

Programmierkurs Java

UE 7 - Arrays

Dr.-Ing. Dietrich Boles

- Motivation
- Anmerkungen
- Array-Erzeugung
- Array-Variablen
- Array-Elemente
- Array-Zerstörung
- for-each-Schleife
- Mehrdimensionale Arrays
- Beispiele
- Zusammenfassung

- bisher: einfache Datentypen (`int`, `float`, `char`, ...)
- Benutzung: Speicherung von "einfachen" Werten in "einfachen" Variablen
- häufig: besonderer Zusammenhang zwischen bestimmten Werten
- Beispiele:
 - Menge von Zahlen, die zu sortieren sind
 - Verwaltung von Personendaten (Name, Alter, Adresse, ...)
- Strukturierte Datentypen: Zusammenfassung mehrerer Werte zu einer Einheit
 - Arrays: Werte desselben Typs
 - Objekte/Klassen: Werte mit u.U. unterschiedlichem Typ

Definition:

Ein **Array** repräsentiert ein **homogenes kartesisches Produkt** zur Aufnahme mehrere Werte (**Elemente**) des **gleichen Typs** (genannt **Elementtyp**), wobei auf die Elemente mit Hilfe eines **Index** zugegriffen wird.

Synonyme:

Feld, Tupel, Matrix, Vektor, Tabelle

	0	1	2	3	4
0	2	-2	67	2	90
1	33	3	-6	5	2
2	4	2	2	78	93

- 3 x 5 - Matrix (2-dimensional)
- Elementtyp: **int**
- 3 Zeilen, 5 Spalten
- Nummerierung beginnt bei 0
 - `m[0][0] == 2`
 - `m[0][2] == 67`
 - `m[5][5] → Laufzeitfehler`

- Alternative zu Arrays: `int v1, v2, ..., v1000; ??????`
 - was ist bei 100.000 Elementen?
 - es existiert kein Zusammenhang zur Laufzeit (Index)!
- Elementtyp ist fest
- Dimension ist fest
- 1-, 2-, ..., n-dimensionale Arrays erlaubt
- Zugriff über n-dimensionalen Index
- random-access-Zugriff (Dateien auslesen → sequentiell)
- Arrays sind in Java Referenzdatentypen (wie Objekte)
- Arrays werden in Java dynamisch erzeugt
- Arrays werden in Java automatisch gelöscht, wenn sie nicht mehr benutzt werden (können)

```
<array-creation> ::= "new" <Typ>  
                    "[" <int-Ausdruck> "]"  
                    { "[" <int-Ausdruck> "]" }
```

Nebenbedingung/Anmerkungen:

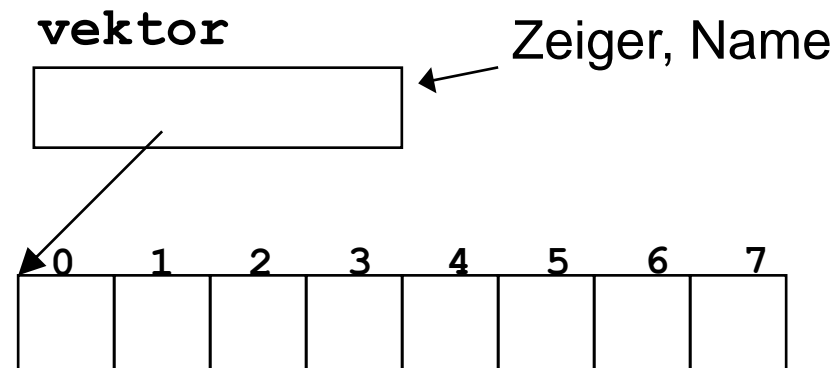
- Reserviert Speicherplatz auf dem Heap
- int-Ausdrücke bestimmen die Anzahl an Elementen der jeweiligen Dimension
- Array-Elemente werden implizit initialisiert mit dem Default-Wert des Elementtyps
- Array-Größe kann nachträglich nicht verändert werden

Beispiele:

```
new int[8]  
    // Reservierung von 8 Speicherplätzen für int-Variablen  
int i = 3;  
new float[i*4][2]  
    // Reservierung von 12*2 Speicherplätzen für float-Variablen
```

