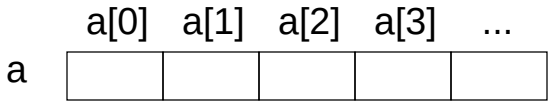
5. Arrays

Eindimensionale Arrays

Array = Tabelle gleichartiger Elemente



- Name *a* bezeichnet das gesamte Array
- Elemente werden über Indizes angesprochen (z.B. *a*[3])
- Indizierung beginnt bei 0
- Elemente sind "namenlose" Variablen

Deklaration

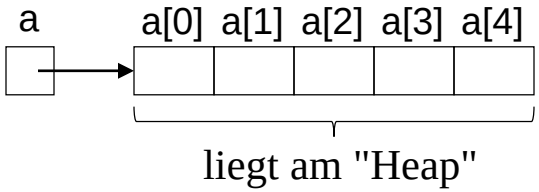
```
int[] a;
float[] b;
```

- deklariert ein Array namens *a* (bzw. *b*)
- seine Elemente sind vom Typ *int* (bzw. *float*)
- seine Länge ist noch unbekannt

Erzeugung

```
a = new int[5];
b = new float[10];
```

- legt ein neues *int*-Array mit 5 Elementen an
- weist seine Adresse *a* zu



Array-Variablen enthalten in Java Zeiger auf Arrays!
(Zeiger = Speicheradresse)

Arbeiten mit Arrays

Zugriff auf Arrayelemente

```
a[3] = 0;
a[2*i+1] = a[i] * 3;
```

- Arrayelemente werden wie Variablen benutzt
- Index muss ein ganzzahliger Ausdruck sein
- Laufzeitfehler, falls Array noch nicht erzeugt wurde
- Laufzeitfehler, falls Index < 0 oder ³ Arraylänge

Arraylänge abfragen

```
int len = a.length;
```

- *length* ist ein Standardoperator, der auf alle Arrays angewendet werden kann.
- Liefert Anzahl der Elemente (hier 5).

Beispiele

```
for (int i = 0; i < a.length; i++) {
    a[i] = In.readInt();
}
```

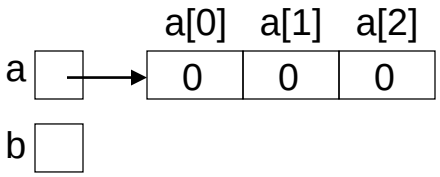
Array einlesen

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

Arrayelemente aufaddieren

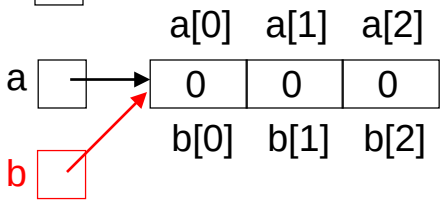
Arrayzuweisung

```
int[] a, b;
a = new int[3];
```



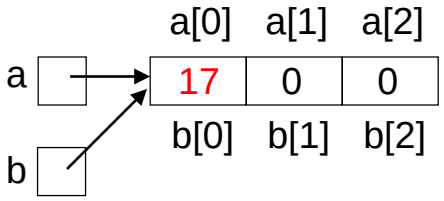
Arrayelemente werden in Java standardmäßig mit 0 initialisiert

```
b = a;
```



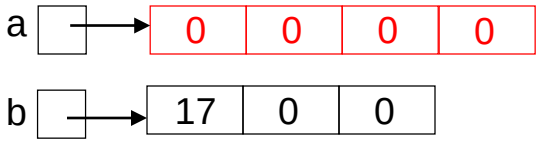
b bekommt denselben Wert wie *a*.
Arrayzuweisung ist in Java Zeigerzuweisung!

```
a[0] = 17;
```



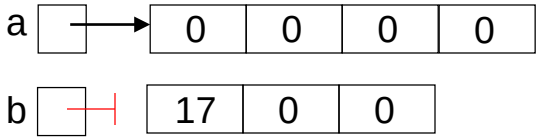
ändert in diesem Fall auch b[0]

