Informatik II
Prüfungsvorbereitungskurs

Tag 1, 20.6.2016

Giuseppe Accaputo

g@accaputo.ch

Über mich

- Giuseppe Accaputo
- 3. Semester RW/CSE Master
- E-Mail: g@accaputo.ch
- Homepage: http://accaputo.ch
- Informatik II Assistenz: http://accaputo.ch/hilfsassistenz/informatik-2-d-baug-2016/

Administratives

- Ort: HIT H51
- Datum: 20.3 – 23.6
- Zeit: 08:10 Uhr – 12:00 Uhr
- 50 Minuten Kurs, 10 Minuten Pause
- Kontakt: g@accaputo.ch

Fragen

- **Fragen stellen ist erwünscht**
- Fragen stellen ist zu jedem Zeitpunkt erlaubt

Themenübersicht

- 20.3: **Java**
- 21.3: **Objektorientierte Programmierung**
- 22.3: **Dynamische Datenstrukturen**
- 23.3: **Datenbanksysteme**

Part 1: **Java**

Primitive Datentypen

Typ	Grösse	Min. Wert	Max. Wert
`byte`	8	-128	127
`short`	16	-32768	32767
`int`	32	-2^{31}	$2^{31}-1$
`long`	64	-2^{63}	$2^{63}-1$
`float`	32	$-3.4 \cdot 10^{38}$	$3.4 \cdot 10^{38}$
`double`	64	$-1.8 \cdot 10^{308}$	$1.8 \cdot 10^{308}$
`boolean`	n.d.	Entweder `true` oder `false`	
`char`	16	`0x0000`	`0xffff`

Arithmetische Zuweisungen

```
a += b   ⟺   a = a + b
```

- Analog für `- * /`

Konstanten

- Konstanten können mit dem Schlüsselwort `final` definiert werden

```
final double pi = 3.14;
final double e = 2.718;

zwei_pi = 2*pi; // Korrekt
pi = 3.1; // Fehler!
```

Ganzzahldivision

Regeln:

- `int / int = int`
- `int / double = double`
- `double / int = double`

```
int i = 5;
int j = 3;
double k = 3.0;
double res1 = i/j; // res1 = 1.0
double res2 = i/k; // res2 = 1.666
```

Präzedenz und Assoziativität

Präzedenz	Operatoren	Assoziativität
Unäre Operatoren	`! + - ++ --`	Links
Arithmetische Operatoren	`* / % + -`	Links
Schiebe-Operatoren	`<< >> >>>`	Links
Vergleichs-Operatoren	`< <= > >= == !=`	Links
Logische Operatoren	`& ^ | && || ?:`	Links
Zuweisungsoperatoren	`=, op=, wobei op: + - / % & ! ^ << >>`	Rechts

Assoziativität

- Linksassoziativ:

```
A op B op C ⇔   (A op B) op C
```

- Rechtsassoziativ:

```
A op B op C ⇔   A op (B op C)
```

Aufgabe: Assoziativität

- Wie lautet das Ergebnis des folgenden Ausdrucks?

```
3 * 10 + 2 * 10 * (3 - 1) / 20 >= 5
```

Lösung Assoziativität

1. `3 * 10 + 2 * 10 * (3 - 1) / 20 >= 5`
2. `3 * 10 + 2 * 10 * 2 / 20 >= 5`
3. `30 + 2 * 10 * 2 / 20 >= 5`
4. `30 + 20 * 2 / 20 >= 5`
5. `30 + 40 / 20 >= 5`
6. `30 + 2 >= 5`
7. `32>= 5`
8. `true`

Aufgabe Assoziativität

- Setze Klammern um Präzedenz anzuzeigen

```
year % 4 == 0 && year % 100 != 0 || year % 400 == 0
```

Lösung Assoziativität

```
(((year % 4) == 0) && ((year % 100) != 0))
     || ((year % 400) == 0)
```

Faustregel:

1. Punkt vor Strich
2. Arithmetisch vor Vergleich
3. Vergleich vor Logisch
4. ! vor && vor ||

Inkrement und Dekrement Operatoren

- **Prä**-Inkrement:

```
y = ++x;      ⇔   x = x + 1; y = x;
```

- **Post**-Inkrement:

```
y = x++;      ⇔   y = x; x = x + 1;
```

- Analog für Dekrement `--x`
- **Wichtig**: Inkrement und Dekrement Operatoren nur auf Variablen verwenden! Z.B. funktioniert `--4` nicht!

Aufgabe Inkrement Operator

- Was hat `z` für einen Wert nach der Ausführung?

```
int x = 20;
int y = 30;
int z = x++ * ++y;
```

Lösung Inkrement Operator

```
int z = x++ * ++y;
```

Präzedenz der
Inkrement Operatoren

```
y = y + 1;
int z = x * y;
x = x + 1;
```

```
z == 620
```

Implizite Typkonversion

- Eine *implizite Typkonversion* findet dann statt, wenn der Zieltyp *grösser* ist als der Ursprungstyp:

```
urpsrungstyp a = ...;
zieltyp b = a;
```

- Reihenfolge (aufsteigend in Grösse):
  `byte` < `short` < `int` < `long` < `float` < `double`

Aufgabe Implizite Typkonversion

- Ist dieser Code korrekt?

```
int i = 10213123;
long l = i ;
byte b = i ;
```

Lösung Implizite Typkonversion

```
int i = 10213123;
long l = i ; // OK:       int < long
byte b = i ; // Fehler:  byte < int
```

Explizite Typkonversion

- Eine *explizite Typkonversion* wird dann benötigt, wenn der Zieltyp *kleiner* ist als der Ursprungstyp:

```
urpsrungstyp a = ...;
zieltyp b = (zieltyp)a;
```

