Wo sind wir?

Was wissen wir schon?

- Objektorientierte Modellierung
- Grundelemente von JAVA
- abstrakte Datentypen: Liste, Keller, Schlange
- Sortierung: Selektionssortierung
- Induktionsbeweis

Was kommt jetzt?

- Rekursion
- Sortierung: Mischsortierung
- Aufwandsabschätzung: O-Notation

Rekursion

- Was hätte ich gern? – Präzise Formulierung des Erfolgskriteriums!
- Wie reduziere ich das Problem? – Kern der Methode
- Rekursiver Aufruf mit dem reduzierten Problem.

Rekursiver Aufruf:

eine Methode ruft sich selbst auf;

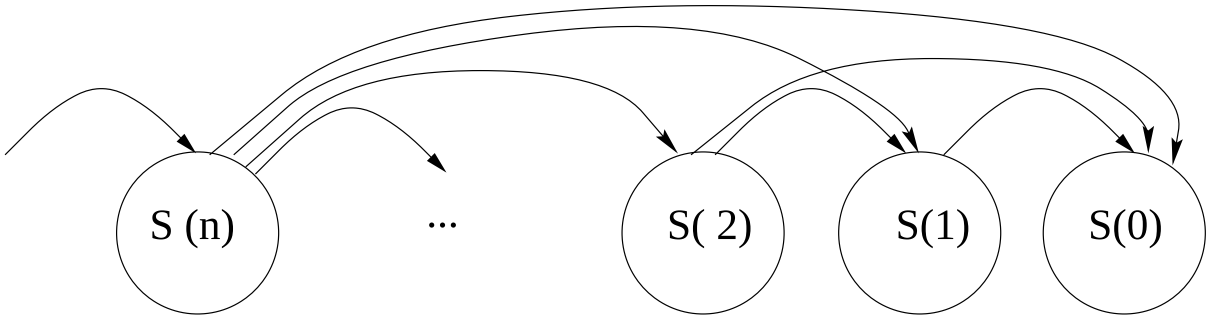
Wichtig:
jeder neuerliche Aufruf der Methode bewältigt ein kleineres Problem und
es gibt eine Folge von Aufrufen derselben Methode, die zu dem Zustand führt, in dem das Erfolgskriterium erfüllt ist.

linear rekursiv: ein einziger Selbst-Aufruf innerhalb der Methode;

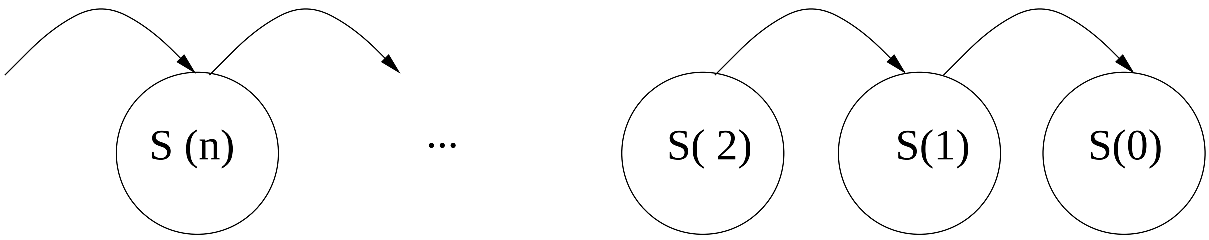
Endrekursion: Selbstaufruf ist letzter Schritt der Methode;

Rekursive Definitionen

Es gibt eine Kette von Fällen, von denen jeder von einem oder allen vorigen Fällen abhängen kann.

