# Einstieg in die Informatik mit Java

## Felder

Gerd Bohlender

Institut für Angewandte und Numerische Mathematik

# Gliederung


1. Was sind Felder?
2. Vereinbarung von Feldern
3. Erzeugen von Feldern
4. Zugriff auf Feldkomponenten
5. Mehrdimensionale Felder
6. Felder als Objekte, Referenzen
   - Kopieren von Feldern mit `System.arraycopy()`
   - Vergleich von Feldern

# Gliederung


1. Was sind Felder?
2. Vereinbarung von Feldern
3. Erzeugen von Feldern
4. Zugriff auf Feldkomponenten
5. Mehrdimensionale Felder
6. Felder als Objekte, Referenzen
   - Kopieren von Feldern mit `System.arraycopy()`
   - Vergleich von Feldern

# Was sind Felder?

Häufig werden viele gleichartige Daten zusammen abgespeichert.

## Beispiele

Vektoren, Matrizen, Tabellen, Monitorbilder als eine Matrix von Pixeln.

- Die Einträge in einem Feld heissen *Elemente* oder *Komponenten*.

## Beispiel

Elemente $a_0$ bis $a_{n-1}$:

| $a_0$ | $a_1$ | $a_2$ | $\ldots$ | $a_{n-1}$ |
|---|---|---|---|---|

- Alle Komponenten sind vom gleichen Typ (*Komponententyp*).
- Felder sind in Java Objekte, die dynamisch erzeugt werden, und sind im Prinzip nur eindimensional, können aber als Komponententyp einen anderen Feldtyp besitzen.

# Gliederung


1. Was sind Felder?
2. Vereinbarung von Feldern
3. Erzeugen von Feldern
4. Zugriff auf Feldkomponenten
5. Mehrdimensionale Felder
6. Felder als Objekte, Referenzen
   - Kopieren von Feldern mit `System.arraycopy()`
   - Vergleich von Feldern

# Vereinbarung von Feldern

## Syntax

*Komponententyp* `[ ]` *Bezeichner*;

Alternativ ist auch noch die C++ Syntax möglich:

*Komponententyp Bezeichner* `[ ];`

## Achtung

Anders als in Pascal und C++ wird durch die Deklaration noch kein Speicher für die Komponenten reserviert, es existiert bislang nur eine Referenz auf das Feld!

## Beispiel

```
int[] a;          // math. Vektor mit ganzzahligen Komp.
double[] b, c;    // math. Vekt. mit Komp. vom Typ double
double[][] m;     // Matrix: Feld von Feldern
```

# Gliederung


1. Was sind Felder?
2. Vereinbarung von Feldern
3. Erzeugen von Feldern
4. Zugriff auf Feldkomponenten
5. Mehrdimensionale Felder
6. Felder als Objekte, Referenzen
   - Kopieren von Feldern mit `System.arraycopy()`
   - Vergleich von Feldern

# Erzeugen von Feldern

Felder können auf die folgenden Weisen erzeugt werden:

(1) Mit dem Operator `new` bei der Initialisierung,

**Beispiel**

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

(2) durch eine Initialisiererliste bei der Deklaration,

**Beispiel**

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

# Erzeugen von Feldern

(3) durch Zuweisung eines mit `new` erzeugten Feldes,

**Beispiel**

```
int[] c;
...
c = new int[3];
```

(4) durch eine Kombination aus (2) und (3).

**Beispiel**

```
int[] d;
...
d = new int[] {1,2,3};
```

# Erzeugen von Feldern, Achtung

## Achtung

Bei der letzten Variante muss `new int[]` stehen (im Gegensatz zur Initialisiererliste bei der Deklaration). Die Anzahl der Feldelemente darf nicht angegeben werden.

## Beispiel

```
int[] d;
...
d = {1,2,3}; // Fehler: new int[] fehlt
d = new int [3] {1,2,3}; // Fehler:
// Dimensionsangabe verboten
```

# Initialisierung von Feldern

Im Gegensatz zu C++ darf die Initialisiererliste auch Variablen und Ausdrücke enthalten!

## Beispiel

```
int x=11, y=22;
int[] d = {x, y+1};
```

# Erzeugen von Feldern

Die Länge eines Feldes kann mit einer ganzzahligen Variablen (allerdings nicht vom Typ `long`) angegeben werden.