- Reihenfolge (absteigend in Grösse):
  `double` < `float` < `long` < `int` < `short` < `byte`
- Bei expliziter Konversion von **`int`** nach **`float`** werden die Nachkommastellen abgeschnitten

Aufgabe Explizite Typkonversion

- Was wird auf der Konsole ausgegeben?

```
System.out.println((double)1/2);
```

Lösung Explizite Typkonversion

```
System.out.println((double)1/2);
```

Konsole:

```
0.5
```

If / Else Anweisungen

```
if(Bedingung)
    Anweisung;
else if (Bedingung)
    Anweisung;
else
    Anweisung;
```

- Bedingung ist ein Ausdruck vom Typ **boolean**
- Beispiel:

```
if(age > 18)
    sellAlcohol();
else
    sendHome();
```

Anweisungsblöcke

```
if(Bedingung){
    // if Zweig
}
else if (Bedingung){
    // else if Zweig
}
else{
    // else Zweig
}
```

- Wo Anweisungen gefordert sind, können Anweisungsblöcke stehen

- Beispiel:

```
if(age > 18){
    sellAlcohol();
    returnChange();
}
```

Aufgabe Anweisungsblöcke

- Kompiliert dieser Code?

```
{
    int x = 0;
}
{
    int x = 10;
}
    int x = 12;
```

Lösung Anweisungsblöcke

Kompiliert!

```
{
    int x = 0;
}
{
    int x = 10;
}
    int x = 12;
```

Aufgabe Anweisungsblöcke

- Kompiliert dieser Code?

```
    int x = 12;
{
    int x = 0;
}
{
    int x = 10;
}
```

Lösung Anweisungsblöcke

Kompiliert nicht! Variablenkonflikt, da x zuoberst definiert wurde!

```
int x = 12;
{
    int x = 0;
}
{
    int x = 10;
}
```

Vorsicht bei Anweisungsblöcken

Wichtig zu beachten bei Anweisungsblöcken:

```
if(x > 10)
    doA();
    doB();
```

=

```
if(x > 10){
    doA();
}
doB();
```

- Gilt `x > 10`, so wird nur `doA()` ausgeführt!

Aufgabe Anweisungsblöcke

- Was wird auf der Konsole ausgegeben?

```
int a = 5;
int b = 10;

if(b > 20)
    a = 30;
    System.out.println(a);
```

Lösung Anweisungsblöcke

```
int a = 5;
int b = 10;

if(b > 20){
    a = 30;
}
    System.out.println(a);
```

Konsole:
5

If / Else Anweisungen: Rückgabe (return)

- Für Funktionen mit einem Rückgabewert gilt:
  Alle Zweige bei einer `if / else if / else` Anweisung müssen zurückkehren

Kompilierfehler

```
if (a > 10)
    return a; // Zweig 1
else if (b > 20)
    return b+a; // Zweig 2
```

Korrekt

```
if (a > 10)
    return a; // Zweig 1
else if (b > 20)
    return b+a; // Zweig 2
else
    return 2*b; // Zweig 3
```

Schleifen: `while`

```
while(Bedingung)
     Anweisung;

while(Bedingung){
     Anweisung1;
     Anweisung2;
     Anweisung3;
}
```

- Bedingung ist ein Ausdruck vom Typ **`boolean`**
- Anweisungsblöcke können auch verwendet werden

Schleifen: do…while

```
do
    Anweisung;
while(Bedingung)

do{
    Anweisung1;
    Anweisung2;
}
while(Bedingung)
```

- Bedingung ist ein Ausdruck vom Typ **boolean**
- Anweisungsblöcke können auch verwendet werden
- Beispiel: Passwortabfrage (Eclipse)

Schleifen: for

```
          1                2                4
for (Initialisierung; Bedingung; Fortschritt){
    Anweisung;
}       3
```

Ausführreihenfolge:

1. Initialisierung (wird nur beim ersten Mal ausgeführt!)
2. Bedingung
3. Anweisung
4. Fortschritt

Beispiel for-Schleife

```
for (int i = 0; i<10; ++i)
{
    System.out.println(i + "*" + i + "=" + i*i);
}
```

```
for (int x = 1; x<=128; x*=2)
{
    System.out.println(x + "=" + x);
}
```

Schleifen: for ⇔ while

```
for (Initialisierung; Bedingung; Fortschritt){
    Anweisung;
}
```

=

```
Initialisierung;
while (Bedingung){
    Anweisung;
    Fortschritt;
}
```

Beispiel for ⇔ while Schleifen

```
for(int i = 0; i < 10; i++)
    System.out.println(i);
```

