

252-0027

Einführung in die Programmierung

3.0 Arrays (Reihen)

Thomas R. Gross

**Department Informatik
ETH Zürich**

Übersicht

Verwandte Themen:

- 3.0 Arrays
- 4.0 Klassen und Objekte

Wichtig um «interessante» Programme zu schreiben:

- 5.0 Input/Output

3.0 Reihen (Arrays)

Based on Slides by Reges et al. Copyright © Pearson 2013.
Copyright © Thomas Gross 2023.
All rights reserved.

Übersicht

3.0 Arrays

3.1 Motivation

3.2 Deklaration und Erstellen

3.3 Arbeiten mit Arrays

3.4 Arrays und Methoden

3.5 Invarianten – Beispiel mit Array

3.1 Ein neues Problem

Sie wollen für die Mitglieder eines Sportvereins (oder einer Gruppe/Team/Sektion) Daten analysieren

z.B. Grösse: Durchschnittsgrösse und Anzahl der Spieler/innen die über dem Durchschnitt liegen

Könnte so aussehen

- Input unterstrichen

Anzahl Mitglieder? 6

Groesse in cm: 165

Groesse in cm: 164

Groesse in cm: 158

Groesse in cm: 163

Groesse in cm: 169

Groesse in cm: 181

Durchschnitt in cm = 166.7

Anzahl \geq Durchschnitt: 2 (33 %) 5

Schritte

- Jeder Wert (Grösse) muss eingegeben werden um den Durchschnitt zu berechnen (Summe dividiert durch Anzahl Mitglieder)
- Nach der Berechnung des Durchschnitts müssen wir zählen wieviele Personen grösser (gleich) als der Durchschnitt sind
 - Aber den Durchschnitt kennen wir erst zum Schluss
 - Müssen die Messwerte bis zum Ende speichern

Was ist problematisch?

- Jeder Wert muss *zweimal* verwendet werden:
 - Um den Durchschnitt zu berechnen
 - Um die Anzahl Personen die grösser (gleich) als der Durchschnitt sind zu finden
- Könnten natürlich jeden Wert in einer Variable speichern ... aber :
 - Wir wissen nicht mit wievielen Personen wir arbeiten bis das Programm ausgeführt wird
 - Wir wissen nicht wieviele Variable wir brauchen
- Wir brauchen einen Weg, mehrere Variable auf einmal zu deklarieren

«Arrays» (Reihen)

- Ein Array erlaubt uns mehrere Werte des *selben Typs* zu speichern
 - Element: Ein Wert in einem Array.
 - Index: Zahl (≥ 0) um ein Element eines Arrays auszuwählen
 - Base: Das erste Element hat den Index 0

«Arrays» (Reihen)

- Ein Array erlaubt uns mehrere Werte des *selben Typs* zu speichern.
 - Element: Ein Wert in einem Array.
 - Index: Zahl (≥ 0) um ein Element eines Arrays auszuwählen
 - Base: Das erste Element hat den Index 0

Index	0	1	2	3	4	5	6	7	8	9
Wert	12	49	-2	26	5	17	-6	84	72	3

Element 0				Element 4						Element 9
-----------	--	--	--	-----------	--	--	--	--	--	-----------

Ein Array für
int Werte

«Arrays» (Reihen)

- Ein Array erlaubt uns mehrere Werte des *selben Typs* zu speichern
 - Element: Ein Wert in einem Array.
 - Index: Zahl (≥ 0) um ein Element eines Arrays auszuwählen
 - Base: Das erste Element hat den Index 0
- Müssen Variable deklarieren um auf Array zugreifen zu können
 - Deklaration in einer Methode (z.B. `main`)

3.2 Array Variable – *Deklaration der Variable und Erstellen des Arrays*

Deklarieren der Variable

Erstellen des Arrays

Initialisieren (Array wird immer initialisiert wenn erstellt)

- **Zwei Varianten**
 - **Deklaration der Variable und Erstellen des Arrays *zusammen***
 - ***Nur* Deklaration der Variable**

type[] *name* = new *type*[*length*]

- *type*: Der Typ der Elemente des Arrays

name verweist auf
Array von *type*
Elementen

Deklaration
und
Initialisierung

type[] *name* = new *type*[*length*]

- *type*: Der Typ der Elemente des Arrays

- Genauer:

type_var[] *name* = new *type_element*[*length*];

- Z. Zt. müssen *type_var* und *type_element* identisch sein

- Beispiel:

- `int[] numbers = new int[10];`

z.Zt.: selber
Typ!

type[] *name* = new *type*[*length*]

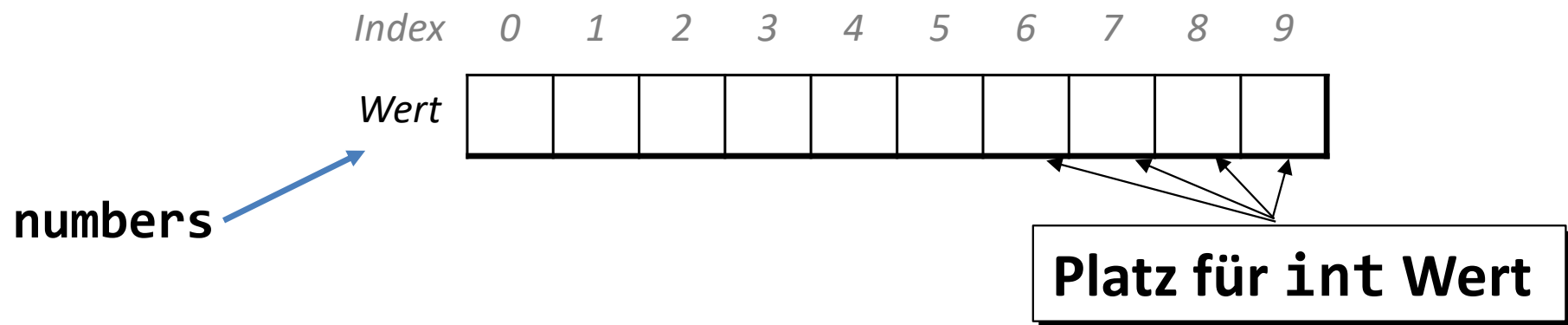
- *type*: Der Typ der Elemente des Arrays
- *name*: Name der Variable mit der Zugriff möglich ist
- **new**: Operator der Array mit Platz für *length* Element erstellt
- *length*: Länge – beliebiger `int` Ausdruck ≥ 0
 - Beispiele:
 - `int[] numbers = new int[10];`
 - `int x = 2 * 3 + 1;`
 - `int[] data = new int[x % 9 + 3];`

