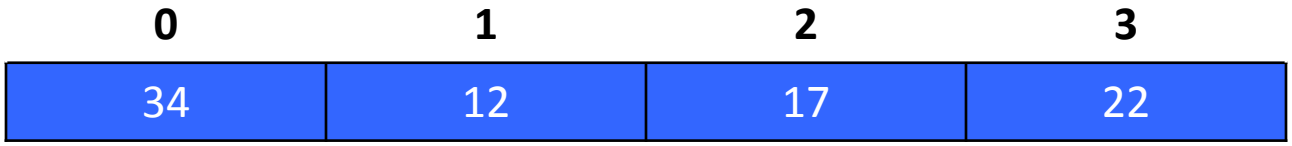
Javakurs FSS 2012

Lehrstuhl Stuckenschmidt

Tag 2 – Arrays, Methoden
und Rekursion

Arrays

- Arrays sind Folgen gleichartiger Variablen:
  - Mathematisch: x_0, x_1, x_2, ..., x_n
  - Java: x[0], x[1], x[2], ..., x[n]
- Arrays haben eine feste Länge



Deklarieren / Initialisieren

- Deklarieren
  - Bekannt machen (z.B. einer Variable)

```
int zahl;
char buchstabe;
```

- Initialisieren
  - Zuweisung eines Wertes

```
zahl = 5;
buchstabe = 'c';
```

Deklarieren eines Arrays

- Deklarieren
  - Bekannt machen (z.B. einer Variable)

```
int[] zahlen;
char[] wort;
```

  - `zahlen` ist eine Folge von ganzen Zahlen
  - `wort` ist eine Folge von Zeichen

Initialisieren eines Arrays

- Initialisieren mit new
  - Ausführliche Schreibweise:

```
int[] zahlen;                  //Deklaration
zahlen = new int[4];           //Default Werte
```

  - Kompakte Schreibweise:

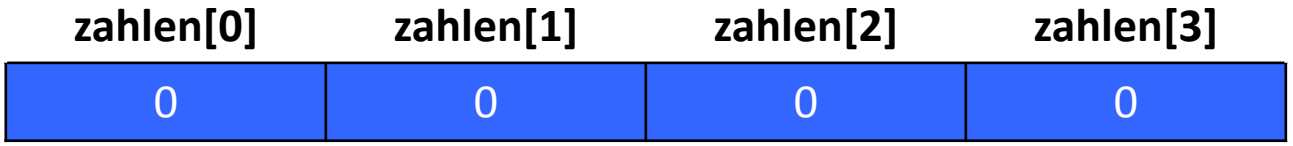
```
int[] zahlen = new int[4];
```

Array besteht aus Default Werten

Initialisierung über die Länge

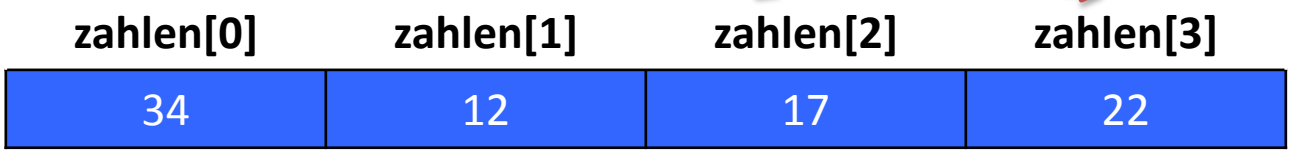
Initialisieren eines Arrays

- Zuweisen von eigenen Werten



  - Ähnlich wie bei primitiven Datentypen
  - Jedes Feld muss separat betrachtet werden

```
int[] zahlen = new int[4];
zahlen[0] = 34;
zahlen[1] = 12;
zahlen[2] = 17;
zahlen[3] = 22;
```



Initialisieren eines Arrays

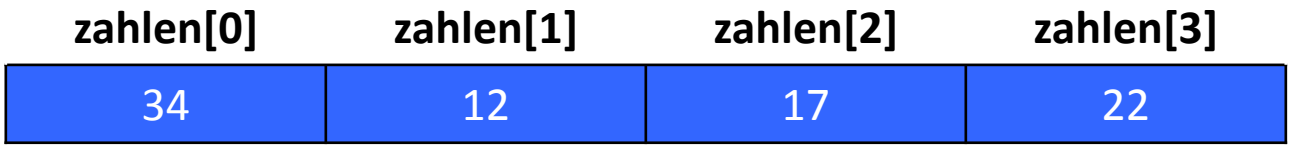
- Geht das nicht schneller?
  - Ausführliche Schreibweise

```
int[] zahlen;
zahlen = new int[]{34, 12, 17, 31};
```

  - Kompakte Schreibweise

```
int[] zahlen =new int[]{34, 12, 17, 31};
```

Keine Angabe der Länge!

