Einführung in Java

Arne Hüffmeier

Michelle Liebers, Dennis Hoffmann

Tilman Lüttje, Jean Wiele

Angelehnt an Java-Vorkurs der Freitagsrunde

Was ist Java

- 1995 von SUN Microsystems entwickelt
- Hieß ursprünglich OAK
- Benannt nach der Lieblings-Kaffeesorte der Programmierer.
- Oracle, das 2010 Sun übernimmt, verschärft die Lizenzbedingungen, aber arbeitet auch stärker an der quelloffenen Variante OpenJDK.
- Aktuelle Version: Java 8

Ziele von Java sind,

- Objektorientiertheit
- Plattformunabhängigkeit
- Sicherheit
- Robustheit

HalloWorld

Ein sehr einfache Programm.

```
public class HelloWorld {
    public static void main (String args[]) {
        System.out.println("Hello World!");
    }
}
```

HalloWorld

erste Zeile

```
public class HelloWorld {
```

Start Positions damit der Rechner weiß wo er die Anfangen soll.

zweite Zeile

```
    public static void main (String args[]) {
```

Die Haupt Funktion also das was das Programm machen soll.

dritte Zeile

```
        System.out.println("Hello World!");
```

Ausgabe von Hello World.

1. Starten von NetBeans
2. File $\rightarrow$ New Project
3. Java Application auswählen
4. Project Namen angeben (z.b. HalloWorld)
5. auf Finisch drücken

Syntax

Weiteres

- Blöcke werden mit {...} umklammert
- Anweisungen enden mit ;
- Leerzeichen und Zeilenumbrüche werden ignoriert
- Groß- und Kleinschreibung wird beachtet
- Einzeilige Kommentare mit //
  Was dahinter steht bis zum Ende der Zeile gehört nicht mehr zum Programm
- Mehrzeilige Kommentare mit /* ... */
  Was dazwischen steht gehört nicht mehr zum Programm

Was können wir jetzt damit machen?

- Wir können Text ausgeben ;-)
- Wir können Rechnen.

- ```
  System.out.println(10 + 2);
  ```
- ```
  System.out.println(10 - 2);
  ```
- ```
  System.out.println(10 + 2);
  ```
- ```
  System.out.println(10 + 3 + 2);
  ```
- ```
  System.out.println(10 % 3);
  ```
  ```

Numerische Operationen

- + Addition
- - Subtraktion
- * Multiplikation
- / Division
- % Modulo (Division mit Rest)

Variable

Aus der Schule kennt ihr

$$f(x) = \ldots$$

Das x ist eine Variable was für eine beliebige Zahl steht.

In der Informatik ist eine Variable ein Zwischenspeiche für einen Wert.

Variablen Typen

Beispiele Typen

boolean:	true oder false
int:	42
double:	3.141
char:	'a' oder 4
String:	"Hallo Welt"

Variablen

Wie benutze man die Variablen?

`int a;`	Deklaration
`a = 2;`	Initialisierung
`int a = 2;`	Beides in einer Zeile

```
int a = 5;
int b = 10;
System.out.println(a + b);
```

```
System.out.println(a - b);
```

```
System.out.println(a + b);
```

```
System.out.println(a % b);
```

```
boolean c = true;
System.out.println(c);
```

```
String d = "Ich bin ein String";
System.out.println(c);
```

Numerische Operationen

+	Addition
-	Subtraktion
*	Multiplikation
/	Division
%	Modulo (Division mit Rest)
++	Inkrement (entspricht + 1)
- -	Dekrement (entspricht - 1)
+=	Addition mit Zuweisung a += b -> a = a + b
-=	Subtraktion mit Zuweisung a -= b -> a = a - b

```
System.out.println(a++);
```

```
a += b;
System.out.println(a);
```

```
int a = 7;
int b = 3;
System.out.println(a / b);
```

Casting

Wieso kann ich 7/3 nicht berechnen?

```
double d = 7/3;
```

d = 1

So geht es

```
double d = (double)7/(double)3;
```

d = 1.5

Das (double) bedeutet, dass wir Java sagen, er soll die Zahl als double behandeln und nicht als int.
So können wir auch aus einem int ein char machen.