=

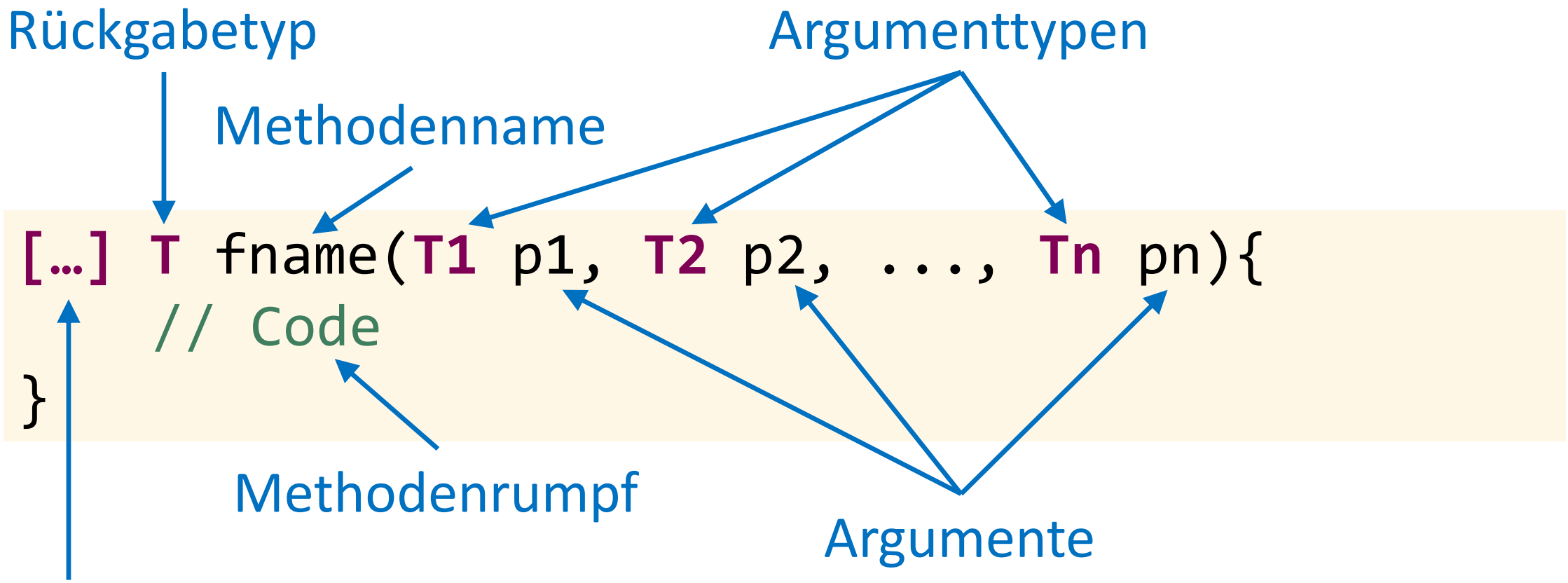
```
int i = 0;
while(i < 10){
    System.out.println(i);
    i ++;
}
```

Prüfung 08.2014 Aufgabe 7b

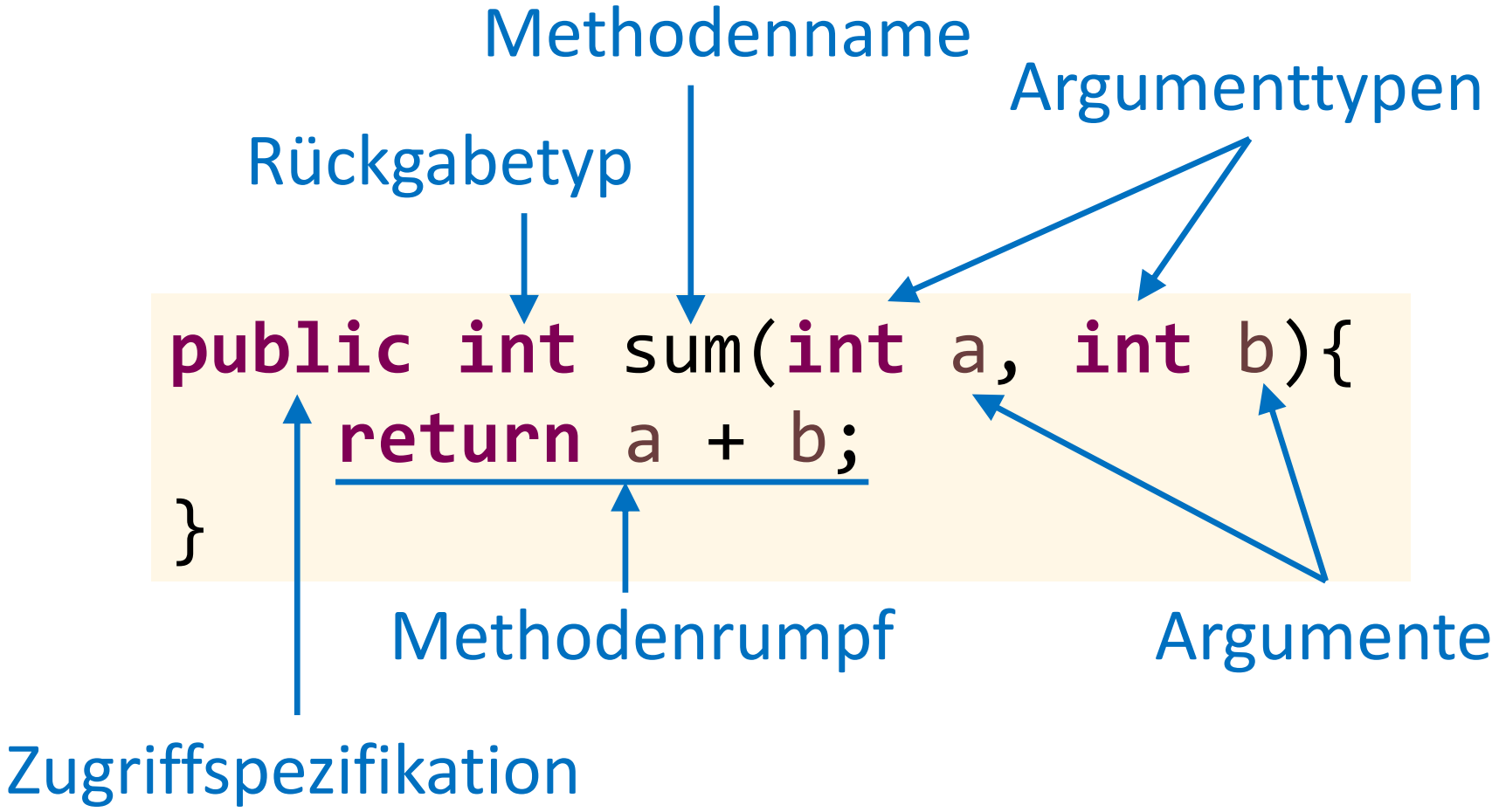
- Wo befindet sich der Fehler? Auswirkung?

```
public int sqr(int n) {
    int r = 1;
    int a = 10000;
    while (a > 0) {
        while (r*r <= n)
            r += a;
        r -= a;
    }
    return r;
}
public static void main(String[] args) {
    int x = sqr (100);
```

