Inhalt

4 Einführung in die Programmiersprache Java (Teil III)

4.5 Referenzdatentypen - Felder

Bisher wurde nur der Umgang mit *Elementardatentypen* behandelt. Für komplexe Anwendungen reichen diese oft nicht aus. Man benötigt zusammengesetzte, sogenannte *strukturierte Daten*. **Referenzdatentypen** ermöglichen den Zugriff auf solche.

Java unterscheidet zwei Arten von Referenzdatentypen, **Felder** und **Klassen**. Im Unterschied zu Elementardatentypen werden auf Werte von Referenzdatentypen *nicht direkt*, sondern *indirekt* über **Referenzen**, zugegriffen. Eine Referenz ist eine Variable, die als Wert eine Adresse besitzt und somit auf einen Speicherbereich verweist (Zeiger).

Definition Feld

Feld der Länge *n* $=_{df}$ ***n*-Tupel über einer Menge von Daten *gleichen* Typs**

***Felder* sind *strukturierte* Datentypen, die für ein Programm eine Einheit bilden und Daten *gleichen* Typs zusammenfassen.**
Die zusammengefassten Daten sind die *Feldkomponenten*.

Arbeitsspeicher

Symbolische Adresse	*Adresse im Speicher*	*Inhalt der Speicherzelle*	*Interpretations-vorschrift*
	...	...	
b	*5e*	00	`short` (2 Byte)
	5f	6a	
	...	...	
r	*65*	a3	`char[]` (Referenz)
	...	...	
	a3	00	`char[0]` (2 Byte)
	a4	41	
	a5	00	`char[1]` (2 Byte)
	a6	75	
	a7	00	`char[2]` (2 Byte)
	a8	74	
	a9	00	`char[3]` (2 Byte)
	aa	6f	
	...	...	...

Elementardatentyp

```
short b = 106;
```

b ist eine Variable des *Elementardatentypen* `short`:
b hat die Speicheradresse $(5e)_{16} = 94$, 2 Byte Länge und den Wert $(00\ 6a)_{16} = 106$.

Referenzdatentyp

```
char[] r = { 'A', 'u', 't', 'o'};
```

r ist eine *Referenzvariable*:

r hat im Speicher die Adresse $(65)_{16}$. Dort steht als *Wert* wiederum eine Adresse $(a3)_{16}$. Diese zeigt auf den Anfang eines *Feldes*. Um das Feld auswerten zu können, muss die *Anzahl* und der *Typ* der *Feldkomponenten* bekannt sein. Im Beispiel handelt es sich um ein Feld mit 4 Komponenten vom Typ `char` . Die einzelnen Feldkomponenten bestehen jeweils aus 2 Byte. Der Speicherbereich, auf den die Referenz verweist, umfasst damit insgesamt 8 Byte und repräsentiert 4 Zeichen im Unicode, hier „Auto“.

4.5.1 Eindimensionale Felder - Vektoren

Analog einfacher Variablen haben Feldvariablen einen **Namen**, einen **Typ und** als **Wert** eine Adresse. Alle durch ein Feld zusammengefasste Daten werden in **Feldkomponenten** abgespeichert.

Deklaration

Mit der Deklaration eines Feldnamens wird dem Compiler mitgeteilt, dass es sich um eine *Referenzvariable* handelt und *Speicherplatz für eine Adresse* zur Verfügung gestellt werden muss.

Komponententyp **[ ]** *Feldname* **;**

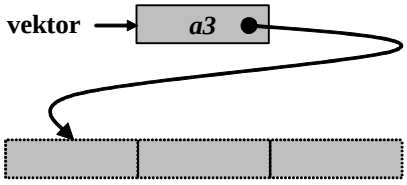
```
float[] vektor;
```

Definition

Um ein Feld für den Gebrauch vorzubereiten, muss der notwendige Speicherplatz für die Komponenten reserviert werden. Dies geschieht mit Hilfe des ***new***-Operators.

Feldname **=** **new** *Komponententyp* **[** *Länge* **] ;**

```
vektor = new float[ 3];
```



oder beides zusammen:

```
float[] vektor = new float[ 3];
```

Zugriff auf die Feldkomponenten

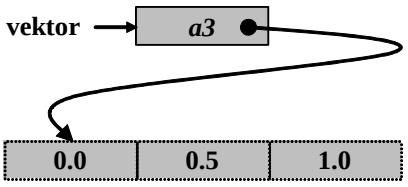
Auf einzelne Feldkomponenten wird über einen **Index** durch den **[ ]-Operator** zugegriffen. Dieser ist ganzzahlig und muss in den Feldgrenzen liegen, d.h. $0 \leq Index < Länge$.

