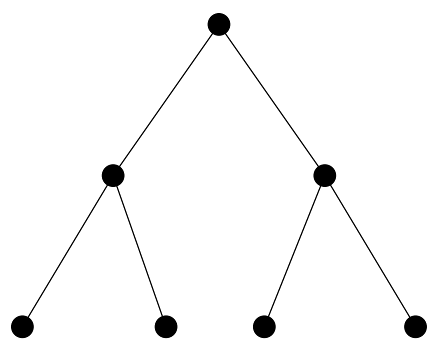
Was Menge, Folge

Wie abstrakter Datentyp Binärbaum

Wozu Suche

Binäre Bäume

Der abstrakte Datentyp *binärer Baum* ist entweder leer, oder besteht aus einem Knoten, dem ein linker und ein rechter binärer Baum zugeordnet ist. Die Operationen sind der Test, ob der (Teil-)baum leer ist, die Rückgabe des linken und die Rückgabe des rechten Teilbaums. Außerdem gibt es eine Operation, die die Wurzel des Baums liefert.



Die Definition ist rekursiv: jeder Unterbaum hat wieder eine Wurzel, an der ein rechter und ein linker Baum hängt. Blätter sind also Bäume, deren rechter und linker Teilbaum leer sind. Dadurch, daß wir den linken und den rechten Unterbaum unterscheiden, ist der Baum *geordnet*. Lins-rechts ist eine **Ordnungsrelation**.

in JAVA

```
package LS8Tools;
import AlgoTools.IO;
public class Baum {
    Object inhalt;                                   // Inhalt
    Baum links, rechts;              // linker, rechter Teilbaum
    public final static Baum LEER = new Baum ();
                                // leerer Baum als Klassenkonst.

    public Baum () {           // konstruiert einen leeren Baum
        inhalt = null;                           // kein Inhalt
        links = null;                                 // keine
        rechts = null;                                // Kinder
    }

    public Baum (Object x) {          // konstruiert ein Blatt
        this (LEER, x, LEER);                   // mit Objekt x
    }

    public Baum (Baum l, Object x, Baum r) {
                                     // konstruiert einen Baum
        inhalt = x;                  // aus einem Objekt x und
        links = l;                     // einem linken Teilbaum
        rechts = r;              // und einem rechten Teilbaum
    }
```

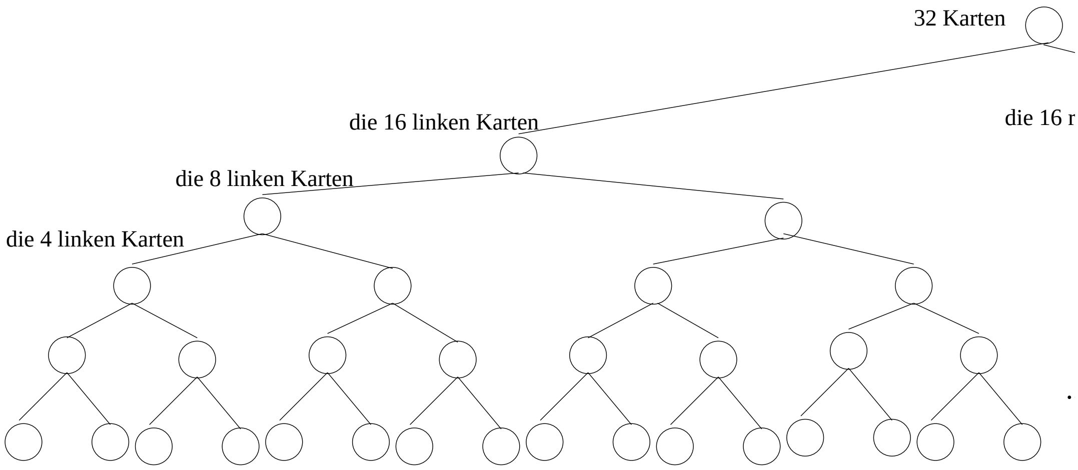
```
    public boolean empty () {                      // liefert true,
        return (inhalt == null);               // falls Baum leer ist
    }

    public Baum left () {              // liefert linken Teilbaum
        if (empty ()) IO.error ("in left: leerer Baum");
        return links;
    }

    public Baum right () {           // liefert rechten Teilbaum
        if (empty ()) IO.error ("in right: leerer Baum");
        return rechts;
    }

    public Object value () {    // liefert Objekt in der Wurzel
        if (empty ()) IO.error ("in value: leerer Baum");
        return inhalt;
    }
}
```

Bekanntes Beispiel

Raten, wo der Pik-Bube in einer verdeckten Reihe von Skatkarten liegt:

in der linken oder in der rechten Hälfte?