type[] *name* = new *type*[*length*]

- Beispiel:

`int[] numbers = new int[10];`

- Deklariert Variable `numbers` und lässt sie auf Array mit 10 Elementen des Typs `int` verweisen

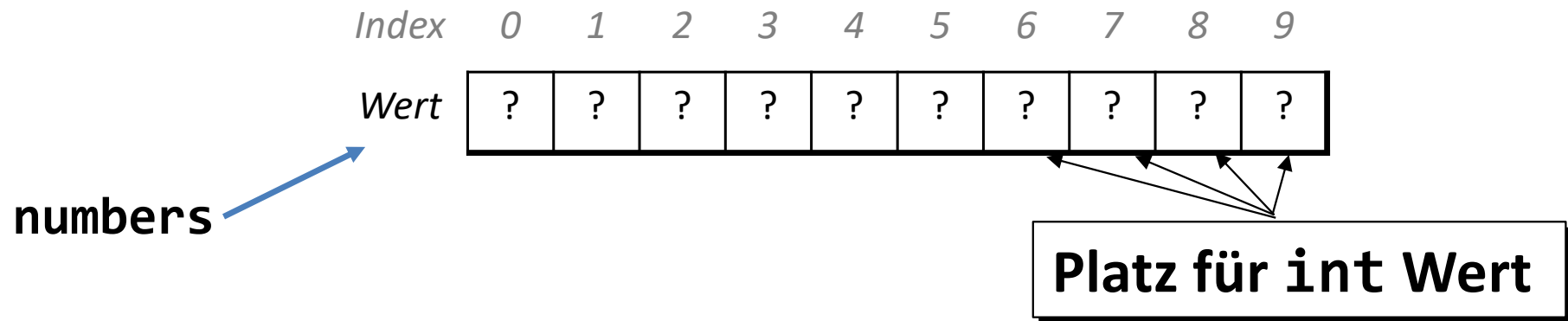


type[] *name* = new *type*[*length*]

- Beispiel:

`int[] numbers = new int[10];`

- Deklariert Variable `numbers` und lässt sie auf Array mit 10 Elementen des Typs `int` verweisen



Array: Deklarieren und Erstellen

- **new**: Operator der Array mit Platz für *length* Element erstellt
- **Was für Werte sollen da (am Anfang) gespeichert werden?**
 - Wollen verhindern dass ein Program Daten eines anderen liest
 - Wollen bei wiederholter Ausführung das *selbe Resultat* (ohne Input...)
 - Daher genau definiert:
- **Jedes Element wird auf einen Wert der Null «entspricht» gesetzt**
 - Voreinstellung («default»)

Type	Default Wert
int	0
long	0
double	0.0
boolean	false
String	null

Array: Deklarieren (der Variable) *ohne* Erstellen (des Arrays)

type[] *name*;

- Deklariert eine Variable (*name*) die auf Arrays mit *type* Elementen verweist
 - Beispiel: `int[] numbers;`
 - Keine Länge nötig
 - Variable `numbers` kann auf Array mit 10 `int` Elementen, auf Array mit 7 `int` Elementen, auf Array mit 42 `int` Elementen ... verweisen
 - Später Zuweisung nötig
 - z.B. `numbers = new int[42];`

«Arrays» (Reihen)

- Ein Array erlaubt uns mehrere Werte des *selben Typs* zu speichern
 - Element: Ein Wert in einem Array.
 - Index: Zahl (≥ 0) um ein Element eines Arrays auszuwählen
 - Base: Das erste Element hat den Index 0
- Zugriff auf Element: Name einer Variable die auf den Array verweist und Index

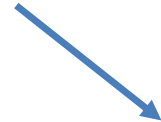
Zugriff auf Elemente: *name[index]* Setzen

name[index] = expression; // Modifikation

- *name*: Name der Variable um auf Arrayelemente zu greifen
- *index*: **int** Wert (Ausdruck der **int** ergibt)
- *expression*: Ausdruck der Wert vom Typ der Elemente des Arrays ergibt

```
int[] data;  
data = new int[10];  
data[0] = 27;  
data[3] = -6;
```

data



Index	0	1	2	3	4	5	6	7	8	9
Wert	27	0	0	-6	0	0	0	0	0	0

Zugriff auf Elemente: *name[index]* Lesen

name[index] liefert Wert des Elements *index*

- *name*: Name der Variable um auf Arrayelemente zu greifen
- *index*: `int` Wert (Ausdruck der `int` ergibt)
- Kann überall auftreten wo ein Wert (dieses Typs) verwendet werden kann

```
int j = data[0];  
int k = data[3];
```

Index	0	1	2	3	4	5	6	7	8	9
Wert	27	0	0	-6	0	0	0	0	0	0

```
System.out.print(j+k)
```

21

Arrays für andere Typen

```
boolean[] results = new boolean[5];  
results[2] = true;  
results[4] = true;
```

<i>Index</i>	<i>0</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>
<i>Wert</i>	false	false	true	false	true

```
String[] address = new String[6];  
address[3] = "Florastrasse 6";
```

<i>Index</i>	<i>0</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>
<i>Wert</i>	null	null	null	"Florastrasse 6"	null	null

Legale Index Werte

Legale Index Werte: zwischen 0 und (Länge des Arrays - 1).