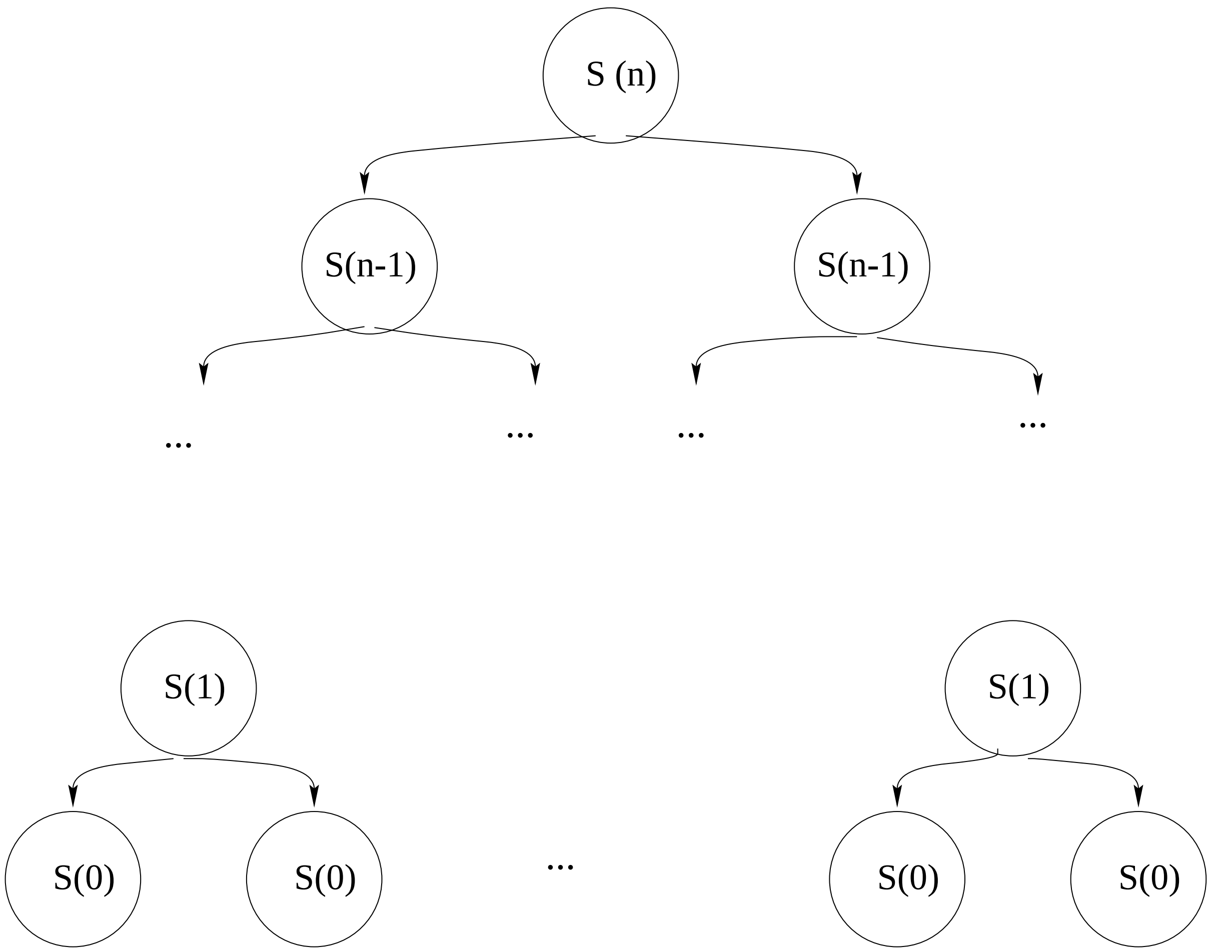


Bei der linearen Rekursion hängt ein Fall nur vom unmittelbar vorhergehenden Fall ab.



Baumartige Rekursion

Bei der baumartigen Rekursion sind mehrere Rekursionsketten verbunden.



Beispiel lineare Rekursion

- Ich hätte gern mein Diplom. Als Erfolgskriterium wird nun einfach das Erreichen des 9.Semesters (oder eines höheren) angegeben. Ist dies Kriterium erfüllt, gibt es das Diplom und die Methode ist beendet.

- Das Problem wird durch das Verstreichen von Semestern reduziert. Der Kern der Methode ist die Schleife und dann das Erhöhen des Jahres um 1.