Nach 5 Fragen ist die Antwort klar – $log_2 32 = 5$

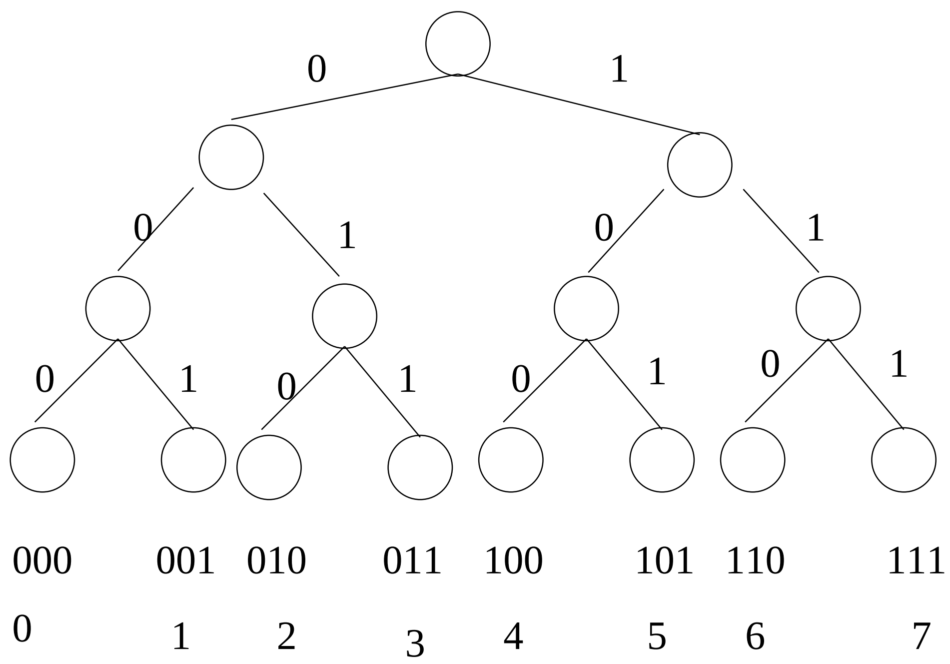


Noch ein Beispiel

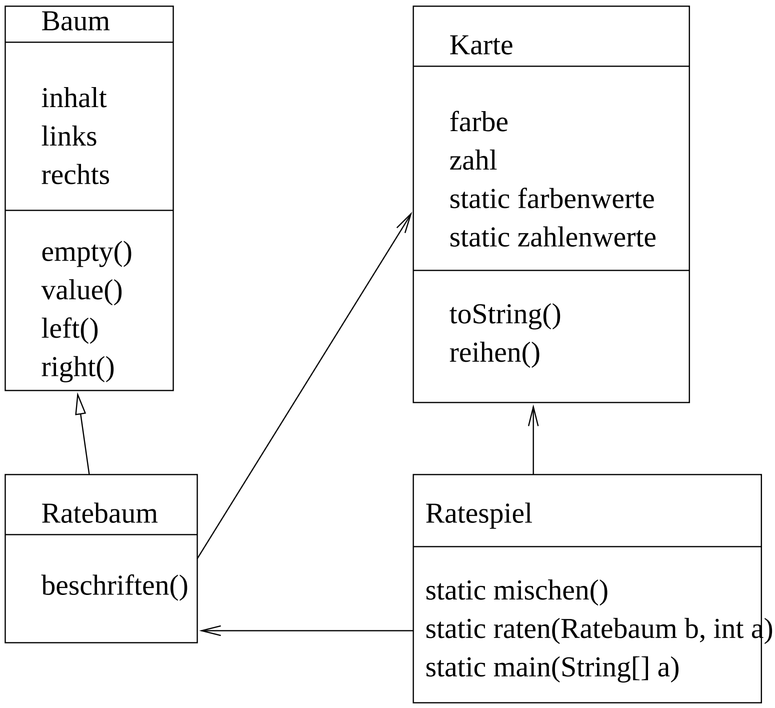
Raten einer Zahl zwischen 0 und 7 in binärer Darstellung als Folge von Entscheidungen:

