• OA3 wird nicht gewertet.

Erinnerung:

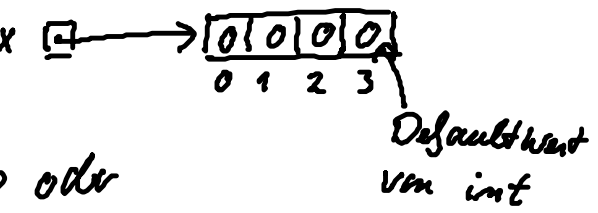
Arrays

(≙ Vektoren in Mathematik)

- Standard Datenstrukter, gibt's in allen Progr. sprachen
- lineare Datenstrukter
- feste Komponentenzahl
- direkter Zugriff mittels Indizes
- homogener Grundtyp (alle Typen möglich)

Arrays sind Referenztypen, d.h. Erzeugung mit "new"

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



alternativ, falls Komponententyp Standardtyp oder String ist, auch Erzeugung mittels Initialisierungsliste

```
double[] y = { 1.5, -2.33, -16.5};
```

Länge eines Arrays muss erst zur Laufzeit festgelegt werden:

```
int[] x;
...
```

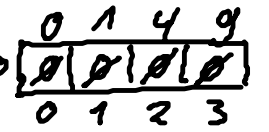
```
// erforderliche Länge n ausrechnen
x = new int[n]; // erst jetzt Länge festlegen
```

Methoden können Arraytypen zurückgeben.

Bsp: Erzeugung der ersten n Quadratzahlen:

2.B. beim Aufruf squares(4)

```
public int[] squares(int n) {
    int[] sq = new int[n];
    for(int i=0; i<n; i++) {
        sq[i] = i*i;
    }
    return sq;
}
```

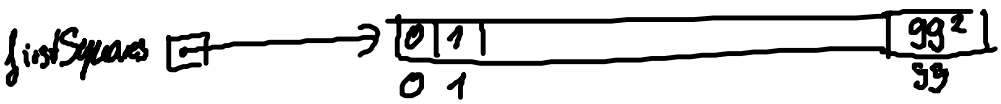


← Referenz auf sq wird zurückgegeben

Verwendung dieser Methode z.B. mit

```
int[] firstSquares = squares(100);
```

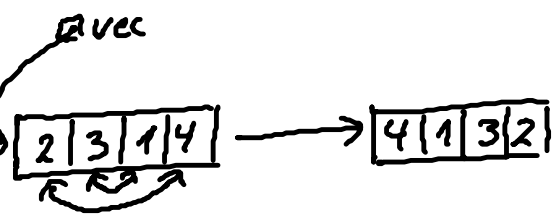
legt den Array an.



Beispiele für Methoden mit Arrays

Umdrehen eines Arrays direkt im Array:

```
int[] vec; // sei bereits initialisiert
```



Idee: die äußeren Komponenten vertauschen, dann 1 „nach innen" gehen, und das wiederholen bis man in der Mitte angekommen ist.

↑ vec.length/2

```
for (int i=0; i<vec.length/2; i++) {
    int tmp = vec[i];
    vec[i] = vec[vec.length-1-i];
    vec[vec.length-1-i] = tmp;
}
```

Vertauschung der Werte der Komponenten i und vec.length − 1 − i.

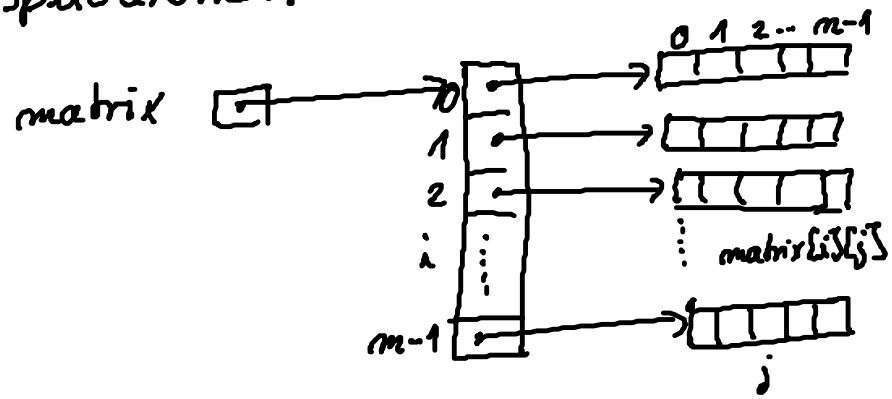
Arrays können als Komponenten jeden beliebigen Typ haben, insbesondere auch wieder Arrays
⟹ mehrdimensionale Arrays

```
double[][] matrix = new double[m][n];
```

