

### 3.3 Felder

Felder (auch Reihungen genannt, engl.: arrays) dienen dazu, mehrere Elemente desselben Datentyps zusammenzufassen. Für ein Feld wird ein fortlaufender Speicherbereich reserviert, der genug Platz bieten muss, um alle einzelnen Elemente aufzunehmen. Der pro Element reservierte Speicherplatz enthält die Daten des jeweiligen Elements (bei Grundtypen) bzw. einen Verweis auf die Elementdaten (bei Referenztypen).

#### Deklaration von Feldern

Die Deklaration von Feldern erfolgt in Java durch Paare von eckigen Klammern, die direkt hinter den Datentyp der Feldelemente geschrieben werden. Die Anzahl der Klammerpaare bestimmt dabei die Dimension des Feldes, d.h. die Anzahl der Indexwerte, die für den Zugriff auf genau ein Element des Feldes benötigt werden. Ein  $n$ -dimensionales Feld kann allgemein als ein  $i$ -dimensionales Feld mit  $n - i$ -dimensionalen Feldern als Elementen betrachtet werden. Mehrdimensionale Felder sind z.B. zur Repräsentierung von Matrizen gut geeignet.

Das folgende Syntaxschema zeigt Variablendeklarationen für ein- bzw. zweidimensionale Felder. Höherdimensionale Felder werden analog mit der entsprechenden Anzahl von Klammerpaaren deklariert.

`<Feldekларation> ::= <Typ> ('[]')+ <Bezeichner> ';' |`

Hinweis: Die Klammerpaare können auch erst hinter den Bezeichner der Feldvariablen gesetzt werden, wobei auch eine Mischung beider Schreibweisen erlaubt ist. Nach der Deklaration ist ein Feld zwar bzgl. seines Datentyps festgelegt, aber es wurde noch kein Speicherplatz für die Feldelemente reserviert. Ein derartiges Feld kann auch als Feld ohne Elemente betrachtet werden. Erst durch die Initialisierung des Feldes wird die Anzahl der Feldelemente festgelegt.

### 3.4 Initialisierung von Feldern

Auch eine Feldvariable kann bereits während der Deklaration initialisiert werden, indem man der Variablen eine mit Kommas getrennte Auflistung ihrer Elemente in geschweiften Klammern zuweist. Auch mehrdimensionale Felder können so initialisiert werden, wenn die entsprechenden Teilfelder innerhalb des Feldes angegeben bzw. initialisiert werden.

```
float[] zahlenFeld = {1.1, 2.2, 3.3, 4.4}; // eindimensionales Feld  
char[][] zeichenFeld = {{ 'a', 'b' }, // eindimensionale Felder als  
                        { 'c', 'd' } }; // Feldelemente
```

Wichtig: Die explizite Angabe von Feldinhalten zwischen geschweiften Klammern ist nur im Rahmen einer Felddeklaration zulässig. Jede andere Verwendung dieser Schreibweise, z.B. in einer separaten Zuweisungsanweisung, führt zu einer Fehlermeldung.

#### Zugriff auf Feldelemente

Der Zugriff auf einzelne Elemente eines eindimensionalen Feldes erfolgt über den Index des Elements. Der Index entspricht der Position des Elements relativ zum Anfang des Feldes, wobei das erste Element im Feld in Java stets den Index 0 erhält. Der Index numeriert die Elemente lückenlos zum Feldende hin aufsteigend durch. Das letzte Element eines  $n$ -elementigen Feldes hat somit den Index  $n - 1$ . Die Anzahl der Elemente eines Feldes wird auch als Länge des Feldes bezeichnet. Die Länge stellt einen Attributwert des Felds dar und kann abgefragt werden, indem man `.length` nach der Angabe des Feldes bzw. seines Namens schreibt. Damit kann zur Laufzeit für jedes beliebige Feld der grösste gültige Feldindex ermittelt werden. Auf Feldelemente

wird zugegriffen, indem der Index des gewünschten Elementes in eckigen Klammern an das Feld bzw. den Feldnamen angehängt wird.

Der Zugriff auf mehrdimensionale Felder verläuft analog zum Zugriff auf eindimensionale Felder, wobei entsprechend der Dimension mehrere Indexwerte zur Adressierung eines Elements benötigt werden. Um auf ein einzelnes Element eines n-dimensionalen Feldes zuzugreifen, ist also ein n-stelliger Index der Form  $[indexwert_1][indexwert_2] \dots [indexwert_n]$  nötig. Allgemein erhält man durch Angabe von m Indexwerten ( $m \leq n$ ) Zugriff auf Teilfelder mit der Dimension  $n - m$ .

```
float[] zahlenFeld = {- 1.1, 2.2, 3.3, 4.4};
zahlenFeld[3] = 5.0; // Wert des 4. Elements zu 5.0 aendern
int letztesElement = zahlenfeld.length -1; // zahlenfeld hat 4 Feldelemente ==>
// Index des letzten Elements ist 3
zahlenfeld[letztesElement] = 4.0; // Wert des letzten Element zu 4.0
// aendern
char[][] zeichenFeld = {{ 'a', 'b' },
                        { 'c', 'd' } };
zeichenFeld[1][0] = zeichenFeld[0][1]; // 'c' wird mit 'b' ueberschrieben
int AnzahlZeichenFelder = zeichenfeld.length; // zeichenfeld hat 2 Feldelemente
// (mit 2 Elementen pro Teilfeld)
int AnzahlZeichen = zeichenfeld[0].length; // zeichenfeld[0] ist Teilfeld
```

