Informatik II
Prüfungsvorbereitungskurs

Tag 1, 6.6.2017

Giuseppe Accaputo

g@accaputo.ch

Über mich

- **Name**: Giuseppe Accaputo
- **Studiengang**: RW/CSE Master
- **E-Mail**: g@accaputo.ch
- **Homepage**: http://accaputo.ch
- Informatik II Assistenz (FS 2017): http://accaputo.ch/hilfsassistenz/informatik-2-d-baug-2017
- Informatik II Assistenz (FS 2016): http://accaputo.ch/hilfsassistenz/informatik-2-d-baug-2016

Administratives

- **Ort**: HIT H 42 (ETH Hönggerberg)
- **Datum**: 6.6.2017 – 9.6.2017
- **Zeit**: 13:00 Uhr – 17:00 Uhr
- **Kursstunde**: 50 Minuten Kurs, 10 Minuten Pause

Fragen stellen

- **Fragen stellen ist überaus erwünscht!**
- Fragen stellen ist zu jedem Zeitpunkt erlaubt
- Bitte schämt euch nicht:
  - Fragen zu angeblich «einfachen» Themen zu stellen
  - bereits beantwortete Fragen nochmals zu stellen weil ihr die Antwort beim ersten Mal nicht verstanden habt

Aufbau des PVK

- **Tag 1**: Java Teil 1
- **Tag 2**: Java Teil 2, Algorithmen & Komplexität
- **Tag 3**: Dynamische Datenstrukturen
- **Tag 4**: Datenbanksysteme & Repetition

Java Teil 1

Variablen

- *Behälter* für einen Wert
- Haben *Datentyp* und eine *Namen*
- Datentyp bestimmt, welche Art von Werten in der Variable erlaubt sind

```
int a = 10, b = 20;
char c = 'd';
float e = 1.2f;
```

- Datentypen:
  `int`, `char`, `float`
- Werte:
  `10`, `20`, `1.2f`, `'d'`
- Namen:
  `a`, `b`, `c`, `e`

Variablen

- Deklaration:

```
int a;
char b;
float c;
```

- Initialisierung:

```
int d = 10
int e = 0;
float f = 3.14f;
```

- Variablen werden mit einem *Defaultwert* initialisiert
  - Defaultwert wird von Java bestimmt, meistens `0` oder `null`
  - In C: «zufälliger» Wert als Initialiwert
- Variablen werden mit einem *bestimmten* Wert initialisiert
  - Variablen lieber selbst initialisieren, da wir sehen, mit welchem Wert initialisiert wird

Repetition Variablen

- Für jede Variable:
  1. Typ
  2. Name
  3. Wert
  4. Nur deklariert?
  5. Initialisiert?

```
int a1;
float a2;
char a3 = '@';
boolean a4 = false;
```

Konstanten

- Schlüsselwort **`final`**
- Der Wert der Variable kann genau einmal gesetzt werden

