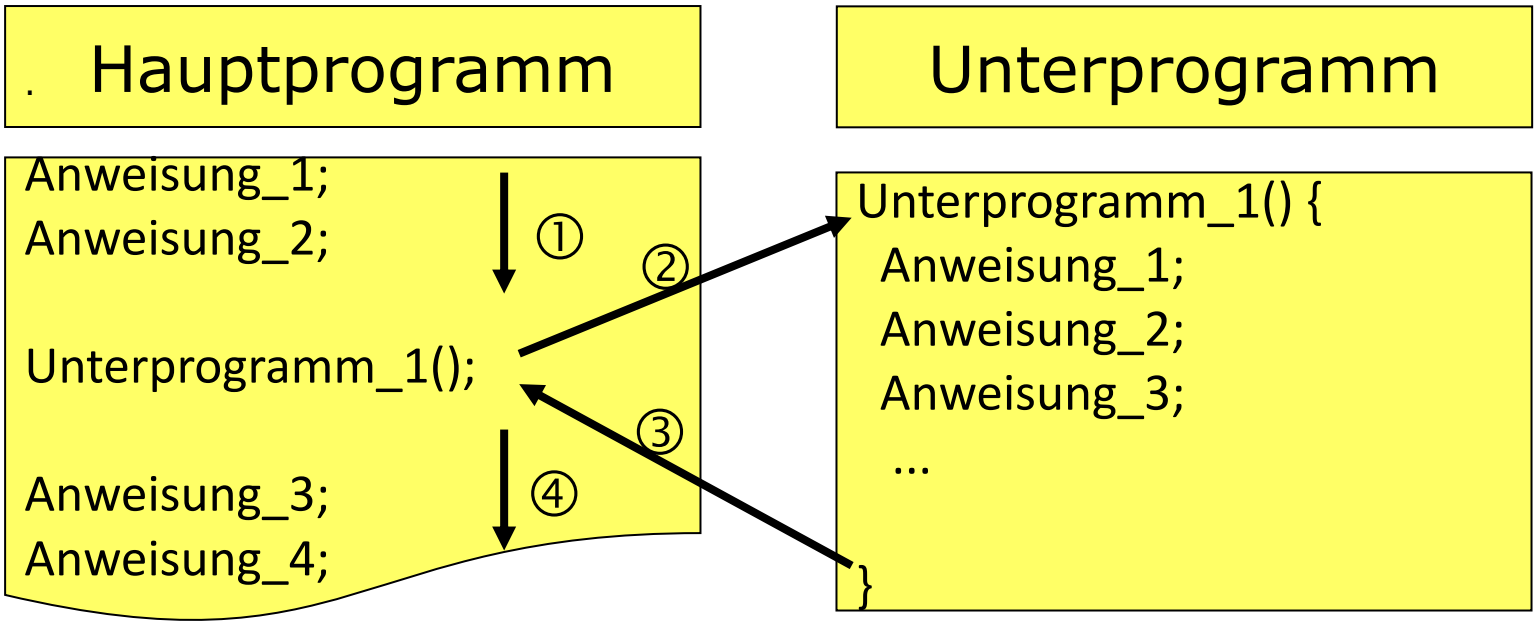
Unterprogramme / Methoden

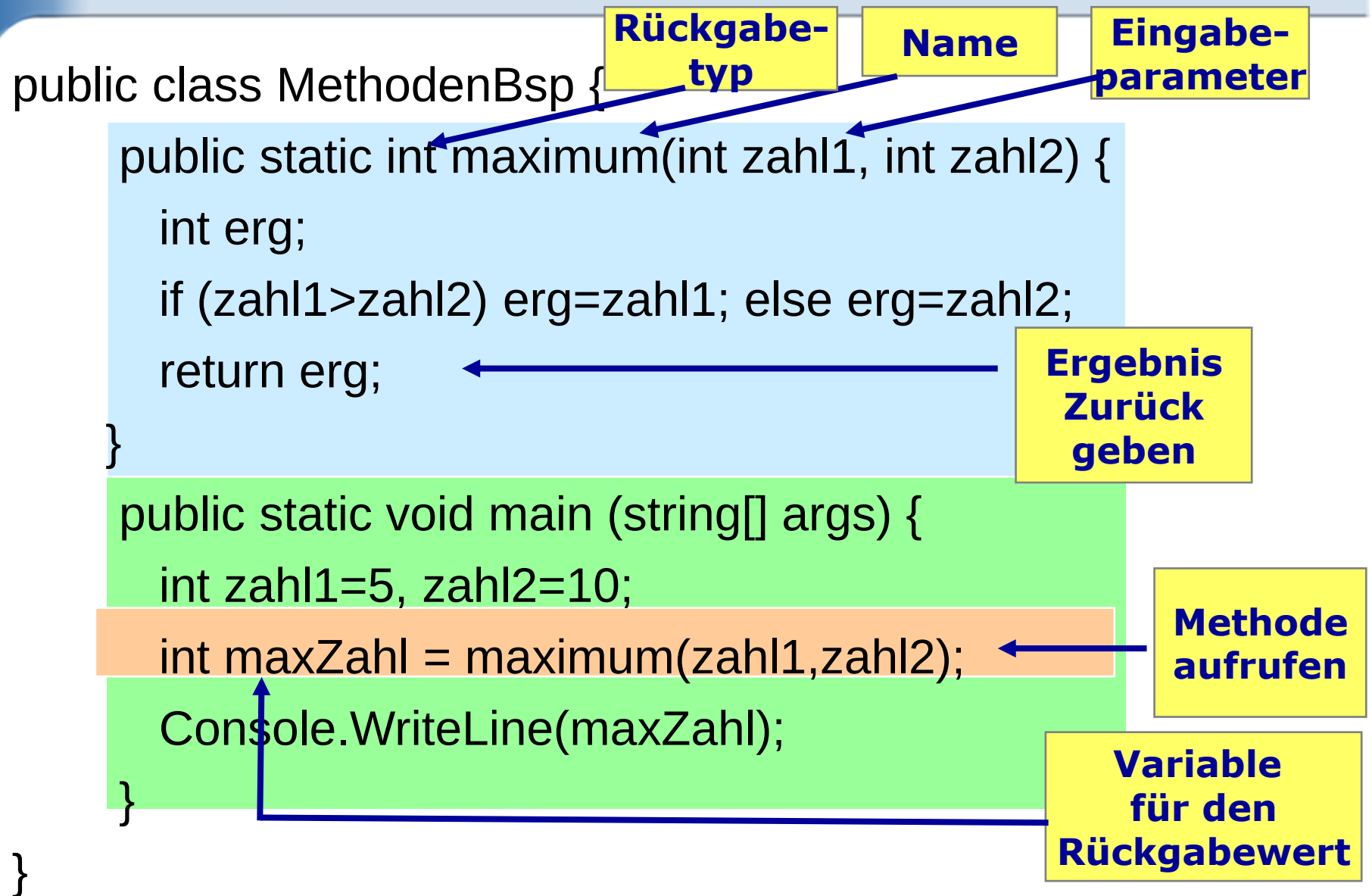
Das Hauptprogramm delegiert eine Aufgabe an ein Unterprogramm



Vorteile von Unterprogrammen

- **Top-Down-Verfahren: “Teile und Herrsche”**
  - Zerlegung eines Problems in Teilprobleme
- **Lesbarkeit**
  - Kompakter, übersichtlicher und selbsterklärener Programmaufbau
- **Wiederverwendbarkeit**
  - Ein Unterprogramm kann für andere ähnliche Aufgaben wieder verwendet werden
- **Änderbarkeit / Erweiterbarkeit**
  - Änderungen nur an einer zentralen Stelle
- **Testbarkeit**
  - Getrennte Tests einzelner Programmteile leicht möglich

Definition und Aufruf



Bemerkungen

Die Reihenfolge der Aufrufparameter ist wichtig.

Eine Methode die nichts zurück gibt, erhält den Rückgabewert **void,** z.B.

```
public static void begruesseBenutzer() { … }
```

Schlüsselunterscheidung
Wert ausgeben vs Wert zurück geben

Wert ausgeben

```
public static void ausgabeM(int zahl1, int zahl2) {
    int erg;
    if (zahl1>zahl2) erg=zahl1; else erg=zahl2;
    CW("Max: " + erg);
}
```

```
...
ausgabeM(5, 10); //Ausgabe: Max: 10
...
```

Der Wert wird nur auf dem Bildschirm ausgegeben.

