- Ende der VL: Zuteilung von 1er zu 2er Gruppen.
- Zwischentest: voraussichtlich am Mi, 12. Dez. zur Übungszeit im Audimax
- Abgabe der theor. HAs Di vor der VL

Erinnerung: Prime.java liest $n \in \mathbb{N}$ ein, testet $z = n+1, n+2, \ldots$ ob Primzahl, gibt z aus.

```
n = scan.nextInt();
z = n
do {
    z++;
    k = 2;
    divisorFound = false;  // so far no divisor of z has been found

    while ((!divisorFound) && (k*k <= z)) {
        // check if k is a divisor of z
        if (z % k == 0) {
            divisorFound = true;
        }
        k++;
    } //end while

} while (divisorFound);
```

äußere Schleife ist do-while Schleife („akzeptierende" Schleife)

innere while-Schleife („abweisende" Schleife)

„Neue" Elemente:

!	$\hat{=}$ logische Negation	z.B. $!(a <= 10) \Leftrightarrow a > 10$
&&	$\hat{=}$ logisches UND	
||	$\hat{=}$ logisches ODER	
%	$\hat{=}$ Rest bei ganzzahliger Division	z.B. $7 \bmod 3 = 1$
	$\hat{=}$ „mod"	$11 \bmod 13 = 11$
/	$\hat{=}$ ganzz. Division	z.B. $7 \text{ div } 3 = 2$
	$\hat{=}$ „div"	$11 \text{ div } 13 = 0$
==	$\hat{=}$ Test auf Gleichheit	

Standardfehler: `if (a=b) {...}`

↑ ist Zuweisung. Muß `a==b` heißen!

++

↑

arithmetischer Operator für ganze Zahlen

a++; ≙ Erhöhung von a um eins

≙ a = a+1;

entsprechend a--;

verwandt: a+=2; ≙ a = a+2;

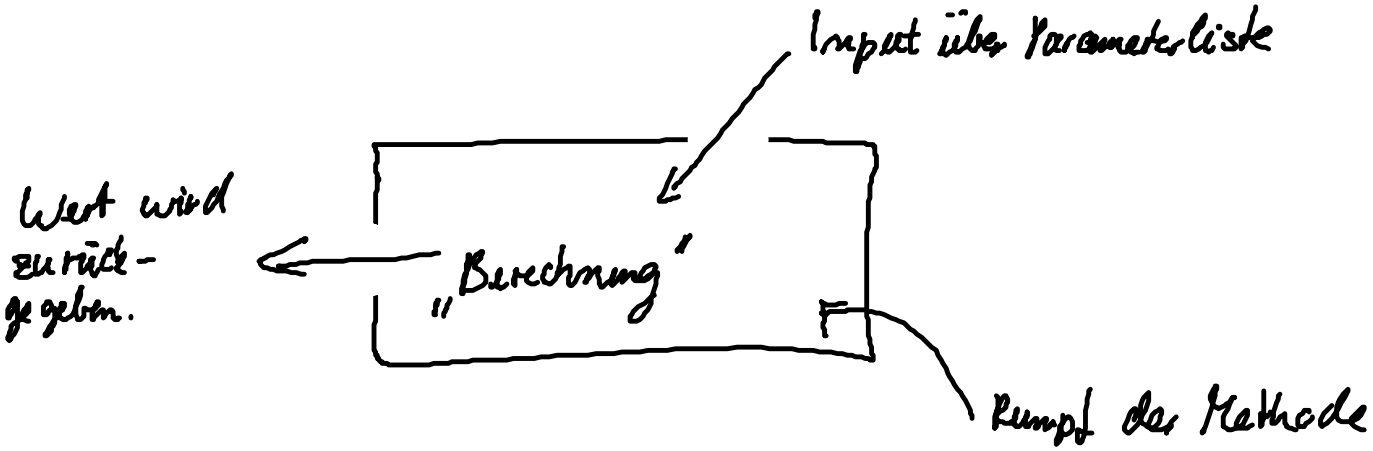
a-=2;

a*=3; ≙ a = a*3;