```java
double a = 1.37637;
System.out.println((int)a);
```

```java
int b = 137;
System.out.println((double)b);
```

```java
char b = 'b';
System.out.println((int)b);
```

if-Anweisung

```
if (<Bedingung>) {
    // Anweisung wenn true
}
```

<Bedingung>

- Muss ein boolescher Ausdruck sein (true oder false)
- z.B. 1<2

Anweisung

Wenn <Bedingung> true ist dann wird der Anweisungsblock ausgeführt

if else

Wo ein **if** da auch manchmal ein **else**

```
if (Bedingung) {
    // Anweisung wenn true
} else {
    // Anweisung wenn falsch
}
```

Es wird das ein **oder** das ander ausgeführt.

```java
boolean a = true;
if (a){
    System.out.println("Wahr");
}else{
    System.out.println("Falsch");
}
```

```java
int b = 16;
if (b == 16){
    System.out.println("Wahr");
}else{
    System.out.println("Falsch");
}
```

```java
boolean a = true;
if (a== false){
System.out.println("Wahr");
}else{
System.out.println("Falsch");
}
```

Logische Operationen

==	Gleichheit
!=	Ungleichheit
<	kleiner als
<=	kleiner gleich
>	größer als
>=	größer gleich
!	nicht
&&	logisches UND
||	logisches ODER

Logische Operationen

Sonderfälle

Einen String vergleicht man mit equals(), also:

```
string.equals(andererString) – Gleichheit oder
!string.equals(andererString) – Ungleichheit
```

else if

Mehrere verschachtelt Anweisungen

```
if (a) {
    // Anweisung
} else {
    if (b) {
        // Anweisung
    } else {
        if (c) {
            // Anweisung
        } else {
            // Anweisung
        }
    }
}
```

Geht das auch einfacher?

JA, geht es!

else if

Hier in schön

```
if (a) {
    // Anweisung
} else if (b) {
    // Anweisung
} else if (c) {
    // Anweisung
} else {
    // Anweisung
}
```

else if ist ein Konstrukt, welches ausgeführt wird, wenn die vorherige Bedingung nicht wahr ist, aber die aktuelle.

```
String b = "Ja";
if (b.equals("Ja")){
    System.out.println("Wahr");
}else{
    System.out.println("Falsch");
}
```

```
String b = "Ja";
if (b.equals("Ja")){
System.out.println("Wahr");
}else{
System.out.println("Falsch");
}
```