Feldname **[** *Index* **]**

```
vektor[ 0]      vektor[ 1]     vektor[ 2]
```

Durchlaufen eines eindimensionalen Feldes

```
for( int i = 0; i < 3; i++)
  vektor[ i] = (float)(i / 2.);
```
[1]



Die **Laufvariable** `i` heißt auch **Indexvariable**, weil `i` alle möglichen Komponenten über den Index des Feldes `vektor` durchläuft.

[1] Da der Ausdruck `(i / 2.)` einen `double`-Wert liefert, muss dieser explizit in einen `float`-Wert umgewandelt werden.

Zu jedem Feld wird dessen *Länge* als ganzzahlige Variable `length` zusätzlich im Speicher abgelegt. Diese kann im Programm abgefragt werden und sollte grundsätzlich, anstatt einer expliziten Längenangabe, verwendet werden.

```
for( int i = 0; i < vektor.length; i++)
  vektor[ i] = (float)(i / 2.);
```

Explizite Anfangswertzuweisung (Initialisierung)
Felder können durch eine Wertemenge bei Ihrer Deklaration definiert und initialisiert werden. Die Länge des Feldes richtet sich nach der Anzahl der Werte in der Wertmenge. Für die Werte wird der notwendige Speicher bereitgestellt.

Komponententyp **[ ]** *Feldname* **=** *Wertmenge* **;**

Mit der folgenden Initialisierung kann der obige Vektor vereinbart werden:

```
float[] vektor = { 0, (float).5, 1};
```

Kopieren eindimensionaler Felder

1. Kopie der Datenstruktur
```
float[] vektor = { 0, .5f, 1};
float[] kopie = new float[ vektor.length];
```

2. Kopie der Daten
```
for( int i = 0; i < vektor.length; i++)
  kopie[ i] = vektor[ i];

for( int i = 0; i < kopie.length; i++) kopie[ i] *= 2;
```

4.5.2 Beispiel „Sortieren eines Vektors“

Beispiel
In einem Feld werden Werte eingegeben, sortiert und anschließend wieder ausgegeben.

Vektor.java (Grobstruktur)
```
public class Vektor
{
  public static void main( String[] args)
  {
    // Deklaration und Definition
    // Vektoreingabe
```

```
      // Sortieren mit SelectSort
      // Vektorausgabe
    }
  }
```

Vektor.java

```
  // Vektor.java                                    MM 2009
  import Tools.IO.*;                              // Eingaben

  /**
   * Sortieren eines Vektors mit Hilfe SelectSort.
   */
  public class Vektor
  {
  /**
   * Eingabe, Sortieren und Ausgabe eines Vektors.
   */
    public static void main( String[] args)
    {
      // Deklaration und Definition
      float[] vektor
      = new float[ IOTools.readInteger( "Anzahl Werte: ")];

      // Vektoreingabe
      for( int i = 0; i < vektor.length; i++)
        vektor[ i] = IOTools.readFloat( "v[" + i + "] = ");

      // Sortieren mit SelectSort
      for( int i = 0; i < vektor.length - 1; i++)
      {
        int k = i;                          // Suche Minimum
        for( int j = i + 1; j < vektor.length; j++)
          if( vektor[ k] > vektor[ j]) k = j;

        if( i != k)                           // Vertausche
        {
          float temp = vektor[ i];
          vektor[ i] = vektor[ k];
          vektor[ k] = temp;
        }
      }

      // Vektorausgabe
      for( int i = 0; i < vektor.length; i++)
        System.out.print( "  " + vektor[ i]);

      System.out.println();
    }
  }
```

4.5.3 Zweidimensionale Felder - Matrizen

Eine Matrix ist ein eindimensionales Feld, deren Komponenten eindimensionale Felder sind. **Deklaration, Definition und Initialisierung** werden als Feld von Feldern ausgeführt. Für die **Deklaration** und **Definition** einer Matrix ergibt sich damit:

Deklaration und Definition

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

bzw.