Methoden


Rückgabetyp
Argumenttypen
Methodenname
[…] T fname(T1 p1, T2 p2, ..., Tn pn){
// Code
}
Methodenrumpf
Argumente
Zugriffspezifikation, z.B. public, private,
weitere Modifizierer, z.B. static

Beispiel Summen-Methode



Aufgabe Methoden

- Definiere eine Funktion, welche ein `String s` auf die Konsole ausgibt
- Definiere eine Funktion, welche überprüft, ob 3 Integer gleich sind und bei Erfolg `true` zurückgibt
- Definiere eine Funktion, welche alle geraden Zahlen im Intervall [0,20] ausgibt (0 ist gerade)

Lösung Methoden

- Definiere eine Funktion, welche ein `String s` auf die Konsole ausgibt

```
void prtStr(String s){
    System.out.println(s);
}
```

Lösung Methoden

- Definiere eine Funktion, welche überprüft, ob 3 Integer gleich sind und bei Erfolg `true` zurückgibt

```
boolean eq3(int a, int b, int c){
    return (a == b) && (a == b);
}
```

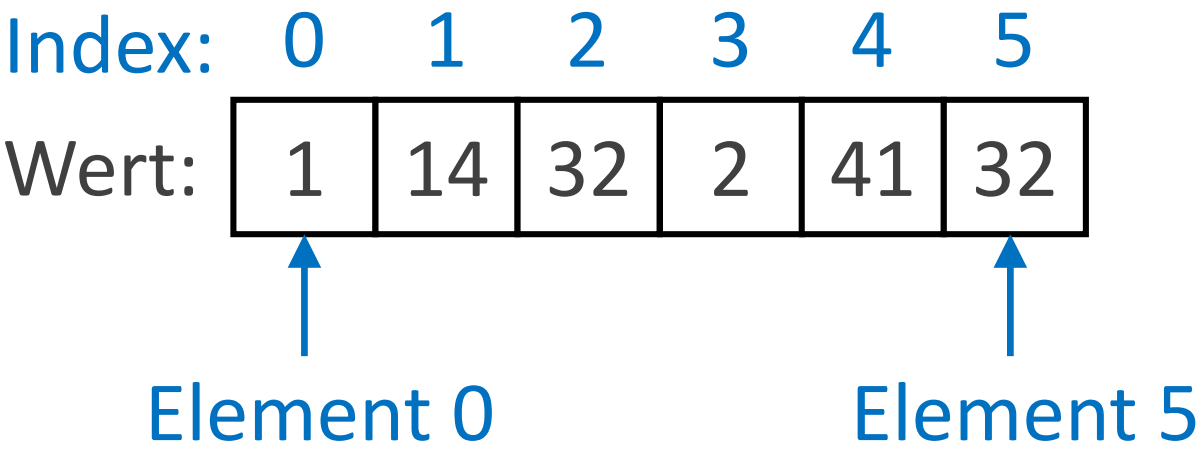
Lösung Methoden

- Definiere eine Funktion, welche alle geraden Zahlen im Intervall [0,20] ausgibt (0 ist gerade)

```
void even(){
    for(int i = 0; i<=20; i+=2){
        System.out.println(i);
    }
}
```

Arrays

- **Array**: Objekt, welches mehrere Werte desselben Typs speichert
- **Element**: Ein Wert an einem Index des Arrays
- **Index**: Position eines Elementes im Array. Startet bei 0