```
int c = 3;
if (c == 1){
    System.out.println("c ist 1");
}else if(c == 2){
    System.out.println("c ist 2");
}else if(c == 3){
    System.out.println("c ist 3");
}else{
    System.out.println("c ist nicht 1,2 oder 3");
}
```

Ein kleines Problem

Stellen wir uns vor wir wollen eine Werte Tabelle für

$$f(x) = \ldots$$

berechnen mit den Werten von -100 bis 100.

Das wären über 200 Variable die erstellet werden müssten.

Die Lösung

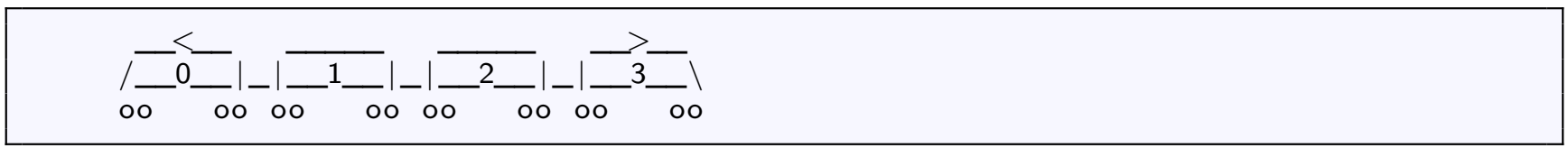
Arrays

Was ist ein Array?

Was ist ein Array?

Was ist ein Array?

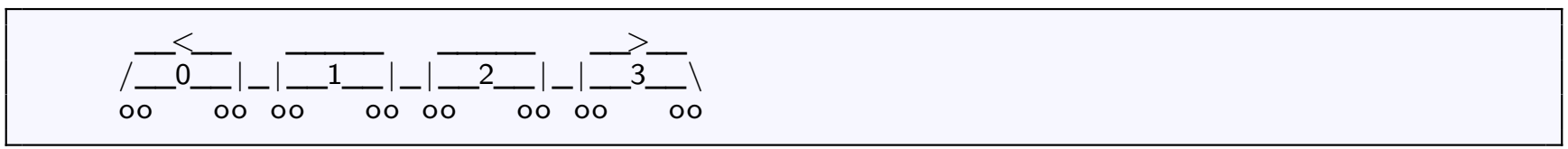
Ein Array ist wie ein Zug



Wir fangen bei 0 an zu zählen.

Ein Zug?

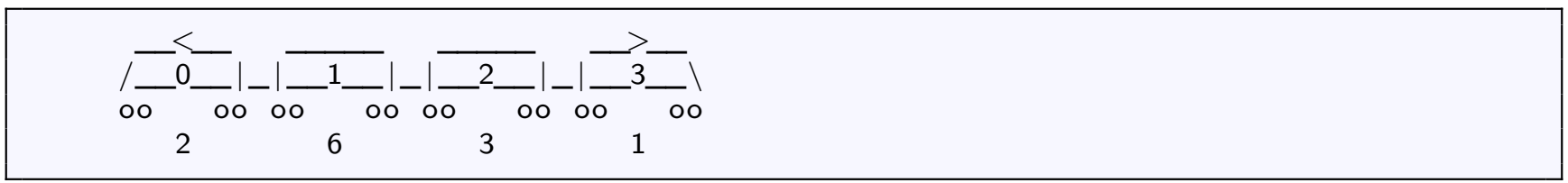
Ein Zug hat viele Wagen, mit unterschiedlich vielen Passagieren darin.



Unser Zug hat auch unterschiedlich viele Personen in den Wagen.

Ein Zug?

Ein Zug hat viele Wagen, mit unterschiedlich vielen Passagieren darin.



Unser Zug hat auch unterschiedlich viele Personen in den Wagen. Zum Beispiel 6 Personen in Wagen 1.

Erstellen eines Arrays

Wie stellen wir nun einen Zug als Datenstruktur da?

```
int[] zug = {2,6,3,1};
```

int ist in diesem Fall unser Datentyp.
Die [] hinter dem Datentyp geben an, dass es ein Array ist.
zug ist der Name unserer Variable, in diesem Fall dem int[].
Mit dem **= {2,6,3,1}** weisen wir unserem **zug** ein Array zu.

Deklarieren und Initialisieren

Und wenn man die Personenzahl noch nicht kennt?

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

Das *new int[4]* bedeutet dass wir ein Array der Länge 4 erstellen.
Also eins mit 4 Feldern.

Genauers zu dem new kommt wenn wir über Objekte sprechen.

Konstante Fahrgastzahlen

Ein Zug, der immer die selbe Anzahl an Fahrgästen hat?

Der Deutschen Bahn würde es vielleicht gefallen, aber es ist ein bisschen unpraktisch.

Darum lernen wir nun, wie wir die Anzahl der Fahrgäste ändern.

Werte zuweisen

Wir haben unseren Zug.

```
int[] zug = {2,6,3,1};
```

Nun steigen im Bahnhof neue Fahrgäste ein.

```
zug[1] = 10;
```

Wie voll ist nun der Zug?

```
{2,10,3,1}
```

Werte zuweisen

Gucken wir uns die Zuweisung noch einmal genau an.

```
zug[1] = 10;
```

Mit *zug[1] = 10;* weisen wir dem Wagen mit der Nummer 1 den Wert 10 zu.

Aber warum ist das nicht das erste Feld?

Weil wir bei 0 anfangen zu zählen!
Der Wagen mit der Nummer 1 ist somit der zweite Wagen.

Array erstellen mal anders

Nun können wir unseren Zug auch anders erstellen.