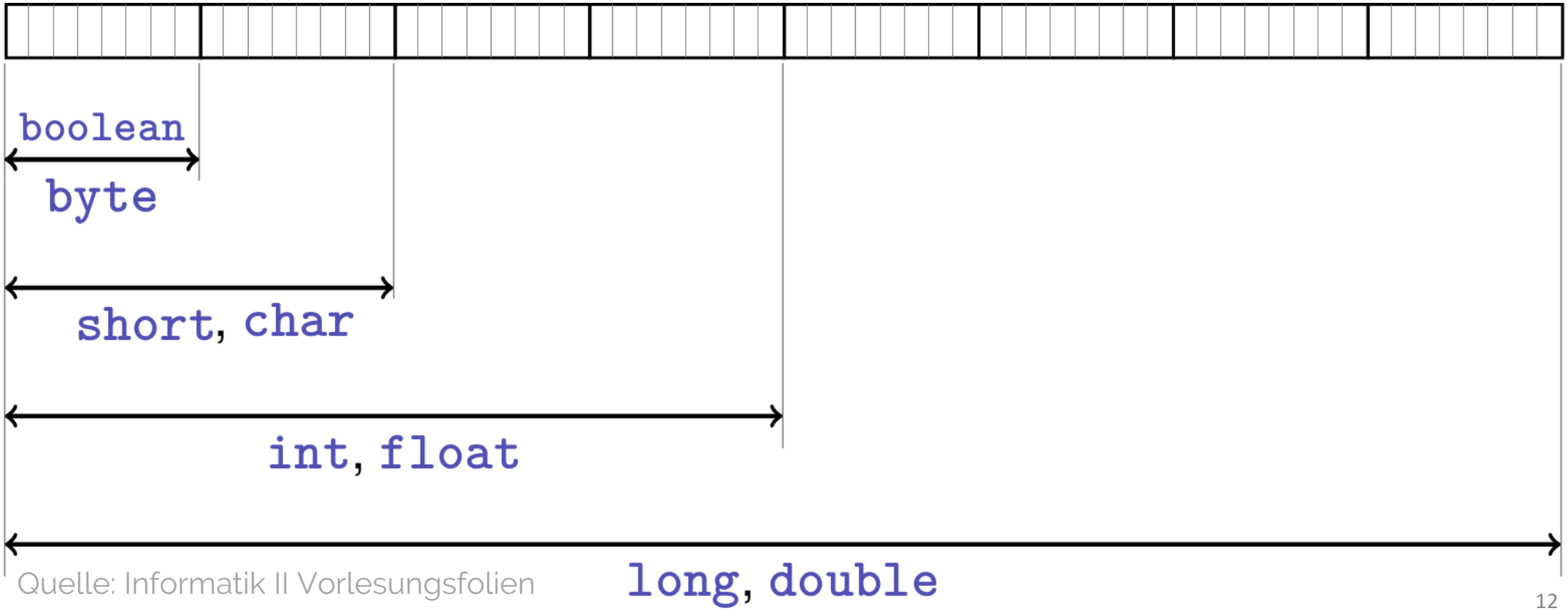
```
final double pi = 3.14;
final double e = 2.718;

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

Standardtypen

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`

Standardtypen und Speicherbelegung

- 1 Byte = 8 Bit



Quelle: Informatik II Vorlesungsfolien

Arithmetische binäre Operatoren

- Binär: `a op b`
- Addition: `a + b`
- Subtraktion: `a - b`
- Multiplikation: `a * b`
- Division: `a / b`
- Modulo: `a % b`

Modulo

```
n % m = b
```

- Modulo berechnet den Rest **b** der Division **n** geteilt durch **m**
- **b** kann nur die Werte 0, 1, ..., m-1 annehmen

`5 % 3 = 2`: «3 passt einmal in 5 und es bleiben 2 übrig»

`-5 % 3 = 1`: $-5 = -2 * 3 + 1$

Präzedenz und Assoziativität

- Absteigend sortiert nach Stelligkeit

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

Präzedenz und Assoziativität

Faustregel:

1. Unäre Operatoren
2. Explizite Klammern
3. Punkt vor Strich
4. Arithmetisch vor Vergleich
5. Vergleich vor Logisch
6. ! vor && vor ||

Analyse eines arithmetischen Ausdrucks

Beispiel `10 * a / 5 + 32 + 4 * p`

- Arithmetischer Ausdruck
- Besteht aus vier Literalen, zwei Variablen und fünf Operatorsymbole

Präzedenz

- Multiplikative Operatoren haben höhere Präzedenz als additive Operatoren

Beispiel

```
10 * a / 5 + 32 + 4 * p
```

```
(10 * a / 5) + 32 + (4 * p)
```

Assoziativität

- Linksassoziativ:

```
A op B op C ⇔      (A op B) op C
```

- Rechtsassoziativ:

```
A op B op C ⇔      A op (B op C)
```

Beispiel

```
(10 * a / 5) + 32 + (4 * p)
```

```
((10 * a) / 5) + 32 + (4 * p)
```

Aufgabe Präzedenz und Assoziativität

- Wie lautet das Ergebnis des folgenden Ausdrucks?

```
6 * 4 + 3 / 4 / 5 * 3 + 7 - 8
```

Lösung Präzedenz und Assoziativität

```
6 * 4 + 3 / 4 / 5 * 3 + 7 - 8
```

Präzedenz: Punkt vor Strich

```
(6 * 4) + (3 / 4 / 5 * 3) + 7 - 8
```

Assoziativität */: Links

```
(6 * 4) + (((3 / 4) / 5) * 3) + 7 - 8
```

Lösung Präzedenz und Assoziativität

```
(6 * 4) + (((3 / 4) / 5) * 3) + 7 - 8
```

```
(6 * 4) + ((0 / 5) * 3) + 7 - 8
```

```
(6 * 4) + (0 * 3) + 7 - 8
```

Assoziativität +-: Links

```
((24 + 0) + 7) - 8
```

23

Aufgabe Präzedenz und Assoziativität

- Wie lautet das Ergebnis des folgenden Ausdrucks?