### 3.4.1 Erzeugung leerer Felder

Zum Zeitpunkt der Deklaration eines Feldes sind oft dessen Grösse bzw. sein Inhalt noch nicht bekannt, so dass eine Initialisierung des Feldes bzw. der Feldelemente an dieser Stelle noch nicht möglich ist. Für solche Situationen bietet Java die Möglichkeit, ein leeres Feld geeigneter Grösse zu erzeugen, sobald die Anzahl der Feldelemente bekannt ist. Mit Hilfe des Operators **new** wird explizit ein Speicherbereich reserviert, der gross genug ist, um das gesamte Feld aufzunehmen. Die Feldelemente selbst können dann auch erst zu einem späteren Zeitpunkt initialisiert werden. Die Syntax zur Erzeugung eines (eindimensionalen) Feldes entspricht folgendem Schema:

<Felderzeugung> ::= 'new' <Datentyp> '[' <Feldlänge> ']'

Die Angabe des Datentyps und der Feldlänge ist erforderlich, damit die Grösse des benötigten Speicherplatzes genau berechnet werden kann. Bei mehrdimensionalen Feldern findet die Initialisierung nach dem selben Prinzip statt, wobei für die weiteren Dimensionen die jeweiligen Längenangaben in zusätzliche eckige Klammerpaare gesetzt werden<sup>4</sup>.

```
char[] feld = new char[80];
int[][] matrix; // Deklaration (Typfestlegung fuer Variablennamen)
matrix = new int[2][2]; // Initialisierung (Festlegung der Feldlaenge)
```

Wichtig: Die Initialisierung von leeren Feldern (mit **new**) betrifft immer nur das Feld selbst, d.h. Anzahl und Art der Feldelemente werden festgelegt. Eine Initialisierung der einzelnen Feldelemente findet dabei nicht statt!

Wird ein Feld bei der Deklaration nicht initialisiert, so entspricht dies einer impliziten Initialisierung mit **null**. Die explizite Initialisierung eines derartigen Feldes könnte dann z.B. so aussehen:

```
int[][] feld; feld = null;
```

Eine solche Initialisierung erzeugt ein Feld mit 0 Elementen, d.h. es muss kein Speicherplatz für die Feldelemente reserviert werden.

Bei mehrdimensionalen Feldern ist auch eine partielle Initialisierung möglich. Wird z.B. bei einem zweidimensionalen Feld eine Initialisierung nur für die erste Dimension angegeben, so entspricht dies der Initialisierung

eines eindimensionalen Feldes mit eindimensionalen Feldern als Elementen. Damit sind auch unregelmässig geformte Felder möglich, da die einzelnen Teilfelder später mit unterschiedlichen Grössen initialisiert werden können.

```
int[][] feld = new int[2][]; // feld hat 2 uninitialisierte Teilfelder
int[] erstesElement = new int[1];
feld[0] = erstesElement; // 1. Teilfeld hat 1 Element
feld[1] = new int[10]; // 2. Teilfeld hat 10 Elemente
String[] textFeld1 = {"das", "ist", "ein", "Textfeld"};
```

Der **new**-Operator wird auch zur dynamischen Erzeugung von einzelnen Objekten verwendet. Hierbei entfällt logischerweise die Angabe irgendwelcher Feldlängen.

```
String[][] textFeld2 = {textFeld1,
                        {"zweidimensionales", "Textfeld"}};
```

Bei der partiellen Initialisierung ist darauf zu achten, dass immer nur komplette Teilfelder uninitialisiert bleiben dürfen. Das bedeutet, dass nach der ersten nichtinitialisierten Dimension keine weiteren Initialisierungsgrössen mehr folgen dürfen. Initialisierungen wie z.B.

```
int[][] initFehler1 = new int[][3]; // FEHLER: Anzahl der Teilfelder fehlt!
double[][][] initFehler2 = new double[10][][10]; // FEHLER: Teilfeld nicht
// komplett uninitialisiert!
```

sind also nicht zulässig. Ist die Länge eines Feldes durch die Initialisierung erst einmal vorgegeben, kann sie nicht mehr geändert werden. Insbesondere kann ein Feld nicht vergrössert werden. Stattdessen wird normalerweise ein zweites Feld mit entsprechend veränderter Grösse erzeugt, in das die ursprünglichen Feldelemente bzw. die noch benötigten Elemente kopiert werden können.