```
int[] zug = new int[4];
zug[0] = 2;
zug[1] = 6;
zug[2] = 3;
zug[3] = 1;
```

Dies ist deutlich näher an der Realität. Ein Zug wird ohne Fahrgäste gebaut und die Passagiere steigen erst später ein.

```
int[] zug = {2,6,3,1};
System.out.println(zug[0]);
zug[0]= 15;
System.out.println(zug[0]);
System.out.println(zug[3]);
```

```
//es steigen Leute in den hinteren Wagen ein:
zug[3]=zug[3]+5;
System.out.println(zug[3]);
```

```
boolean[] d = {true, false, true};
```

```
String[] d = new String[4];
d[0]="ich";
d[1]="bin";
d[2]="ein";
d[3]="Satz";
```

Matrix (Mehrdimensionales Array)

Man kann auch mehr als nur Arrays von Zahlen machen.
Oder mit mehreren Dimensionen, wie bei einer Matrix. Weisen wir mal feldweise die Zahlen zu

```
int[][] matrix = new int[3][4];
  matrix[0][0] = 1;
  matrix[0][1] = 2;
  matrix[0][2] = 3;
  matrix[0][3] = 4;
  matrix[1][0] = 2;
  matrix[1][1] = 3;
                .
                .
                .
  matrix[3][3] = 1;
  matrix[3][4] = 2;
```

- Wir erstellen ein Array mit 3x4 Feldern.
- Wir können beliebig viele Dimensionen nutzen.

Matrix (Mehrdimensionales Array)

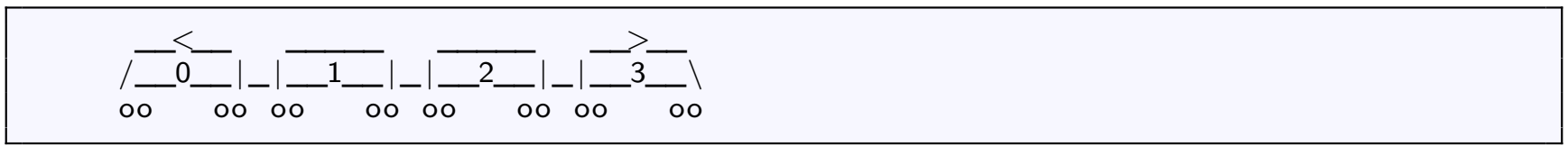
Es geht auch anders mit dem Zuweisen
zum Beispiel zeilenweise

```
int[][] matrix = new int[3][4];
       int[] a =  { 1 , 2 , 3 , 4 };
       int[] b =  { 2 , 3 , 4 , 1 };
       int[] c =  { 3 , 4 , 1 , 2 };
     matrix[0] = a;
     matrix[1] = b;
     matrix[2] = c;
```

Oder mit nur einer einzigen Zuweisung

```
int[][] matrix = {{ 1 , 2 , 3 , 4 },
                  { 2 , 3 , 4 , 1 },
                  { 3 , 1 , 4 , 2 }};
```

Zurück zum Zug



Was passiert eigentlich, wenn wir in unserem Zug auf Wagen 4 zugreifen wollen?

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 4
at Zug.main(Zug.java:4)
```

Was bedeutet dies?

ArrayIndexOutOfBoundsException

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 4
at Zug.main(Zug.java:4)
```

Dies bedeutet:

Exception: Ein Fehler ist aufgetreten

ArrayIndexOutOfBounds: Wir versuchen auf einen nicht vorhandenen Wagen zuzugreifen

Wir haben keinen Wagen mit der Nummer 4.
Mit der *length* Angabe des Arrays könnten wir dies erfahren.

length

Gucken wir uns mal *zug.length* an

```
System.out.println(zug.length);
```

Ja, aber da steht doch 4!

Das stimmt, aber es gibt an wie viele Wagen wir haben. Da wir mit 0 anfangen zu zählen, ist die größte Adresse *zug.length - 1*, somit bei uns 3.

So können wir verhindern, dass wir zu weit schreiben.