- Unterscheidung zwischen den **eigentlichen Arrays** und den **Array-Variablen**
- Array-Variablen speichern die Adresse des eigentlichen Arrays: Referenz auf das Array (**Array-Variablen sind Referenzvariablen!**)
- Zugriff auf ein Array erfolgt immer über eine Array-Variable

```
vektor = new int[8];
```



```
<arrayvar-def> ::= <Typ>                ← Elementtyp  
                 "[" "]" { "[" "]" }      ← Dimension  
                 <Bezeichner> { "," <Bezeichner> }  
                 ";"
```

Beispiele:

```
int[] vektor1;           // ElemTyp: int,    Dim: 1  
float[][] matrix1;     // ElemTyp: float, Dim: 2  
char[] buchstaben;     // ElemTyp: char,   Dim: 1  
  
double[][][] quader1, quader2;  
                    // ElemTyp: double, Dim: 3  
  
char ziffern[];        // Definitionsalternative!
```


- bei der Definition einer Array-Variablen wird **kein** Speicherplatz für das Feld selbst angelegt!!!!
- vielmehr wird ein Speicherbereich reserviert, dem mittels des **new**-Operators Adressen zugewiesen werden können (→ Zeiger, Felderzeugung)
- man kann die Adressen weder auslesen noch auf den Adressen Operationen ausführen (wie bspw. in C oder C++); Ausnahme: `==` und `!=`
- Default-Wert einer Array-Variablen: `null` (Literal)
- explizite Initialisierung: `int[] zahlen = null;`
- Konstrukte wie `int[]` oder `double[][]` sind Typen

Nebenbedingung/Anmerkungen:

- Elementtyp der Array-Variablen und Elementtyp des Arrays müssen gleich sein!
- Dimension der Array-Variablen und Dimension des Arrays müssen gleich sein!

Beispiele:

```
int[] vektor1;
```

```
vektor1 = new int[8];
```

```
int i = 3;
```

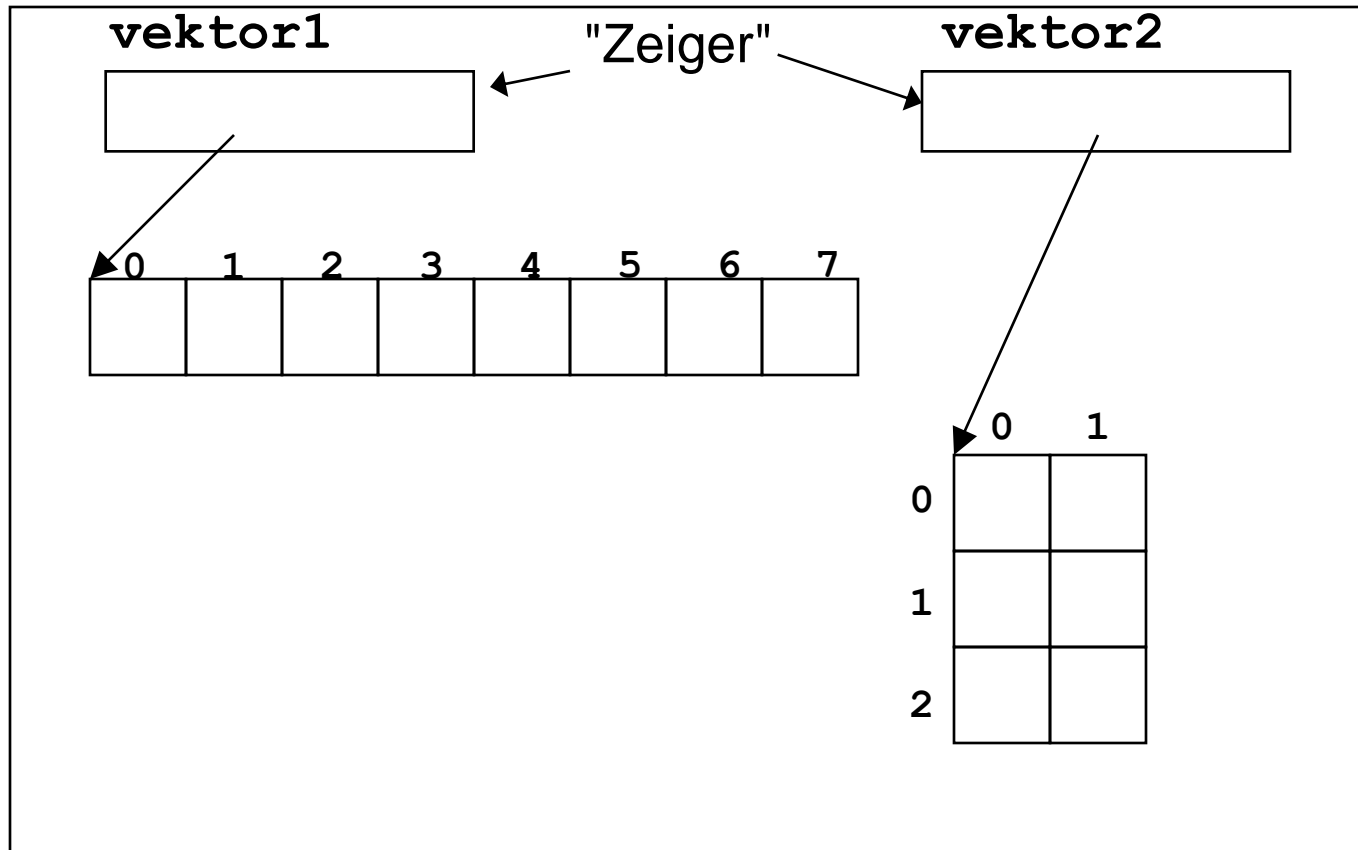
```
float[][] vektor2 = new float[i][2];
```

```
char[] alphabet = new double[26];           // Fehler
```

```
boolean[] aussagen = new boolean[2][3];    // Fehler
```

```
int[] vektor1 = new int[8];  
float[][] vektor2 = new float[3][2];
```