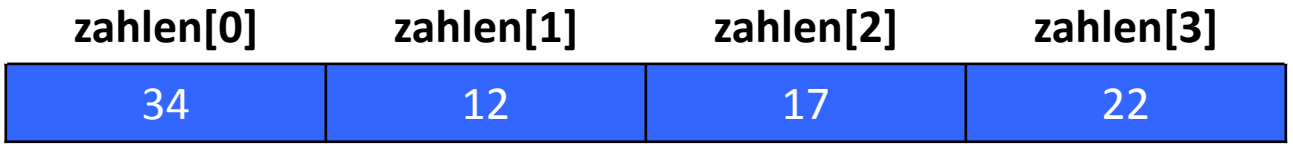


Initialisieren eines Arrays

- Nicht schnell genug?
  - Kompakte Schreibweise

```
int[] zahlen = {34, 12, 17, 31};
```

Vorsicht: Ausführliche Schreibweise existiert nicht!



Länge eines Arrays

- Jeder Array hat eine Variable `length`
  - Entspricht der Länge des Arrays
  - `int` und `final`
    - Auslesen der Länge:

```
int[] zahlen = {34, 12, 17, 31};
System.out.println(zahlen.length);
```

    - Konsolenausgabe:

```
4
```

Das letzte Feld eines Arrays ist `length-1`!!

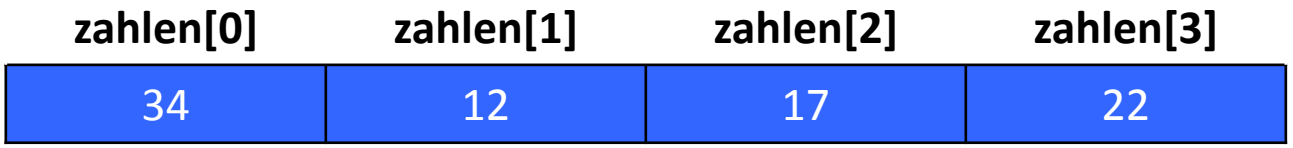
Aufrufen von Elementen

- Zugriff auf die Elemente über den Index

```
int[] zahlen = {34, 12, 17, 22};
System.out.println(zahlen[1]);
System.out.println(zahlen[2]);
System.out.println(zahlen[4]);
```

- Konsolenausgabe:

```
12
17
```



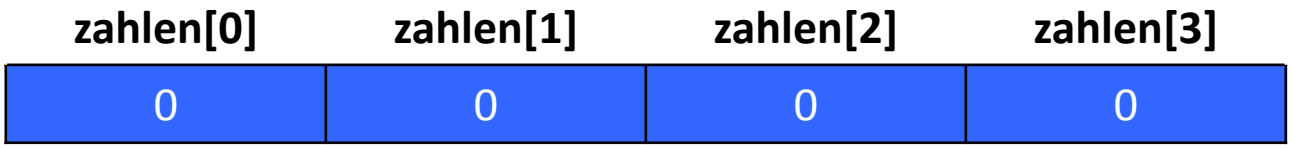
Aufrufen von Elementen

- Zugriff auf die Elemente über den Index

```
int[] zahlen = new int[4];
System.out.println(zahlen[1]);
System.out.println(zahlen[2]);
System.out.println(zahlen[4]);
```

- Konsolenausgabe:

```
0
0
```



Arrays und for-Schleifen

- Perfekt zum Durchlaufen von Arrays

```
char[] alphabet = {'a', 'b', 'c', ..., 'y', 'z'};
for(int n = 0; n < alphabet.length; n++){
   System.out.print(char[n] + " ");
}
```

- Konsolenausgabe:

```
a b c d e f g h i j k ...
```

Arrays und for-Schleifen

- Berechnung der Summe

```
int[] zahlen = {34, 12, 17, 22};
int sum = 0;
for(int n = 0; n < zahlen.length; n++){
    sum += zahlen[n];
}
System.out.println(sum);
```