```
(2 + 3 + 4) * (4 - 3 * 2) / 2
```

Lösung Präzedenz und Assoziativität

```
(2 + 3 + 4) * (4 - 3 * 2) / 2
```

Präzedenz: Klammern zuerst

```
(2 + 3 + 4) * (4 - 3 * 2) / 2
```

Präzedenz: Punkt vor Strich

```
(2 + 3 + 4) * (4 – (3 * 2)) / 2
```

Lösung Präzedenz und Assoziativität

```
(2 + 3 + 4) * (4 - (3 * 2)) / 2
```

Tiefste Klammer zuerst

```
(2 + 3 + 4) * (4 - (3 * 2)) / 2
```

```
(2 + 3 + 4) * (4 - 6) / 2
```

Assoziativität */: Links

```
(9 * -2) / 2
```

-9

Aufgabe Präzedenz und Assoziativität

- Setze Klammern um die Präzedenz zu visualisieren:

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

Lösung Assoziativität

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

Arithmetische Zuweisung

```
a += b    ⟺    a = a + b
```

- Analog für `-, *, /, %`

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, x--`
- **Wichtig**: Inkrement und Dekrement Operatoren nur auf Variablen anwenden! Folgendes geht nicht: `--42`

Aufgabe Inkrement Operator

- Was hat z für einen Wert nach der Ausführung?

```
int x = 7;
int y = 9;
int z = x++ * ++y;
```

Lösung Inkrement Operator

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

Präzedenz: Inkrement Operatoren

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

```
z = 70
```

Aufgabe Inkrement/Dekrement Operator

- Was haben `z1` und `z2` für einen Wert nach der Ausführung?

```
int a = 2;
int b = 3;
int z1 = (4 + 2) * 3 / 7 - a++ + ++b;
int z2 = ++a * ++b - 4 * 7;
```

Aufgabe Inkrement/Dekrement Operator

Was gibt das Programm aus?

```
public class Ausdruckswert {
    public static void main(String[] args) {
        int i = 4;
        int j = ++i;
        i += j++;
        System.out.println(i * j);  // ?
    }

}
```

Quelle: Informatik II Vorlesung

Typsystem

- Bei Java müssen alle Typen deklariert werden (statisches Typsystem)

```
int a1;
float a2;
char a3 = …;
boolean a4 = …;
```

Implizite Typkonvertierung

- Eine *implizite Typkonvertierung* 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**

Explizite Typkonvertierung

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

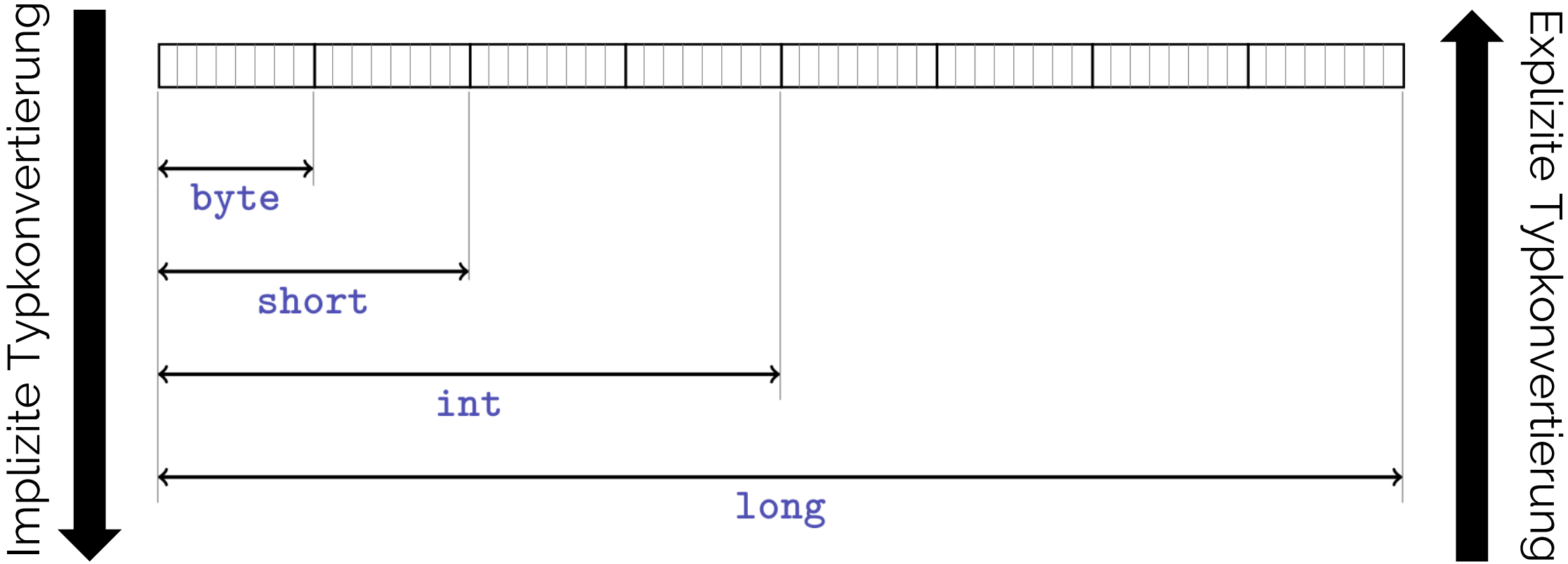
```
urpsrungstyp a = ...;
zieltyp b = (zieltyp)a;
```

- Reihenfolge (absteigend in Grösse):

  `double` < **`float`** < **`long`** < **`int`** < **`short`** < **`byte`**

- Bei expliziter Konversion von **`int`** nach **`float`** werden die Nachkommastellen abgeschnitten (*Informationsverlust*)

Typkonvertierung bei Ganzzahlen



Quelle: Informatik II Vorlesungsfolien

Aufgabe Typkonvertierung

- Ist dieser Code korrekt? Wenn nicht, wo liegt der Fehler?

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

Lösung Typkonvertierung

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

Aufgabe Typanalyse

- Was ist problematisch am folgenden Code?