Speicher



```

<array-access> ::= <Bezeichner>
                    "[" <int-Ausdruck> "]"
                    { "[" <int-Ausdruck> "]" }
    
```

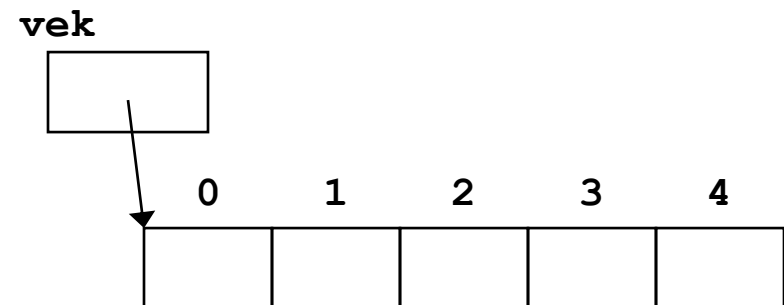
Nebenbedingungen / Anmerkungen:

- Bezeichner muss gültige Array-Variable sein
- Array-Variable muss ein Array referenzieren (→ Laufzeitfehler!)
- int-Ausdruck muss Wert zwischen 0 und der Anzahl der Elemente - 1 der jeweiligen Dimension liefern (→ Laufzeitfehler!)
- Zugriff auf Array-Elemente nennt man **Indexierung**

Beispiele:

```

int[] vek = new int[5];
vek[0] = -23;
vek[1] = vek[0] + 25;
vek[vek[1]] = -4;
vek[5] = 56; // Laufzeitfehler!
    
```

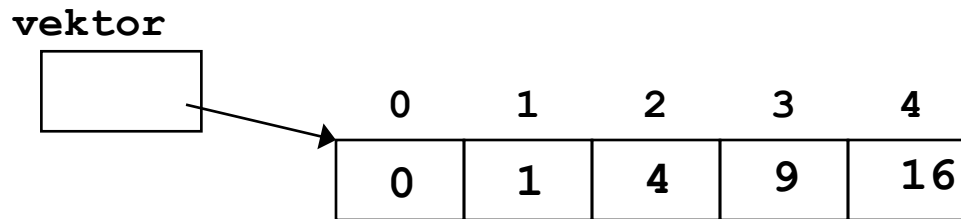


Array-Elemente / Initialisierung (1)