- Mit dem neuen Jahr und dem Monat 1 ruft sich die Methode selbst auf.

```
import ballbeispiel.*;
import AlgoTools.IO;

class Studierend extends Mensch {
    int semester,monat,jahr;
    public Studierend () {
        super ();
        semester = IO.readInt (“Wievieltes Semester? “);
        monat = IO.readInt (“Monat des Jahres? (Zahl) “);
        jahr = IO.readInt (“In welchem Jahr? “);
    }

    public void studieren () {
        for (int i=this.monat; 13 >i;i++) {          //Monate
            if ((i!=this.monat) && (i==4 | i==10)) {
                                               //Semester zaehlen
                this .semester++;
                System.out.println (this.name + “ist“
                + this.jahr + “im Semester “ + semester);  }
        }
        this .jahr++;                                  //Jahre
        this .monat=1;
        if (9>semester)   //Studienende noch nicht erreicht?
        studieren ();                       //dann weiterstudieren
        else System.out.println (“Und jetzt das Diplom!“);
    }
}
```

```
class Studi {
    private static void main (String argv[]) {
        Studierend stud;
        stud = new Studierend ();
        stud.studieren ();
        System.out.println (stud.name+
                            "bekommt das Diplom "+stud.jahr);
    }
}
```

noch ein Beispiel lineare Rekursion

Wir wollen wissen, wie oft ein Name in einer Namensliste vorkommt.

Wenn wir alle Elemente der Liste mit dem Namen verglichen haben, sind wir fertig. Wir machen den Vergleich konsumtiv, d.h. wir entfernen verglichene Elemente. Wir sind fertig, wenn die Liste leer ist.

Das Problem läßt sich aufteilen in zwei Teilprobleme:

vergleiche das erste Element der Liste mit dem Namen und

vergleiche den Rest der Liste mit dem Namen.

Das Problem wird reduziert, in dem die Liste verkürzt wird.

in JAVA

```
import AlgoTools.*;
import java.util.*;

class Namensliste extends LinkedList {
    String name;
    int count=0;

    public Namensliste () {
        super ();
    }

    public void addName (String _name) {
        addFirst (_name);
    }

    public String removeFirstName () {
        return (String) removeFirst ();

    }

    public boolean compareName (String _name) {
        if (removeFirstName ().equals( _name))
        return true;
        else return false;
    }
```

```
    public int countName (String _name) {
        if (compareName (_name))
        count++;
        if (size ()>0)
        countName (_name);
        return count;
    }
}
class Namenfinden  {

    static void main (String[] argv) {
        Namensliste liste= new Namensliste ();
        while (IO.readString ("Namen
eingeben?").equals("ja")) {
            liste.add (IO.readString ("Ein Name: "));
        }
        System.out.println (argv[0]+"wurde "+
                    liste.countName(argv[0])+"gefunden!");
    }

}
```

Ergebnis

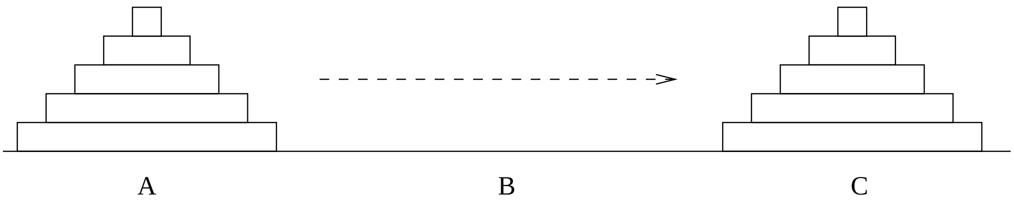
java Namenfinden uta
Wollen Sie Namen eingeben?ja
Bitte geben Sie einen Namen ein uta
Wollen Sie Namen eingeben?ja
Bitte geben Sie einen Namen ein ralf
Wollen Sie Namen eingeben?ja
Bitte geben Sie einen Namen ein uta
Wollen Sie Namen eingeben?ja
Bitte geben Sie einen Namen ein sascha
Wollen Sie Namen eingeben?ja
Bitte geben Sie einen Namen ein uta
Wollen Sie Namen eingeben?nein

uta, ralf, uta, sascha, uta
1
ralf, uta, sascha, uta
uta, sascha, uta
2
sascha, uta
uta
3
uta wurde 3 gefunden!