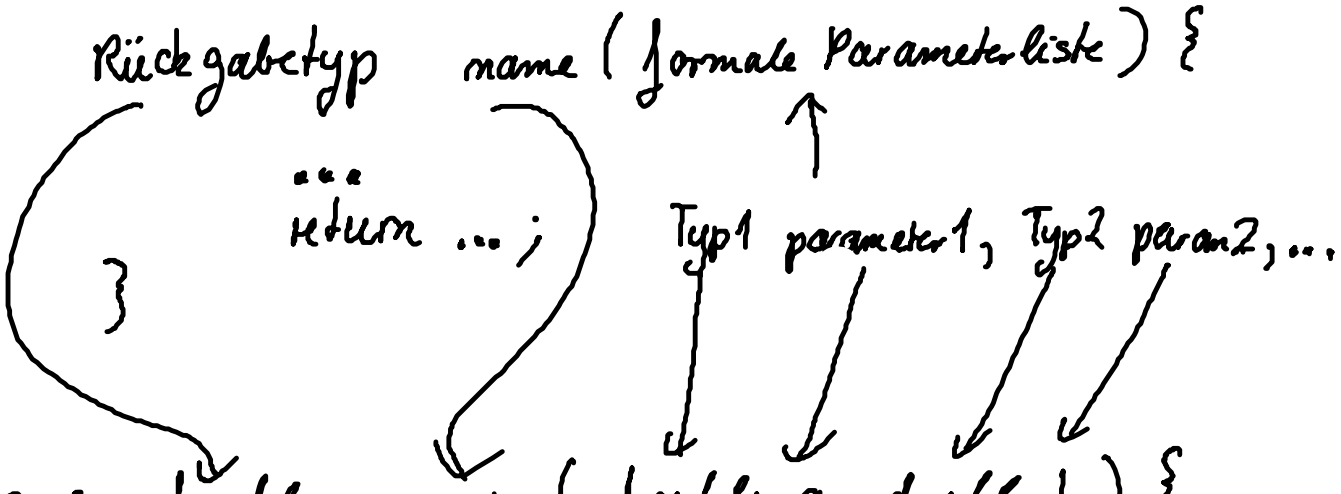
Zur besseren Strukturierung des Programms bietet es sich oft an Methoden zu verwenden.

Methoden: „ähneln" Funktionen in der Mathematik

- strukturieren Programmcode
- machen Code wiederverwendbar



Methode wird deklariert über



Bsp: double max (double a, double b) {
    if (a > b)
        return a;
    else
        return b;
}

- Methode gibt genau einen Wert vom Rückgabetyp zurück (mit return-Anweisung)
- Parameterliste (...) kann leer sein
- Rückgabetyp kann „void" sein (≙ keine Rückgabe)
  ↖ eigentlich kein Typ
  Dann wird die Abarbeitung mit return; abgebrochen.
- Methoden können in beliebigen Ausdrücken verwendet werden.

Bsp: double z = max (10, 20);
z = z * max (10, 20) + 3;

- Aufruf einer Methode erfolgt mit konkreten Werten für die formalen Parameter.
  - formalen Parameter müssen Typ-verträglich sein!
  - konkreten Werten: „aktuelle" Parameter

Bsp: Berechnung für $n!$ für $n \in \mathbb{N} \setminus \{0\}$

```
long factorial (long n) {
    long f = 1;
    long k = 2;
    while (k <= n) {
        f *= k;
        k++;
    }
    return f;
}
```

Verwendet:

```
long z = factorial (15);
long a = 3 * factorial (2*z + 3);
```

- Jeder Aufruf einer Methode wird mit genau einer return-Anweisung beendet, gibt Wert vom Rückgabetyp zurück (außer bei „void"):

```
double max (double a, double b, double c) {
    if ((a >= b) && (a >= c)) return a;
    if (b >= c) return b;
    return c;
}
```

„else" kann hier fehlen

Methoden mit verschiedenen Parameterlisten können denselben Namen haben, der Compiler kann sie unterscheiden → Signatur ≙ name + Parameterliste

„Modifier"

```
// check if z >= 3 is prime
private boolean isPrime(long z) {
    if (z % 2 == 0) return false;
    if (z == 3 || z == 5) return true;
    long k = 3;
    while (k*k <= z) {
        if (z % k == 0) return false;
        k +=2;
    }
    return true;
}
```

Rückgabetyp: boolean

Parametertyp: long

ab hier ist $z \geq 7$ ungerade

es werden nur die Zahlen $k = 3, 5, 7, 9, 11, \ldots$ auf Teiler überprüft.

Bsp für Methode ohne Parameter:

```
double pi() {
    return 3.14159...;
}
```

Verwendung: `double a = 2 * pi();`

Kapitel 3: Ausdrücke, Anweisungen und Kontrollstrukturen

Variablen und Ausdrücke haben immer einen Typ.

```
double a;
```

`2 * a * a`

in Java: 1. Standardtypen (primitive Typen; eingebaute Typen)

2. Referenztypen

Standardtypen	Wertebereich		
boolean	true, false		
char	alle Unicode-Zeichen (der zugehörige Zahlenwert $0, \dots, 2^{16}-1$)	16 bits	
byte	$-128, \dots, 127$	8 bits	ganze Zahlen
short	$-2^{15}, \dots, 2^{15}-1$	16 bits	
int	$-2^{31}, \dots, 2^{31}-1$	32 bits	
long	$-2^{63}, \dots, 2^{63}-1$	64 bits	
float		32 bits	Fließkommazahlen
double		64 bits	

void ist jedoch kein Typ.

```
double a;
int b;
```

`3 * a * b`

Bei arithmetischen Ausdrücken wird der Typ des Ausdrucks immer durch den „breitesten" vorkommenden Typ bestimmt.

Organisatorisches:

- alle 1er- und 2er Gruppen nach vorne
  - ↙ links
  - ↘ rechts (von mir)
- Eintragungen in Datenbank: bei Madeleine Theile
- Selbstorganisieren + Forum benutzen