```
int celsius = 1;
float fahrenheit = 9 * celsius / 5 + 32;
```

Lösung Typanalyse

- Da wir nur mit ganzen Zahlen arbeiten, jedoch ein Fliesskommazahl-Ergebnis erhalten möchten müssen wir einen Operanden als in einen float umwandeln:

```
int celsius = 1;
float fahrenheit = 9.0f * celsius / 5 + 32;
```

Regeln für binäre Operanden

Regeln für op = {+, -, /, *, %}

1. Ganzzahl **op** Ganzzahl = **Ganzzahl**
2. Ganzzahl **op** Fliesskommazahl= **Fliesskommazahl**
3. Fliesskommazahl **op** Ganzzahl = **Fliesskommazahl**
4. Fliesskommazahl **op** Fliesskom. = **Fliesskommazahl**

Beispiel

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

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();
}
```

Vorsicht bei Anweisungsblöcken

Wichtig zu beachten bei Anweisungsblöcken:

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

=

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

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

Schleifen: `for`

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

Ausführreihenfolge:

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

Schleifen: for

Beispiel

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

Beispiel

```
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;
}
```

Schleifen: `for` ⇔ `while`

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

=

Beispiel

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

Boolesche Ausdrücke

- Ausdruck, der zu `true` oder `false` ausgewertet wird
- Für die Bedingungen in den Schleifen und Kontrollstrukturen werden boolesche Ausdrücke verwendet

Vergleichsoperatoren

- Folgende Operatoren existieren:

`<` `>` `<=` `>=` `==` `!=`

- Resultat vom Typ **`boolean`**
- `<` `>` `<=` `>=` binden stärker als `==` `!=`

Boolesche Operatoren

- **&&** (binär): Konjunktion (logisches «und»)
- **||** (binär): Disjunktion (logisches «oder»)
- **!** (unär): Negation
- Präzedenz: **!** vor **&&** vor **||**

Boolesche Ausdrücke

Beispiel

- Befindet sich eine Zahl im Interval `[3,5]`?

```
int x = 4;
boolean isInInterval = (x >= 3) && (x <= 5);
```

Aufgabe Boolesche Ausdrücke

- Definiere einen booleschen Ausdruck, der `true` ist wenn eine Variable `x` durch 5 und 7 teilbar ist