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



a zeigt jetzt auf neues Array.

```
b = null;
```



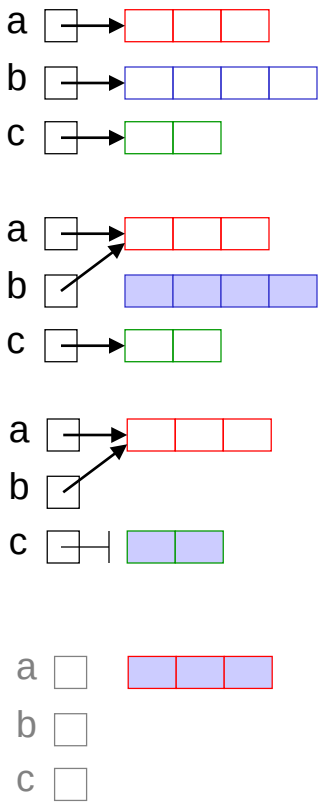
null: Spezialwert, der auf kein Objekt zeigt; kann jeder Arrayvariablen zugewiesen werden

Freigeben von Arrayspeicher

Garbage Collection (Automatische Speicherbereinigung)

Objekte, auf die kein Zeiger mehr verweist, werden automatisch eingesammelt.
Ihr Speicher steht für neue Objekte zur Verfügung (wird dem Heap zurückgegeben)

```
static void P() {
    int[] a = new int[3];
    int[] b = new int[4];
    int[] c = new int[2];
    ...
    ...
    b = a;
    ...
    ...
    c = null;
    ...
    ...
    ...
}
```



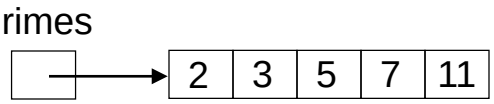
kein Zeiger mehr auf dieses Objekt
Þ wird eingesammelt

kein Zeiger mehr auf dieses Objekt
Þ wird eingesammelt

Am Methodenende werden lokale Variablen freigegeben Þ Zeiger *a*, *b*, *c* fallen weg
Þ Objekt wird eingesammelt

Initialisieren von Arrays

```
int[] primes = {2, 3, 5, 7, 11};
```



identisch zu

```
int[] primes = new int[5];
primes[0] = 2;
primes[1] = 3;
primes[2] = 5;
primes[3] = 7;
primes[4] = 11;
```

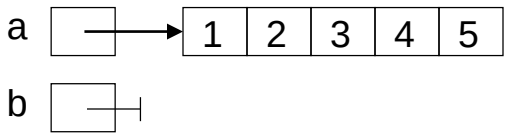
Initialisierung kann auch bei der Erzeugung erfolgen

```
int[] primes;
...
primes = new int[] {2, 3, 5, 7, 11};
```

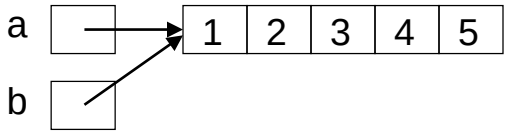
keine Länge angegeben, dafür Initialwerte

Kopieren von Arrays

```
int[] a = {1, 2, 3, 4, 5};
int[] b;
```

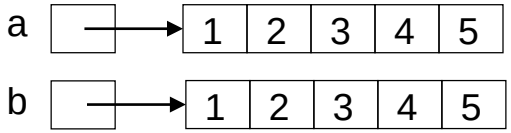



```
b = a;
```



kein Kopieren,
nur Zeigerzuweisung

```
b = (int[]) a.clone();
```



Typumwandlung nötig, da *clone* etwas vom Typ *Object*[] liefert

5. Arrays

Foreach-Schleife

Vereinfachtes Durchlaufen eines Arrays

Variable durchläuft dabei alle Datenwerte des Arrays

Beispiele

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

a | 2 | 3 | 5 | 7 |

```
double[] b = {3.7, 4.2, 7.8};
```

b | 3.7 | 4.2 | 7.8 |

Aufsummieren des Arrays

Gewöhnliche For-Schleife

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

```
double sum = 0.0;
for (int i = 0; i < b.length; i++) {
    sum = sum + b[i];
}
```

Foreach-Schleife

```
int sum = 0;
for (int val: a) {
    sum = sum + val;
}
```

sprich: for each *int val* in *a*

val durchläuft die Werte: 2, 3, 5, 7

```
double sum = 0.0;
for (double val: b) {
    sum = sum + val;
}
```

