PGdP

Style Guide

Algorithmen

Stack

Rekursion

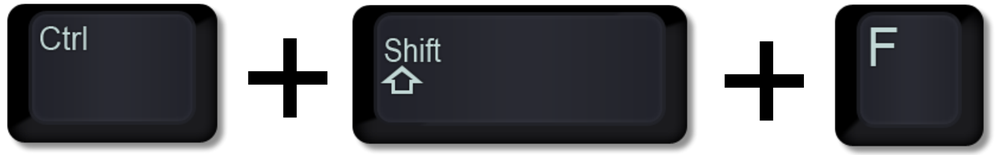
P-Aufgaben

Folien: go.tum.de/904005

Tobias Schamel

Style Guide

- Vor der Abgabe einmal (bei Eclipse) *Str+Shift+F* drücken, um die **Einrückungen** richtig zu machen



- Sinnvolle Datentypen verwenden (Bsp. für Einwohnerzahl int, nicht double)
- Sinnvolle Variablennamen verwenden (One Character Namen nur für Laufvariablen z.B. in for-Schleifen)
- **Attribut-, Methoden- und Variablennamen klein** schreiben
  > für zusammengesetzte Wörter **Camel Case** verwenden
  z.B. calcSumOfTwoInt();

Style Guide

- **Klassennamen** werden **groß** geschrieben (zusammengesetzte Worte Camel Case, Bsp. PascalDreieck.class)
- Einrückungen konsistent halten (Str+Shif+F), nicht mehr als **ein Befehl pro Zeile**, max. **eine Leerzeile**, Zeilen nicht breiter als **120 Zeichen**
- Kommentierung des 'nicht offensichtlichen', **vor** der zu erklärenden Stelle

Algorithmen

Eventuell nochmal in H-Aufgaben relevant, bisher **noch** nie in der Klausur.

- Sortieralgorithmen
- Übersicht Algorithmen

Stack

Erfunden vom TUM-Professor Friedrich L. Bauer

- FIFO (First-In-First-Out)
- zwei Operationen

```
push(data); //auf den Stack legen
pop();  //vom Stack nehmen
```

Stack

Erfunden vom TUM-Professor Friedrich L. Bauer

- FIFO (First-In-First-Out)
- zwei Operationen

```
push(data); //auf den Stack legen
pop();  //vom Stack nehmen
```

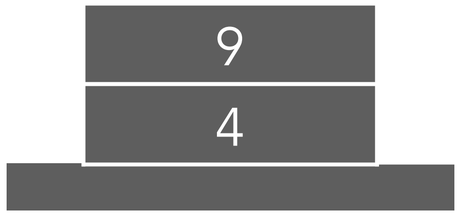
`push(4)`

Stack

Erfunden vom TUM-Professor Friedrich L. Bauer

- FIFO (First-In-First-Out)
- zwei Operationen

```
push(data); //auf den Stack legen
pop();  //vom Stack nehmen
```



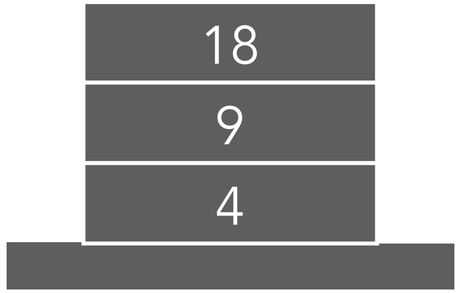

```
push(9)
```

Stack

Erfunden vom TUM-Professor Friedrich L. Bauer

- FIFO (First-In-First-Out)
- zwei Operationen

```
push(data); //auf den Stack legen
pop();  //vom Stack nehmen
```




```
push(18)
```

Stack

Erfunden vom TUM-Professor Friedrich L. Bauer

- FIFO (First-In-First-Out)
- zwei Operationen

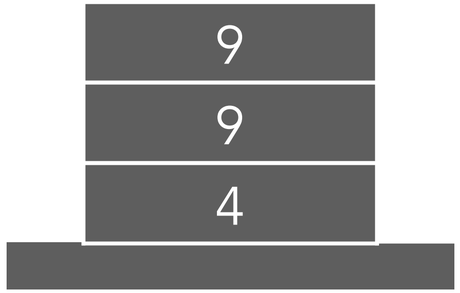
```
push(data); //auf den Stack legen
pop();  //vom Stack nehmen
```

Stack

Erfunden vom TUM-Professor Friedrich L. Bauer

- FIFO (First-In-First-Out)
- zwei Operationen

```
push(data); //auf den Stack legen
pop();  //vom Stack nehmen
```