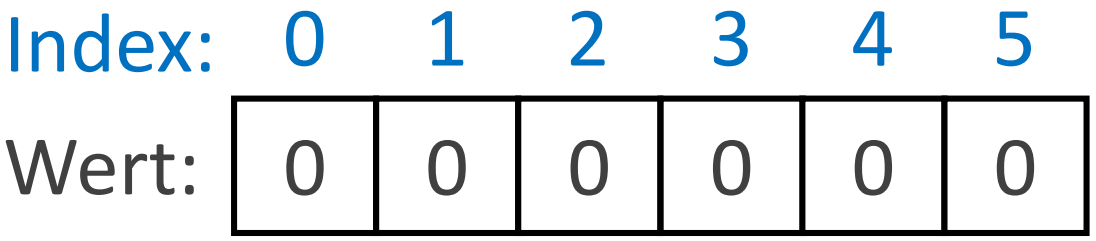


Arrays: Deklaration

```
Typ[] name = new Typ[Länge];
```

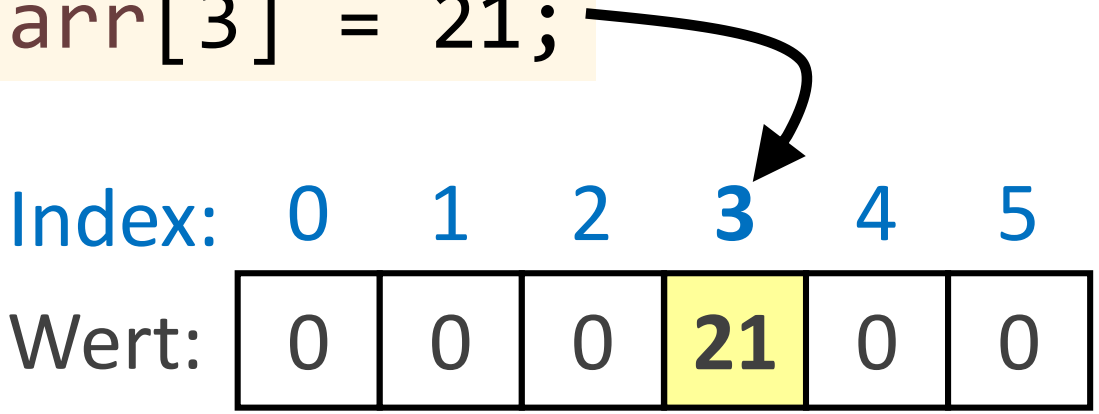
- Beispiel:

```
int[] arr = new int[6];
```



Arrays: Elementzugriff

- Element lesen: `int a = arr[index];`
- Element speichern: `arr[3] = 21;`



- Erste Position auf Index `0`, letzte auf `arr.length - 1`
- **Wichtig**: Exception wird geworfen bei Fehlzugriff!

Arrays: Durchiterieren

- Arrays besitzen die Instanzvariable `length`, welche die Länge des Arrays abspeichert
- Um durch ein Array zu iterieren, kann man eine `for`-Schleife und das `length` Attribut verwenden:

```
int[] arr = new int[3];
arr[0] = 1; arr[1] = 2; arr[2] = 3;

for(int i = 0; i < arr.length; i++)
    System.out.println(arr[i] + "");
```

Konsole:

```
1 2 3
```

Prüfung 08.2014 Aufgabe 7a

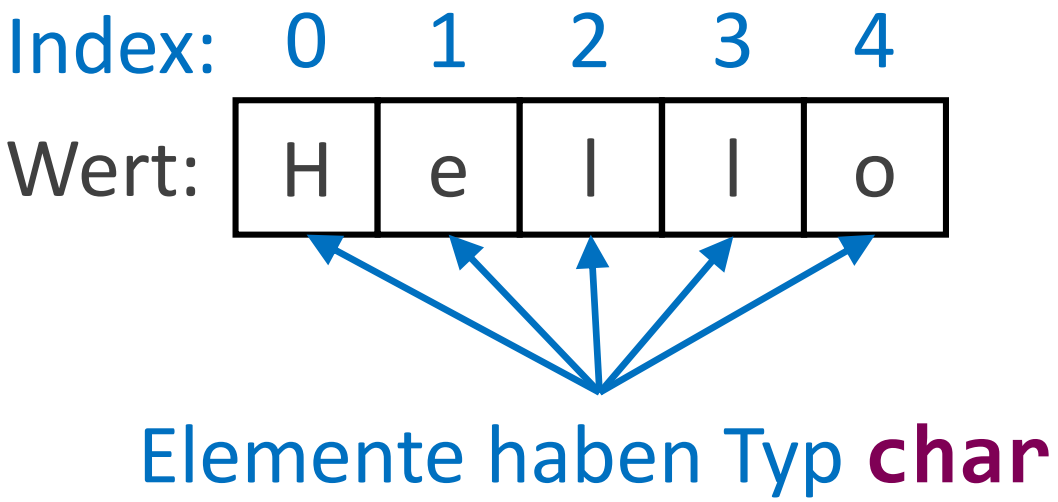
- Fehler? Auswirkung?

```
public int sum(int[] numbers) {
    int s = 0;
    for (int i=0; i<=numbers.length; ++i)
        s += numbers[i];
    return s;
}
public static void main(String[] args) {
    int a[] = {1,2,3};
    int s = sum(a);
    // ...
}
```

Strings

- String: Objekt, dass eine Zeichenkette speichert

```
String name = "Hello";
```



Strings

- String Objekte sind *immutable*, d.h. sie können nicht verändert werden:

```
String s = "Hello, World!";
s[1] = 'H'; // Kompilierfehler
```

String Operationen

- Konkatenation zweier Strings mittels +-Operator:

```
String s = "Hello";
String t = ", World!";
String u = t + s;
// u = "Hello, World!"
```

- `i`-ter Buchstabe eines Strings auslesen:

```
String s = "Hello";
char c = s.charAt(1); // c == 'e'
```

String Operationen

- Bestimmte Buchstaben ersetzen:

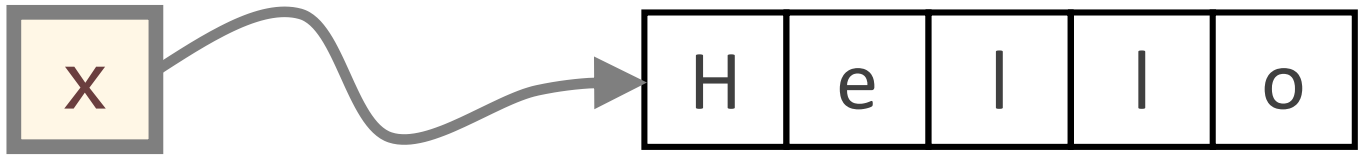
```
String s = "Hello";
String t = s.replace('e', 'a');
// t = "Hallo"
// s = "Hello" (nicht verändert)
```

  - **Wichtig**: Da Strings *immutable* sind, arbeitet `replace` auf einer Kopie des Strings `s`, welche wir in `t` dann zwischenspeichern. `replace` führt also kein in-place Replacement durch!