Keine weitere Verarbeitung möglich.

Wert zurück geben

```
public static int maximum(int zahl1, int zahl2) {
    int erg;
    if (zahl1>zahl2) erg=zahl1; else erg=zahl2;
    return erg;
}
```

```
...
int maxZahl = maximum(5, 10);
maxZahl=maxZahl/2;
...
```

Der Wert wird von der Methode zurück geliefert und meistens in einer Variable gespeichert.

Der Wert kann weiter verarbeitet werden (z.B. in eine Formel eingesetzt werden).

Arbeitsauftrag

Sternefigur Hohlquadrat in Unterprogramme aufteilen

Ändern Sie Ihr Programm so ab, dass zusammenhängende Programmteile in Unterprogrammen ausgeführt werden (siehe Übungsblatt arbeitsblatt-1-rechteck).

Arbeitsauftrag

Berechnung der Fakultät

Definieren Sie eine Methode, die die Fakultät
n! = 1*2*3*…*n
berechnet und rufen diese mit verschiedenen Parametern auf.

Weitere Details zu Methoden

Siehe
Präsentation
00-methoden-ausführlich

Dokumentation (summary)

Dokumentation
Kommentare in den Quellcode schreiben

```
///<summary>
///Methode Note in Text au
///1 -> "sehr gut"
///</summary>
1 Verweis
public static String notenText(int note)
{
    String erg = "";
    switch(note)
    {
        case 1:
            erg = "sehr gut";
            break;
        case 2:
            erg = "gut";
            break;
```

```
///<summary>
///...
///</summary>
```

Doku-Kommentar

string Program.notenText(int note)
Methode Note in Text ausgeben. 1 -> "sehr gut"

Typumwandlungen (Typecasting)

Implizites und explizites Typecasting

int →long

```
int iZahl=1000;
    long loZahl=1000000;
    loZahl=iZahl; // funktioniert implizit
```

long → int

```
iZahl = (int) loZahl; // explizit durch den Typecast-Operator
```

float →double

```
float fZahl=3.14f;
    double dZahl=77.7;
    dZahl = fZahl; // funktioniert implizit
```

double → float

```
fZahl = (float) dZahl; // explizit durch den Typecast-Operator
```

Explizites Typecasting zwischen verschiedenen Datentypen

int ←→ float

```
Int iZahl1=5;
    float fZahl1=10.8f;

    fZahl1 = (float) iZahl1;
    iZahl1 = (int) fZahl1; // danach hat iZahl1 den Wert 10
```

String ←→int, String ←→float

```
String sZahl1="56";
    int iZahl1=5;
    float fZahl1=10.8f;

    iZahl1 = Convert.ToInt32(sZahl1);
    sZahl1 = Convert.ToString(iZahl1);
    fZahl1 = Convert.ToDouble(sZahl1);
```

Typecasting Beispiele 1

```
iZahl = (int) fZahl1 * (int) fZahl2;
    oder
    iZahl = (int) (fZahl1 * fZahl2); // Klammern sind hier wichtig!

dZahl2 = fZahl1; // geht implizit

fZahl = (float)(iZahl1 + iZahl2); // expliziter Typecast-
    Operator

sZahl1 = Convert.ToString(loZahl1 + (long) iZahl1);

fZahl1= 45.765f;
    iZahl1= (int) fZahl1; // erhält nur den ganzteiligen Anteil, also 45
```

Typecasting Beispiele 2

Runden einer float-Zahl

```
float fZahl1=45.49f;
int iZahl1 = (int) (fZahl1+0.5f);
```

Runden auf 3 Kommastellen

```
float fZahl1=45.123456f;
float fZahl3 =  (int)(fZahl1*1000+0.5f)/1000f;
```

Arbeitsauftrag

Würfel-Simulation schreiben

Rekursion

Was passiert hier?