- Konsolenausgabe:

```
85
```

Arrays und for-Schleifen

- Erweiterte for-Schleife

```
int[] zahlen = {34, 12, 17, 22};
int sum = 0;
for(int zahl : zahlen){
    sum += zahl;
}
System.out.println(sum);
```

- Konsolenausgabe:

```
85
```

Arrays und for-Schleifen

- Erweiterte for-Schleife
  - Kurze Schreibweise

```
for(int zahl : zahlen){
   //bla
}
```

  - Entspricht:

```
for(int n = 0, zahl; n < zahlen.length; n++){
   zahl = zahlen[n];
   //bla
}
```

Arrays und for-Schleifen

- Typische Suche in einem Array

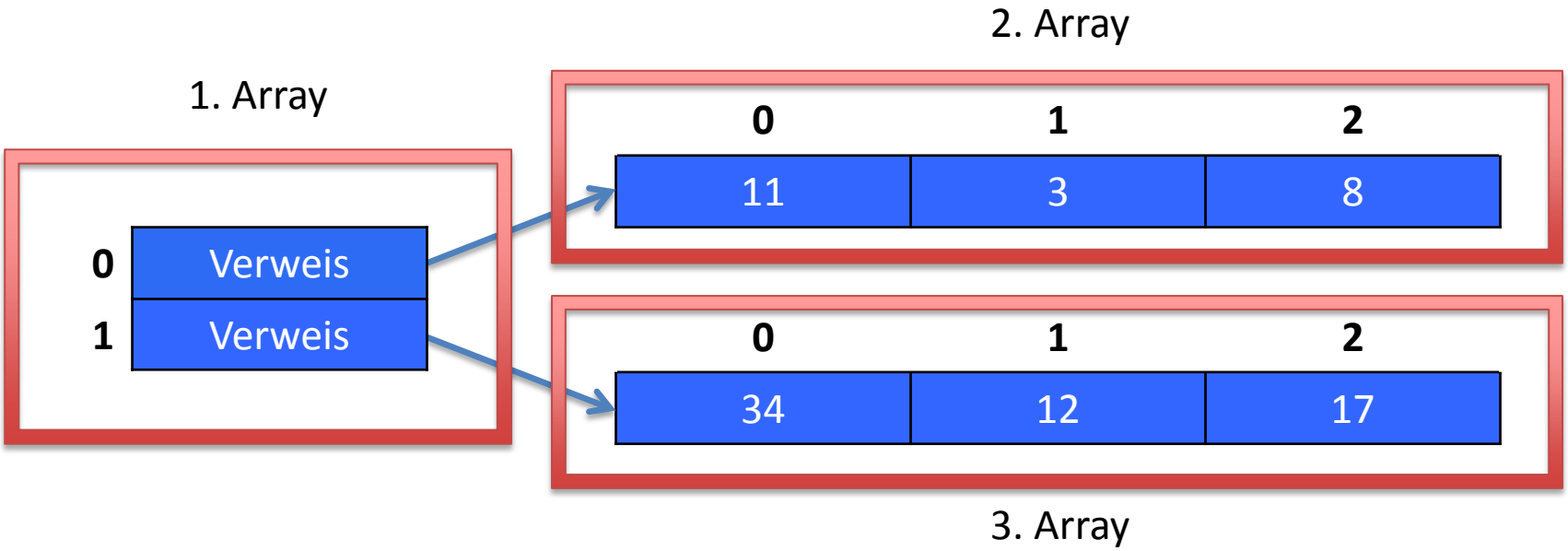
```
int position = 0;
for(int n = 0; n < alphabet.length; n++){
   if(alphabet[n] == 'd'){
      position = n + 1;
      break;
   }
}
System.out.println(position);
```

- Konsolenausgabe:

```
4
```

Mehrdimensionale Arrays

- Arrays von Arrays
  - Erste Dimension stellen die Zeilen dar
  - Zweite Dimension stellen die Spalten dar



Mehrdimensionale Arrays

- Nichtrechteckige Arrays
  - Jede Zeile kann eine eigene Größe haben!

