Vorlesung an der

Berufsakademie Oldenburg

Unterrichtseinheit 17

Arrays

Dr. Dietrich Boles

Gliederung

- Strukturierte Datentypen
- Felder
  - Motivation
  - Anmerkungen
  - Feldvariablen
  - Felderzeugung
  - Feldelemente
  - Feldzerstörung
  - „neue“ for-Schleife
  - Mehrdimensionale Felder
- Beispiele
- Aufgaben

Strukturierte Datentypen

- bisher: einfache Datentypen (`int`, `float`, `char`, ...)
- Benutzung: Speicherung von "einfachen" Daten in "einfachen" Variablen

- häufig: besonderer Zusammenhang zwischen bestimmten Daten
- Beispiele:
  - Menge von Zahlen, die zu sortieren sind
  - Verwaltung von Personendaten (Name, Alter, Adresse, ...)

- Strukturierte Datentypen: Zusammenfassung mehrerer Daten zu einer Einheit
  - Felder: Daten desselben Typs
  - Verbunde/Klassen: Daten mit u.U. unterschiedlichem Typ

Motivation

Definition:

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

Synomyme:

> 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
```

Anmerkungen

- Alternative zu Arrays: **`int v1, v2, v3, ..., 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)

Feldvariablen / Definition (1)

```
<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!
```

Feldvariablen / Definition (2)

- Unterscheidung zwischen den **eigentlichen Felder** und den **Feldvariablen**
- Feldvariablen speichern die Adresse des eigentlichen Feldes: Referenz auf das Feld (**Feldvariablen sind Referenzvariablen**!)
- bei der Definition einer Feldvariablen wird **kein** Speicherplatz für das Feld selbst angelegt!!!!!
- vielmehr wird ein Speicherbereich reserviert, dem mittels des **new**-Operators Adressen zugewiesen werden können (→ Zeiger, Felderzeugung)
- Felder werden auf dem Heap angelegt
- man kann die Adressen weder auslesen noch auf den Adressen Operationen ausführen (wie bspw. in C oder C++)
- Default-Initialisierung einer Feldvariablen mit dem Literal `null`
- auch explizite Initialisierung möglich: **`int[] zahlen = null;`**

Felderzeugung

```
<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
- Feldelemente werden nicht implizit initialisiert!

Beispiele:

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

Felderzeugung / Feldvariablenzuweisung

CARL VON OSSIETZKY universität OLDENBURG

Nebenbedingung/Anmerkungen:

- Elementtyp der Feldvariablen und Elementtyp des Feldes müssen gleich sein!
- Dimension der Feldvariablen und Dimension des Feldes müssen gleich sein!