ist die Lösung links (0) oder rechts (1)

für alle 3 Stellen.



Wie realisieren wir das Ratespiel in JAVA?



Karte

- Wie stellen wir dar, dass nur einige bestimmte Zeichenketten die Werte der Variablen farbe vom Typ **String** sein dürfen? JAVA hat keinen Aufzählungstyp!

class **Karte** {
String farbe=“ungueltige Farbe“;
String zahl= “ungueltige Zahl“;
static String[] farbenwerte= { “Kreuz“,“Pik“, “Herz“, “Karo“ } ;
static String[] zahlenwerte= { “sieben“, “acht“,“neun“,“zehn“,“Bube“, “Dame“, “Koenig“, “As“ } ;

Karten erzeugen

- Wie erzeugen wir eine Karte?
- Wie erzeugen wir alle Karten?

```
public Karte (String _farbe, String _zahl) {
    for (int i=0;i<4;i++) {
        if (_farbe.equals(farbenwerte[i]))
        farbe=_farbe;
    }
    for (int i=0;i<8;i++) {
        if (_zahl.equals(zahlenwerte[i]))
        zahl=_zahl;
    }
}
public static Karte[] reihen () {
    Karte[] kartenreihe=new Karte[32];
    for (int f=0;f<4;f++) {
        for (int z=0;z<8;z++)
        kartenreihe[8*f+z]=new Karte
(farbenwerte[f],zahlenwerte[z]);
    }
    return kartenreihe;
}}
```

Kartenreihe mischen

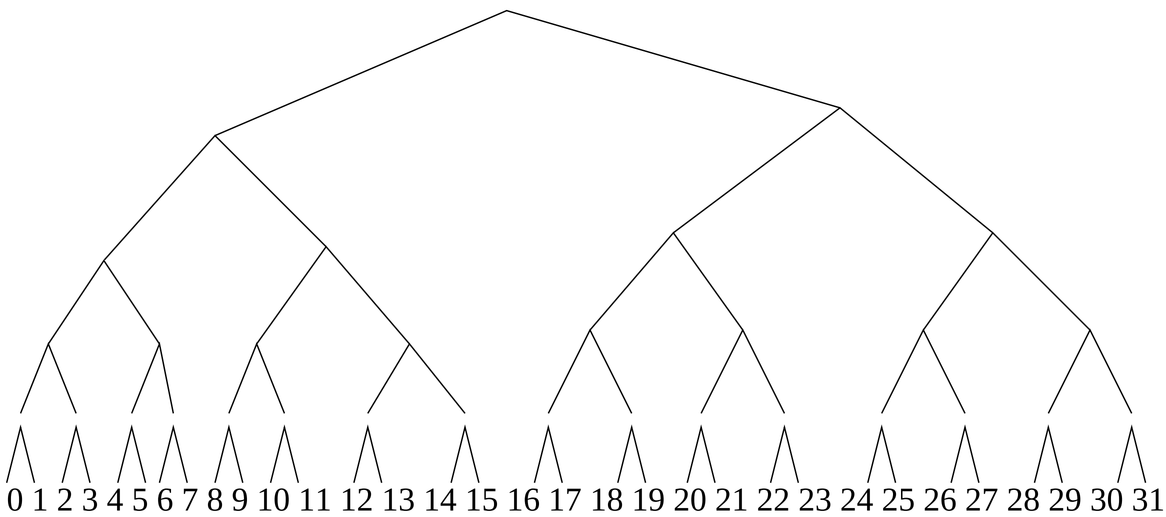
Vertauschen zufälliger Positionen

```
public static Karte[] mischen (Karte[] kartenreihe) {
    Random r = new Random ();
    Karte tmp;
    for (int j=0;j<20;j++) {
        int i = r.nextInt ()%32;
        if (i<0)
        i = -i;
        int k = r.nextInt ()%32;
        if (k<0)
        k = -k;
        tmp = kartenreihe[i];
        kartenreihe[i] = kartenreihe[k];
        kartenreihe[k] = tmp;
    }
    return kartenreihe;
}
```