Beispiel baumartige Rekursion

Mehrere Aufrufe einer Methode innerhalb dieser Methode.

Türme von Hanoi

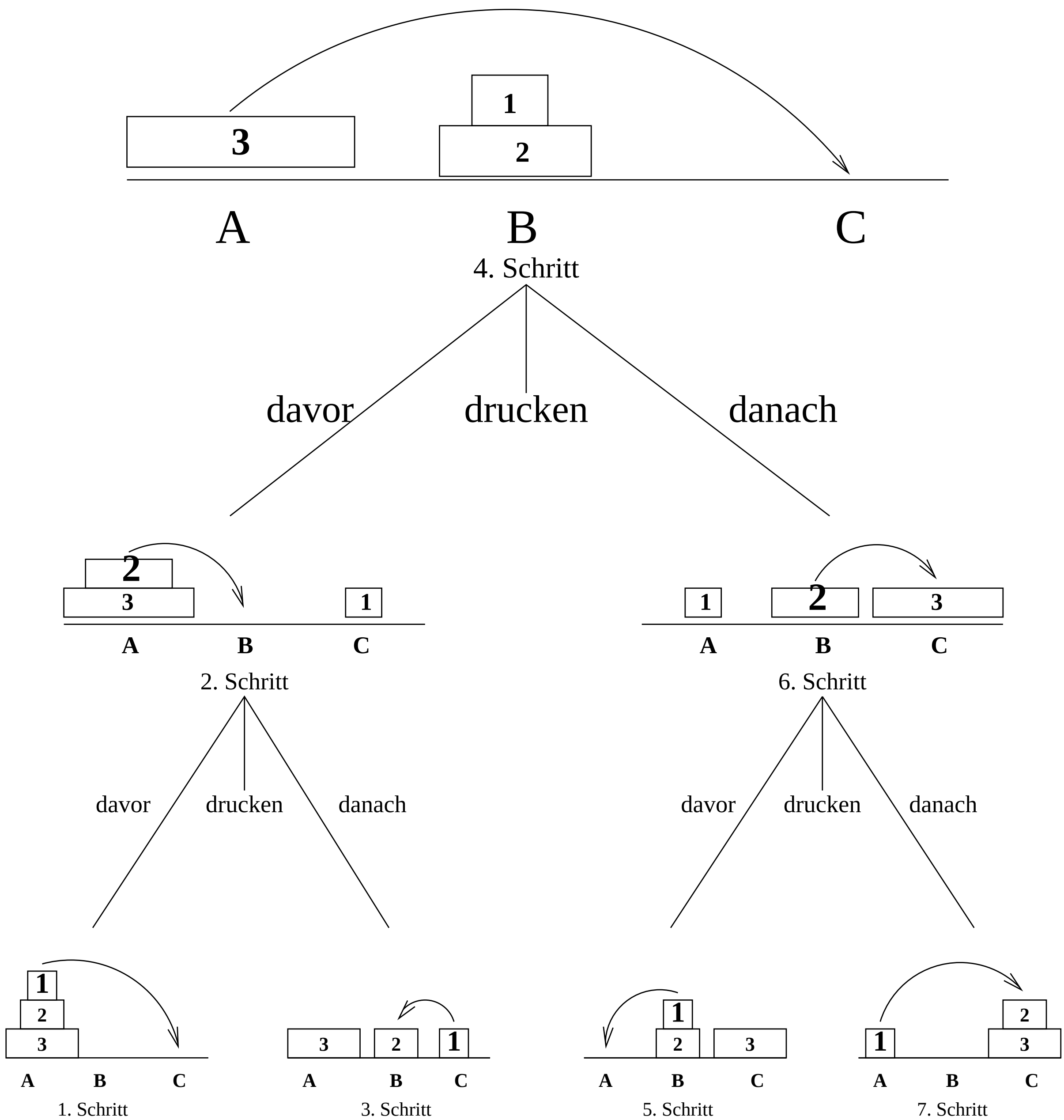


- Das Problem ist gelöst, wenn wir die Scheibe 1 auf den geordneten Stapel auf Platz C legen.