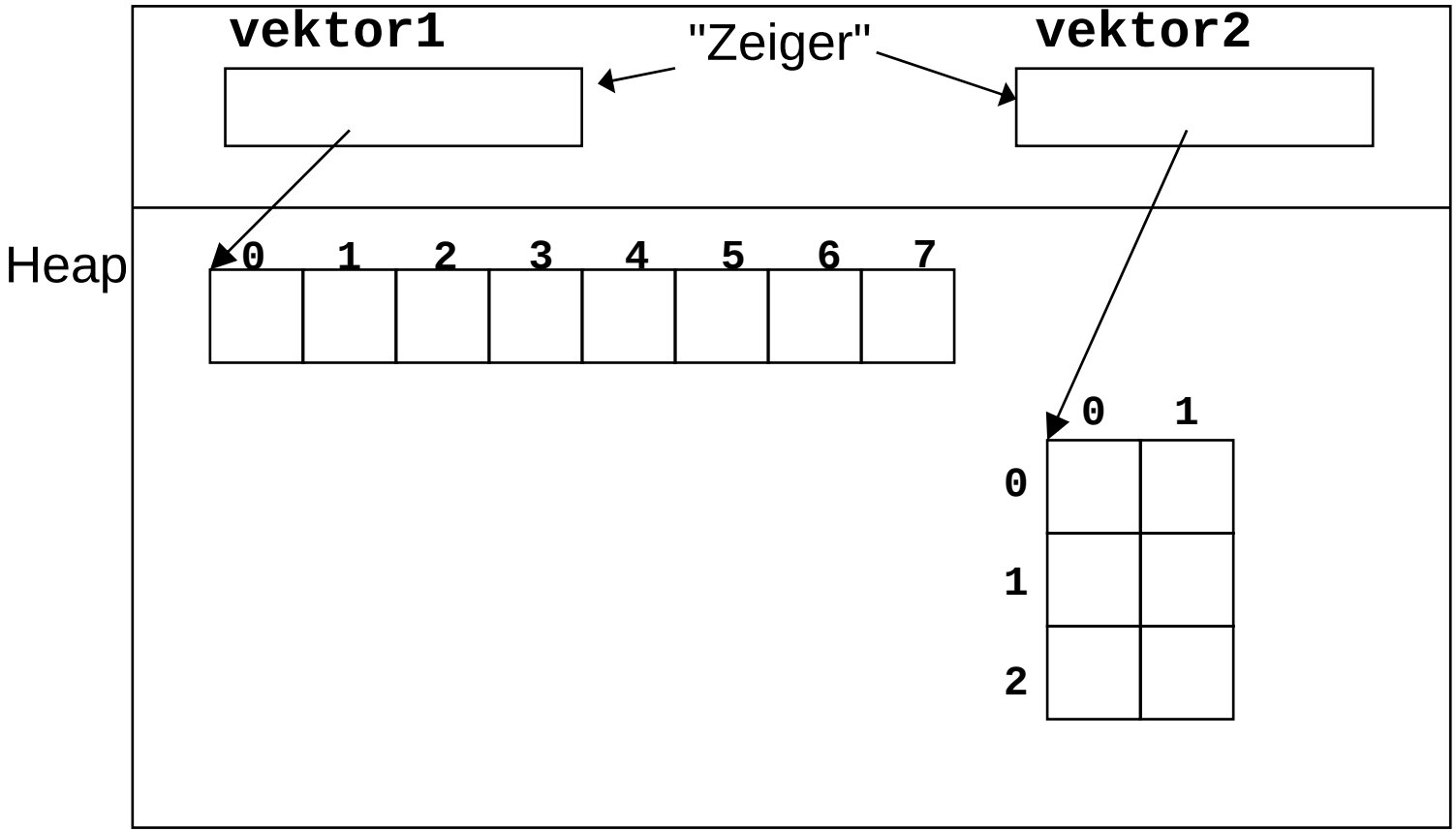
Beispiele:

```
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
```

Felderzeugung / Schema

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

Stack oder Heap



Feldelemente / Zugriff

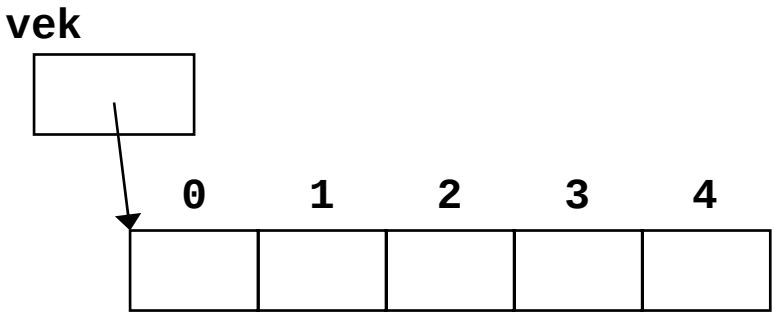
```
<array-access> ::= <Bezeichner>
                   "[" <int-Ausdruck> "]"
                   { "[" <int-Ausdruck> "]" }
```

Nebenbedingungen / Anmerkungen:

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

Beispiele:

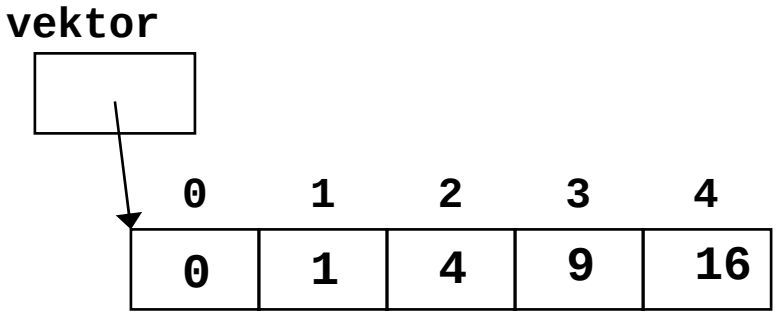
```
int[] vek = new int[5];
vek[0] = -23;
vek[1] = vek[0] + 25;
vek[vek[1]] = -4;
vek[5] = 56; // Laufzeitfehler!
```



Feldelemente / Initialisierung (1)

Explizite Initialisierung:

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



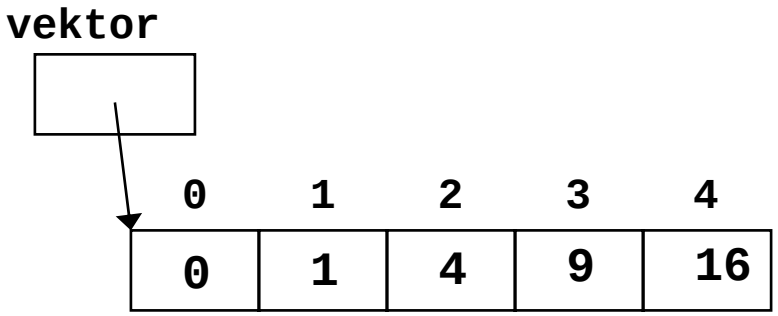
length:

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

Feldelemente / Initialisierung (2)

Implizite Erzeugung und Initialisierung:

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

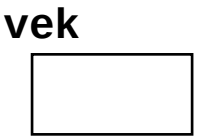


- erzeugt Feld mit entsprechender Elementanzahl
- initialisiert die Elemente
- Initialisierungswerte können durch Ausdrücke des entsprechenden Elementtyps gebildet werden (häufig Literale)

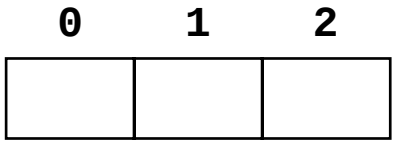
Feldzerstörung

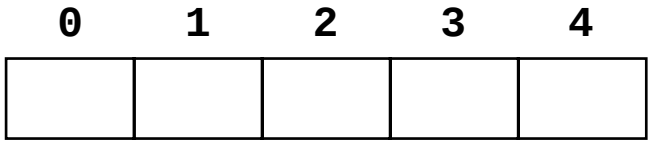
- Java: automatisches Garbage-Collection!
- Kein **`delete`**-Operator
- Speicherplatz wird automatisch freigegeben, wenn nicht mehr auf Heap-Speicherplatz referenziert ("gezeigt") wird

Beispiel:




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

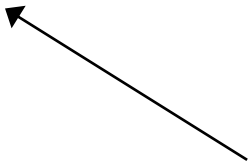




„neue“ for-Schleife (1)

Beispiel: Berechnen der Summe eines Zahlen-Arrays

```
public static void main(String[] args) {
  int[] zahlen = {2, 4, 6, 5, 1, 2};
  System.out.println(summe(zahlen));
}

static int summe(int[] zahlen) {
  int sum = 0;
  for (int i = 0; i < zahlen.length; i++) {
    sum += zahlen[i];
  }
  return sum;
}
```



Durchlauf über alle Array-Elemente

„neue“ for-Schleife (2)

Beispiel: Berechnen der Summe eines Zahlen-Arrays

```
public static void main(String[] args) {
  int[] zahlen = {2, 4, 6, 5, 1, 2};
  System.out.println(summe(zahlen));
}

static int summeNeu(int[] zahlen) {
  int sum = 0;
  for (int zahl : zahlen) {
    sum += zahl;
  }
  return sum;
}
```