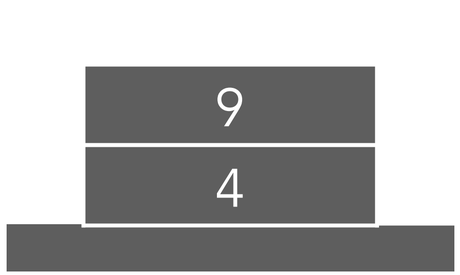

```
push(9)
```

Stack

Erfunden vom TUM-Professor Friedrich L. Bauer

- FIFO (First-In-First-Out)
- zwei Operationen

```
push(data); //auf den Stack legen
pop();  //vom Stack nehmen
```



Stack

Erfunden vom TUM-Professor Friedrich L. Bauer

- FIFO (First-In-First-Out)
- zwei Operationen

```
push(data); //auf den Stack legen
pop();  //vom Stack nehmen
```

Stack

Erfunden vom TUM-Professor Friedrich L. Bauer

- FIFO (First-In-First-Out)
- zwei Operationen

```
push(data); //auf den Stack legen
pop();  //vom Stack nehmen
```

Stack

Erfunden vom TUM-Professor Friedrich L. Bauer

- FIFO (First-In-First-Out)
- zwei Operationen

```
push(data); //auf den Stack legen
pop();  //vom Stack nehmen
```

`pop()` ⟶ Fehler

Rekursion

Anwendung: Rekursion

- Rekursionsanfang
- Rekursionsschritt
- Abbruchbedingung

Tail-Rekursion funktioniert läuft in der Realität ohne Stack! (Compiler optimiert)

Bsp: Fakultät

$$f(n) = \begin{cases} n = 1 & 1 \\ n > 1 & n \cdot f(n-1) \end{cases}$$

Rekursion

Anwendung: Rekursion, Bsp. Fakultät

```
int fac(int n) {
  if(n == 1)
    return 1;
  else
    return n*fac(n-1);
}
```

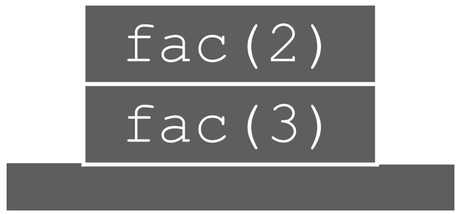
Tail-Rekursion funktioniert läuft in der Realität ohne Stack! (Compiler optimiert)

Rekursion

Anwendung: Rekursion, Bsp. Fakultät

```
int fac(int n) {
  if(n == 1)
    return 1;
  else
    return n*fac(n-1);
}
```

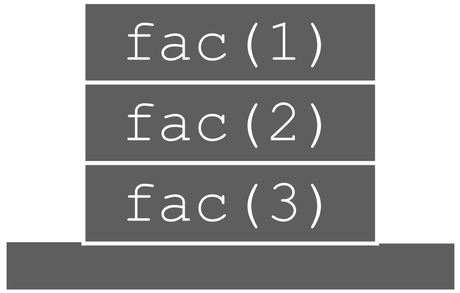
Tail-Rekursion funktioniert läuft in der Realität ohne Stack! (Compiler optimiert)



Rekursion

Anwendung: Rekursion, Bsp. Fakultät

```
int fac(int n) {
  if(n == 1)
    return 1;
  else
    return n*fac(n-1);
}
```



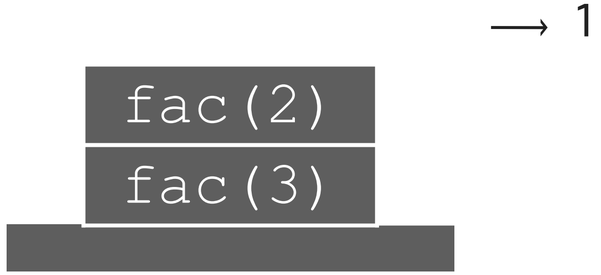
Tail-Rekursion funktioniert läuft in der Realität ohne Stack! (Compiler optimiert)

Rekursion

Anwendung: Rekursion, Bsp. Fakultät

```
int fac(int n) {
  if(n == 1)
    return 1;
  else
    return n*fac(n-1);
}
```

Tail-Rekursion funktioniert läuft in der Realität ohne Stack! (Compiler optimiert)



Rekursion

Anwendung: Rekursion, Bsp. Fakultät

```
int fac(int n) {
  if(n == 1)
    return 1;
  else
    return n*fac(n-1);
}
```

Tail-Rekursion funktioniert läuft in der Realität ohne Stack! (Compiler optimiert)

⟶ 1•2

Rekursion

Anwendung: Rekursion, Bsp. Fakultät

```
int fac(int n) {
  if(n == 1)
    return 1;
  else
    return n*fac(n-1);
}
```

Tail-Rekursion funktioniert läuft in der Realität ohne Stack! (Compiler optimiert)