Deklarieren und Initialisieren

- Deklarieren
  - Äquivalent wie beim eindimensionalen Array

```
int[][] matrix;
char[][] halloWelt;
```

  - Keine Unterschied zwischen rechteckigen und nichtrechteckigen Arrays

Initialisieren eines Arrays

- Initialisieren mit new
  - Rechteckige Arrays:

```
int[][] matrix = new int[2][3]
```

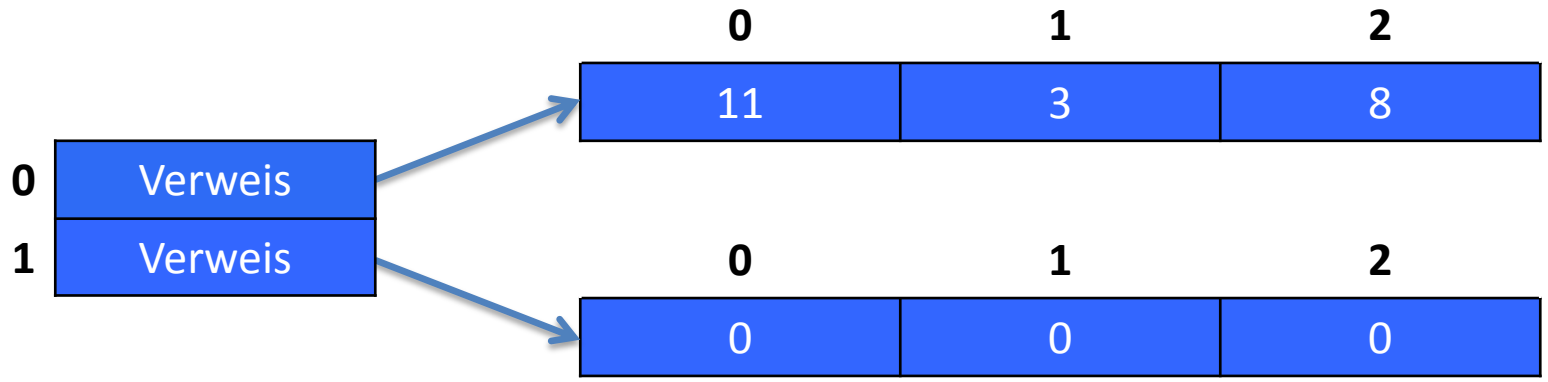
  - Nichtrechteckige Arrays

```
char[][] halloWelt = new char[2][];
halloWelt[0] = new char[5];
halloWelt[1] = new char[6]
```

Initialisieren eines Arrays

- Zuweisen von eigenen Werten
  - Ähnlich wie bei eindimensionalen Array
  - Jedes Feld muss separat betrachtet werden

```
int[][] matrix = new int[2][3];
zahlen[0][0] = 11;
zahlen[0][1] = 3;
zahlen[0][2] = 8;
```