for each *double val* in *b*

Syntax

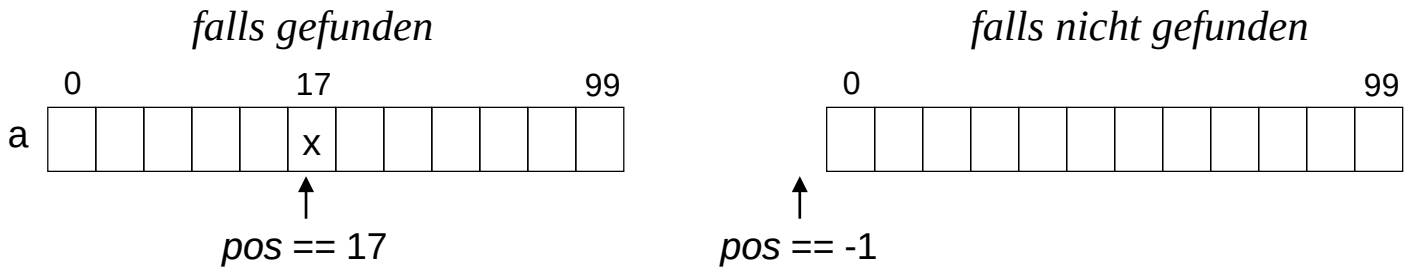
```
ForeachStatement = "for" "(" Type ident ":" ident ")" Statement.
```

Laufvariable — Array

5. Arrays

Beispiel: sequentielles Suchen

Suchen eines Werts *x* in einem Array




```
int[] a = new int[100];
...      // Annahme: a mit Zahlen gefüllt

int pos = a.length - 1;
while ( pos >= 0 && a[pos] != x ) pos--;
// pos == -1 || a[pos] == x
```

gewünschtes Ergebnis

Primzahlenberechnung: Sieb des Erathostenes

1. "Sieb" wird mit den natürlichen Zahlen ab 2 gefüllt

2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, ...

2. Erste Zahl im Sieb ist Primzahl. Entferne sie und alle ihre Vielfachen

2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, ...

3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, ...

3. Wiederhole Schritt 2

3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, ...

5, 7, 11, 13, 17, 19, 23, 25, ...

... Wiederhole Schritt 2

5, 7, 11, 13, 17, 19, 23, 25, ...

7, 11, 13, 17, 19, 23, ...

Implementierung

Sieb = *boolean*-Array, Zahl *i* im Sieb Û sieve[i] == true

0	1	2	3	4	5	6	7	8	9	
false	false	true	true	true	true	true	true	true	true	...

Zahl *i* entfernen: *sieve*[*i*] = *false*

0	1	2	3	4	5	6	7	8	9	
false	false	false	true	false	true	false	true	false	true	...

```
// print primes from 2 to max
boolean[] sieve = new boolean[max + 1];
for (int i = 2; i <= max; i++) sieve[i] = true;
for (int i = 2; i <= max; ) {
   Out.print(i + " ");  // i is prime
   for (int j = i; j <= max; j = j + i) sieve[j] = false;
   while (i <= max && !sieve[i]) i++;
}
```

Beispiel: Monatstage berechnen

Bisher mit Switch-Anweisung gelöst

```
switch (month) {
   case 1: case 3: case 5: case 7: case 8: case 10: case 12:
      days = 31; break;
   case 4: case 6: case 9: case 11:
      days = 30; break;
   case 2:
      days = 28;
}
```

Besser mit Tabelle

Dummy

```
int[] days = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
...
int d = days[month];
```