- Default-Initialisierung der Elemente mit Default-Wert des Elementtyps

Explizite Initialisierung:

```
int[] vektor = new int[5];
for (int i = 0; i < vektor.length; i++)
    vektor[i] = i*i;
```

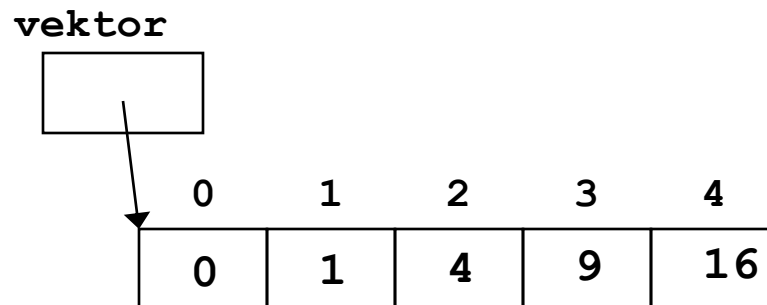


length:

- liefert Anzahl an Elementen (der angegebenen Dimension)
- nur lesbar!

Implizite Erzeugung und Initialisierung:

```
int i = 3;  
int[] vektor = {0, 1, 4, i*i, 16};
```

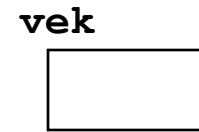


- erzeugt Array mit entsprechender Elementanzahl
- initialisiert die Elemente
- Initialisierungswerte können durch Ausdrücke des entsprechenden Elementtyps gebildet werden (häufig Literale)

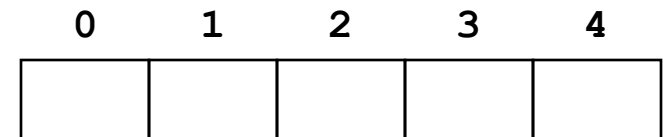
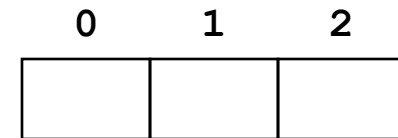
Array-Zerstörung

- Java: automatisches Garbage-Collection!
- Kein `delete`-Operator
- Speicherplatz wird automatisch freigegeben, wenn nicht mehr auf Heap-Speicherplatz referenziert ("gezeigt") wird

Beispiel:

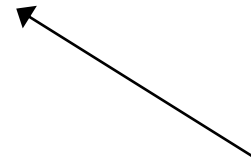


```
int[] vek = new int[3];
vek[0] = 4;
// ...
vek = new int[5];
vek[2] = 45;
vek[4] = -5;
```



Beispiel: Berechnen der Summe eines Zahlen-Arrays

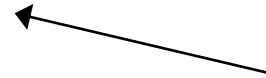
```
public static void main(String[] args) {  
    int[] zahlen = {2, 4, 6, 5, 1, 2};  
  
    int sum = 0;  
    for (int i = 0; i < zahlen.length; i++) {  
        sum += zahlen[i];  
    }  
  
    System.out.println(sum);  
}
```



Durchlauf über alle Array-Elemente

Beispiel: Berechnen der Summe eines Zahlen-Arrays

```
public static void main(String[] args) {  
    int[] zahlen = {2, 4, 6, 5, 1, 2};  
  
    int sum = 0;  
    for (int zahl : zahlen) {  
        sum += zahl;  
    }  
  
    System.out.println(sum);  
}
```



Durchlauf über alle Array-Elemente
beginnend bei 0

```
for (<variable> : <array>) {  
    <variable> nimmt Wert des  
    jeweiligen Elementes an  
}
```