**Achtung**

Das heißt natürlich nicht, dass sich die Feldlänge bei einer Änderung der Variablen ebenfalls ändert!

**Beispiel**

```
int dim = 5;
int[] e = new int[dim];
// Feld e hat 5 Komp.
dim     = 10;
// e hat immer noch 5 Komp.
```

# Gliederung


1. Was sind Felder?
2. Vereinbarung von Feldern
3. Erzeugen von Feldern
4. Zugriff auf Feldkomponenten
5. Mehrdimensionale Felder
6. Felder als Objekte, Referenzen
   - Kopieren von Feldern mit `System.arraycopy()`
   - Vergleich von Feldern

# Zugriff auf Feldkomponenten

Ist `i` ein `int`–Ausdruck, so kann mit `f[i]` auf das `i`-te Feldelement von `f` zugegriffen werden.

## Beispiel

```
int[] f = { 1, 2, 3 };
```

| f[0] | f[1] | f[2] |
|---|---|---|
| 1 | 2 | 3 |

Damit ergibt sich für die Komponente `f[0]` der Wert 1, für die Komponente `f[1]` der Wert 2 und für die Komponente `f[2]` der Wert 3.

## Achtung

Die Indizierung der Feldelemente beginnt in Java immer bei Null! Für das obige Beispiel bedeutet dies, dass die Komponente `f[3]` nicht existiert!
Im Gegensatz zu C++ werden Zugriffe auf nichtexistente Feldelemente geprüft und führen zu einem Laufzeitfehler!

# Zugriff auf Feldlänge

Die Länge eines Feldes `f` kann durch `f.length` bestimmt werden.

## Beispiel

```
char [] vokale = {'a', 'e', 'i', 'o', 'u'};
...
for (int i=0; i<vokale.length; ++i)
  System.out.println (vokale[i]);
```

# Beispiel Vektoraddition

```
import java.util.*;
public class AddVek {
   public static void main (String[] args) {
      Locale.setDefault (Locale.US);
      Scanner sc = new Scanner (System.in);
      double[] a, b;
      System.out.print ("Dimension = ");
      int dim = sc.nextInt ();
      a = new double[dim];
      b = new double[dim];
      for (int i=0; i<a.length; ++i) {
         System.out.print ("a [" +i+ "]: ");
         a[i] = sc.nextDouble ();
         System.out.print ("b [" +i+ "]: ");
         b[i] = sc.nextDouble ();
      }
      for (int i=0; i<a.length; ++i)
         System.out.println("Summe["+i+"]: "+(a[i]+b[i]));
   }
}
```

# Gliederung


1. Was sind Felder?
2. Vereinbarung von Feldern
3. Erzeugen von Feldern
4. Zugriff auf Feldkomponenten
5. Mehrdimensionale Felder
6. Felder als Objekte, Referenzen
   - Kopieren von Feldern mit `System.arraycopy()`
   - Vergleich von Feldern

# Mehrdimensionale Felder

Mehrdimensionale Felder sind genau genommen eindimensionale Felder, deren Komponenten selbst wieder Felder sind.

- **Vereinbarung**:
  Komponententyp [] Name;
  wobei der Komponententyp ein Feld ist.
- **Erzeugung**:
  Die geläufigste Methode zur Erzeugung ist die mittels des `new`–Operators. Daneben findet gelegentlich auch die Erzeugung mit Hilfe einer geschachtelten Initialisiererliste
  Anwendung: `{{Zeile0}, {Zeile1}, ...}`
- **Zugriff auf Komponenten**:
  Feldname [ $Index_1$ ] [ $Index_2$ ] ...

**Achtung**

Die Schreibweise Feldname [ $Index_1$ , $Index_2$ ] ist nicht erlaubt.