Baumaufbau

Rekursive Konstruktion:

- Wenn es ein Blatt ist, dann wird der Inhalt des Baumes ein Feld von Karte[].
- Sonst wird der linke und der rechte Teilbaum konstruiert und der Inhalt auf “falsch” gesetzt.



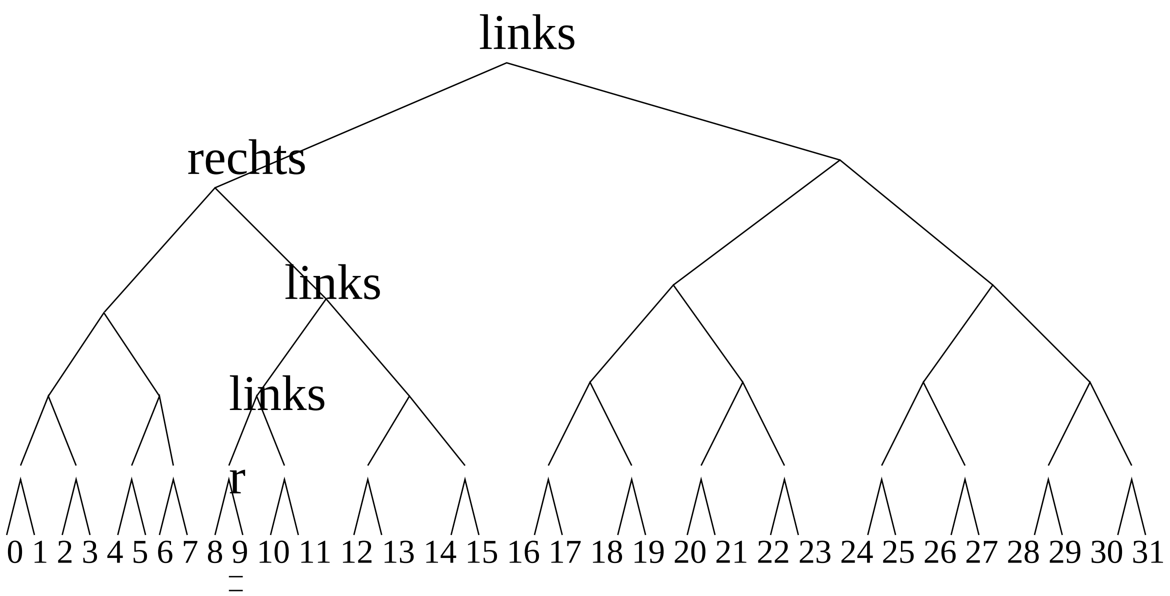
in JAVA

```
public Ratebaum (Karte[] reihe, int start, int ende) {
    if (start == ende−1)
    inhalt = reihe[start];
    else {
        links = new Ratebaum (reihe, start, (start+ende)/2);
        inhalt = “falsch“;
        rechts = new Ratebaum (reihe, (start+ende)/2, ende);
    }
}
```

Beschriften des Baumes

Rekursives Setzen der Knoteninhalte:

- Wenn das Blatt der Pik-Bube ist, liefere `true`.
- Sonst
  wenn der Pik-Bube im linken Teil ist, beschrifte mit “links”,
  wenn der Pik-Bube im rechten Teil ist, beschrifte mit “rechts”



in JAVA

```
public boolean beschriften () {
    if (value () instanceof Karte) {
        Karte karte=(Karte) value ();
        return ( karte.farbe.equals ("Pik") &&
karte.zahl.equals ("Bube") );
    }
    else if (((Ratebaum)left ()).beschriften()) {
        inhalt = "links";
        return true;
    }
    else if (((Ratebaum)right ()).beschriften()) {
        inhalt = "rechts";
        return true;
    }
    else return false;
}
```

Raten

Rekursiv immer nach links oder rechts fragen, die linke oder rechte Hälfte nehmen und wieder fragen, bis es nur noch ein Blatt gibt.