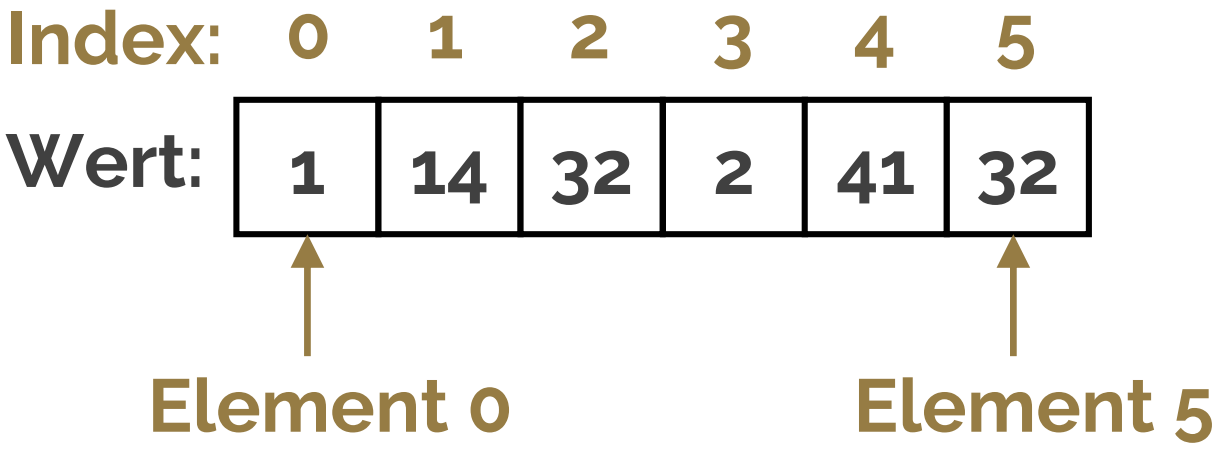
Prüfung 08.2014 Aufgabe 7b

- 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);
    // ...
}
```

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

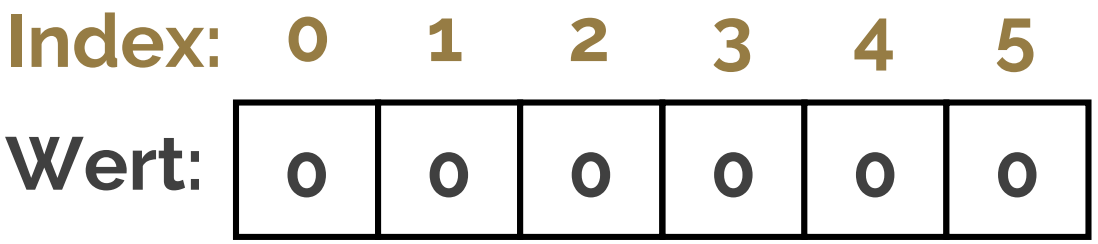


Arrays: Deklaration und Initialisierung

```
Typ[] name = new Typ[Länge];
```

Beispiel

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

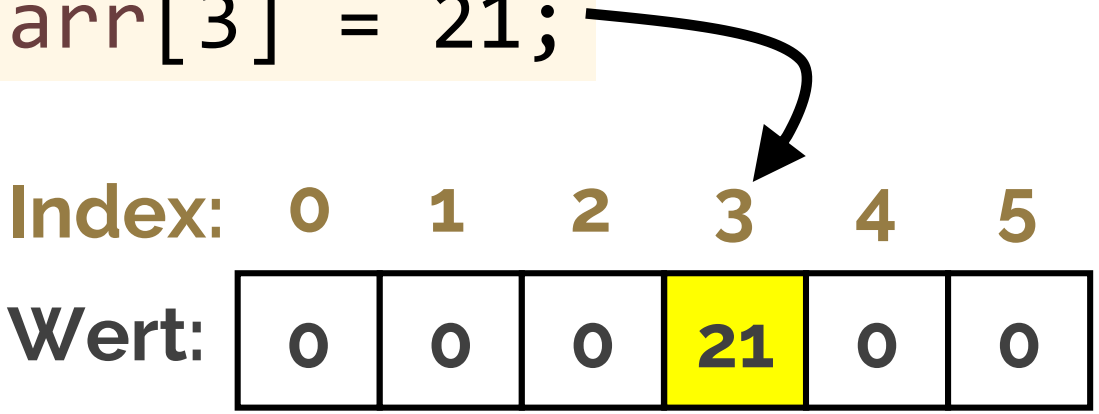


Arrays: Elementzugriff

- Element lesen:
- Element speichern:

```
int a = arr[index];
```

```
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

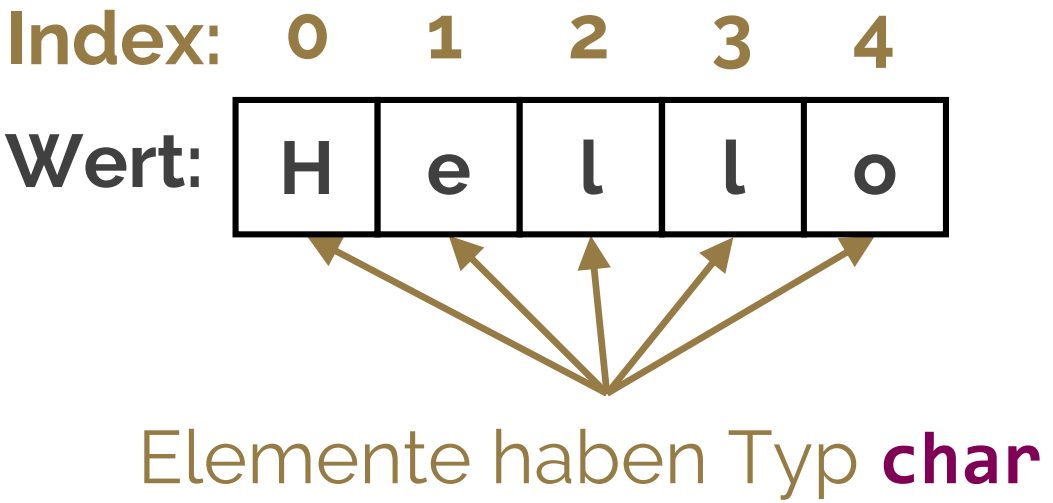
- Fehler? Auswirkung?