⟶ (1•2)•3

P04.01

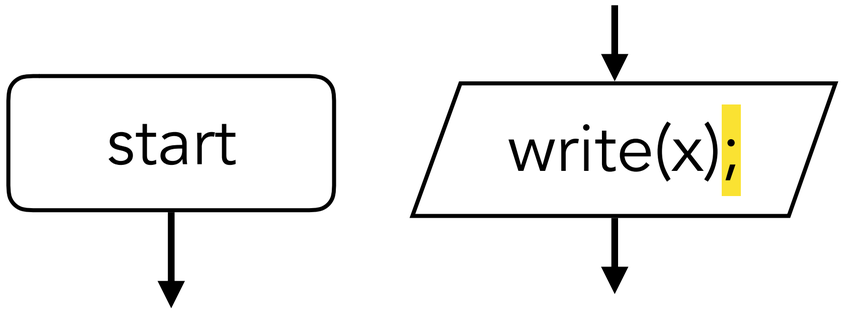
Kontrollflussdiagramm für folgendes Programm:

```
1  int x,r;
2  int n;
3  r = 1;
4  n = 1;
5  x = readInt();
6
7
8
9
```

```
10 while (n < x) {
11     if (r % 1 == 0)
12         r = r * n;
13     else {
14         r = r * (-n);
15         n = n + 1;
16     }
17   write (r);
18 }
```