- BenutzerIn sagt die richtige Hälfte, also wird diese als neue Ausgangsmenge genommen;
- BenutzerIn sagt die falsche Hälfte, also bleibt die aktuelle Ausgangsmenge.

in JAVA

```
public static void raten (Ratebaum b, int anz) {
    IO.println ("In einer Reihe von "+anz);
    IO.println ("Spielkarten habe ich einen Pik−Buben.\n");
    IO.println ("Raten Sie, in welcher Haelfte, links oder rechts, er ist!");
    if (IO.readString ("In welcher Haelfte?").equals(b.value ())) {
        IO.println (b.value ()+"Richtig!");
        if (anz == 2)
        IO.println ("Ende");
        else if (b.value ().equals("links"))
        raten ((Ratebaum)b.left (), anz/2);
        else if (b.value ().equals("rechts"))
        raten ((Ratebaum)b.right (), anz/2);
    }
```

```
    else {
        IO.println ("Falsch!");
        raten (b, anz);
    }
}
```

Ratespiel

```
public static void main (String[] argv)  {
    Karte[] reihe = mischen (Karte.reihen ());
    Ratebaum baum = new Ratebaum (reihe,0,32);
    baum.beschriften ();
    raten (baum,32);
}
```

Suche

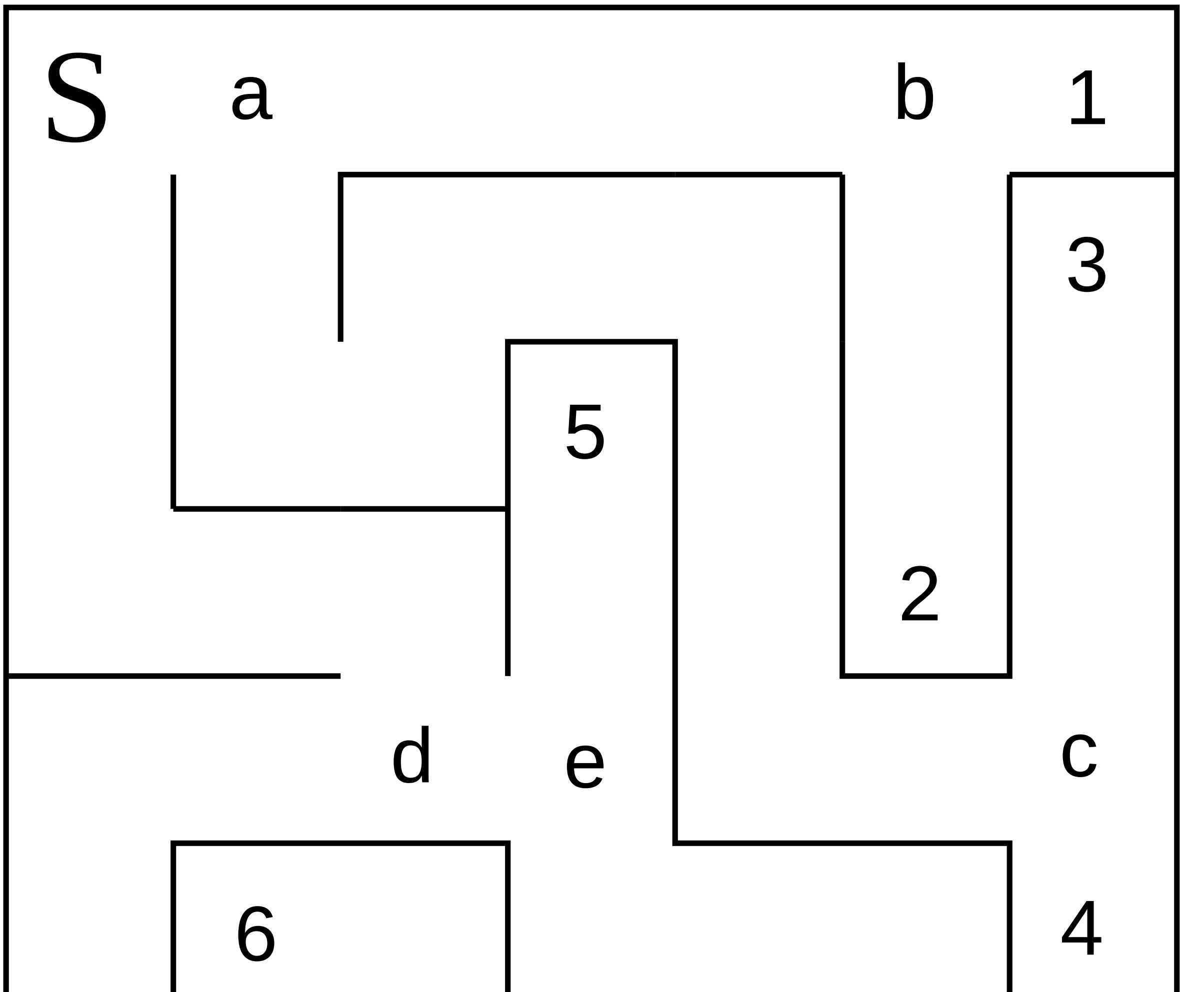
Erst haben wir den Pik-Buben beim Beschriften gesucht und eine Spur im Inhalt der Knoten angelegt.