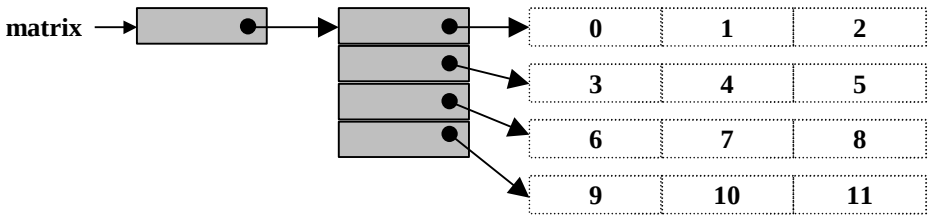
```
int[][] matrix = new int[ 4][ 3];
```

Zugriff auf die Feldkomponenten

```
matrix[ 0][ 0]      matrix[ 0][ 1]      matrix[ 0][ 2]
matrix[ 1][ 0]      matrix[ 1][ 1]      matrix[ 1][ 2]
matrix[ 2][ 0]      matrix[ 2][ 1]      matrix[ 2][ 2]
matrix[ 3][ 0]      matrix[ 3][ 1]      matrix[ 3][ 2]
```

Durchlaufen eines zweidimensionalen Feldes

```
int k = 0;
for( int z = 0; z < matrix.length; z++)
  for( int s = 0; s < matrix[ 0].length; s++)
    matrix[ z][ s] = k++;
```



Das Durchlaufen einer Matrix erfolgt zeilenweise und erfordert eine verschachtelte `for`-Schleife mit zwei Indexvariablen (hier: `z` Zeilenindex, `s` Spaltenindex). Dabei gibt `matrix.length` die *Anzahl der Zeilen* von `matrix` an. Um die *Anzahl der Spalten* je Zeile zu ermitteln, wird mit `matrix[ 0].length` die Länge der 0. Zeile von `matrix` zu Hilfe genommen.

Explizite Anfangswertzuweisung (Initialisierung)

Matrizen können durch eine Menge von Mengen bei Ihrer Deklaration initialisiert werden. Die Zeilen- und Spaltenanzahl richtet sich nach der Anzahl der Wertmengen und der Anzahl der Werte in der Wertmengen. Für die Werte wird der notwendige Speicher bereit gestellt.

```
int[][] matrix
= { { 0, 1, 2}, { 3, 4, 5}, { 6, 7, 8}, { 9 , 10, 11}};
```

Im Beispiel wurde analog eine 4 x 3 - Matrix `matrix` deklariert, definiert und mit Werten initialisiert.

Eine Matrix ist eine *Referenz auf ein Feld von Referenzen.*

Kopieren zweidimensionaler Felder

1. Kopie der Datenstruktur:

```
int[][] matrix
  = { { 0, 1, 2}, { 3, 4, 5}, { 6, 7, 8}, { 9 , 10, 11}};
int[][] kopie
  = new int[ matrix.length][ matrix[ 0].length];
```

2. Kopie der Daten:

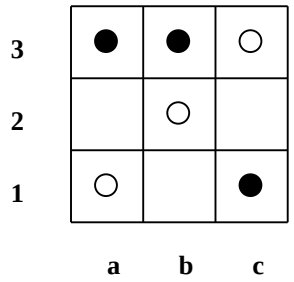
```
for( int z = 0; z < matrix.length; z++)
  for( int s = 0; s < matrix[ 0].length; s++)
    kopie[ z][ s] = matrix[ z][ s];
```

4.5.4 Beispiel „Tic Tac Toe“

Tic Tac Toe ist ein Spiel, welches auf einem Spielbrett mit $n * n$ Karos und mit weißen und schwarzen Steinen gespielt wird. In der Ausgangsspielsituation sind alle Karos leer. Weiß und Schwarz besetzen abwechselnd ein leeres Karo, Weiß beginnt. Das Spiel wird in zwei Varianten gespielt:

1. Gewonnen hat der Spieler, dem es als erstem gelingt, m der eigenen Steine in eine waagerechte, senkrechte oder diagonale Reihe zu bringen.
2. Verloren hat der Spieler, der als erster m der eigenen Steine in eine waagerechte, senkrechte oder diagonale Reihe setzen muss.

Spielsituation auf einem 3*3 Brett, bei der 3 weiße Steine in die Diagonale gesetzt wurden:



Mit einem guten Partner wird das Spiel am 3*3 - Brett für beide Spielvarianten unentschieden enden.