Strings: Zeichenketten vergleichen

- Variablen sind Referenzen auf Werte im Speicher

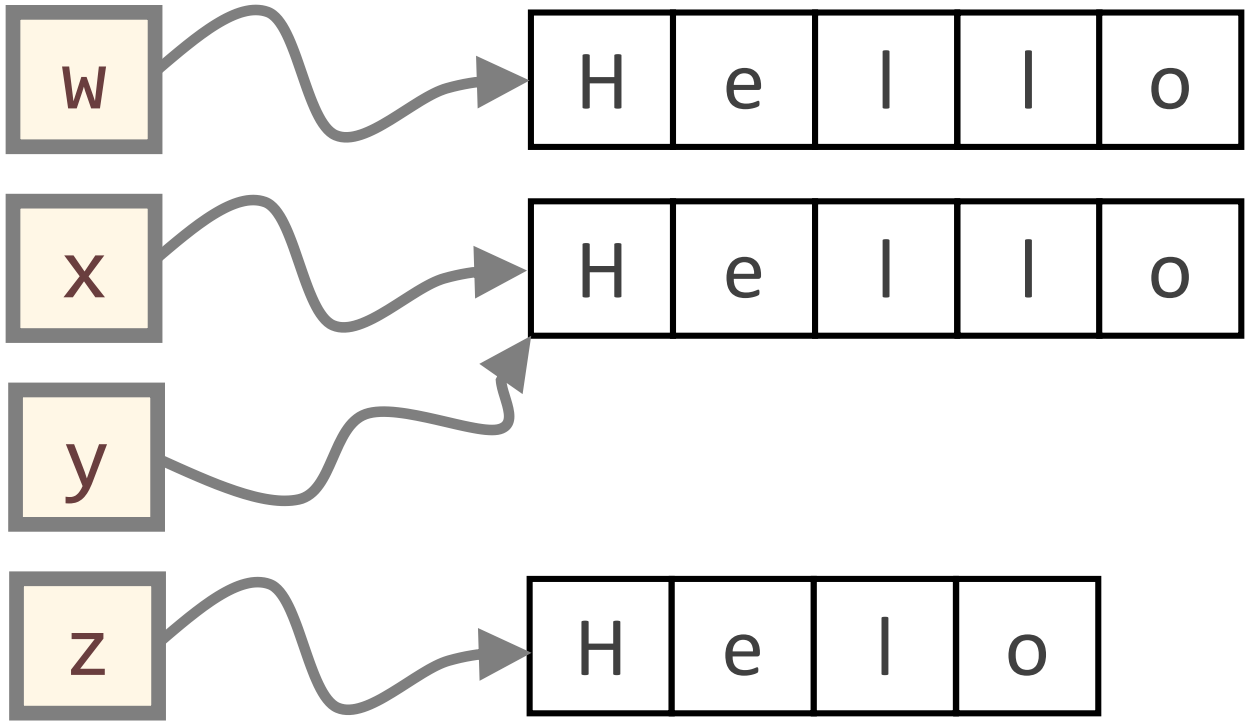
```
String x = "Hello";
```



- == und != Operatoren auf Strings machen einen *Referenzvergleich* und keinen Zeichenkettenvergleich
- `equals`-Methode verwenden um Zeichenketten zu vergleichen

Strings: Zeichenkettenvergleiche

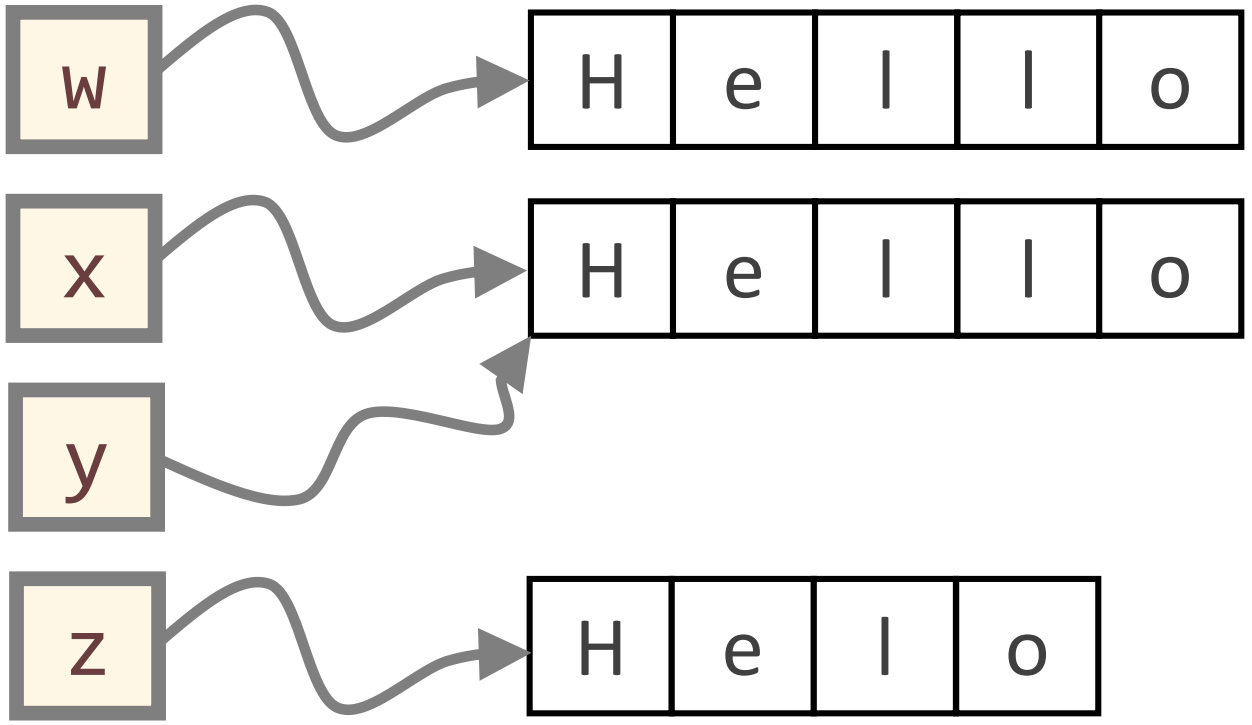
```
String w = "Hello";
String x = "Hello";
String y = x;
String z = "Helo";
```




```
w == w → true
w == x → false
x == y → true
```