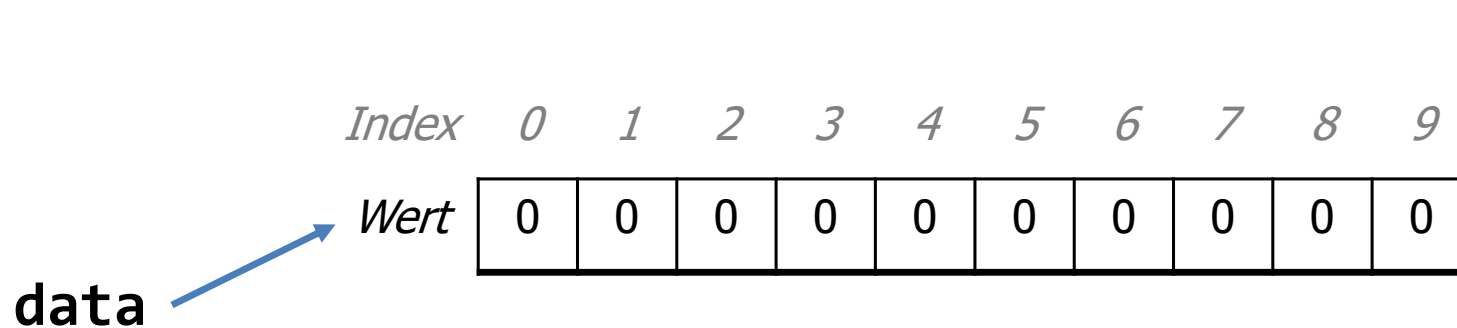
- Einschliesslich
- Lesen oder Schreiben (Zugriff, «access») mit einem Index ausserhalb dieses Bereichs resultiert in einer `ArrayIndexOutOfBoundsException`
- «Out-of-bounds» Fehler

...	-3	-2	-1	Index	0	1	2	3	4	5	6	7	8	9	10	11	12	...
				Wert	0	0	0	0	0	0	0	0	0	0				

Out-of-bounds Zugriffe

- Example:

```
int[] data = new int[10];  
System.out.println(data[0]);           // okay  
System.out.println(data[9]);           // okay  
System.out.println(data[-1]);          // exception  
System.out.println(data[10]);          // exception
```



↑
**Exception bewirkt
Abbruch der
Ausführung dieses
Programms**


Arrays und Zugriffe auf Elemente

```
int[] numbers = new int[8];  
numbers[1] = 3;  
numbers[4] = 99;  
numbers[6] = 2;
```

```
int x = numbers[1];  
numbers[x] = 42;  
numbers[numbers[6]] = 11; // use numbers[6] as index
```

x

3

numbers		Index	0	1	2	3	4	5	6	7
		Wert	0	3	11	42	99	0	2	0

Arrays und for-Schleifen

- Oft wird auf Arrayelemente in einer «for»-Schleife zugegriffen

```
for (int i = 0; i < 8; i++) {  
    System.out.print(numbers[i] + " ");  
}  
System.out.println(); // output: 0 3 11 42 99 0 2 0
```


- Eine Schleife eignet sich auch zur Zuweisung an jedes Element eines Arrays.

```
for (int i = 0; i < 8; i++) {  
    numbers[i] = 2 * i;  
}
```

Index 0 1 2 3 4 5 6 7

Wert

0	2	4	6	8	10	12	14
---	---	---	---	---	----	----	----

numbers 

33

Das `length` Attribut

Das `length` Attribut eines Arrays *name* liefert die Anzahl der Elemente

name.length

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

// output: 0 2 4 6 8 10 12 14

Array Initialisierung

- (Richtige) Programme lesen Dateien um Array zu initialisieren
- Wir können einen Loop verwenden ...

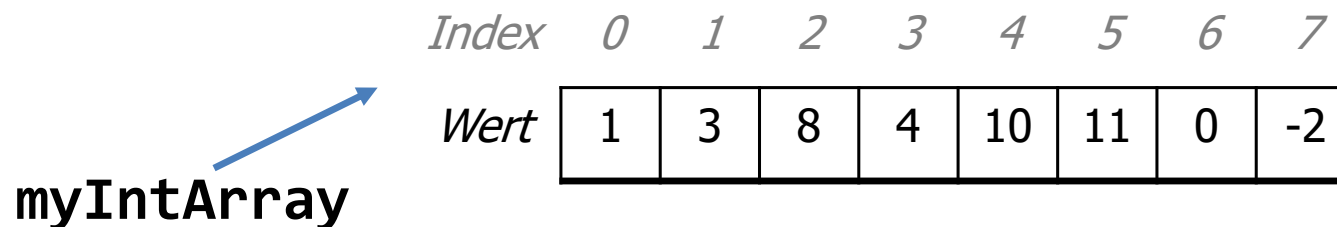
```
for (int i = 0; i < 8; i++) {  
    numbers[i] = 2 * i;  
}
```
- Manchmal mühsam
- Java erlaubt Initialisierung mit Konstanten

Array Initialisierung

type[] name = { value₁, value₂, ..., value_N }

- Deklariert und initialisiert Array *name* mit N Elementen
- *value_i*: Typ muss mit *type* übereinstimmen

```
int[] myIntArray = { 1, 3, 8, 4, 10, 11, 0, -2 }
```




	<i>Index</i>	<i>0</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>	<i>7</i>
myIntArray	<i>Wert</i>	1	3	8	4	10	11	0	-2