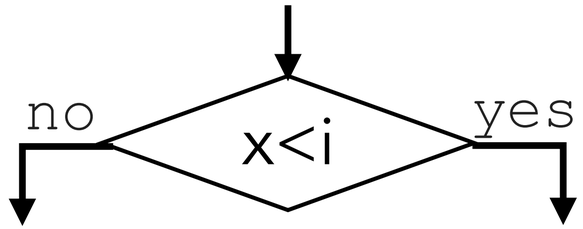
P04.01

Recap: Kontrollflussdiagramme

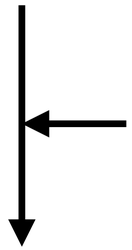


Startknoten

Ausgabe

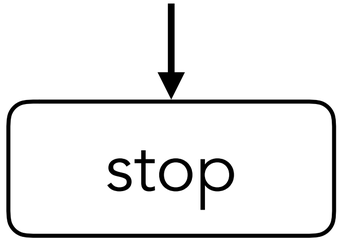


Bedingte Verzweigung

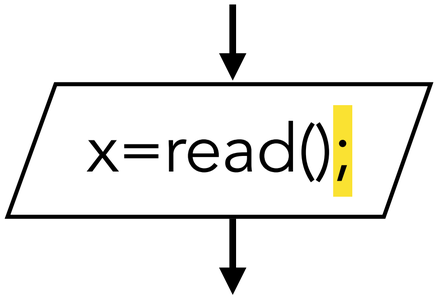


Zusammenlauf

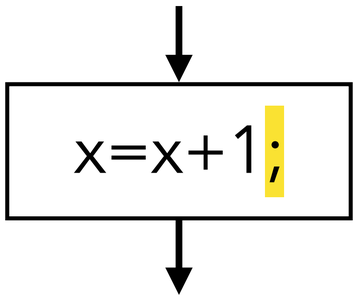
Endknoten



Eingabe

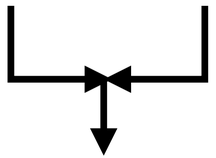


Zuweisung

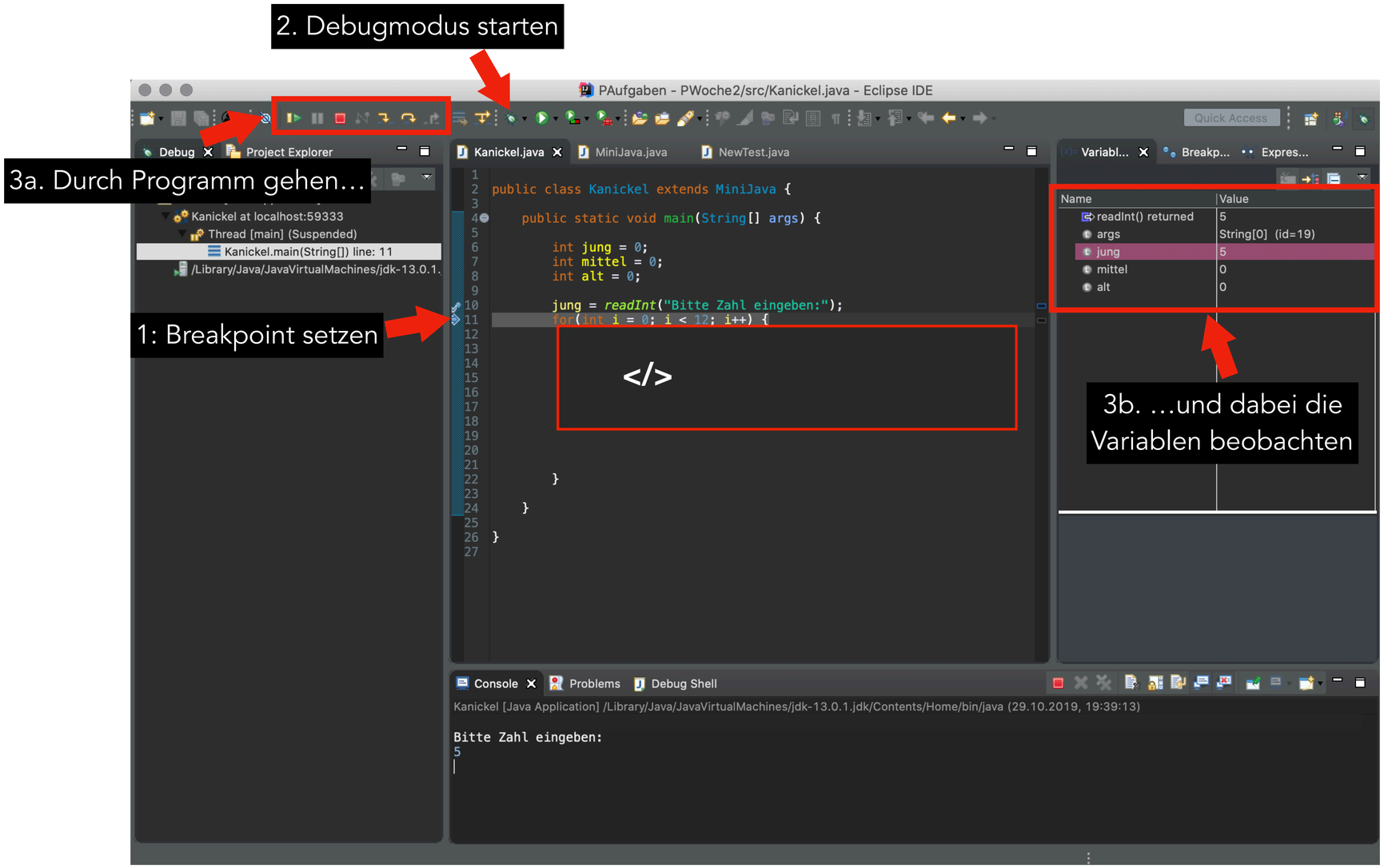


Zusammenlauf

Nach if/else



P04.01



P04.01

Zustandstabellen, Breakpoint an allen Zuweisungen

```
1  int x, r, n;
2  r = 1;
3  n = 1;
4  x = readInt();
5  while (n < x) {
6      if (r % 1 == 0)
7          r = r * n;
8      else {
9          r = r * (-n);
10         write(r);
11     }
12     n = n + 1;
13 }
14 write(r);
```

Z.	x	r	n
2	–	1	–
3	–	1	1
4	3	1	1
7	3	1	1

Usw…

P04.02

Recap: Arrays & For-Schleifen

Deklaration

```
int[] a = new int[length];
```

Länge eines Arrays

```
a.length
```

For-Schleife

```
for(init; cond; modify) {stmt}
```

Startwert, Endwert, Schrittweite

```
for(int i=0; i<a.length; i++)
```

Arrays werden von 0 bis $\text{length} - 1$ indiziert.

P04.02

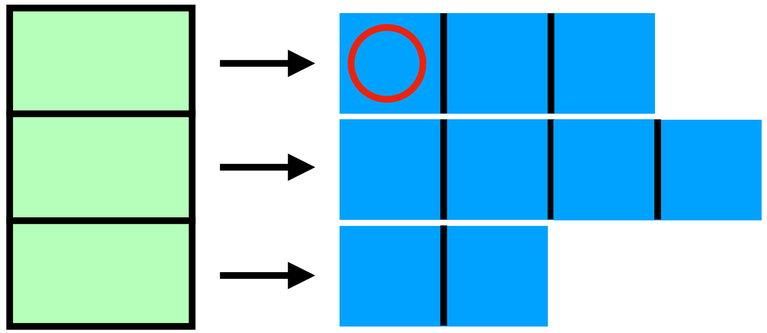
Recap: Arrays & For-Schleifen

Schleifen über zweidimensionale Arrays

```
int[][] arr = new [...];
for(int i = 0; i < arr.length; i++)
  for(int j = 0; j < arr[i].length; j++)
    write(arr[i][j]);
```