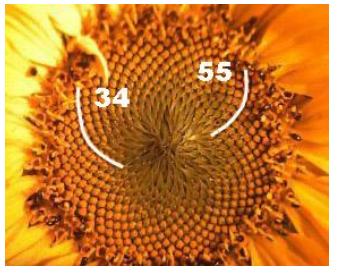
Quelle: de.forwallpaper.com/wallpaper/artist-art-recursion-the-picture-is-dip-brush-177932.html

Fibonacci-Zahlen

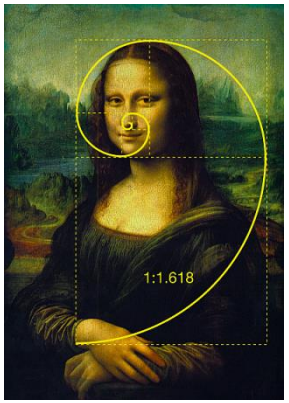
Die erste Fib-Zahl ist 1.

Die zweite Fib-Zahl ist 1.

Die n-te Fib-Zahl ist die Summe
der zwei vorherigen.



Also: 1 1 2 3 5 8 13 21 …



Formal:

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

Frage: Welchen Wert hat **fib(90)**?

Arbeitsauftrag

Berechnung der 90-ten Fibonacci-Zahl

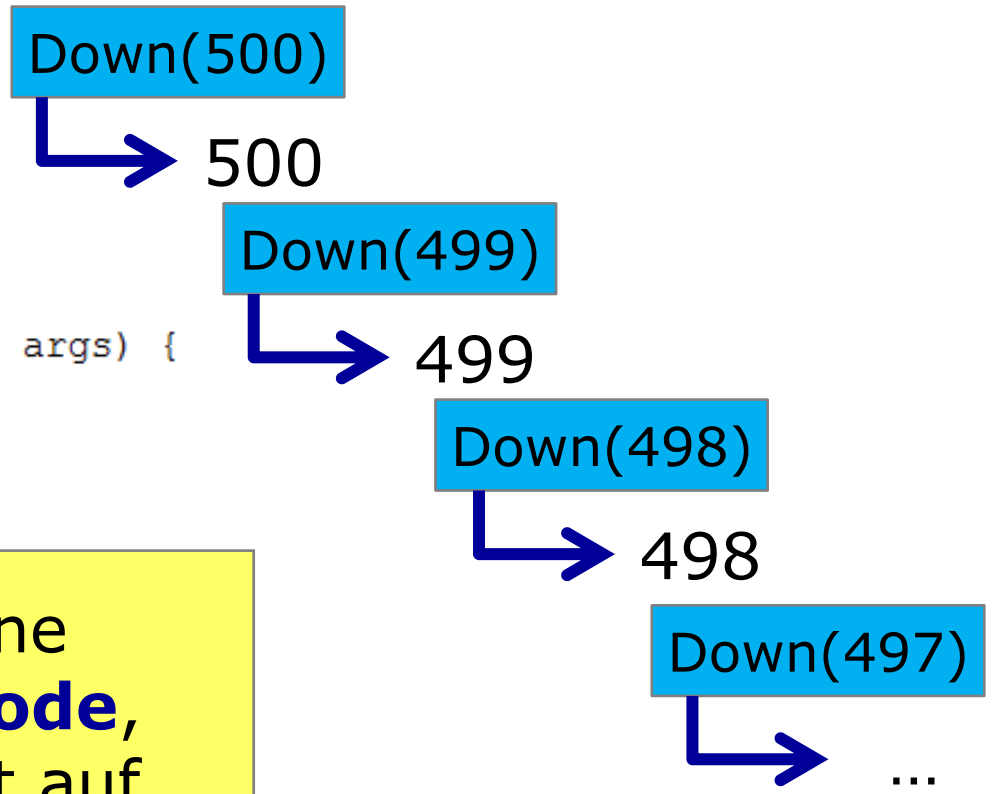
Definieren Sie eine Methode, die mithilfe eines Arrays die 90-te Fibonacci-Zahl errechnet.

Iterative Lösung

```
1  public class Fibonacci {
2
3    public static long fib(int n) {
4      long[] a = new long[n];
5      a[0]=1;
6      a[1]=1;
7      for(int i=2; i<n; i++) {
8        a[i]=a[i-1]+a[i-2];
9        //System.out.println(a[i]);
10     }
11     return a[n-1];
12   }
13
14   public static void main(String[] args) {
15     System.out.println(fib(90));
16   }
17 }
```

Was macht dieses Programm?

