(b01a, negD); // b01a: -12345,678000
10       printf(b01b, posD); // b01b: +12345,678
11       printf(b01b, negD); // b01b: -12345,678
12       printf(b01c, posD); // b01c:    +12345,68
13       printf(b01c, negD); // b01c:    -12345,68
14       printf(b01d, posD); // b01d:     12345,68
15       printf(b01d, negD); // b01d:    -12345,68
```

Vereinbaren Sie die Formatstrings `b01a` bis `b01d`.

Üb05-4:

```
 1       // Ein paar Bruchzahlen:
 2       double posD = +12345678.901234;
 3       double negD = -12345678.901234;
 4
 5       // Die printf-Befehle (und ihre Soll-Ausgabe als Kommentar):
 6                              //       1   5   10   15   20   25   30
 7                              //       |   |    |    |    |    |    |
 8       printf(b02a, posD); // b02a:  12.345.678,901234
 9       printf(b02a, negD); // b02a: -12.345.678,901234
10       printf(b02b, posD); // b02b:       +12.345.678,90
11       printf(b02b, negD); // b02b:       -12.345.678,90
12       printf(b02c, posD); // b02c:    +12.345.678,90123
13       printf(b02c, negD); // b02c:    -12.345.678,90123
14       printf(b02d, posD); // b02d:          12.345.679
15       printf(b02d, negD); // b02d:         (12.345.679)
```

Vereinbaren Sie die Formatstrings `b02a` bis `b02d`.

Üb05-5:

```
 1       // Diverse Variablen
 2       String  s01 = "Hallo!";
 3       String  s02 = null;
 4       boolean b01 = true;
 5       boolean b02 = false;
 6       Boolean b03 = new Boolean(true);
 7       Boolean b04 = new Boolean(false);
 8       JButton j01 = new JButton("Klick");
 9
10       // Die printf-Befehle (und ihre Soll-Ausgabe als Kommentar):
11                                 //       1   5   10   15   20   25   30
12                                 //       |   |    |    |    |    |    |
13       printf(d01a, s01, s02); // d01a: Hallo!: true,   null:false
14       printf(d01a, b01, b02); // d01a:   true: true,  false:false
15       printf(d01a, b03, b04); // d01a:   true: true,  false:false
16       printf(d01b, j01);      // d01b: javax.swing.JButton[,0,0,0x0,i ...
```

Vereinbaren Sie die Formatstrings `d01a` und `d01b`.

Diesen Java-Quelltext (ergänzt um einige Befehle für Testausgaben) finden Sie auch in der Datei `UebPrintf.java` im Archiv `DateienFuerPr2.zip`.