double[] — Komponententyp
[] — Arraytyp

Zwei-dim. Array modelliert Matrizen aus der Mathematik

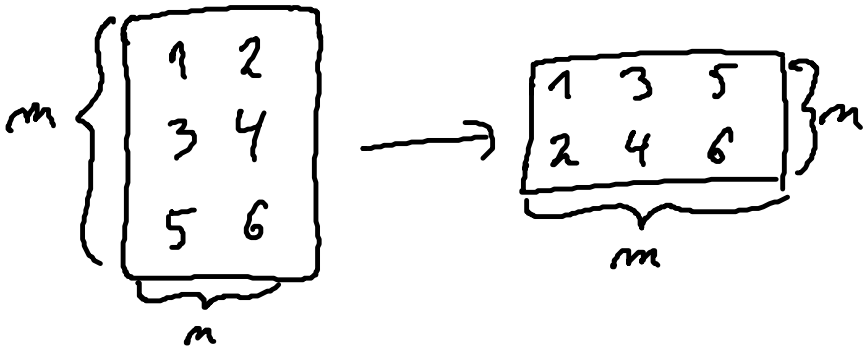
Speicherbild:



ACHTUNG:
Zuerst Komponentenzahl des äußeren Arrays.
„Zugeständnis an Mathematik"

$$\stackrel{\wedge}{=} \begin{pmatrix} a_{00} & a_{01} & \cdots & a_{0,n-1} \\ a_{10} & & & \vdots \\ \vdots & & a_{ij} & \vdots \\ a_{m-1,0} & \cdots & & a_{m-1,n-1} \end{pmatrix}$$

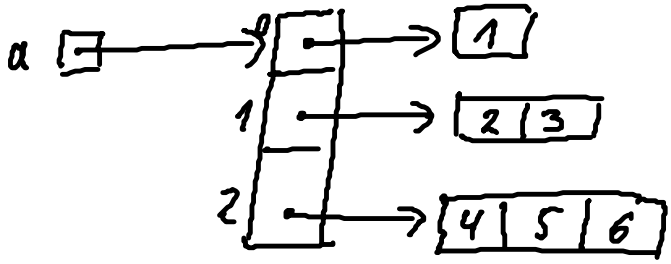
Beispiel: Matrix transponieren




```
double[][] transpose (double[][] matrix) {
    double[][] transpose = new double[matrix[0].length]
                                     [matrix.length];
    for (int i=0; i < matrix[0].length; i++) {
        for (int j=0; j < matrix.length; j++) {
            transpose[i][j] = matrix[j][i];
        }
    }
    return transpose;
}
```