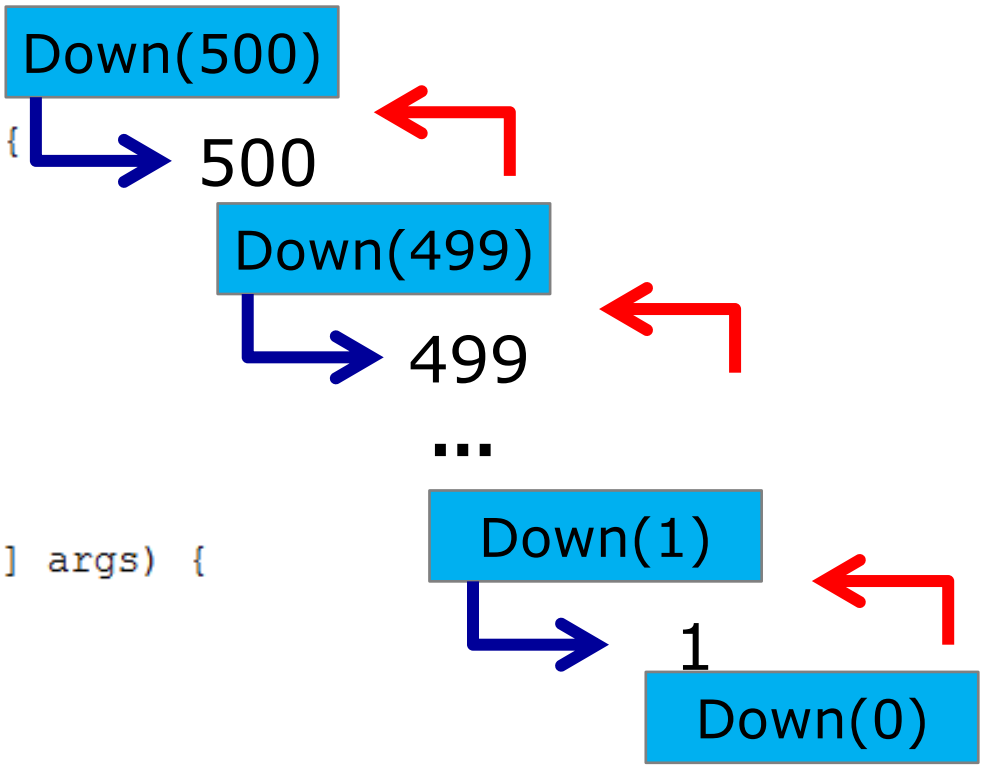
```
1  public class Test1 {
2
3    public static void down(int n) {
4      System.out.println(n);
5      down(n-1);
6    }
7
8    public static void main(String[] args) {
9      down(500);
10   }
11 }
```



down(int) ist eine **rekursive Methode**, sie ruft sich selbst auf.

down(int) mit Abbruchkriterium

```
1  public class Test1 {
2
3    public static void down(int n) {
4      //Abbruchkriterium
5      if (n==0) return;
6      //Verarbeitung
7      System.out.println(n);
8      //Rekursionsschritt
9      down(n-1);
10   }
11
12   public static void main(String[] args) {
13     down(500);
14   }
15 }
```



down(int n)
mit **Abbruchkriterium**
verhindert
unendliche Laufzeit.

Teile:
- Abbruchkriterium
- Verarbeitung
- Rekursionsschritt

Lassen sich die Fibonacci-Zahlen rekursiv berechnen?