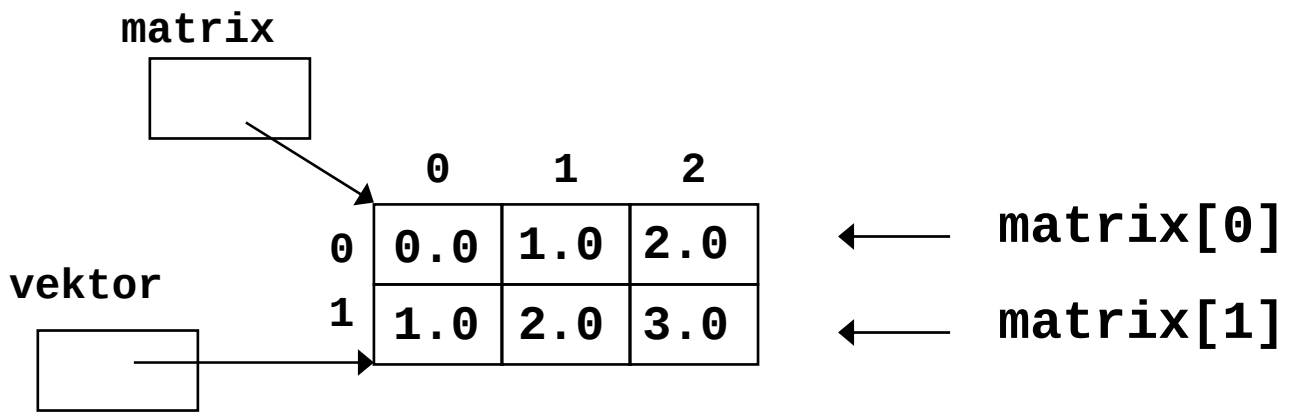
Durchlauf über alle Array-Elemente beginnend bei 0

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

Mehrdimensionale Felder (1)

Normalfall: Anzahl an Elementen pro Dimension ist identisch

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

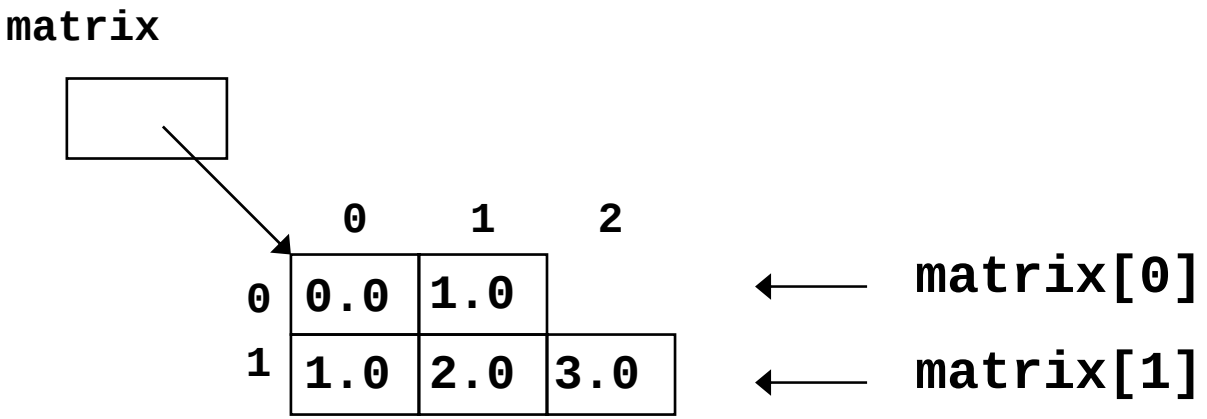



```
float[] vektor = matrix[1];
vektor[1] = 3.4F;
```

Mehrdimensionale Felder (2)