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

- `i`-ter Buchstabe eines Strings auslesen:

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

String Operationen

- Bestimmte Buchstaben ersetzen:

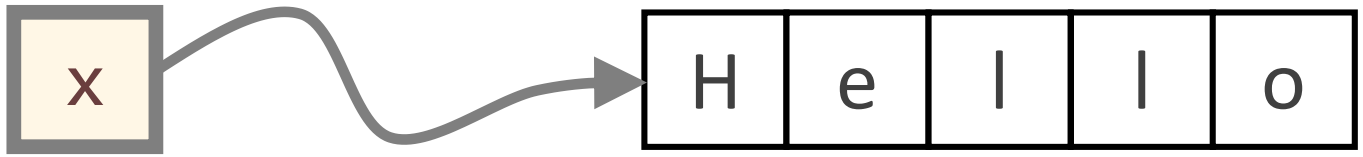
```
String s = "Hello";
String t = s.replace('e', 'a');
// t = "Hallo"
// s = "Hello" (nicht verändert)
```

  - **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

- Variable ist Referenzen auf Speicherblock

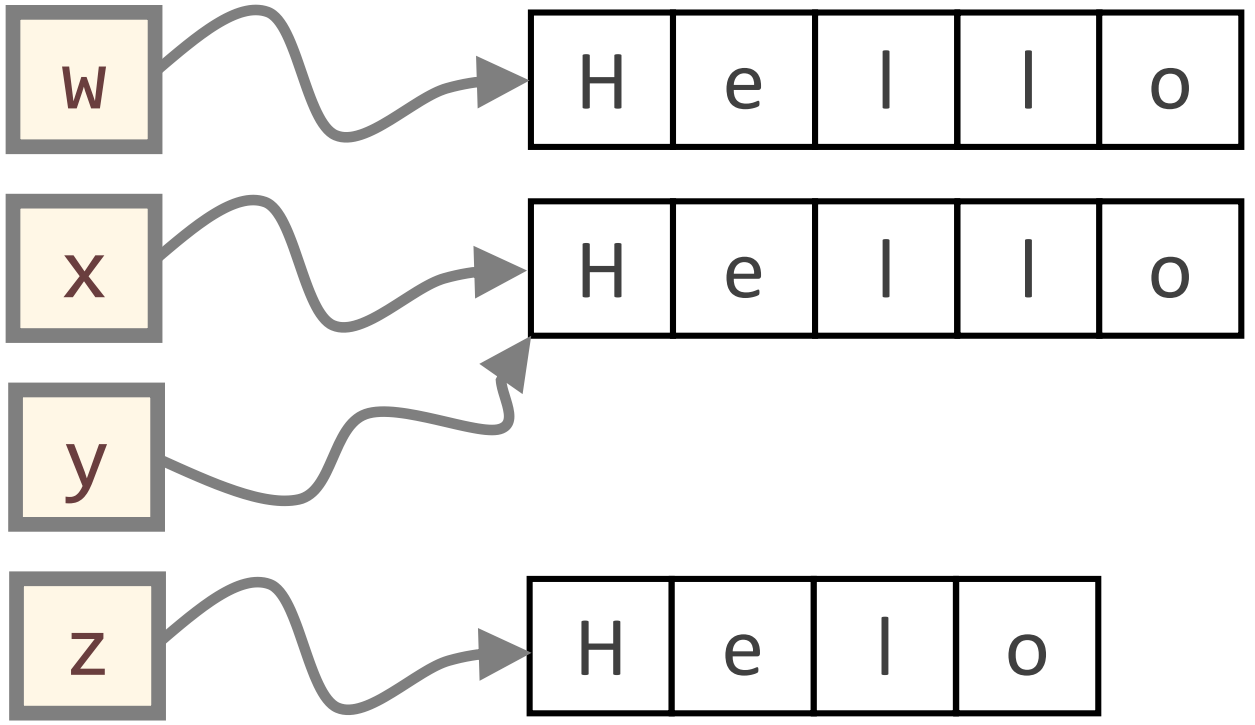
```
String x = "Hello";
```



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

Strings: Zeichenkettenvergleiche

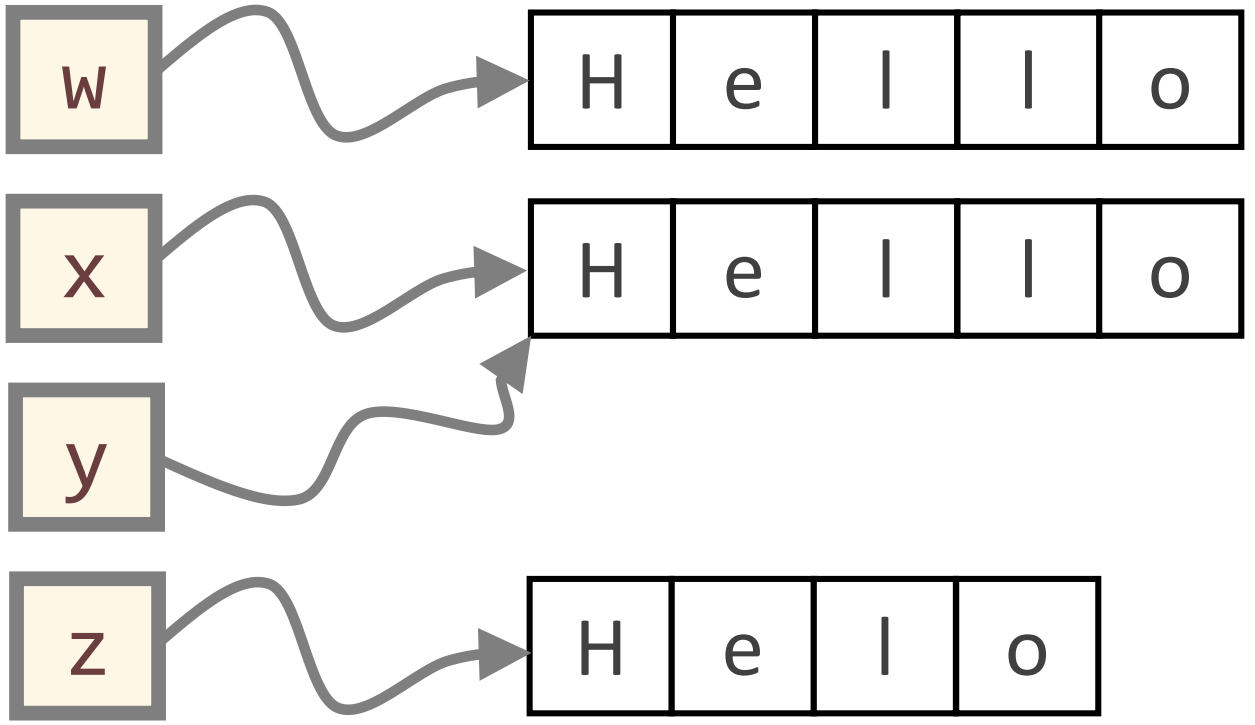
```
String w = "Hello";
String x = "Hello";
String y = x;
String z = "Helo";
```




```