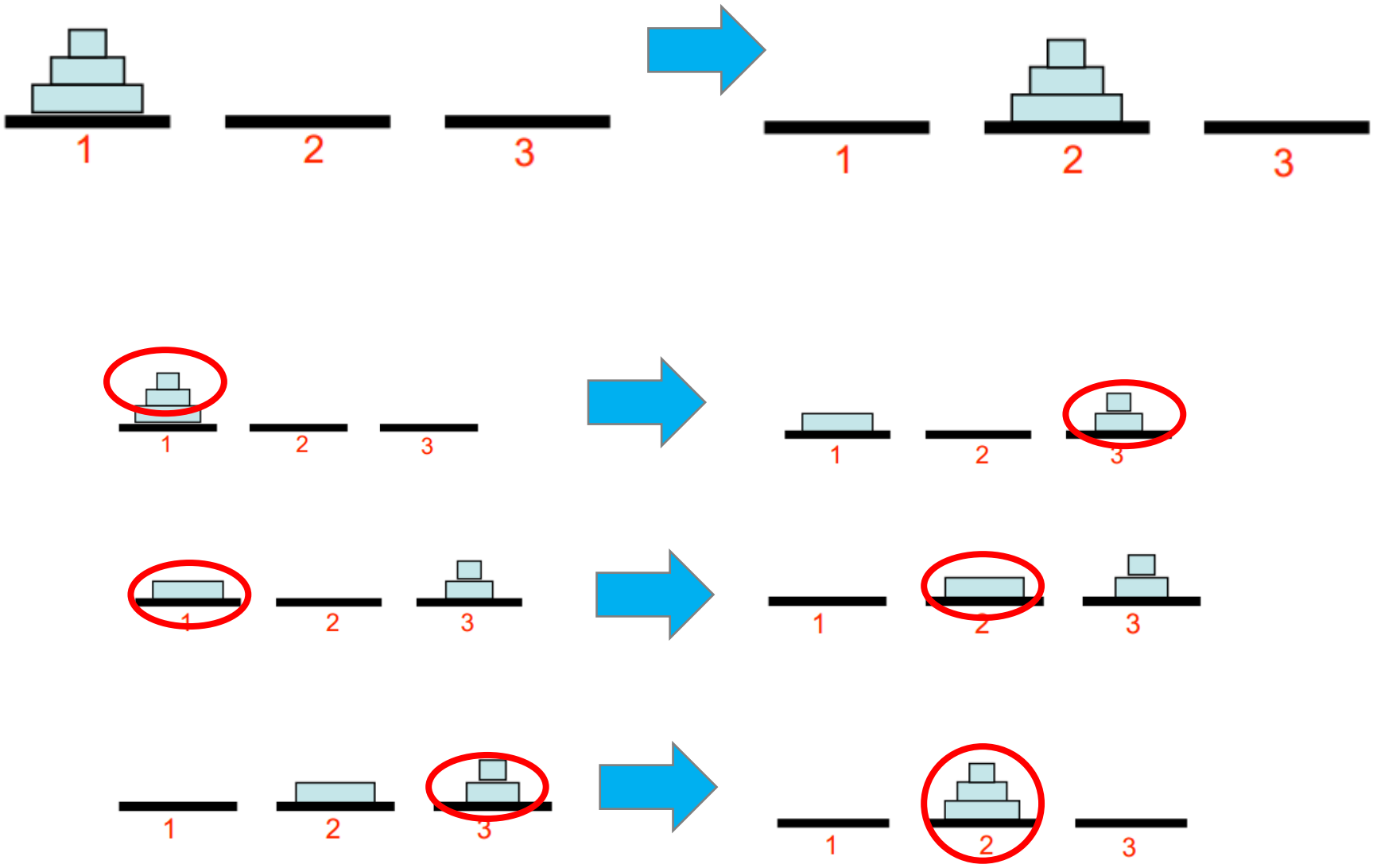
fib(0)=1, fib(1)=1
fib(n) = fib(n-1)+fib(n-2)

Ja, sie sind
schon rekursiv definiert.

```
1  public class Fibonacci2 {
2
3    public static long fib(int n) {
4      //Abbruchkriterium
5      if (n==0) return 1;
6      if (n==1) return 1;
7      //Rekursionsschritt
8      return fib(n-1)+fib(n-2);
9    }
10
11   public static void main(String[] args) {
12     System.out.println(fib(90));
13   }
14 }
```