Array Initialisierung für andere Typen

type[] name = { value₁, value₂, ..., value_N }

double[] myDoubleArray = {1.0, 0.0, 0.5, 0.999999999999};

		<i>Index</i>	<i>0</i>	<i>1</i>	<i>2</i>	<i>3</i>
myDoubleArray		<i>Wert</i>	1.0	0.0	0.5	0.999999999999

boolean[] myBooleanArray = {true, true, true, false};

		<i>Index</i>	<i>0</i>	<i>1</i>	<i>2</i>	<i>3</i>
myBooleanArray		<i>Wert</i>	true	true	true	false

String[] myStringArray = {"ein", "Versuch"};

		<i>Index</i>	<i>0</i>	<i>1</i>
myStringArray		<i>Wert</i>	"ein"	"Versuch"

Unser Beispiel am Anfang (Messwertanalyse)

Ein Array hilft uns, dieses Programm zu schreiben

```
Anzahl Mitglieder? 6  
Groesse in cm: 165  
Groesse in cm: 164  
Groesse in cm: 158  
Groesse in cm: 163  
Groesse in cm: 169  
Groesse in cm: 181  
Durchschnitt in cm = 166.7  
Anzahl  $\geq$  Durchschnitt: 2 (33 %)
```

Programm

```
// Liest Groessen, berechnet Durchschnitt, gibt Anzahl und
// Prozentsatz >= Durchschnitt aus

import java.util.Scanner;

public class Analyse {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Anzahl Mitglieder? ");
        int members = input.nextInt();

        int[] data = new int[members];
        double sum = 0.0;
        for (int i = 0; i < members; i++) { // Groesse einlesen
            System.out.print("Groesse in cm: ");
            data[i] = input.nextInt();
            sum += data[i];
        } // Fortsetzung naechste Seite
```


Programm , Fortsetzung

```
// compute results
double avg = (double) sum/members;
System.out.println("Durchschnitt in cm = " + avg);
int count = 0;
for (int i = 0; i < members; i++) {
    if (data[i] >= avg) {
        count++;
    }
}
// report results
System.out.println("Anzahl >= Durchschnitt: " +
    count + " (" +
    (double)count/members*100 + " %)");
// Formatierung koennte besser sein ...
}
```

```
}
```

Diskussion

- **Idealerweise können wir diese Anweisungen in einer Methode organisieren**
 - **Möchten den Array der Messwerte als Parameter übergeben können**
- **Wir ignorieren Formatierung der Ergebnisse**

3.3 Arbeiten mit Arrays

Java Variable – Referenztyp

- Eine Referenzvariable (auch: «reference type variable», Variable eines Referenztyps) *erlaubt den Zugriff* auf einen Array

```
int[] myArray;
```

- `myArray` ist eine Referenzvariable (kann für Arrays von `int` Werten gebraucht werden)
 - Nicht für andere Arten von Arrays (dazu später mehr)
 - Speichert/liefert Information die Zugriff auf Array erlaubt
 - Nach der Deklaration gibt es noch keinen Platz für die Daten

Java Variable – Referenztyp

- `int[] myArray` deklariert eine Referenzvariable (kann für Arrays von `int` Werten gebraucht werden)
 - Speichert/liefert Information die Zugriff auf Array erlaubt
 - Noch gibt es keinen Bereich der Daten speichert
- Erst Zuweisung verknüpft Referenzvariable mit einem Array
 - Entweder in der Deklaration `int[] myArray = new int[10];`
 - Oder in separaten Schritten

```
int[] myArray;
```

```
myArray = new int[10];
```

```
int[] myArray = {0, 1, 4, 9};
```

Direkte Initialisierung nur mit Deklaration möglich

Java Variable – Referenztyp

- `int[] myArray` deklariert eine Referenzvariable (kann für Arrays von `int` Werten gebraucht werden)

- Erst Zuweisung verknüpft Referenzvariable mit einem Array

- Wiederholte Zuweisungen sind möglich

```
int[] myArray = {0, 1, 4, 9}; // Deklaration
                                // myArray[2]==4
```

```
myArray = new int[3]; // myArray verweist auf {0,0,0}
```

- Auch Zuweisung auf anderen (existierenden Array) ist möglich

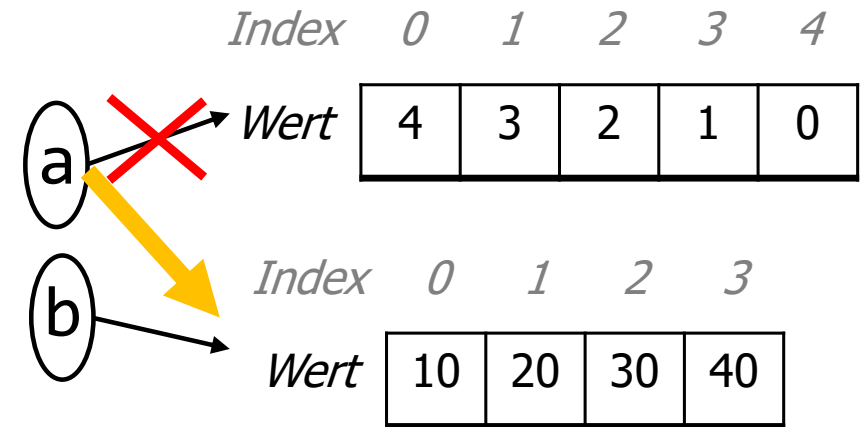
```
int[] yourArray = myArray;
```

- Durch eine Zuweisung kann eine Referenzvariable auf einen anderen Array verweisen

```
int[] a = { 4, 3, 2, 1, 0};
```

```
int[] b = { 10, 20, 30, 40};
```

```
a = b;
```



```
//a:[10, 20, 30, 40]
```