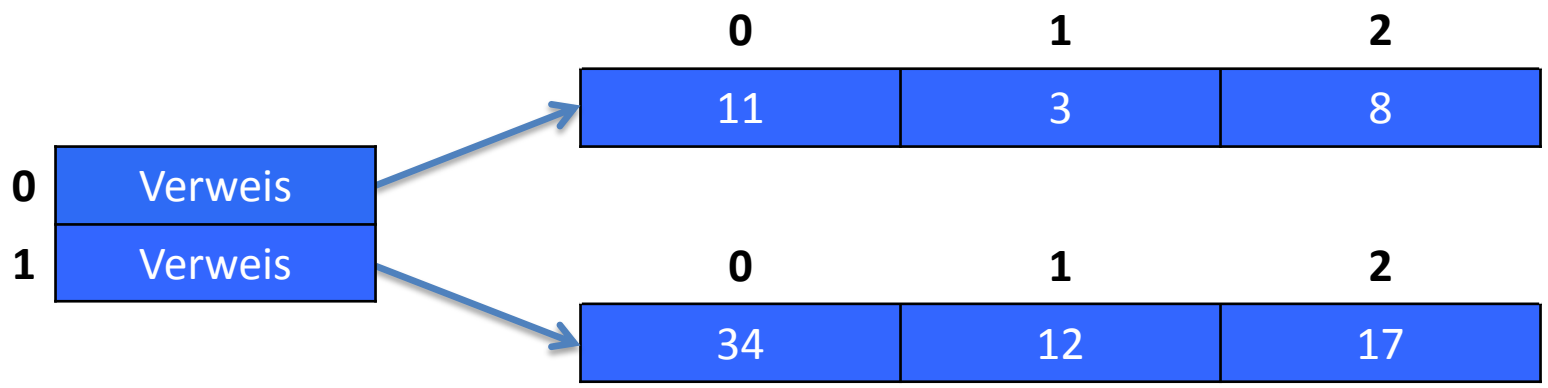


Initialisieren eines Arrays

- Geht das nicht schneller?

```
int[] matrix = {{11, 3, 8},{34, 12, 17}};
```

  - Mengenklammern trennen Dimensionen



Hallo Welt

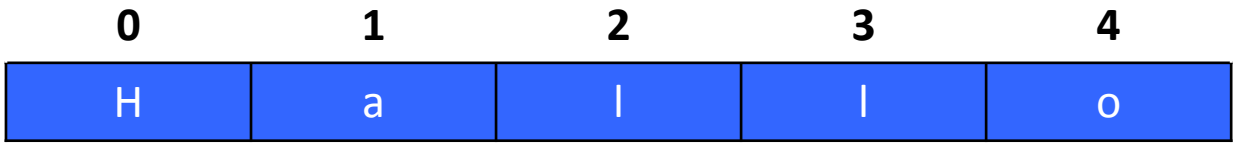
```
char[][] halloWelt =
                {{'H', 'a', 'l', 'l', 'o'},
                 {'W', 'e', 'l', 't'}};

for(char[] wort : halloWelt){
   for(char c : wort){
      System.out.print(c);
   }
}
```

String in Array

- toCharArray()
  - Wandelt einen String in einen Array um

```
String str = "Hallo";
str.toCharArray();
```



Array Exceptions

- Array Index Out Of Bounds Exception

```
int[] zahlen = {12, 32, 24, 17};
System.out.println(zahlen[4]);
```

  - Index im Array nicht vorhanden

- Null Pointer Exception

```
char[][] halloWelt = new char[2][];
System.out.println(halloWelt[0][0]);
```

  - 2te Dimension noch nicht initialisiert

Methoden

- Funktionen einer Klasse/Objektes

- Aber warum Methoden?
  - Zerlegung komplexer Programme in kleine Teile
  - Wiederkehrende „Programmteile“ schon verfügbar

```
public static void halloWelt(){
   System.out.println(“Hallo Welt!“);
}
```

Aufbau von Methoden

- Methoden bestehen aus:
  - Methodenkopf

```
public static void halloWelt()
```

```
private static double mwSt(double betrag)
```

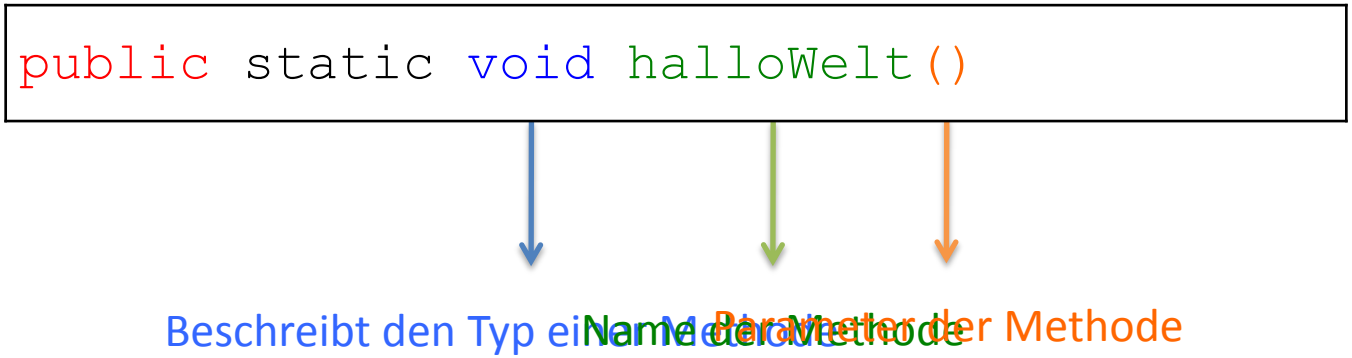
  - Methodenrumpf

```
{
   System.out.println(“Hallo Welt!“);
}
```

```
{
   return betrag * 0.19;
}
```