Vorteil: elegante Definition
Nachteil: schlechte Laufzeit

Warum also Rekursion?
Beispiel: Die Türme von Hanoi



Die Türme von Hanoi
Java-Programm

```
1  public class TuermeVonHanoi1 {
2
3    public static void tauscheTurm(int n,int start, int ziel, int tmp) {
4      //Abbruchbedingung
5      if (n==1) {
6        System.out.println("Von "+start+" nach "+ziel);
7      } else {
8      //Rekursionsschritt
9      tauscheTurm(n-1,start,tmp,ziel);
10     System.out.println("Von "+start+" nach "+ziel);
11     tauscheTurm(n-1,tmp,ziel,start);
12     }
13   }
14
15   public static void main(String[] args) {
16     tauscheTurm(3,1,2,3);
17   }
18 }
```

Lösungsidee:

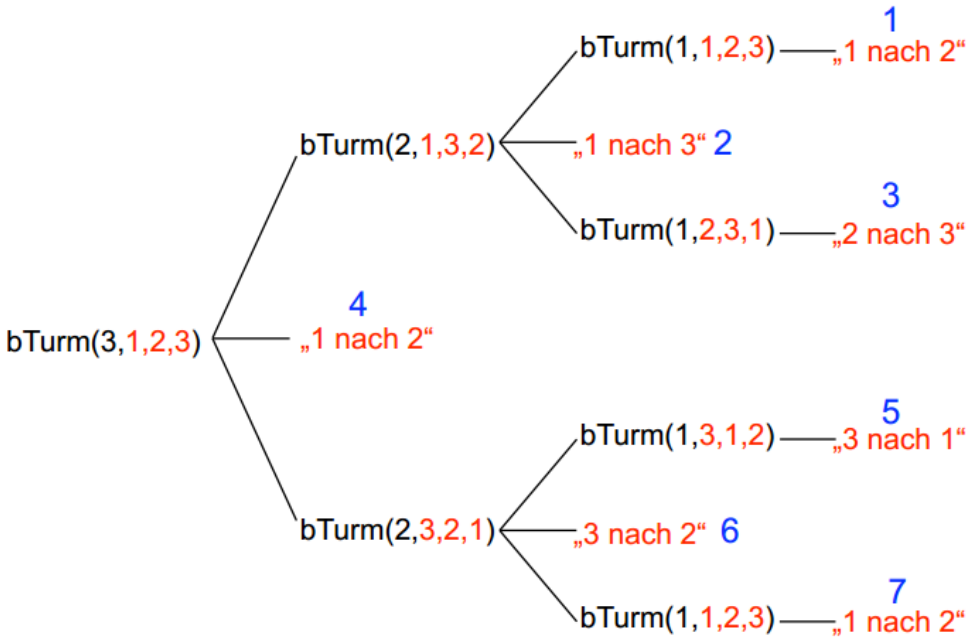
- Verschiebe den n-1-ten Stapel von 1 nach 3

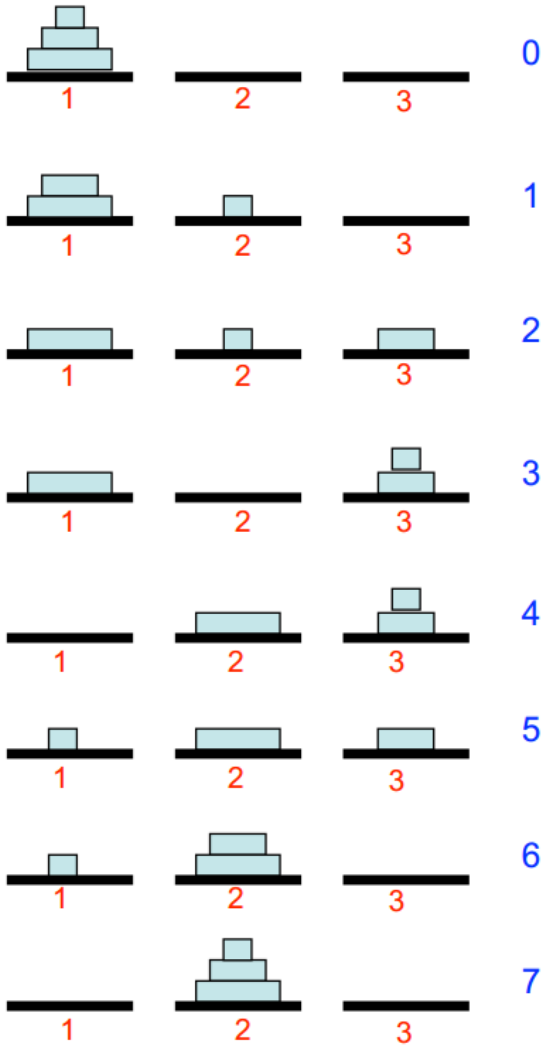
- Verschiebe die verbliebene Scheibe von 1 nach 2