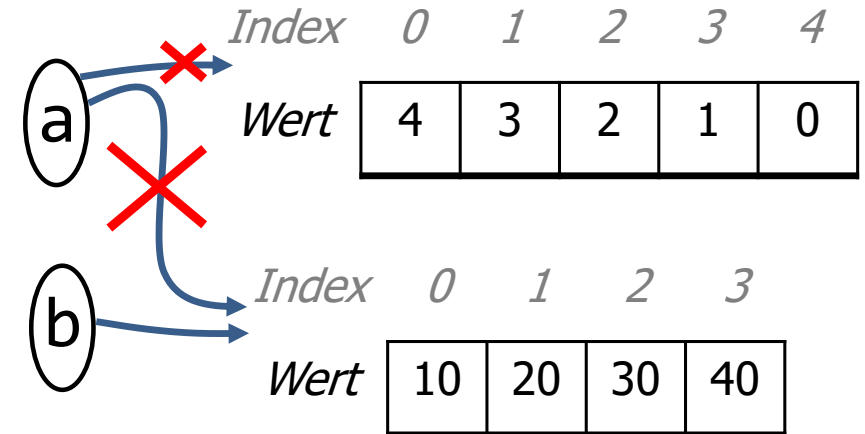
- Die rechte Seite einer Zuweisung zu einer Referenzvariable muss auch eine Referenzvariable sein.
 - Der Typ der Elemente muss übereinstimmen
 - Die Anzahl der Elemente muss *nicht* übereinstimmen
 - (Oder Ergebnis des `new` Operators)

- **Durch eine Zuweisung kann eine Referenzvariable auf einen anderen Array verweisen**

```
int[] a = { 4, 3, 2, 1, 0};
```

```
int[] b = { 10, 20, 30, 40};
```

```
a = b;
```



- **Und wenn a nicht mehr auf diesen Array verweisen soll?**

- Zuweisung des besonderen Wertes `null`

```
a = null;
```

- `null` heisst das `a` auf keinen Array verweist
- Nur noch `b` verweist jetzt auf den Array

Zugriff auf Elemente des Arrays ...

erfordert [..] (mit legalem Index) (wenn Array existiert)

```
myArray[1] = 99;
```

```
int b = myArray[7];
```

```
System.out.println(myArray[1]); //99
```

[..] ist auch ein Operator (mit höchstem Rang)

Sonst macht `myArray[1]+1` keinen Sinn

Java Referenzvariable

- **Wir müssen unterscheiden zwischen**
 - Zugriff auf ein Element (mittels `name[index]`)
 - Zugriff auf den Array auf den die Referenzvariable verweist
 - z.B. um andere Referenzvariable zu setzen (= name)
- **Referenzvariable ist eine Kurzform um zu sagen: dies ist eine Java Variable des Typs «Reference auf»**

```
long[] longArray;    //Verweist auf Array mit long Elementen  
int[]  intArray;     //Verweist auf Array mit int Elementen
```

null

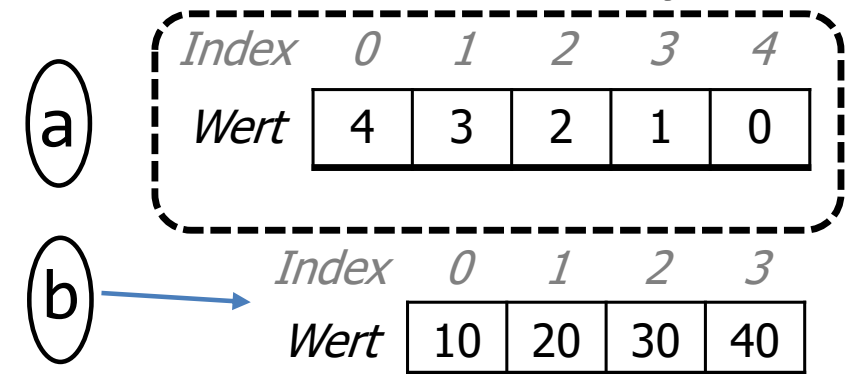
- **null** ist der Wert einer Referenzvariable die auf *keinen* Array verweist.

```
int[] a = new int[10];  
a[0] = 1; ✓  
System.out.println(a.length); ✓  
a = null;  
System.out.println(a.length); ✗  
a[1] = 2; ✗  
System.out.println(a); ✓ //null
```

Was passiert mit dem Array {4, 3, 2, 1, 0}?

Nach Zuweisung von `null` verweist eine Referenzvariable auf keinen Array

```
int[] a = { 4, 3, 2, 1, 0};  
int[] b = { 10, 20, 30, 40};  
a = b;  
a = null;
```



Wenn keine weitere Referenzvariable auf den Array verweist
dann ist der Array `[]` unerreichbar

- Keine direkte Auswirkung auf Programmausführung
- Kann (und wird) irgendwann vom Java System entfernt werden (nicht unser Thema, abhängig von Version, Speicherplatz, Host, ...)
 - Wir können uns auf das Erstellen eines korrekten Programms «beschränken»

Reference Semantics

- Wenn eine Referenzvariable als Operand in einer Zuweisung erscheint ($x=y$) dann wird der Array *nicht kopiert* sondern beide Variablen beziehen sich nun (verweisen nun) auf den selben Array: *Reference Semantics*!
 - Variablen x und y enthalten nun eine Referenz («reference», Verweis) auf den selben Array. Veränderungen der Elemente einer Referenzvariable *bewirken* eine Änderung der Werte der anderen Referenzvariable.
 - *Aliasing*: zwei (oder mehr) Referenzvariable verweisen auf den selben Array

Verweise und Arrays

- **Arrays verwenden Reference Semantics. Warum?**
 - *Effizienz.* Das Kopieren grosser Arrays kostet (zu)viel Zeit.
 - *Programmstruktur.* Es ist oft sinnvoll (und logisch) wenn verschiedene Methoden mit einem (gemeinsamen) Array arbeiten.

```
int[] dataSet = new int[ ... ];  
step1(dataSet);  
step2(dataSet);
```