Möglich: Anzahl an Elementen pro Dimension ist unterschiedlich

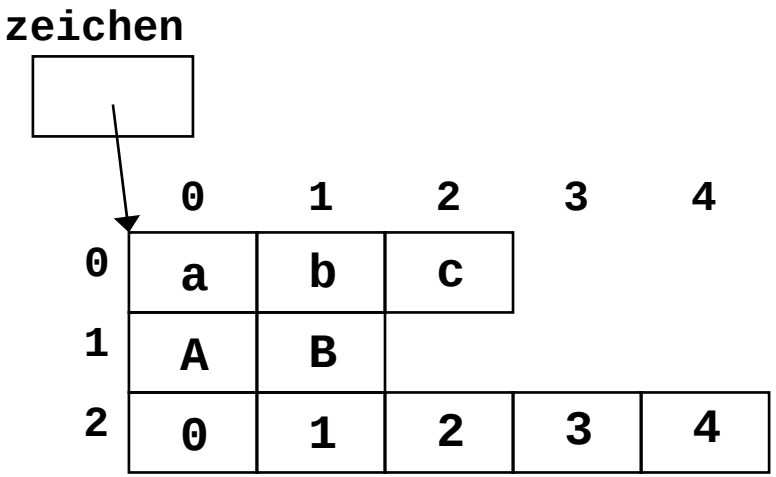
```
float[][] matrix = new float[2][];
matrix[0] = new float[2];
matrix[1] = new float[3];
for (int z=0; z < matrix.length; z++)
  for (int s=0; s < matrix[z].length; s++)
    matrix[z][s] = z + s;
```



Mehrdimensionale Felder (3)

Implizite Erzeugung und Initialisierung:

```
char[][] zeichen = { {‘a‘, ‘b‘, ‘c‘},
                     {‘A‘, ‘B‘},
                     {‘0‘, ‘1‘, ‘2‘, ‘3‘, ‘4‘}
                   };
```



Mehrdimensionale Felder (4)

Beispiel: Ausgabe einer Matrix

```
static void print (int[][] matrix) {
  for (int z=0; z<matrix.length; z++) {
      for (int s=0; s<matrix[z].length; s++) {
         IO.print(matrix[z][s] + " ");
      }
      IO.println();
  }
}
public static void main(String[] args) {
  int[][] zahlen = {{1,2},{3,4},{5,6}};
  print(zahlen);
}
```

```
1 2
3 4
5 6
```

Mehrdimensionale Felder (5)

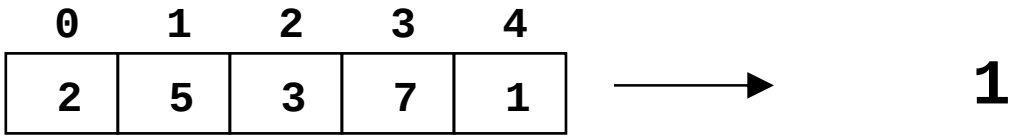
Beispiel: Ausgabe einer Matrix (mit „neuer“ for-Schleife)

```
static void print (int[][] matrix) {
  for (int[] array : matrix) {
    for (int elem : array) {
      IO.print(elem + " ");
    }
    IO.println();
  }
}

public static void main(String[] args) {
  int[][] zahlen = {{1,2},{3,4},{5,6}};
  print(zahlen);
}
```

```
1 2
3 4
5 6
```

Beispiel 1

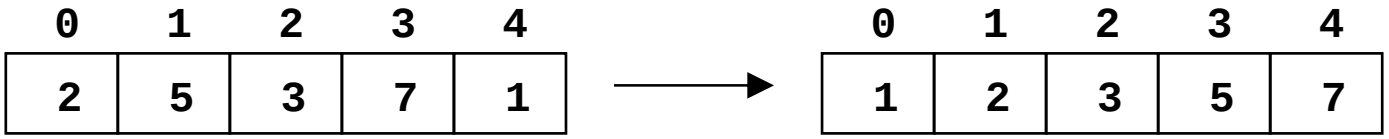
Minimum-Suche:




```
public static void main(String[] args) {
    int[] feld = {2,5,3,7,1};
    System.out.println(minimum(feld));
}

static int minimum(int[] vektor) {
    int min = vektor[0];
    for (int i=1; i<vektor.length; i++) {
        if (vektor[i] < min)
            min = vektor[i];
    }
    return min;
}
```

Beispiel 2 (1)