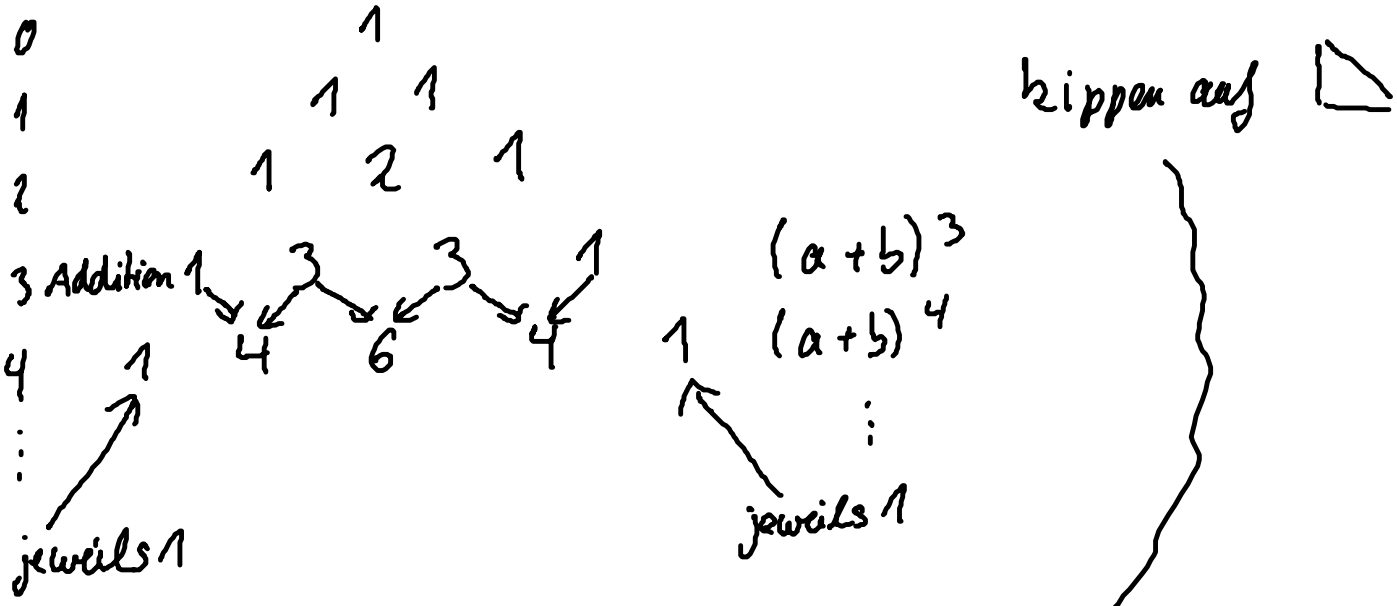
Bemerkung: Komponenten eines mehrdimensionalen Arrays müssen nicht alle gleich sein.

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



Beispiel: Pascalsches Dreiecke ← Darstellung der Koeffizienten bei Ausmultiplikation von $(a+b)^n$

n



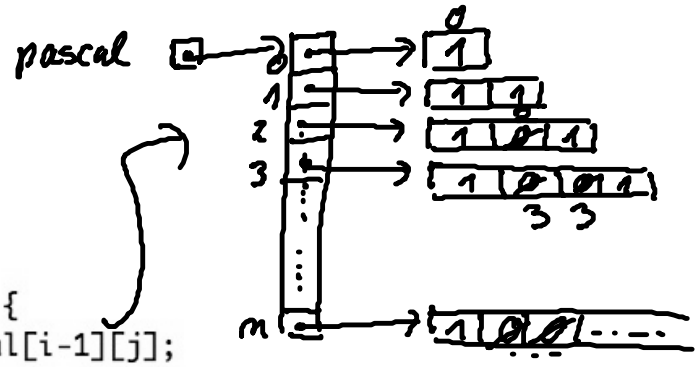
Asymmetrisches Pascalsches Dreieck bis zur Reihe 12:

```
Zeilennr:

   0      1
   1      1   1
   2      1   2   1
   3      1   3   3   1
   4      1   4   6   4   1
   5      1   5  10  10   5   1
   6      1   6  15  20  15   6   1
   7      1   7  21  35  35  21   7   1
   8      1   8  28  56  70  56  28   8   1
   9      1   9  36  84 126 126  84  36   9   1
  10      1  10  45 120 210 252 210 120  45  10   1
  11      1  11  55 165 330 462 462 330 165  55  11   1
  12      1  12  66 220 495 792 924 792 495 220  66  12   1
```

Methode zur Erzeugung des Pascalschen Dreiecks bis Zeile n:

```
public int[][] pascalTriangle(int n) {
    int[][] pascal = new int[n+1][];
    for (int i=0; i<pascal.length; i++) {
        pascal[i] = new int[i+1];
        pascal[i][0] = 1;
        pascal[i][i] = 1;
    }
    for (int i=2; i<pascal.length; i++) {
        for (int j=1; j<pascal[i].length-1; j++) {
            pascal[i][j] = pascal[i-1][j-1]+pascal[i-1][j];
        }
    }
    return pascal;
}
```



§6. Algorithmen auf Arrays

6.1. Sequentielle und Binäre Suche

6.2. Matrizenmultiplikation

6.3. Kürzeste Wege in Graphen