- **Objekte verwenden auch Reference Semantics**

Arrays sind Objekte

- ... aber viele Operatoren sind für Objekte nicht definiert

- Der Additionsoperator + ist nur für Strings definiert

```
String s = "hello"; String t = "world";  
s = s + " " + t;           // "hello world"  
Scanner sc = new Scanner(System.in);  
Scanner ec = new Scanner(System.in);
```

```
sc = sc + ec; // operator + cannot be applied
```

- Der Multiplikationsoperator * ist auch für Strings nicht definiert

```
s = 2 * s;           // operator * cannot be applied
```

- Auch die meisten Vergleichsoperatoren sind nicht definiert
- Und Output auch nicht so wie wir es wollen

Arrays

- Manchmal wünschen wir uns (vielleicht) dass (z.B.) + definiert ist

```
int[] a = new int[10];
```

```
int[] b = new int[10];
```

```
a = a + b; // error
```

```
if ( a >= b ) { ... } // error
```

- Aber Java unterstützt dies nicht.

Vergleiche von Arrays

- Leider gibt es kein `equals` für Arrays
 - Wie für Strings: `if ("hello".equals(myString)) { ... }`
 - Man kann zwei Arrays weder mit `==` noch mit `equals` vergleichen um zu prüfen ob sie die selben Elemente haben

```
int[] a1 = {42, -7, 1, 15};  
int[] a2 = {42, -7, 1, 15};  
if (a1 == a2) { ... }           // incorrect!  
if (a1.equals(a2)) { ... }      // incorrect!
```

3.4 Arrays und Methoden

- **Array Parameter**
- **Array Rückgabe Werte**

Filter

- Schreiben Sie ein Programm(segment) das in einem Array alle Elemente < 0 auf 0 setzt
 - Zum Beispiel für diesen Array:
[11, -42, -5, 27, 0, 89]
 - Sollte der gefilterte Array so aussehen:
[11, 0, 0, 27, 0, 89]
- Das Programm sollte für `int` Arrays jeder Grösse funktionieren.

Filter — Variante 1

- Der Array der Elemente ist sichtbar, Name der Referenzvariable numbers

```
// filter the array numbers
```

```
for (int i = 0; i < numbers.length; i++) {  
    if (numbers[i] < 0) {  
        numbers[i] = 0;  
    }  
}
```

Filter, Verbesserte Anforderungen

- Schreiben Sie (ausgehend von dem Programm Segment das einen Array filtert) eine `filter` Methode
 - Soll den Array, dessen Werte zu filtern sind, als Parameter akzeptieren.

```
int[] numbers = {11, -42, -5, 27, 0, 89};  
filter(the_numbers);
```

Filter, Verbesserte Anforderungen

- Schreiben Sie (ausgehend von dem Programm Segment das einen Array filtert) eine `filter` Methode

- Soll den Array, dessen Werte zu filtern sind, als Parameter akzeptieren.

```
int[] numbers = {11, -42, -5, 27, 0, 89};  
filter(the_numbers);
```

- Fragen:

1. Wie schreiben wir eine Methode die einen Array als Parameter hat?
2. (Wie) Müssen wir den Inhalt des geänderten Arrays nach Verarbeitung zurück geben?

Array Parameter (Deklaration)

return_type *methodName* (*type* [] *name*)

- Deklaration einer Methode *methodName* die einen Array als Parameter akzeptiert
 - ... und dann auch erfordert
 - mehrere Parameter möglich (wie bisher), auch gewöhnliche (skalare) Parameter
- *return_type*: Typ des Rückgabewertes
 - Die bekannten Regeln (werden demnächst erweitert)
- *type*: Typ der Elemente des Arrays
- *name*: Name des formalen Parameters

Array Parameter (Deklaration)

return_type methodName (*type[] name*)

- Beispiel:

```
// Returns the average of the given array of numbers
int average(int[] numbers) {
    int sum = 0;
    for (int i = 0; i < numbers.length; i++) {
        sum += numbers[i];
    }
    return sum / numbers.length;
}
```


Array Parameter (Deklaration)

*return_type methodName(**type**[] **name**)*

- In der Deklaration steht dass ein Array Parameter gebraucht wird
 - Keine Angabe der Grösse
 - Angabe des Typs der Elemente
 - Obwohl die Länge des Arrays nicht in der Parameterliste erscheint so kann die Methode die Länge jedoch herausfinden
 - length Attribut `name.length`

Aufruf einer Methode mit Array Parameter

- Wie gehabt (Aufruf einer Methode mit Basistyp Parametern)

Array Parameter (Aufruf)

methodName (*arrayName*) ;

- *arrayName*: Name des aktuellen Parameters
- Beispiel:

```
// figure out the average TA IQ
int[] iq = {126, 84, 149, 167, 95};
int avg = average(iq);
System.out.println("Average IQ = " + avg);
```

Array Parameter (Aufruf)

methodName (*arrayName*) ;

Keine [] wenn der
Array übergeben wird!

- *arrayName*: Name des aktuellen Parameters
- Beispiel:

```
// figure out the average TA IQ  
int[] iq = {126, 84, 149, 167, 95};  
int avg = average(iq);  
System.out.println("Average IQ = " + avg);
```

Array Rückgabe (Deklaration)

- Methoden können auch Arrays zurückgeben

return_type [] *methodName* (ParameterListe) {

return_type: Typ der Elemente des zurückgegebenen Arrays

- Wieder verwenden wir ein `return` Statement um anzugeben, was als Ergebnis einer Methode zurückgegeben wird.