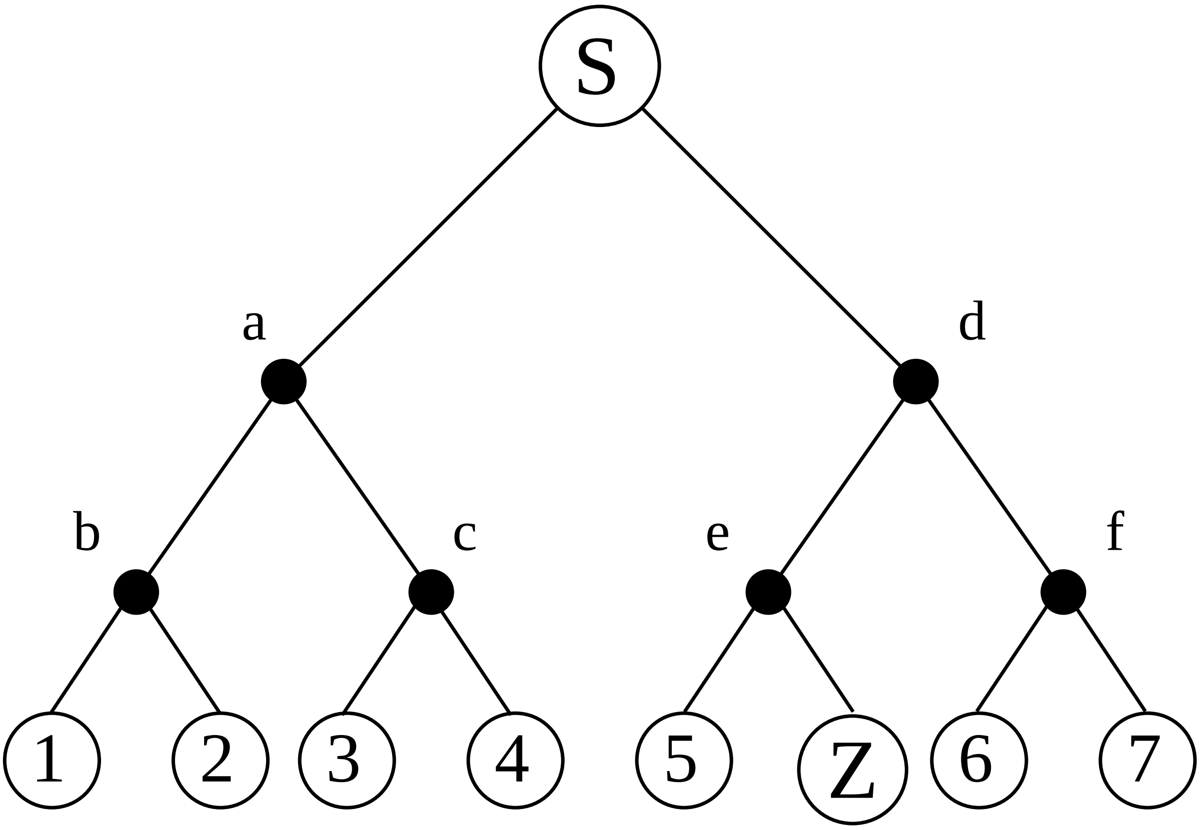
Dann sucht der Benutzer den Pik-Buben und das Programm geht wieder den Baum durch – jetzt aber nicht erschöpfend.