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
```

Primitive Datentypen

- Primitive Datentypen:
  `boolean`, `byte`, `char`, `short`, `int`, `long`, `float`, `double`
- Variable zeigt auf einen einzelnen, konkreten Wert

Nicht-primitive Datentypen

- Nicht-primitive Datentypen:
  Alle anderen Datentypen, z.B. `String`, `int[]`, etc.
- Instanzen werden meistens mittels **`new`** (dynamisch) erzeugt:

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

  - Ausnahme: z.B. `String`

```
String w = "Hello";
```

- Variable ist eine Referenzen die auf einen Speicherblock zeigt

Vergleiche

- Bei primitiven Datentypen:
  == vergleicht Werte der beiden Variablen miteinander

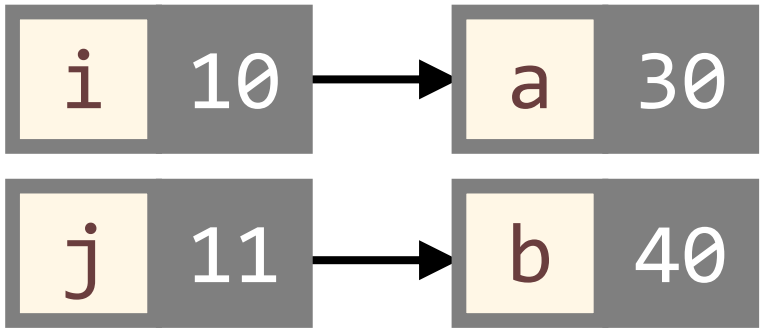
```
char c1 = ‘h’;
char c2 = ‘h’;
boolean b = (c1 == c2); // true
```