```
if (a < zug.length) {
  zug[a] = 4;
}
```

```java
int[] zug = {1,2,3,4,5,6};
System.out.println(zug.length);
```

```java
boolean[] test = {true, false, true, true};
System.out.println(test.length);
```

Neues Problem

Wir können nun Arrays erstellen und die Werte in den Feldern ändern, jedoch wie sieht es mit dem auslesen aus?

Vielleicht möchte die Bahn wissen, wie viele Fahrgäste in den Wagen sind.

Fahrgastzahlen

Natürlich könnten wir die Ausgabe wie folgt realisieren:

```
System.out.println(zug[0]);
System.out.println(zug[1]);
System.out.println(zug[2]);
System.out.println(zug[3]);
```

Für unseren Zug geht das ja noch, aber nun kommt ein langer ICE!

```
int[] ice = new int[50];
```

Was machen wir nun?

Passagiere im ICE

Wir könnten das auch von Hand ausgeben lassen

```
System.out.println(ice[0]);
System.out.println(ice[1]);
System.out.println(ice[2]);
                .
                .
                .
System.out.println(ice[49]);
```

Jedoch wäre das sehr zeit rauben.

Wenn wir nun einen Güterzug mit 200 Wagen hätten, würde man ewig daran sitzen ihn auszulesen.

Passagiere im ICE

Das geht auch einfacher!

Mit Hilfe einer while-Schleife!

while-Schleife

Die while-Schleife wiederholt Anweisungen solange die Bedingung erfüllt ist

```
while (Bedingung) {
    // Anweisung
}
```

while erklärt

Bedingung ist ein boolescher Ausdruck. Solange dieser true ist, wird die Anweisung ausgeführt.

Anweisung kann beliebig sein, sie kann auch leer sein.

Passagiere im ICE

Für unseren ICE könnten wir so ganz einfach die Fahrgastzahlen ausgeben

```
int i = 0;
while (i < ice.length) {
    System.out.println(ice[i]);
    i = i + 1;
}
```

Das funktioniert nicht nur mit dem ICE, sondern mit jedem beliebigen Array.

```
int[] ice = {1,2,34,5,7,3,76,32,6,2};
int i = 0;
while (i < ice.length) {
    System.out.println(ice[i]);
    i = i + 1;
}
```

```
int[] ice = {1,2,34,5,7,3,76,32,6,2};
int i = 0;
while(i < ice.length){
    ice[i] = ice[i] + 1;
    i = i + 1;
}
while (i < ice.length) {
    System.out.println(ice[i]);
    i = i + 1;
}
```

```
int i = 0;
while (i < 500) {
    System.out.println("Hallo");
    i = i + 1;
}
```

Hallo sagen

```
int i = 0;
while (i < 500) {
    System.out.println("Hallo");
    i = i + 1;
}
```

Auch das kann man noch vereinfachen.

for-schleife

Die for-schleife macht es möglich.

```
for (Laufvariable; Bedingung; Iterationsschritt) {
        // Anweisung
}
```

for erklärt

Laufvariable ist eine Variable, über die gezählt (iteriert) wird.
z.B. int i = 0

Bedingung ist ein boolescher Ausdruck, hier kann alles hin, was true oder false ergibt.
z.B. $i < 5$

Iterationsschritt ist die Anweisung zum weiterzählen.
z.B. i++ ($i = i + 1$)

Wir können auch for nutzen

Wir können es also nun auch wie folgt darstellen

```
for (int i = 0; i < 500; i++) {
      System.out.println("Hallo");
}
```

Oder unseren ICE ausgeben lassen

```
for (int i = 0; i < ice.length; i++) {
      System.out.println(ice[i]);
}
```

Wir sehen, wir haben mit Schleifen ein paar mächtige Werkzeuge.

Für Ästhetiker nun in elegant

Wir können es auch bei Arrays wie folgt machen

```
for (int wagon : ice) {
      System.out.println(wagon);
}
```

Kurze Erklärung der Bedeutungen

int wagon : ice Für jedes Feld in ***ice***, welches wir nun ***wagon*** nennen, tun wir was in der Schleife steht.
Also geben wir hier für jeden Wagon unseres ICE die Passagierzahl aus.

Das müsst ihr nicht unbedingt verstehen, der andere Weg geht genau so gut und ist genau so richtig.

Verschachtelte Schleifen

Nun wollen wir unsere Matrix ausgeben