0

0



P04.02

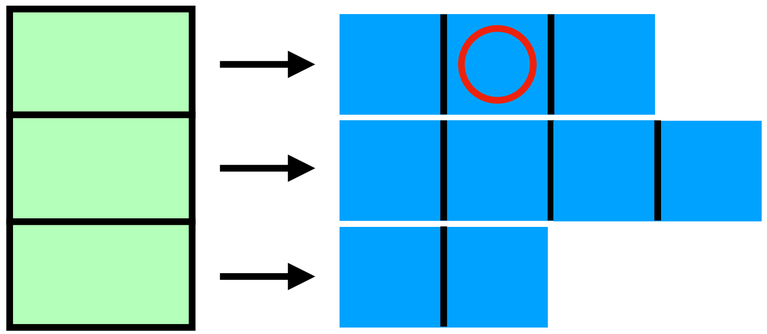
Recap: Arrays & For-Schleifen

Schleifen über zweidimensionale Arrays

```
int[][] arr = new [...];
for(int i = 0; i < arr.length; i++)
  for(int j = 0; j < arr[i].length; j++)
    write(arr[i][j]);
```

0

1



P04.02

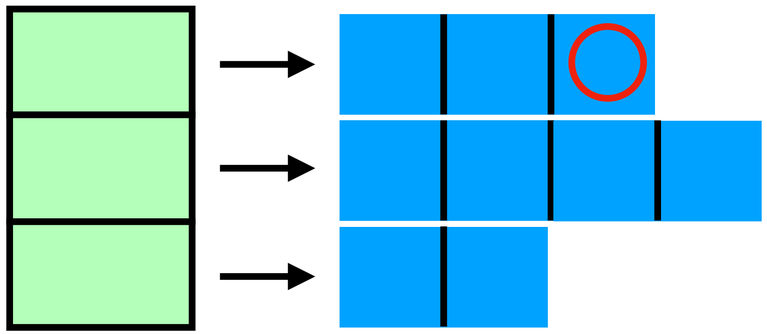
Recap: Arrays & For-Schleifen

Schleifen über zweidimensionale Arrays

```
int[][] arr = new [...];
for(int i = 0; i < arr.length; i++)
  for(int j = 0; j < arr[i].length; j++)
    write(arr[i][j]);
```

0

2



P04.02

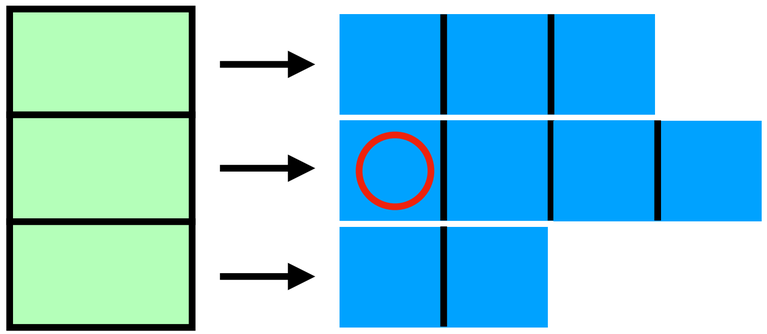
Recap: Arrays & For-Schleifen

Schleifen über zweidimensionale Arrays

```
int[][] arr = new [...];
for(int i = 0; i < arr.length; i++)
  for(int j = 0; j < arr[i].length; j++)
    write(arr[i][j]);
```

1

0



P04.02

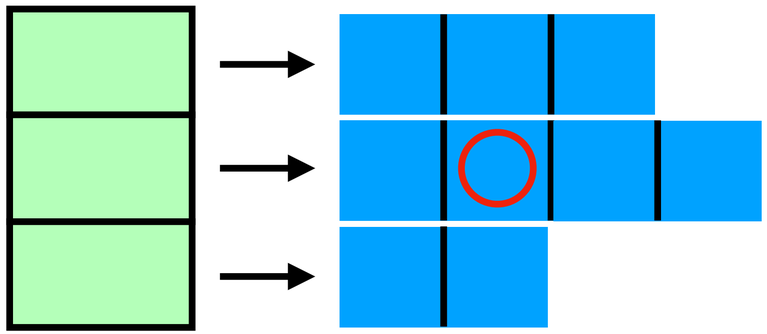
Recap: Arrays & For-Schleifen

Schleifen über zweidimensionale Arrays

```
int[][] arr = new [...];
for(int i = 0; i < arr.length; i++)
  for(int j = 0; j < arr[i].length; j++)
    write(arr[i][j]);
```

1

1



P04.02

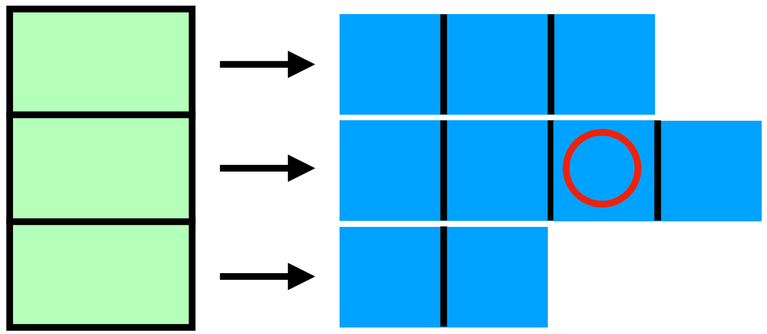
Recap: Arrays & For-Schleifen

Schleifen über zweidimensionale Arrays

```
int[][] arr = new [...];
for(int i = 0; i < arr.length; i++)
  for(int j = 0; j < arr[i].length; j++)
    write(arr[i][j]);
```