- Bei nicht-primitiven Datentypen:
  == 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;
}

… void main(String[] args){
      int i = 10, j = 11;
      do(i,j);
}
```



Nach Aufruf von do(i,j):

i == 10

j == 11

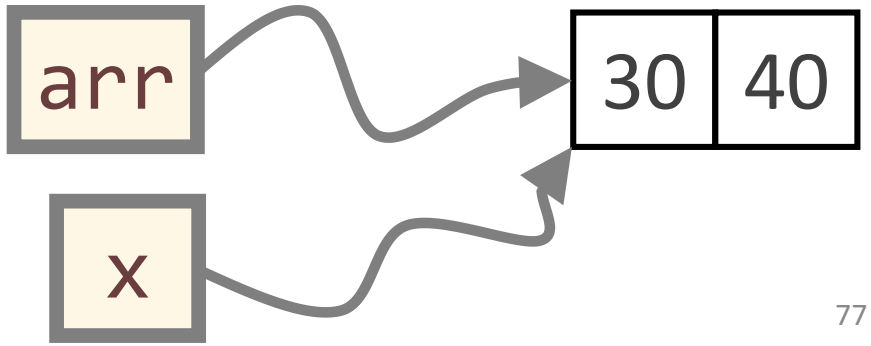
Java ist Pass by Value

- Parameter: Referenzen auf Objekte werden kopiert

```
void do(int x[]){
      x[0] = 30;
      x[1] = 40;
}
… void main(String[] args){
      int[] arr = new int[2];
      arr[0] = 10;arr[1] = 11;
      do(arr);
}
```

Zu Beginn:

Nach Aufruf von do(arr):

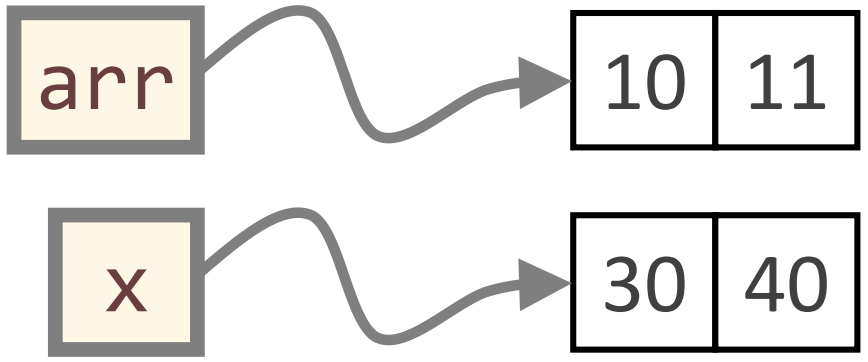


Java ist Pass by Value

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

… void main(String[] args){
      int[] arr = new int[2];
      arr[0] = 10;arr[1] = 11;
      do(arr);
}
```

Zu Beginn:

Nach Aufruf von do(arr):



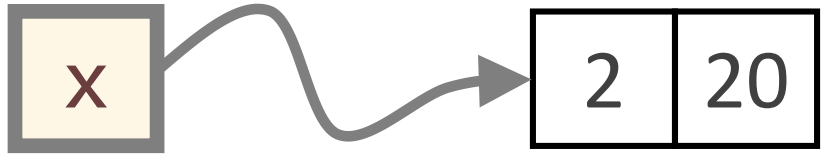
Aufgabe Pass by Value

- Welche Elemente befinden sich im Array x?

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

… void main(String[] args){
      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;
}
…

… void main(String[] args){
      int[] x = new int[2];
      x[0] = 100; x[1] = 200;
      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]);
}
```

Aufgabe Pass by value

```
static void I(int x, int y) {
   int temp = x;
   x = y;
   y = temp;
}
static void A(int[] x) {
   x[0] = x[1];
}
static void S (String x, String y) {
   x = y;
}
public static void main(String[] args) {
   int[] x = {1,2};
   String name = "ETH";
   I(x[0],x[1]);
   A(x);
   S(name, "EPFL");
   System.out.println(x[0]+","+x[1]+","+name);
}
```

Quelle: Informatik II Vorlesung

Was wird ausgegeben?

(1) 1,1,ETH

(2) 1,2,ETH

(3) 2,1,ETH

(4) 2,2,ETH

(5) 1,1,EPFL

(6) 1,2,EPFL

(7) 2,1,EPFL

(8) 2,2,EPFL