Normalfall: Anzahl an Elementen pro Dimension ist identisch

```
double[][] matrix = new double[2][3];  
for (int z = 0; z < 2; z++)  
    for (int s = 0; s < 3; s++)  
        matrix[z][s] = z + s;
```

matrix

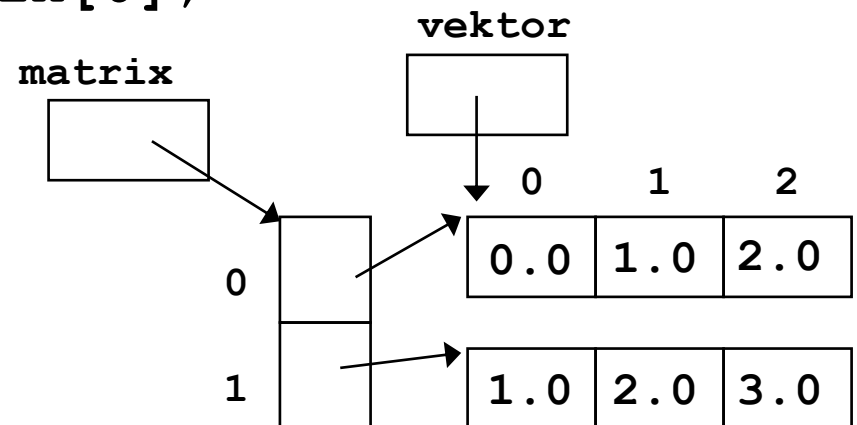
	0	1	2
0	0.0	1.0	2.0
1	1.0	2.0	3.0

Mehrdimensionale Arrays (2)

- Interne Realisierung mehrdimensionaler Arrays: Array von Array!

```
double[][] matrix = new double[2][3];
for (int z = 0; z < 2; z++)
    for (int s = 0; s < 3; s++)
        matrix[z][s] = z + s;
```

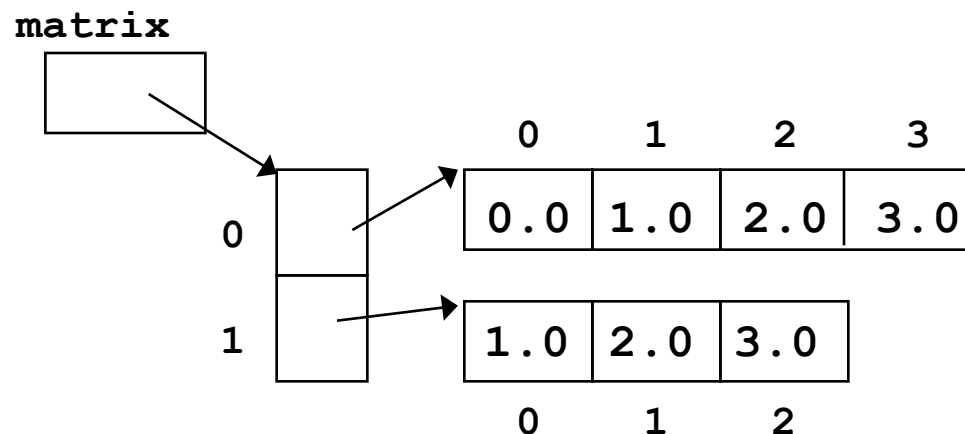
```
double[] vektor = matrix[0];
vektor[1] = 3.4;
```



Mehrdimensionale Arrays (3)