1

2



P04.02

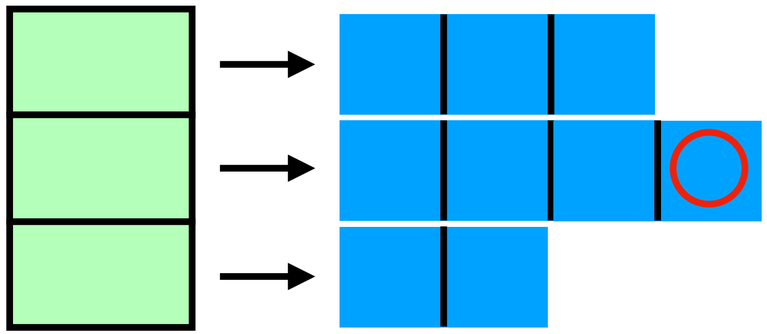
Recap: Arrays & For-Schleifen

Schleifen über zweidimensionale Arrays

```
int[][] arr = new [...];
for(int i = 0; i < arr.length; i++)
  for(int j = 0; j < arr[i].length; j++)
    write(arr[i][j]);
```

1

3



P04.02

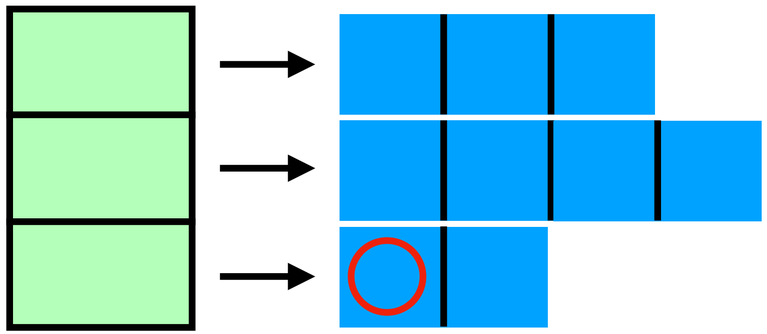
Recap: Arrays & For-Schleifen

Schleifen über zweidimensionale Arrays

```
int[][] arr = new [...];
for(int i = 0; i < arr.length; i++)
  for(int j = 0; j < arr[i].length; j++)
    write(arr[i][j]);
```

2

0



P04.02

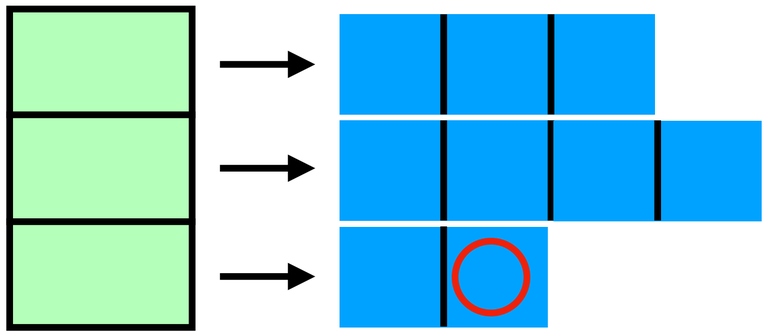
Recap: Arrays & For-Schleifen

Schleifen über zweidimensionale Arrays

```
int[][] arr = new [...];
for(int i = 0; i < arr.length; i++)
  for(int j = 0; j < arr[i].length; j++)
    write(arr[i][j]);
```