Array Rückgabe Beispiel

```
// Returns a new array with two copies of each value.  
// Example: [1, 4, 0, 7] -> [1, 1, 4, 4, 0, 0, 7, 7]
```

```
int[] duplicateElements(int[] numbers) {  
    int[] result = new int[2 * numbers.length];  
    for (int i = 0; i < numbers.length; i++) {  
        result[2 * i] = numbers[i];  
        result[2 * i + 1] = numbers[i];  
    }  
    return result;  
}
```

Array Rückgabewerte

- Die aufrufende Methode sollte den Rückgabewert entgegen nehmen
 - Dafür brauchen wir eine passende Variable

return_type **[]** *name* = *methodName* (parameters) ;

```
int[] iq = {126, 84, 149, 167, 95};  
int[] iqd = duplicateElements(iq);
```

Filter, Verbesserte Anforderungen

- Schreiben Sie (ausgehend von dem Programm Segment das einen Array filtert) eine `filter` Methode

- Soll den Array, dessen Werte zu filtern sind, als Parameter akzeptieren.

```
int[] numbers = {11, -42, -5, 27, 0, 89};  
filter(the_numbers);
```

- Fragen:

1. Wie schreiben wir eine Methode die einen Array als Parameter hat?
2. (Wie) Müssen wir den Inhalt des geänderten Arrays nach Verarbeitung zurück geben?

Filter – Variante 2

- Schreiben Sie (ausgehend von dem Programm Segment das einen Array filtert) eine `filter` Methode.

- Soll den Array, dessen Werte zu filtern sind, als Parameter akzeptieren.

```
int[] numbers = {11, -42, -5, 27, 0, 89};  
int[] filteredNumbers = filter(numbers);
```

- Antwort auf Frage 1:

```
int[] filter (int[] inputArray) {  
    ...  
    return filtered;    // result Array  
}
```

Filter – vollständige Variante 2

```
int[] filter (int [] numbers) {  
    int [] filtered = new int[numbers.length];  
  
    for (int i = 0; i < numbers.length; i++) {  
        if (numbers[i] < 0) {  
            filtered[i] = 0;  
        }  
    }  
    return filtered;  
}
```

Filter – vollständige Variante 2

```
int[] filter (int [] numbers) {  
    int [] filtered = new int[numbers.length];  
  
    for (int i = 0; i < numbers.length; i++) {  
        if (numbers[i] < 0) {  
            filtered[i] = 0;  
        }  
    }  
    return filtered;  
}
```

- Vergisst (leider) die Werte die ≥ 0 sind !

Filter – Variante 3: Lösung mit Einschränkungen

```
int[] filter (int[] numbers) {  
    int[] filtered = new int[numbers.length];  
  
    for (int i = 0; i < numbers.length; i++) {  
        if (numbers[i] < 0) {  
            filtered[i] = 0;  
        } else {  
            filtered[i] = numbers[i];  
        }  
    }  
    return filtered;  
}
```

Diskussion

- Lösung korrekt (gibt uns den gewünschten Array)
- Nehmen wir an der Array hat die Länge 8'500'000 und 20 Elemente < 0
 - Wie viele Zuweisungen werden ausgeführt?
 - Wie viele davon sind absolut notwendig?
- Eine Methode möchte Elemente eines Arrays modifizieren *ohne* die Elemente kopieren zu müssen
 - Ein Beispiel von vielen ...

Array Parameter

- **Wir wollen einer Methode erlauben, einen Array (oder ein Objekt) als Parameter zu erhalten *ohne* dass die Arrayelemente kopiert werden müssen**
 - **Wir sparen Zeit.**
 - **Wir sparen Platz.**
 - **Wir können Veränderungen («Updates») direkt («in place») vornehmen**
- **Dann brauchen wir (evtl) auch keinen Rückgabewert**

Wir erinnern uns: Reference Semantics

- **Reference Semantics: Eine Variable enthält eine Referenz («reference», Verweis) auf einen Array**
 - Wenn eine Referenzvariable als Operand in einer Zuweisung erscheint dann wird der Array nicht kopiert sondern beide Variablen beziehen sich nun (verweisen nun) auf den *selben* Array.
 - Veränderungen der Elemente einer Referenzvariable *bewirken* eine Änderung der Werte der anderen Referenzvariable.

Übergabe eines Array Parameters

- Wie eine Zuweisung

```
int[] localA = new int[3];  
int[] localB = localA;
```

- ```
// irgendwo:
method(localA);
```

```
void method(int[] pA) { ... }
```

- ```
// beim Aufruf: int[] pA = localA;
```

- Wie geht das? Das System kopiert die Information die besagt, wo sich der Array befindet

Reference Semantics

- Wenn wir einen Array als Parameter übergeben kann die aufgerufene Methode den Array verändern *und* die Veränderungen sind für den Aufrufer sichtbar
- Bei Variablen eines Basistyps (`int`, `double`, `boolean`, ...) ist das anders

Filter, vollständige Lösung

```
public static void filter (int [] numbers) {  
  
    for (int i = 0; i < numbers.length; i++) {  
        if (numbers[i] < 0) {  
            numbers[i] = 0;  
        }  
    }  
}
```

Filter

```
public static void filter (int[] numbers) {  
  
    for (int i = 0; i < numbers.length; i++) {  
        if (numbers[i] < 0) {  
            numbers[i] = 0;  
        }  
    }  
}  
  
int[] data = {11, -42, -5, 27, 0, 89};  
filter(data);  
  
// Nach Ende von filter:  data ist Array mit Werten  
// [11, 0, 0, 27, 0, 89]
```

Array als Parameter

- Wenn ein Array als Parameter übergeben wird, dann wird der Array *nicht* kopiert. Der Parameter verweist auf den selben (ursprünglichen) Array.
 - Wenn der Array modifiziert wird, dann wird auch der Array, den die aufrufende Methode sieht, verändert.