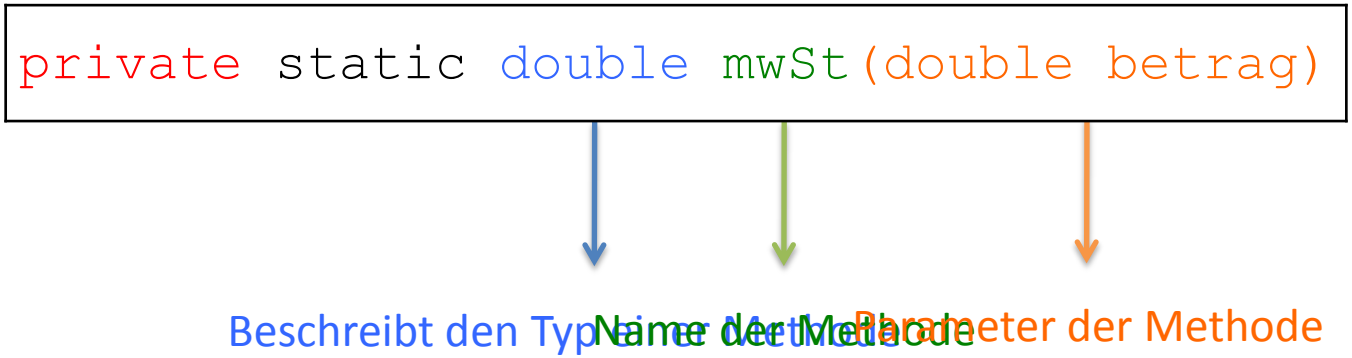
Aufbau von Methoden

- Methodenkopf
  - Sichtbarkeit
  - Methodentyp (Rückgabetyp)
  - Methodenname
  - Methodenparameter



Aufbau von Methoden

- Methodenkopf
  - Sichtbarkeit
  - Methodentyp (Rückgabetyp)
  - Methodenname
  - Methodenparameter



Aufbau von Methoden

- Methodenrumpf
  1. Öffnenden Klammer
  2. Folge von Anweisungen
  3. `return` (optional bei `void`)
  4. Schließende Klammer

```
{
    System.out.println("Hallo Welt!");
}
```

```
{
    return betrag * 0.19;
}
```

Return

- `return`
  - Ein Rückgabewert pro Methode
  - Beendet Methode (vorzeitig)

```
static void sqrt(double d){
    if ( d < 0){
        return;
    }
    System.out.println(Math.sqrt(d));
}
```

```
static int soNicht(){
    int i = 0;
    return i;
    i = 2;                    //Unreachable Code
}
```

Methodentypen

- `void`
  - Verwendung für Methoden **ohne** Rückgabewerte
  - Bzw für Methoden die keinen Wert berechnen
  - `return` optional ohne Wert

```
public static void halloWelt()
{
   System.out.println(“Hallo Welt!“);
}
```

Methodentypen

- Datentypen
  - Verwendung für Methoden **mit** Rückgabewerte
  - Benötigt `return`
  - Rückgabewert muss der selbe Datentyp sein
    - » `int`, `char`, `double`... aber auch Objekte

```
public static double mwSt(int betrag)
{
   return betrag * 0.19;
}
```

Methodenparameter

- Parametrisierte Funktion
  - Übergabe von Werten an eine Methode
  - Methode kann diese Werte verwenden
  - Generische Methoden möglich

```
public static double mwSt(double betrag, double satz)
{
   System.out.println("Methode mit variabler MwSt!");
   return betrag * satz;
}
```

Methodenparameter

- Wertübergabe
  - Übergabe von Werten an Methode bei Aufruf
  - Bei gleichem Methodennamen wird mit Parameter zwischen den einzelnen Methoden differenziert

```
public static void main(String[] args){
    System.out.println(mwSt(10));
    System.out.println(mwSt(10,0.19));
}
```

  - Konsolenausgabe:

```
Methode mit fester MwSt!
1.9
Methode mit variabler MwSt!
1.9
```

Rekursion

- „Zurücklaufen“
  - Eine Funktion durch sich selbst zu definieren

```
static void inBinaer(int dezimal){
   while(dezimal > 0){
      System.out.print(dezimal % 2);
      dezimal = dezimal / 2;
   }
}
```

  - Problem Binärzahl ist spiegelverkehrt!