- Wir reduzieren das Problem, indem wir immer kleinere Stapel von Scheiben betrachten.

- Das Problem, n Scheiben von A unter Verwendung von B nach C zu verlegen, läßt sich in zwei Probleme aufteilen: verlege $n-1$ Scheiben vom Start- zum Zwischenplatz und verlege $n-1$ Scheiben vom Zwischen- zum Zielplatz. Wir schreiben also eine Methode für n Scheiben, die sich selbst zweimal für $n-1$ Scheiben aufruft.

Ziele

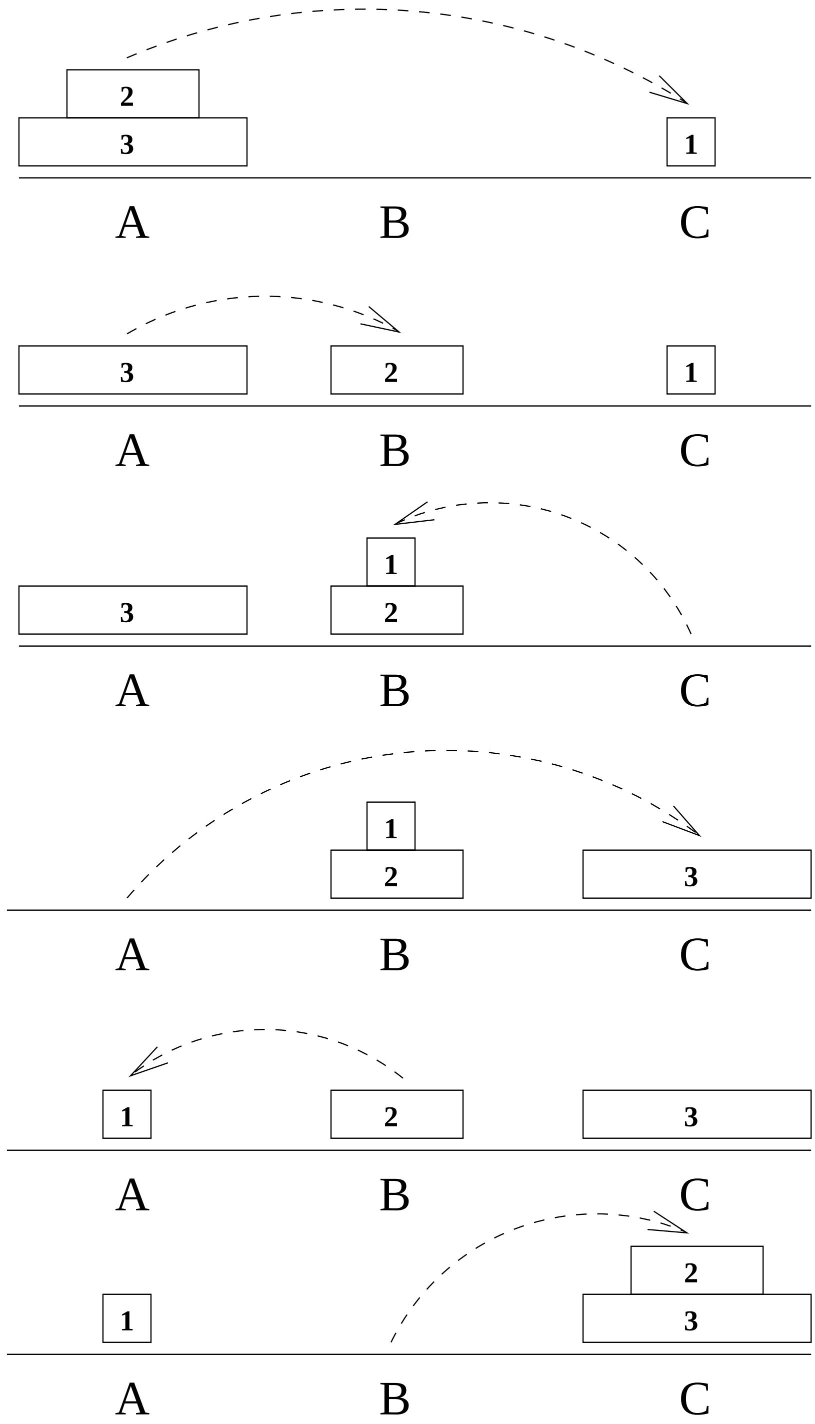

3
1
2
A
B
C
4. Schritt
davor
drucken
danach
2
3
1
A
B
C
2. Schritt
1
2
3
A
B
C
6. Schritt
davor
drucken
danach
davor
drucken
danach
1
2
3
A
B
C
1. Schritt
3
2
1
A
B
C
3. Schritt
1
2
3
A
B
C
5. Schritt
2
1
3
A
B
C
7. Schritt

... in JAVA

```
import AlgoTools.IO;
public class Hanoi {
    static void verlege (
                        // drucke die Verlegeoperationen, um
    int n,                                   // n Scheiben
    char start,                              // vom Startort
    char zwischen, // unter Zuhilfenahme eines Zwischenortes
    char ziel)                               // zum Ziel bringen
     {
        if (n == 1)                          // Erfolgskriterium
        IO.println ("Scheibe 1 von "+ start + "nach "+ ziel);
        else {
            verlege (n-1,start, ziel, zwischen);          //1.
            IO.println ("Scheibe "+ n +"von "+ start +
                                          "nach "+ ziel);
            verlege (n-1,zwischen, start, ziel);          //2.
        }
    }
```

```
public static void main (String argv[])  {
        int n;
        do  {
            n = IO.readInt ("Zahl der Scheiben (n>0): ");
        }  while (n <= 0);                    solange n unzulässig
        verlege (n,'A','B','C');
    }
}
```