```
for (int i = 0; i < matrix.length; i++) {
    for (int j = 0; j < matrix[i].length; j++) {
        System.out.print(matrix[i][j] + " ");
    }
    System.out.println("");
}
```

Kurze Erklärung

System.out.print() Gibt auch das in den Klammern aus, aber ohne eine neue Zeile anzulegen, es geht also in der gleichen Zeile weiter.

1.
```
for(int i =0;i<9001;i++){
    System.out.println(i+1 + ".␣Hello");
}
```

2.
```
int [] ice = {20,5,2,3,5,3,5,5,6,4,8,5,2,7,3,4,3,6,8,9};
for(int i = 0; i < ice.length;i++){
    System.out.println("Der␣" + (i+1) +".␣Wagon␣hat␣" + ice[i] + "␣
        Passagiere.");
}
```

3.
```
int [] ice = {20,5,2,3,5,3,5,5,6,4,8,5,2,7,3,4,3,6,8,9};// instead of new
    int[20]

for(int wagon: ice){
        System.out.println(wagon);
}
```

4.
```
 int [][] matrix = new int [10][10];

for(int a=0;a<matrix.length;a++){
    for(int b = 0; b <matrix[a].length; b++){
        System.out.print(matrix [a][b] +"␣");
    }
    System.out.println("");
}
```

break

Manchmal möchte man eine Schleife auch vorzeitig abbrechen. Zum Beispiel überprüft man eine Zahl, ob sie eine Primzahl ist

```
int k = 1337;
boolean bool = true;
for (int i = 2; i < k; i++) {
    if (k % i == 0) {
        bool = false;
        break;
    }
}
System.out.println(bool);
```

Kurze Erklärung

k %i == 0 Gibt an, ob k ganzzahlig durch i teilbar ist, wenn ja, ist k keine Primzahl.

break Lässt uns aus der Schleife ausbrechen. Alles was nach der Schleife kommt, wird noch abgearbeitet.

Beispiele

```java
int k=1337;
boolean bool =true;
for (int i=2; i<k; i++){
    if (k%i ==0){
        //Gibt an, ob k ganzzahlig durch i teilbar ist
        //wenn ja: k ist keine Primzahl
        bool=false;
        break;
        //Laesst uns aus der Schleife ausbrechen
    }
}
System.out.println(bool);
```

```java
int[] zug = {1,2,3,4,5,6,7,8};
for (int i = 0; i < zug.length; i++){
    System.out.println(zug[i]);
    if (i >= (zug.length/2)){
        break;
    }
}
```

Was kann schon schiefgehen?

Gucken wir uns unseren Zug wieder an

```
int i = 0;
while (i < zug.length) {
    System.out.println(zug[i]);
}
```

Es hört nicht auf und gibt immer nur die gleiche Zahl aus.

Warum?

Wir haben vergessen i weiter zu zählen. So funktioniert es

```
int i = 0;
while (i < zug.length) {
    System.out.println(zug[i++]);
}
```

Lustige Fehlersuche II

Wieder haben wir eine Endlosschleife, woran liegt es?

```
int grenze = 0;
  int zahl = 1;

while (zahl != grenze) {
    zahl++;
}
```

Lustige Fehlersuche II

Eine Lösung wäre

```
int grenze = 0;
  int zahl = 1;

while (zahl < grenze) {
    zahl++;
}
```

$<$ statt $!=$ nutzen

Alternativ können wir auch mit ***break*** entkommen

```
int grenze = 0;
  int zahl = 1;

while (zahl != grenze) {
    if (zahl > grenze) {
        break;
    }
    zahl++;
}
```

Lustige Fehlersuche III

Auch das ist nicht ganz richtig

```
for (int i = 0; i < 10; i++) {
    System.out.println("Ich␣sagte␣" + i);
}
System.out.println("Ich␣sagte␣aber␣nicht␣" + i);
```

Die Ursache ist, dass wir *i* nur im Block der for-Schleife haben, außerhalb des Blocks gibt es kein *i*.

Am besten gar nicht erst Fehler machen.