Rekursion

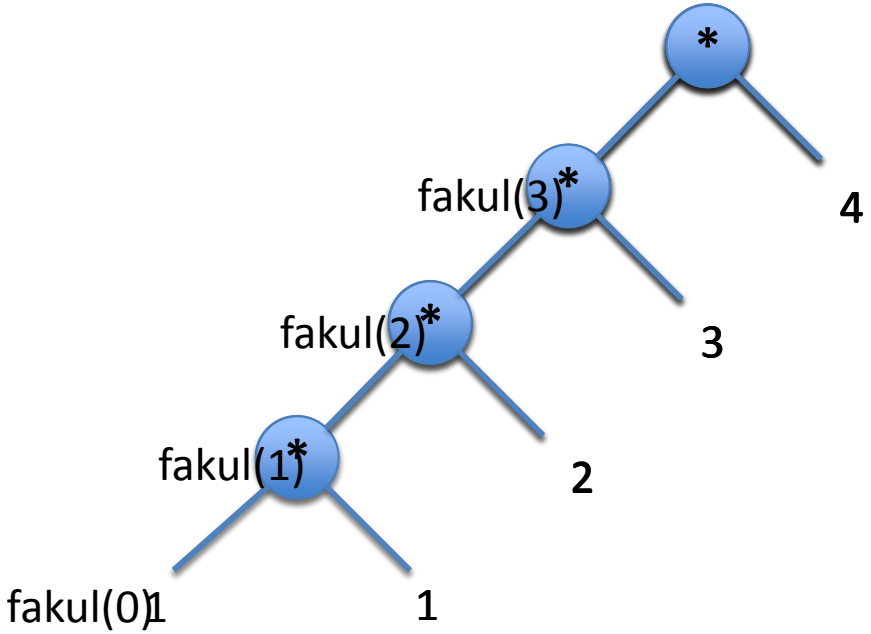
- „Zurücklaufen“
  - Eine Funktion durch sich selbst zu definieren

```
static void inBinaer(int dezimal){
   if(dezimal < 2)
      System.out.print(dezimal);
   else{
      inBinaer(dezimal / 2);
      System.out.print(dezimal % 2);
   }
}
```

Wie funktioniert Rekursion?

```
static int fakul(int zahl) {
    if(n == 0)
        return 1;
    else
        return fakul(n-1)* n;
}
```

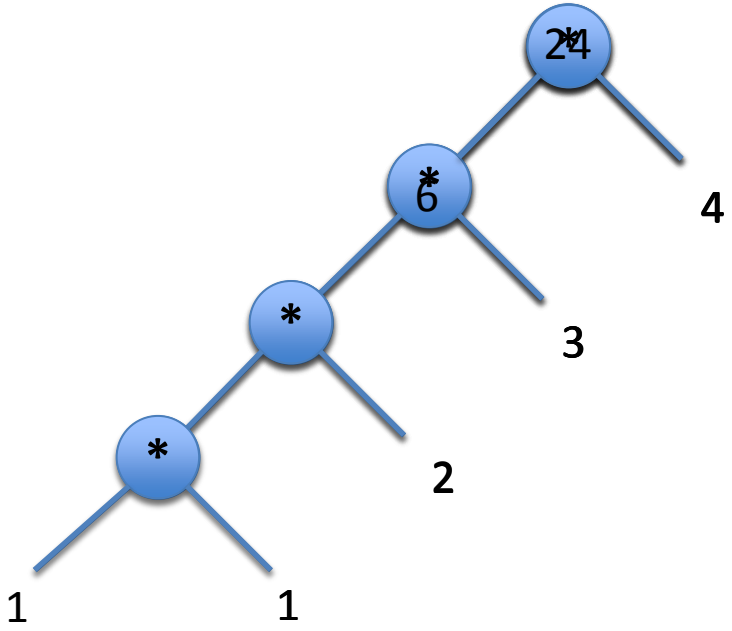
fakul(4) =



Wie funktioniert Rekursion?

```
static int fakul(int zahl) {
    if(n == 0)
        return 1;
    else
        return fakul(n-1)* n;
}
```

fakul(4) =



Wechselseitige Rekursion

- Zwei oder mehr Funktionen
  - Wechselseitigen Bezug aufeinander

```
static boolean istUngerade(int zahl) {
    if(zahl == 0) {
        return false;
    }
    return istGerade(zahl-1);
}

static boolean istGerade(int zahl) {
    if(zahl == 0){
        return true;
    }
    return istUngerade(zahl-1);
}
```

Endlose Rekursion

- Terminiert nicht (Ähnlich wie bei Schleifen)

Fragen?