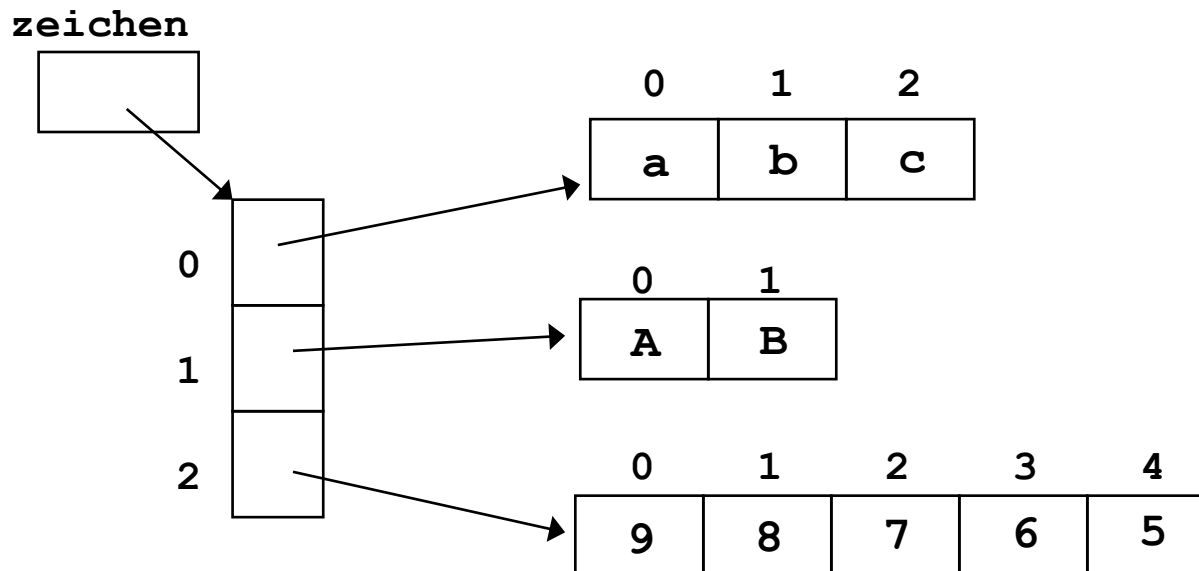
Möglich: Anzahl an Elementen pro Dimension ist unterschiedlich

```
double[][] matrix = new double[2][];
matrix[0] = new double[4];
matrix[1] = new double[3];
for (int z=0; z < matrix.length; z++)
    for (int s=0; s < matrix[z].length; s++)
        matrix[z][s] = z + s;
```

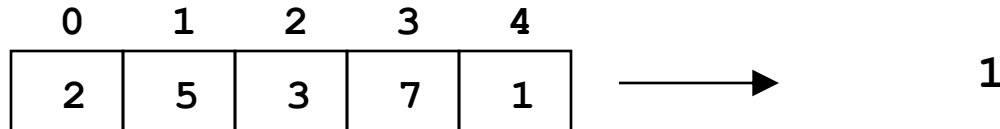


Implizite Erzeugung und Initialisierung:

```
char[][] zeichen = { {'a', 'b', 'c'},  
                    {'A', 'B'},  
                    {'9', '8', '7', '6', '5'}  
                    };
```



Minimum-Suche:



```
public static void main(String[] args) {  
    int[] vektor = {2,5,3,7,1};  
  
    int min = vektor[0];  
    for (int i = 1; i < vektor.length; i++) {  
        if (vektor[i] < min)  
            min = vektor[i];  
    }  
  
    System.out.println(min);  
}
```

Sortierung von Zahlen:

```
int[] zahlen = { 2, 5, 3, 7, 1 };
```

```
boolean tausch = false;
```

```
do {
```

```
    tausch = false;
```

```
    for (int i = 0; i < zahlen.length - 1; i++) {
```

```
        if (zahlen[i] > zahlen[i + 1]) {
```

```
            int speicher = zahlen[i];
```

```
            zahlen[i] = zahlen[i + 1];
```

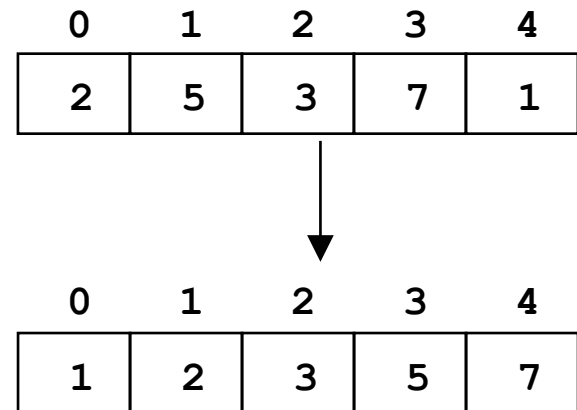
```
            zahlen[i + 1] = speicher;
```

```
            tausch = true;
```

```
        }
```

```
    }
```

```
} while (tausch);
```



Bubblesort-Algorithmus

- Array: Zusammenfassung mehrerer Variablen desselben Typs zu einer Einheit
- Array-Variable: Variable zur Referenzierung eines Arrays
- Zugriff auf die einzelnen Variablen (Elemente) via Array-Variable über einen Index
- Arrays müssen erzeugt werden
- Arrays werden automatisch gelöscht