- erschöpfende Suche
- gezielte Suche (Finden)
- heuristische Suche

In welcher Reihenfolge werden die Knoten besucht?

Labyrinth




S
a
d
b
c
e
f
1
2
3
4
5
Z
6
7

in JAVA

```
import AlgoTools.IO;
import LS8Tools.Baum;
import LS8Tools.Schlange;
class Traversierung {
    public static boolean tiefensuche (Baum b) {
        boolean amZiel = false;                    // Variable, ob wir das Ziel gefunden haben

        if (b.empty ()) return false;              // Leerer Teilbaum ? Raus.

        if (tiefensuche (b.left ())) amZiel = true;                      // Ist Ziel links ?
        if (!amZiel && tiefensuche (b.right ())) amZiel = true;          // oder rechts ?
```

```
if (!amZiel) {                                    // Ziel nicht in Teilbaeumen gefunden
            IO.println ("Knoten: "+ b.value ());  // Akt. Knoten ausgeben

            if (b.value ().equals("Ziel")) {      // Test, ist das Ziel hier ?
                IO.println ("Ziel erreicht !");
                amZiel = true;
            }
        }
        return amZiel;
    }
```

```
public static void breitensuche (Baum wurzel) {
        Baum b;                                              // Hilfsvariable
        boolean amZiel = false;                              // Wert, ob wir schon im Ziel sind

        Schlange zuBesuchen = new Schlange (20);             // Schlange der Knoten
                                                             // die noch nicht besucht wurden
        if (!wurzel.empty()) zuBesuchen.enq (wurzel);        // Mit Wurzel starten

        while (!zuBesuchen.empty() && !amZiel) {             // Solange noch Knoten
                                                             // vorhanden und Ziel nicht erreicht,
            b = (Baum)zuBesuchen.front ();                   // obersten Knoten aus Schlange
            zuBesuchen.deq ();                               // nehmen und loeschen

            IO.println ("Knoten: "+ b.value ());             // und ausgeben.
```

```
if (b.value ().equals("Ziel")) {                          // Sind wir hier am Ziel ?
                amZiel = true;
                break ;                    // Ziel gefunden, while−Schleife verlassen
            }

                                 // Eventuelle Nachfolger hinten an Schlange haengen
            if (!b.left().empty()) zuBesuchen.enq (b.left ());
            if (!b.right().empty()) zuBesuchen.enq (b.right ());
        }
        if (amZiel)
        IO.println ("Ziel erreicht !");
        else
        IO.println ("Habe mich verlaufen !");
    }
```

```
public static void main (String[] argv) {
        Baum wurzel, l, r, x, y;

                                                        // Baum aufbauen
        l = new Baum (new Baum ("1"), "b", new Baum("2"));
        r = new Baum (new Baum ("3"), "c", new Baum("4"));
        x = new Baum (l, "a", r);

        l = new Baum (new Baum ("5"), "e", new Baum("Ziel"));
        r = new Baum (new Baum ("6"), "f", new Baum("7"));
        y = new Baum (l, "d", r);

        wurzel = new Baum (x, "Start", y);
```

```
                                        // Ziel mit Tiefensuche finden
        IO.println ("Tiefensuche :");
        tiefensuche (wurzel);

                                        // Ziel mit Breitensuche finden
        IO.println ("\nBreitensuche :");
        breitensuche (wurzel);
    }

}
```