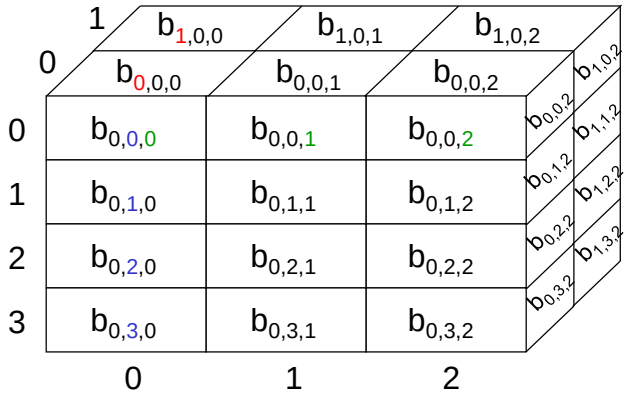
5. Arrays

Mehrdimensionale Arrays

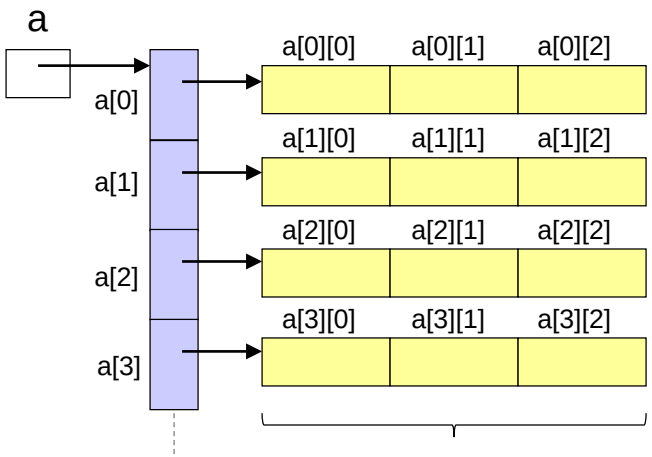
Zweidimensionales Array ° Matrix

	0	1	2
0	$a_{0,0}$	$a_{0,1}$	$a_{0,2}$
1	$a_{1,0}$	$a_{1,1}$	$a_{1,2}$
2	$a_{2,0}$	$a_{2,1}$	$a_{2,2}$
3	$a_{3,0}$	$a_{3,1}$	$a_{3,2}$

Dreidimensionales Array ° Cubus



In Java als Array von Arrays implementiert



Array von Zeilen jede Zeile: Array von Spalten

Deklaration und Erzeugung

```
int[][] a;
a = new int[4][3];
```

```
int[][][] b;
b = new int[2][4][3];
```

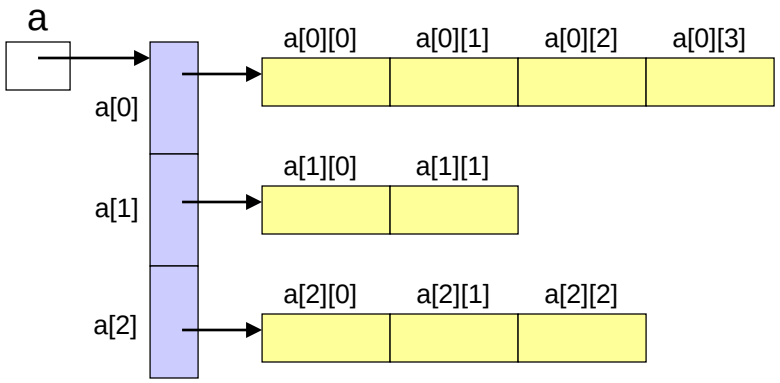
Zugriff

```
a[i][j] = a[i][j+1];
```

```
b[i][j][k] = a[i][j][k+1];
```

Mehrdimensionale Arrays

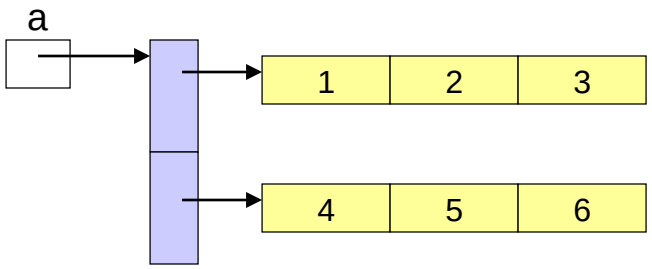
Zeilen können unterschiedlich lang sein (obwohl selten sinnvoll)




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

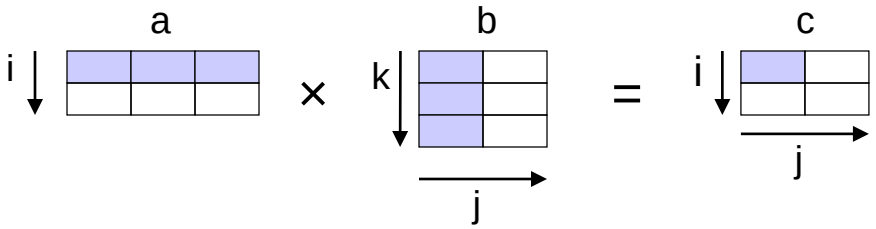
Initialisierung

```
int[][] a = {{1, 2, 3}, {4, 5, 6}};
```



5. Arrays

Beispiel: Matrixmultiplikation



$c_{0,0} = a_{0,0}*b_{0,0} + a_{0,1}*b_{1,0} + a_{0,2}*b_{2,0}$

```
float[][] a = new float[2][3]; ... // mit Werten füllen
float[][] b = new float[3][2]; ... // mit Werten füllen
...
float[][] c = new float[a.length][b[0].length];
for (int i = 0; i < a.length; i++) {
    for (int j = 0; j < b[0].length; j++) {
        float sum = 0;
        for (int k = 0; k < b.length; k++)
            sum += a[i][k] * b[k][j];
        c[i][j] = sum;
    }
}
```