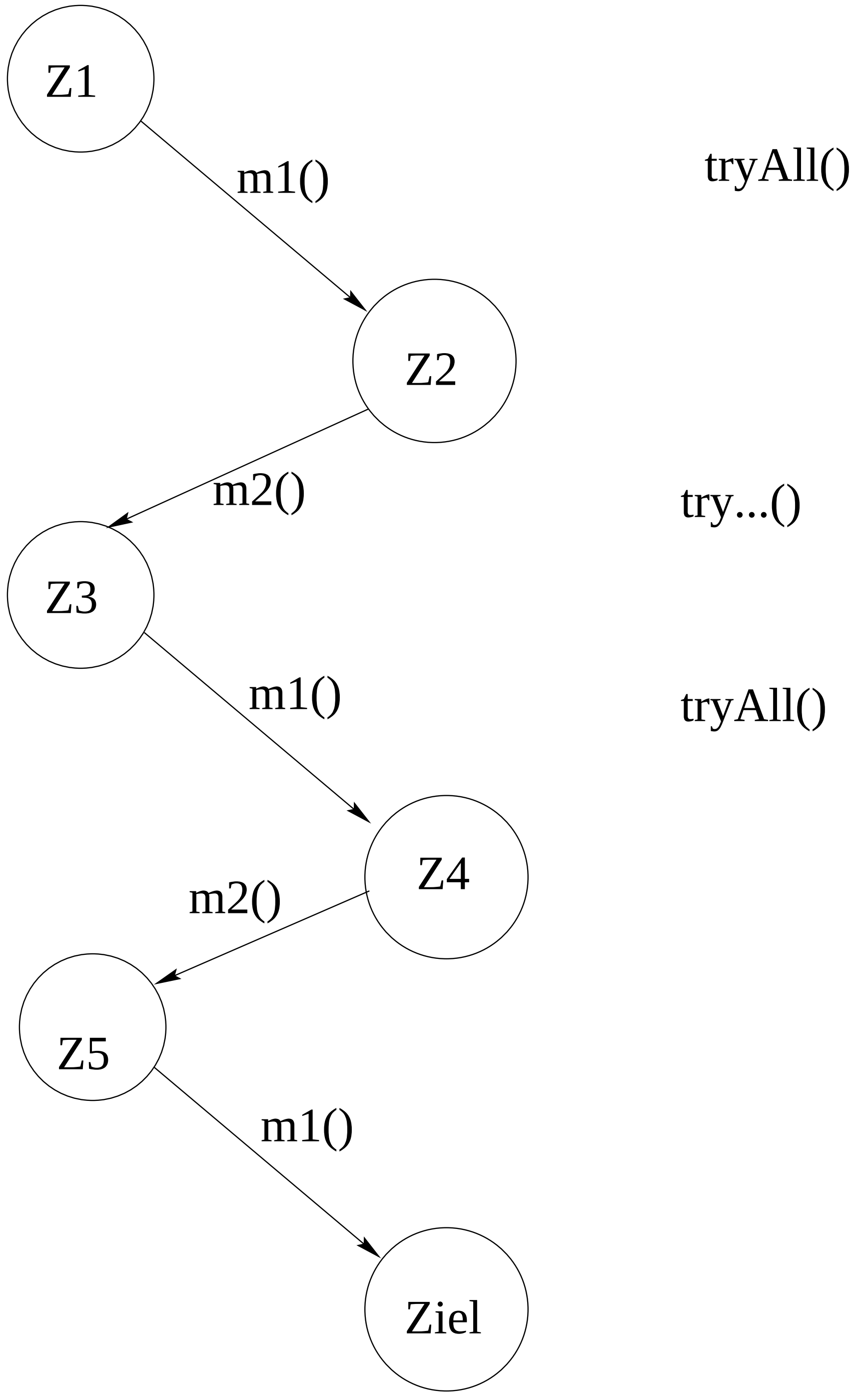
Ergebnis



formale Aufrufe	Werte
verlege(n,Start,Zwischen,Ziel)	verlege(3,A,B,C)
? n=1	
verlege(n-1,Start,Ziel,Zwischen)	verlege(2,A,C,B)
| $\hat{=}$ verlege(n,Start,Zwischen,Ziel)	verlege(2,A,C,B)
| ? n=1	
| verlege(n-1,Start,Ziel,Zwischen)	verlege(1,A,B,C)
| | $\hat{=}$ verlege(n,Start,Zwischen,Ziel)	verlege(1,A,B,C)
| | ?! n=1	
| | drucke 'Scheibe 1 von Start nach Ziel'	Scheibe 1 von A nach C
| drucke 'Scheibe n von Start nach Ziel'	Scheibe 2 von A nach B
| verlege(n-1,Zwischen,Start,Ziel)	verlege(1,C,A,B)
| | $\hat{=}$ verlege(n,Start,Zwischen,Ziel)	verlege(1,C,A,B)
| | ?! n=1	
| | drucke 'Scheibe 1 von Start nach Ziel'	Scheibe 1 von C nach B

```
drucke 'Scheibe n von Start nach Ziel'                 Scheibe 3 von A nach C
verlege(n-1,Zwischen,Start,Ziel)                       verlege(2,B,A,C)
|≙  verlege(n,Start,Zwischen,Ziel)                     verlege(2,B,A,C)
    ? n=1
    verlege(n-1,Start,Ziel,Zwischen)                   verlege(1,B,C,A)
    |≙  verlege(n,Start,Zwischen,Ziel)                 verlege(1,B,C,A)
        ?! n=1
        drucke 'Scheibe 1 von Start nach Ziel'         Scheibe 1 von B nach A
    drucke 'Scheibe n von Start nach Ziel'             Scheibe 2 von B nach C
    verlege(n-1,Zwischen,Start,Ziel)                   verlege(1,A,B,C)
    |≙  verlege(n,Start,Zwischen,Ziel)                 verlege(1,A,B,C)
        ?! n=1
        drucke 'Scheibe 1 von Start nach Ziel'         Scheibe 1 von A nach C
```