Strings: Zeichenkettenvergleiche

```
String w = "Hello";
String x = "Hello";
String y = x;
String z = "Helo";
```




```
w.equals(x) → true
x.equals(y) → true
y.Equals(z) → false
```

Allgemein: Vergleiche

- Bei primitiven Datentypen (`boolean`, `byte`, `char`, `short`, `int`, `long`, `float`, `double`):
  == vergleicht Werte der beiden Variablen miteinander

```
char c1 = 'h';
char c2 = 'h';
boolean b = (c1 == c2); // true
```

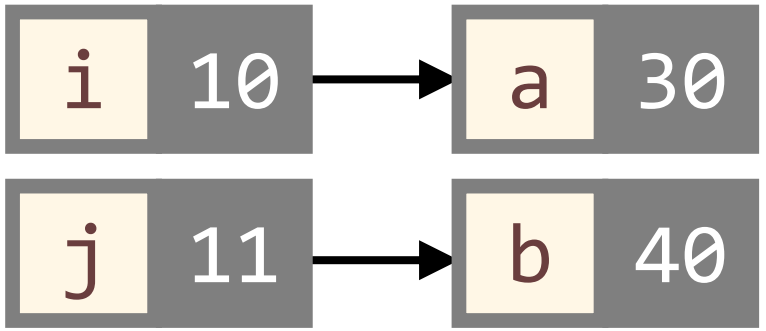
- Bei den restlichen Datentypen (`String`, `MyClass`, etc.):
  == vergleicht die Referenzen der Variablen miteinander

Java ist Pass-By-Value

- Parameter: Primitive Datentypen werden kopiert

```
void do(int a, int b){
    a = 30;
    b = 40;
}

int i = 10, j = 11;
do(i,j);
```



Nach Aufruf von do(i,j):

i == 10

j == 11

Java ist Pass-By-Value

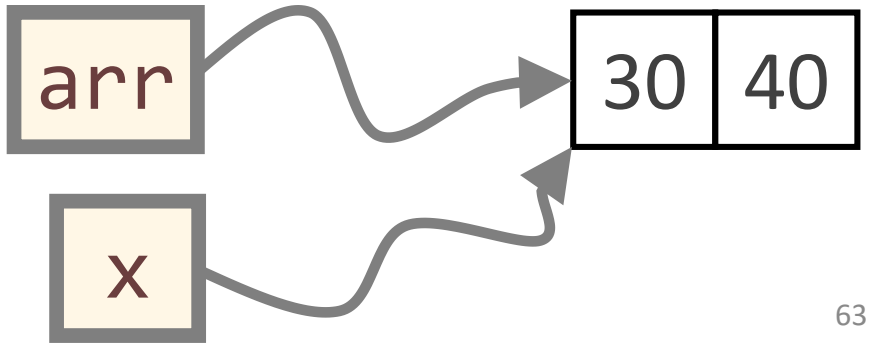
- Parameter: Referenzen auf Objekte werden kopiert

```
void do(int x[]){
    x[0] = 30;
    x[1] = 40;
}

int[] arr = new int[2];
arr[0] = 10; arr[1] = 11;
do(arr);
```

Zu Beginn:

Nach Aufruf von do(arr):



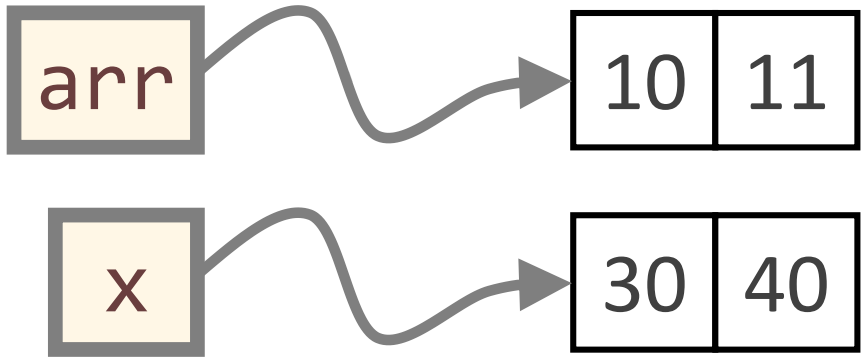
Java ist Pass-By-Value

```
void do(int x[]){
    x = new int[2];
    x[0] = 30;
    x[1] = 40;
}

int[] arr = new int[2];
arr[0] = 10; arr[1] = 11;
do(arr);
```

Zu Beginn:

Nach Aufruf von do(arr):



Aufgabe Pass-By-Value

- Welche Elemente befinden sich im Array x?