- Verschiebe den n-1-ten Stapel von 3 nach 2

Die Türme von Hanoi
Aufrufreihenfolge

Animation des Aufrufs bewegeTurm(3,1,2,3):





Plätze

Zeitschritte

Quelle: O. Bittel; Sept. 2007

Zusammefassung Rekursion

Rekursive Methoden rufen sich selbst auf.

Sie brauchen eine Abbruchbedingung.

Sie sind bei bestimmten Problemen leicht zu formulieren.

Allerdings sind sie ineffizient bei langer Laufzeit.

Technik: Hilfsvariable in den Eingabeparametern

```
12  public class ErgebnisInEingabeparameter {
13
14      public static String ohneZiffern(String input,String output) {
15          //Abbruchbedingung der Rekursion
16          if (input.equals("")) return output;
17
18          //Eingabedaten aufspalten in erstesZeichen und Rest
19          String erstesZeichen=input.substring(0,1);
20          String rest = input.substring(1);
21
22          //Rekursion mit dem Rest, erstes Zeichen eventuell sammeln
23          if (!Character.isDigit(erstesZeichen.charAt(0)))
24            return ohneZiffern(rest,output+erstesZeichen);
25          else
26            return ohneZiffern(rest,output);
27      }
28
29
30      public static void main(String[] args) {
31          String input="a1b243c43d";
32          String output=ohneZiffern(input,"");
33          System.out.println("Eingabe:"+input);
34          System.out.println("Ausgabe:"+output);
35
36      }
37
38  }
```

Variable zum Sammeln des Ergebnises

Rückgabe der Variablen beim Abbruch

Variable output wird in der Rekursion erweitert

Zusatz
Eliminierung der Rekursion

Allgemeines Schema

Funktion mit endrekursivem Aufruf

```
RT f(T x)
{
    if ( Basisfall )
        return r;
    else
    {
        A
        return f(a);
    }
}
```

Iterative Funktion

```
RT f(T x)
{
    while (! Basisfall )
    {
        A
        x = a;
    }
    return r;
}
```

RT steht für einen für einen beliebigen Rückgabewerttyp.
Der Rückgabewerttyp kann auch void sein.

T steht für beliebigen Parametertyp. Im allgemeinen kann die Funktion f auch mehrere Parameter haben.

A steht für einen beliebigen Anweisungsblock.

Quelle: O. Bittel; Sept. 2007

Zusatz
Eliminierung der Rekursion mit einem Keller

```
void bewegeTurm(int n, int s, int z, int h)
{
  if (n == 1)
    cout << "Bewege Scheibe " << s " nach " << z ;
  else
  {
    bewegeTurm(n-1,s,h,z);
    cout << "Bewege Scheibe von " << s " nach " << z ;
    bewegeTurm(n-1,h,z,s);
  }
}
```

Statt rekursiver Aufrufe werden die Parameter für die spätere Bearbeitung eingekellert.

Man beachte die umgekehrte Reihenfolge der push-Befehle gegenüber den rekursive Aufrufen

```
void bewegeTurm(int n, int s, int z, int h)
{
  Stack s;
  s.push(n,s,z,h);

  while (! s.empty()) {
    s.pop(n,s,z,h);
    if (n == 1)
      cout << "Bewege Scheibe von " << s " nach " << z ;
    else {
      s.push(n-1,h,z,s);
      s.push(1,s,z,h);
      s.push(n-1,s,h,z);
    }
  }
}
```

Um die Darstellung kurz zu halten, wird eine push- bzw. pop-Operation vorausgesetzt, die 4 int-Werte in einem Schritt ein- bzw. auskellert.

Quelle: O. Bittel; Sept. 2007