Indirekte Rekursion



Umwandlung Rekursion in Schleife

Linear-rekursive Programme können einfach in iterative umgewandelt werden:

Schleife statt Rekursion!

```
public void studierenI () {
    while (9 > this.semester) {          //Studienende erreicht?
                                         //dann weiterstudieren
        for (int i=this.monat; 13>i; i++) {
                                         // Monate zaehlen
            if ((i!=this.monat) && (i==4 | i==10)) {
                                         // Semester zaehlen
                this.semester++;
                System.out.println
                        (this.name + "ist " + this.jahr
                    + "im " + semester + ". Semester");
            }
        }
        this.jahr++;                     // Jahre zaehlen
        this.monat=1;
    }

    System.out.println ("Und jetzt das Diplom!");
}
```

Namen finden iterativ

```
import AlgoTools.*;
import java.util.*;

class Namensliste extends LinkedList {
    String name;
    int count=0;

    public Namensliste () {
        super ();
    }

    public void addName (String _name) {
        addFirst (_name);
    }

    public String removeFirstName () {
        return (String) removeFirst ();
    }

    public boolean compareName (String _name) {
        if (removeFirstName ().equals( _name))
        return true;
        else return false;
    }
}
```

```
class IterativFinden {

    static void main (String[] argv) {
        Namensliste liste= new Namensliste ();

        while
        (IO.readString ("Namen eingeben?").equals("ja"))
         {
            liste.add (IO.readString ("Bitte Namen: "));
        }

        for (int i=0;liste.size ()>0;i++) {
            if (liste.compareName (argv[0]))
            liste.count++;
        }
        System.out.println (argv[0]+
                            "wurde "+liste.count+"gefunden!");
    }
}
```

Umwandlung baumartig rekursiver Methoden in iterative

Aufrufe werden als Zustände aufgefaßt: Methode und aktuelle Parameter;

Der Anfangszustand wird auf den Keller gelegt.

Iteration: Der oberste Zustand wird vom Keller gelesen, entfernt und seine Nachfolgezustände der Reihe nach auf den Keller gelegt.

Der Keller ist abgearbeitet, oder die Iteration wird erneut durchlaufen.

Die Kellerzustände

Keller nach der ersten Iteration:

2, A, C, B
print(3, A, B, C)
2, B, A, C
3, A, B, C

Keller nach der zweiten Iteration:

1, A, B, C
1, C, A, B
print(3, A, B, C)
2, B, A, C
3, A, B, C

Hanoi iterativ

```
import java.util.*;
import AlgoTools.*;

class HanoiIterativ {
    public static void main (String argv[]) {
        int n=IO.readInt("Bitte geben Sie die Anzahl der Scheiben an: ");
        new Zustand (false,n,'A','B', 'C').verlegen(new Stack ());
    }
}
```

```
class Zustand {
    int n;
    char start;
    char zwischen;
    char ziel;
    boolean drucken;

    public Zustand (boolean _drucken, int _n, char _start, char _zwischen, char _ziel) {
        n=_n;
        drucken=_drucken;
        start=_start;
        zwischen=_zwischen;
        ziel=_ziel;
    }
```

```
void drucke () {
        System.out.println (n+"von "+start+"nach "+ziel);
    }

    public void verlegen (Stack keller) {
        keller.push (this );

        while (!keller.empty())
        ((Zustand)keller.pop ()).rekursionsersatz(keller);

        System.exit (0);
    }
```

```
    public void rekursionsersatz (Stack keller) {
        if (drucken || n == 1)
        drucke ();
        else {
            drucken=true;
            keller.push (new Zustand (false, n-1, zwischen, start,ziel));
            keller.push (this );
            keller.push (new Zustand (false, n-1,start, ziel,zwischen));
        }
    }
}
```