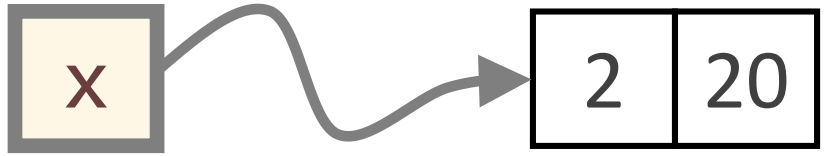
```
public static void d(int[] y){
   y[0] = 2;
   y = new int[2];
   y[1] = 3;
}
…

int[] x = new int[2];
x[0] = 100; x[1] = 200;
d(x);
```

Lösung Pass-By-Value

```
public static void d(int[] y){
   y[0] = 2;
   y = new int[2];
   y[1] = 3;
}
…

int[] x = new int[2];
x[0] = 10; x[1] = 20;
d(x);
```



Prüfung 08.2014 Aufgabe 6b

- Was wird auf der Konsole ausgegeben?

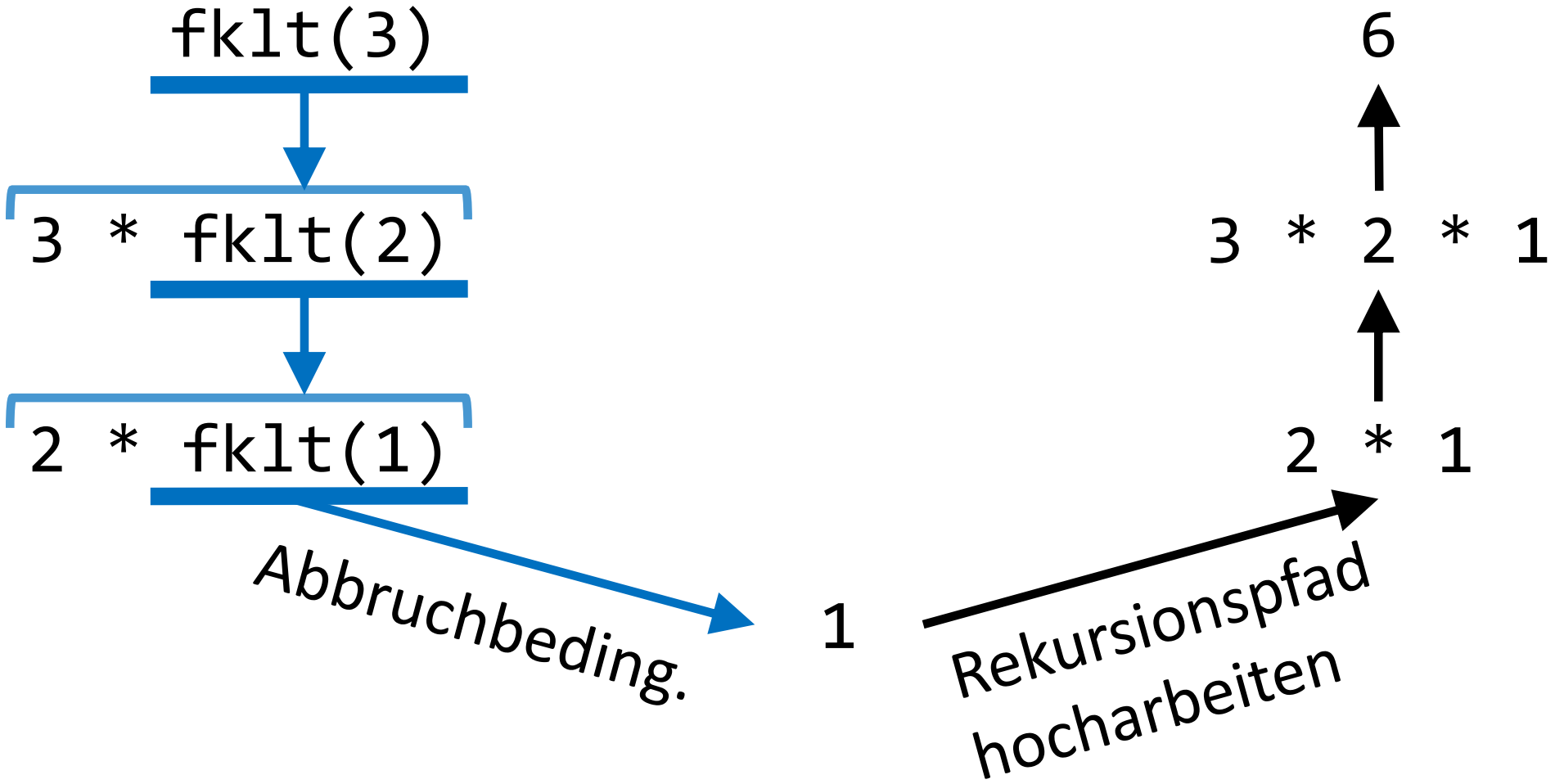
```
public static void main(String[] args) {
    String a[] = new String[2];
    String b[] = a;
    String c[] = a;
    a[0] = "Hund";
    a[1] = "Katze";
    b[0] = "Maus";
    c = new String[2];
    c[1] = "Elefant";
    for (int i = 0; i<a.length; ++i)
        System.out.println(a[i]);
}
```

Rekursion

- Rekursive Funktion um Fakultät einer Zahl n zu berechnen:

```
public static double fklt(double n){
    if(n <= 1)
        return 1;
    else
        return n * fklt(n-1);
}
```

Rekursion: Beispiel Aufruf



Prüfung 08.2015 Aufgabe 1b

- Was wird auf der Konsole ausgegeben?

```
static double recursive(double x){
    if (x > 1)
        return ((int)x) * recursive(x-1);
    return x;
}

System.out.println(recursive(2.5));

System.out.println(recursive(5.2));
```