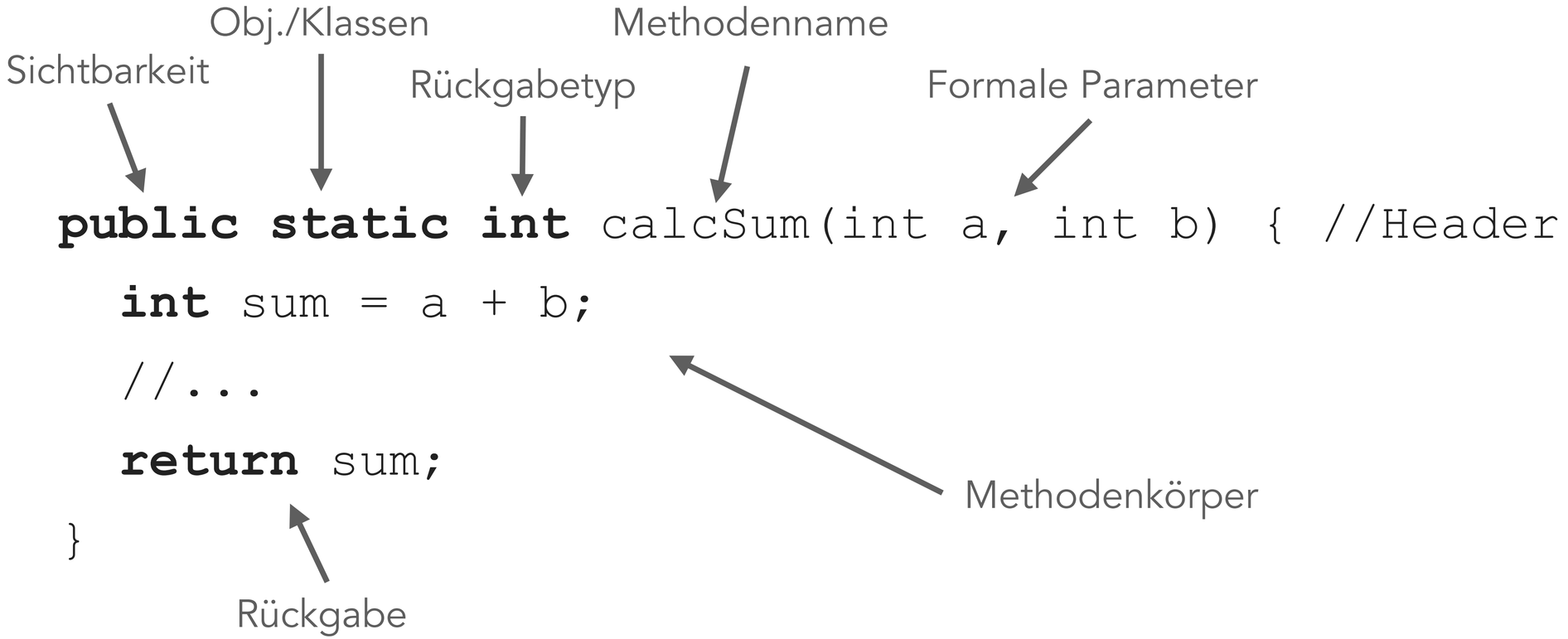
2

1



P04.02

Recap: Methoden



P04.02

Grundlegende Array-Funktionen

1: Array auf Konsole ausgeben

```
public static void print(int[] feld)
print(new int[] {1, 2, 3});
⇒{1, 2, 3}
```

2: Minimum & Maximum auf Konsole Ausgeben

```
public static void minUndMax(int[] feld)
minUndMax(new int[] {1, -3, 8, 99, -43});
⇒Minimum = -43, Maximum = 99
```

P04.02

Grundlegende Array-Funktionen

Plausibilitätsabfrage fehlt bei allen Lösungen!

1: Array auf Konsole ausgeben

```
public static void print(int[] feld) {
  writeConsole("{");
  for(int i = 0; i < feld.length-1; i++)
    writeConsole(feld[i]+", ");
  writeConsole(feld[feld.length-1]+"}");
  return; //optional
}
```

P04.02

Grundlegende Array-Funktionen

2: Minimum & Maximum

Plausibilitätsabfrage fehlt bei allen Lösungen!

```
public static void minUndMax(int[] feld) {
  int min = feld[0];
  int max = feld[0];
  for(int i = 0; i < feld.length; i++){
    if(min > feld[i]) min = feld[i];
    if(max < feld[i]) max = feld[i];
  }
writeConsole("Minimum = "+min+", Maximum = "+max);
}
```

P04.02

Grundlegende Array-Funktionen

3: Invertieren

```
public static int[] invertieren(int[] feld)
invertieren(new int[] {1, 2, 3});
//returns {3, 2, 1}
```

Kein Konsolenoutput sondern Rückgabe der Referenz auf das neue Array!

P04.02

Grundlegende Array-Funktionen

3: Invertieren

Plausibilitätsabfrage fehlt bei allen Lösungen!

```
public static void invertieren(int[] feld) {
  for(int i = 0; i < feld.length/2; i++) {
    int speicher = feld[i];
    feld[i] = feld[feld.length-i-1];
    feld[feld.length-i-1] = speicher;
  }
}
```

Funktioniert das?

P04.02

Grundlegende Array-Funktionen

Plausibilitätsabfrage fehlt bei allen Lösungen!