Sortierung von Zahlen:




```
public static void main(String[] args) {
  int[] feld = {2,5,3,7,1};
  print(feld);
  sortieren(feld);
  print(feld);
}
static void print(int[] vektor) {
  IO.print("Vektor: ");
  for (int i=0; i<vektor.length; i++)
    IO.print(vektor[i] + " ");
  IO.println();
}
```

Beispiel 2 (2)

```
// Bubblesort
static void sortieren(int[] vektor) {
  boolean veraendert = false;
  do {
    veraendert = false;
    for (int i=1; i<vektor.length; i++) {
      if (vektor[i-1] > vektor[i]) {
        int help = vektor[i-1];
        vektor[i-1] = vektor[i];
        vektor[i] = help;
        veraendert = true;
      }
    }
  } while (veraendert);
}
```

0	1	2	3	4
2	5	3	7	1

http://olli.informatik.uni-oldenburg.de/fpsort/Animation.html

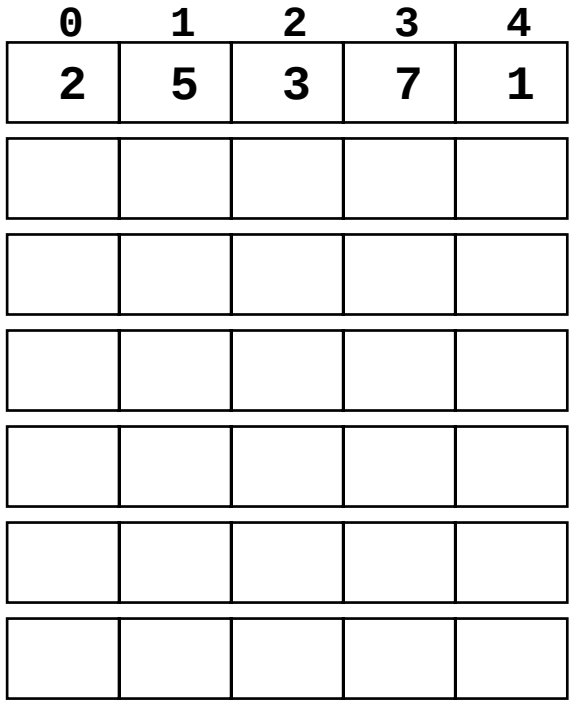
Beispiel 2 (3)

```
// Selectionsort
static void sortieren(int[] zahlen) {
  for (int aktI = 0; aktI < zahlen.length - 1; aktI++) {
    int minI = aktI;
    for (int suchI = aktI+1; suchI<zahlen.length; suchI++) {
      if (zahlen[suchI] < zahlen[minI]) {
        minI = suchI;
      }
    }
    int speicher = zahlen[aktI];
    zahlen[aktI] = zahlen[minI];
    zahlen[minI] = speicher;
  }
}
```

0	1	2	3	4
2	5	3	7	1

Beispiel 2 (4)

```
// Insertionsort
static void sortieren(int[] zahlen) {
  for (int aktI = 1; aktI < zahlen.length; aktI++) {
    int aktZ = zahlen[aktI];
    int vglI = aktI - 1;
    while (vglI >= 0 && zahlen[vglI] > aktZ) {
      zahlen[vglI + 1] = zahlen[vglI];
      vglI--;
    }
    zahlen[vglI + 1] = aktZ;
  }
}
```



Aufgabe 1

Schreiben Sie eine Funktion `mehrfach`, die als Parameter ein Array mit `int`-Werten übergeben bekommt und die genau dann `true` liefert, wenn ein Wert mehrfach im Array vorhanden ist.

```
public static void main(String[] args) {
  int[] feld = {2,-5,1,3,7,1}; // zum Testen
  if (!mehrfach(feld))
      IO.println("nicht ");
  IO.println("mehrfach");
}

static boolean mehrfach(int[] zahlen) ?
```

Aufgabe2

Schreiben Sie ein Programm, bei dem 2 Spieler gegeneinander das bekannte Spiel *4-Gewinnt* spielen können. Vereinfachen Sie die Regeln zunächst dahingehend, dass ein Spieler das Spiel gewonnen hat, wenn er in der Horizontalen eine 4er Reihe bilden konnte (Vertikale und Diagonale fallen weg).