# Mehrdimensionale Felder – Beispiele

```
double[][][] quader = new double[4][2][3];
// 4x2x3 = 24 double-Zahlen, Zugriff z.B.:
quader[1][1][2] = 3.14;

double[][] quadrat   = {{ 1, 2 },{ 3, 4 }};
// 2x2 Matrix, Zugriff z.B.:
System.out.println (quadrat[0][0]);
// gibt 1 aus

double[][] dreieck   = {{ 1 },{ 2, 3 },{ 4, 5, 6 }};
// Dreiecksmatrix, Zugriff z.B.:
dreieck[2][2] = dreieck[0][0] + 6;
// aus 6 wird 7
```

**Achtung**

Wie im dritten Beispiel erkennbar, darf die Zeilenlänge durchaus auch unterschiedlich sein!

# Mehrdimensionale Felder

Die Dimensionsangaben dürfen ganzzahlige Ausdrücke sein (jedoch nicht vom Typ `long`).

## Beispiel

```
int dim = scanner.nextInt();
double[][] mat1 = new double[dim][dim];
double[][] mat2 = new double[dim+1][2*dim];
```

Die Angabe der ersten Dimension ist notwendig, die weiteren können entfallen.

## Beispiel

```
double[][] matrixA = new double[3][];
```

Es wird nur ein Vektor von Referenzen auf die Zeilen angelegt, die Zeilen selber existieren noch nicht!

# Mehrdimensionale Felder – Schrittweise Erzeugung
# double[][] matrixB = new double[3][5];

```
double[][] matrixB;
```

matrixB → ???  (Variable matrixB ist nicht initialisiert)

```
matrixB = new double[3][];
```

matrixB →

| null |
|---|
| null |
| null |

```
for (int i=0; i<3; ++i) matrixB[i] = new double[5];
```

matrixB →

| | | | | | |
|---|---|---|---|---|---|
| | → | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| | → | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| | → | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

# Gliederung


1. Was sind Felder?
2. Vereinbarung von Feldern
3. Erzeugen von Feldern
4. Zugriff auf Feldkomponenten
5. Mehrdimensionale Felder
6. Felder als Objekte, Referenzen
   - Kopieren von Feldern mit `System.arraycopy()`
   - Vergleich von Feldern

# Felder als Objekte, Referenzen

Im Gegensatz zu den Grundtypen (Zahlen, Zeichen und boolsche Typen) werden Felder wie alle Objekte als Referenzen behandelt.
Das bedeutet, dass eine Zuweisung über den Zuweisungsoperator = nur eine neue Referenz (einen *Aliasnamen* für das Feld) erzeugt, jedoch keine echte Kopie.

**Achtung**

Bei Kopien mit Hilfe des Zuweisungsoperators wird bei Grundtypen der Wert kopiert, bei Referenztypen nur die Referenz, nicht der Wert!

# Felder als Objekte, Referenzen

## Beispiel

```
int    i = 1;
int    j = i;   // ergibt i=1, j=1
       i = 2;   // ergibt i=2, j=1
int[] a = {1};
int[] b = a;   // ergibt a[0]=1, b[0]=1
   a[0] = 2;   // ergibt a[0]=2, b[0]=2
```

`i` und `j` sind zwei unabhängige Variablen, `a` und `b` sind lediglich zwei unabhängige Referenzen.
Abhilfe schafft die Methode `clone()`.

## Achtung

`clone()` bearbeitet nur die erste Ebene, bei mehrdimensionalen Feldern muss `clone()` mehrfach aufgerufen werden!

# Kopieren von Feldern mit `System.arraycopy()`

## Syntax

`System.arraycopy (`*Quelle*`,`*Quellindex*`,`
*Ziel*`,`*Zielindex*`,`*Laenge*`);`

## Beispiel

```
int[] a = {0, 1, 2, 3, 4, 5};
int[] b = new int[10];          // b[0]=0,...,b[9]=0
System.arraycopy(a,2,b,4,3);    // b[4]=2,b[5]=3,b[6]=4
```

b →

| 0 | 0 | 0 | 0 | 2 | 3 | 4 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|

## Achtung

Felder werden wie alle Objekte automatisch initialisiert!

# Vergleich von Feldern

Mit dem logischen Vergleich `a==b` werden nur die Referenzen verglichen, nicht jedoch der Inhalt der Felder. Dasselbe gilt bei der Verwendung der Methode `equals()`.

## Beispiel

```
int[] a = {1};
int[] b = {1};
if (a==b) ...          // Bedingung ergibt false
if (a.equals(b)) ...   // Bedingung ergibt false
```