3: Invertieren

```
public static int[] invertieren(int[] feld) {
  int[] inverted = new int[feld.length];
  for(int i = 0; i < feld.length; i++)
    inverted[i] = feld[feld.length-i-1];
  return inverted;
}
```

P04.02

Grundlegende Array-Funktionen

4: Schneiden

```
public static int[] schneiden(int[] feld,
  int laenge)
schneiden(new int[] {1, 2, 3}, 2);
//returns {1, 2}

schneiden(new int[] {1, 2, 3}, 4);
//returns {1, 2, 3, 0}
```

P04.02

Grundlegende Array-Funktionen

4: Schneiden

```
public static int[] schneiden(int[] feld,
    int laenge) {
    int[] geschnitten = new int[laenge];
    for(int i = 0; i < feld.length &&
        i < laenge; i++)
        geschnitten[i] = feld[i];
    return geschnitten;
}
```

P04.02

Grundlegende Array-Funktionen

5: Linearisieren

```
public static int[] linearisieren(int[][] feld)
linearisieren(new int[][] {
{4, 8, 6},
{1, 2}
});
//returns {4, 8, 6, 1, 2}
```

P04.02

Grundlegende Array-Funktionen

5: Linearisieren

```
public static int[] linearisieren(int[][] feld) {
  int length = 0;
  for(int i = 0; i < feld.length; i++)
    length += feld[i].length;
  int[] linear = new int[length]; int c = 0;
  for(int i = 0; i < feld.length; i++)
    for(int j = 0; j < feld[i].length)
      linear[c++] = feld[i][j];
  return linear; } //bad style
```

P04.03

Zahleneigenschaften

1: Primzahl

```
public static boolean isPrime(int n)
isPrime(5);
//returns true
isPrime(14);
//returns false
```

P04.03

Zahleneigenschaften

1: Primzahl

```
public static boolean isPrime(int n) {
  boolean isPrim = true;
  for(int teiler = 2; teiler < n; teiler++)
    if(n%teiler == 0)
      isPrim = false;
  return isPrim;
}
```

P04.03

Zahleneigenschaften

2: Quersumme

```
public static int crossSum(int n)
crossSum(5678);
//returns 26
crossSum(14);
//returns 5
```

P04.03

Zahleneigenschaften

2: Quersumme

```
public static int crossSum(int n) {
    //siehe P03.03 > erhalte int[] nArr
    int sum = 0;
    for(int i = 0; i < nArr.length; i++)
      sum+=nArr[i];
    return sum;
}
```

P04.03

Zahleneigenschaften

3: Quadratzahl

```
public static int squareOf(int n)
squareOf(4);
//returns 2
squareOf(196);
//returns 8
```

P04.03

Zahleneigenschaften

3: Quadratzahl

```
public static int squareOf(int n) {
  int testing = 1;
  while(testing*testing < n)
    testing++;
  if(testing*testing == n)
    return testing;
  else
    return -1;
}
```

P04.03

Zahleneigenschaften

4: Betrag

```
public static int abs(int n)
abs(245);
//returns 245
abs(-14);
//returns 14
```

P04.03

Zahleneigenschaften

4: Betrag

```
public static int abs(int n) {
  if(n < 0)
    return n*(-1);
  else
    return n;
}
```

P04.04

Arrayoperationen

1: Skalarprodukt

```
public static int scalarProduct(int[] v)
```

$$\langle a, a \rangle = a \cdot a = \sum_{j=1}^{n} (a_j \cdot a_j) \ : \ a \in \mathbb{R}^n$$

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} \cdot \begin{pmatrix} a \\ b \\ c \end{pmatrix} \quad = \quad a \cdot a \ + \ b \cdot b \ + \ c \cdot c$$

P04.04

Arrayoperationen

1: Skalarprodukt

```
public static int scalarProduct(int[] v) {
  int prod = 0;
  for(int i = 0; i < v.length; i++)
    prod+= v[i]*v[i];
 return prod;
}
```

P04.04

Arrayoperationen

2: Skalarmultiplikation

```
public static int[][]
    scalarMultiplication(int[][] m, int s)
```

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \cdot s = \begin{pmatrix} s \cdot a & s \cdot b & s \cdot c \\ s \cdot d & s \cdot e & s \cdot f \\ s \cdot g & s \cdot h & s \cdot i \end{pmatrix}$$

P04.04

Arrayoperationen

2: Skalarmultiplikation

```
public static int[][]
  scalarMultiplication(int[][] m, int s) {
  int[][] newM = new int[m.length][];
  for(int i = 0; i < m.length; i++) {
    newM[i] = new int[m[i].length];
    for(int j = 0; j < m[i].length; j++)
      newM[i][j] = m[i][j]*s;
  }
  return newM;
}
```

P04.04

Arrayoperationen

3: Transposition

```
public static int[][] transpose(int[][] m)
```

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}^T = \begin{pmatrix} a & d & g \\ b & e & h \\ c & f & i \end{pmatrix}$$

$$a^T_{i,j} = a_{j,i}$$

P04.04

Arrayoperationen

3: Transposition

```
public static int[][] transpose(int[][] m) {
    int[][] toReturn = new int[m[0].length][];
        for(int i = 0; i < m[0].length; i++)
            toReturn[i] = new int[m.length];
        for(int i = 0; i < m.length; i++)
            for(int j = 0; j < m[i].length; j++)
                toReturn[j][i] = m[i][j];
        return toReturn;
}
```