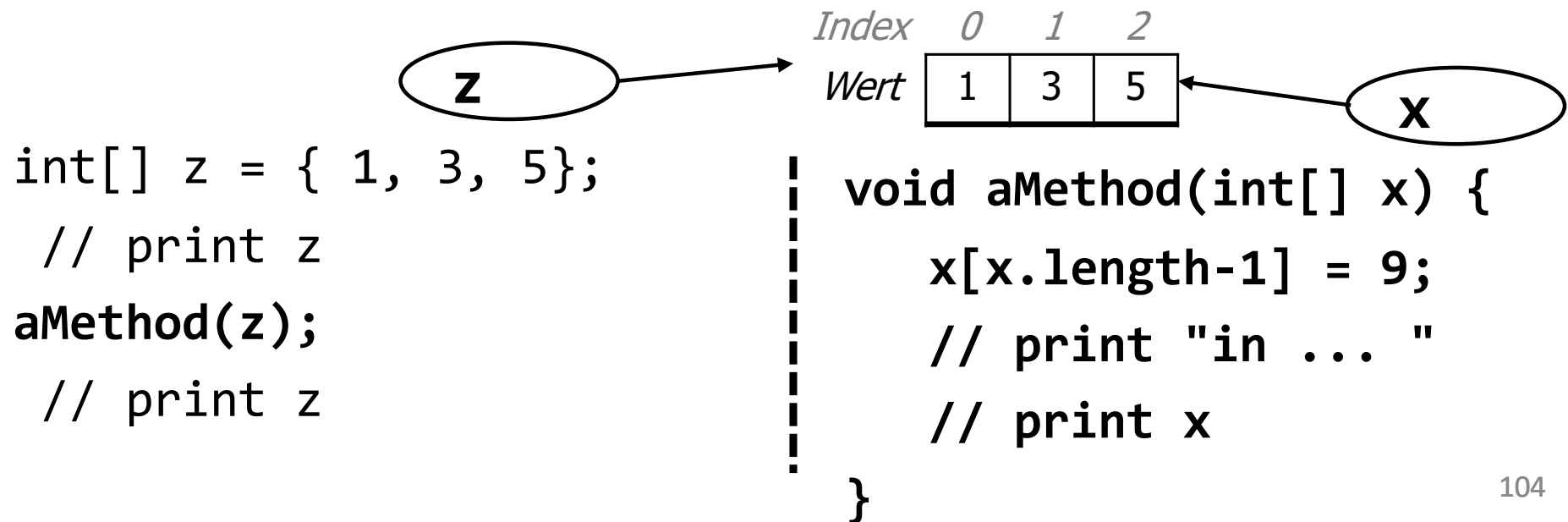
Array Parameter verwenden Reference Semantics

- Wir sagen dass Array Parameter «*by reference*» übergeben werden.
 - Veränderungen in einer Methode werden von der aufrufenden Methode gesehen.

Reference Semantics

Output: {1,3,5}
in ... {1,3,9}

- Wird eine **Array Variable** (als Argument) **übergeben** bezieht sich der Parameter in der aufgerufenen Methode auf den selben Array
 - Wie eine Zuweisung von einer Referenzvariablen an eine andere



Arrays und Output

- Auch für Arrays gibt es eine Default Darstellung

```
int[] myArray = new int[5];
```

```
System.out.println(myArray);
```

Arrays und Output

- Auch für Arrays gibt es eine Default Darstellung

```
int[] myArray = new int[5];
```

```
System.out.println(myArray);
```

Output

I@2a139a55

- Nur leider hilft uns diese nicht weiter

Die Klasse Arrays

- Die Klasse `Arrays` in der Bibliothek `java.util` enthält einige Methoden, die wir in einer `static` Methode aufrufen können

Method name	Description
<code>binarySearch(array, value)</code>	returns the index of the given value in a <i>sorted</i> array (or <code>< 0</code> if not found)
<code>copyOf(array, length)</code>	returns a new copy of an array
<code>equals(array1, array2)</code>	returns <code>true</code> if the two arrays contain same elements in the same order
<code>fill(array, value)</code>	sets every element to the given value
<code>sort(array)</code>	arranges the elements into sorted order
<code>toString(array)</code>	returns a string representing the array, such as "[42, -7, 1, 15]"

Die Arrays Klasse

- **Syntax:** `Arrays.methodName(parameters)`
 - *methodName*: Service den wir brauchen
 - *parameter(s)* : Ein oder mehr Parameter
- **Beispiel**

```
import java.util.Arrays;

int[] a1 = {42, -7, 1, 15};
int[] a2 = {42, -7, 1, 15};
if (Arrays.equals(a1, a2)) {
    ...
}
```

Besonders praktisch: Arrays.toString

- Die Klasse `Arrays` in der Bibliothek `java.util` enthält einige Methoden, die wir in einer `static` Methode aufrufen können

Method name	Description
<code>binarySearch(array, value)</code>	returns the index of the given value in a <i>sorted</i> array (or <code>< 0</code> if not found)
<code>copyOf(array, length)</code>	returns a new copy of an array
<code>equals(array1, array2)</code>	returns <code>true</code> if the two arrays contain same elements in the same order
<code>fill(array, value)</code>	sets every element to the given value
<code>sort(array)</code>	arranges the elements into sorted order
<code>toString(array)</code>	returns a string representing the array, such as "[42, -7, 1, 15]"

Arrays.toString

- **Arrays.toString** nimmt einen Array als Parameter und liefert einen String mit den Array Elementen

```
int[] e = {0, 2, 4, 6, 8};  
e[1] = e[3] + e[4];  
System.out.println("e is " + Arrays.toString(e));
```

Output:

```
e is [0, 14, 4, 6, 8]
```

- **Einfacher Weg String mit den Werten des Arrays zu erhalten**