Tic Tac Toe für ein 3*3 – Brett und der ersten Spielvariante:

TicTacToe.java (Grobstruktur)

```
public class TicTacToe
{
  public static void main( String[] args)
  {
    // Spielerlaeuterung

    // leeres Spielbrett
    // 2 Spieler
```

```
      // Spielstart
      boolean fertig = false;
      // Spielrunde
      do
      {
        // aktueller Spieler
        // Stein setzen
        // Spielbrett zeigen
        // Auswerten
          // Zeilentest
          // Spaltentest
          // positiver Diagonalentest
          // negativer Diagonalentest
        // Fertig
      } while( !fertig);
      // Spielauswertung
    }
  }
```

TicTacToe.java

```
  // TicTacToe.java                                    MM 2009
  import Tools.IO.*;                                // Eingaben

  /**
   * TicTacToe - Spiel fuer zwei Personen.
   */
  public class TicTacToe
  {
  /**
   * Spieler setzen abwechseln drei Steine,
   * gewonnen hat der Spieler, der seine Steine
   * in einer Zeile, Spalte oder Diagonalen hat.
   */
    public static void main( String[] args)
    {
      // Spielerlaeuterung
      System.out.println
      ( "Spieler setzen abwechseln drei Steine, ");
      System.out.println
      ("gewonnen hat der Spieler, der seine Steine ");
      System.out.println
      ("in einer Zeile, Spalte oder Diagonalen hat.");

      // leeres Spielbrett
      final int laenge = 3;
      char[][] brett = new char[ laenge][ laenge];

                                        // Spielbrett leeren
      for( int z = 0; z < brett.length; z++)
        for( int s = 0; s < brett[ 0].length; s++)
          brett[ z][ s] = ' ';
```

```
// 2 Spieler
char[] spieler = { 'x', 'o'};

// Spielstart
int dran = 0;                   // aktueller Spieler
int runde = 0;                      // Rundenzaehler
boolean fertig = false;
boolean gewonnen = false;

// Spielrunde
do
{
  // aktueller Spieler
  dran = 1 - dran;
  runde++;

  System.out.println
  ( "Spieler " + spieler[ dran] + " ist dran!");
  System.out.println();

  // Stein setzen
  int z, s;
  do
  {
    do
    {
      z =
      IOTools.readInteger( "Zeile  (0<=z<=2), z = ");
    } while( z < 0 || z > 2);
    do
    {
      s =
      IOTools.readInteger( "Spalte (0<=s<=2), s = ");
    } while( s < 0 || s > 2);
  } while( brett[ z][ s] != ' ');

  brett[ z][ s] = spieler[ dran];

  // Spielbrett zeigen
  System.out.println();
  for( z = 0; z < brett.length; z++)
  {
    System.out.print( " | ");
    for( s = 0; s < brett[ 0].length; s++)
      System.out.print( brett[ z][ s] + " | ");
    System.out.println();
  }
  System.out.println();

  // Auswerten
  for( z = 0; z < brett.length; z++)  // Zeilentest
    if( brett[ z][ 0] != ' ' &&
```

```
            brett[ z][ 0] == brett[ z][ 1] &&
            brett[ z][ 0] == brett[ z][ 2])
        {
           gewonnen = true;
           break;
        }

      if( !gewonnen)                        // Spaltentest
        for( s = 0; s < brett[ 0].length; s++)
          if( brett[ 0][ s] != ' ' &&
              brett[ 0][ s] == brett[ 1][ s] &&
              brett[ 0][ s] == brett[ 2][ s])
          {
             gewonnen = true;
             break;
          }
                                // positiver Diagonalentest
      if( !gewonnen && brett[ 0][ 0] != ' ' &&
          brett[ 0][ 0] == brett[ 1][ 1] &&
          brett[ 0][ 0] == brett[ 2][ 2])
          gewonnen = true;
                                // negativer Diagonalentest
      if( !gewonnen && brett[ 0][ 2] != ' ' &&
          brett[ 0][ 2] == brett[ 1][ 1] &&
          brett[ 0][ 2] == brett[ 2][ 0])
          gewonnen = true;

      // Fertig
      fertig = runde == laenge * laenge || gewonnen;
    } while( !fertig);

    // Spielauswertung
    if( gewonnen)
      System.out.println
      ( "Sieger: Spieler " + spieler[ dran]);
    else
      System.out.println( "Patt!");

    System.out.println( "Spiel beendet");
  }
}
```

4.6 Referenzdatentypen - Klassen

Eigenschaften von Feldern

- Ein Feld fasst Daten zusammen, die für ein Programm eine Einheit bilden.
- **Alle Komponenten eines Feldes besitzen den *gleichen* Typ.**

Der letzte Punkt ist eine Einschränkung, unterschiedliche Typen sind oft wünschenswert. Im Gegensatz zu Feldern sind **Strukturen** Sammlungen von Daten *verschiedenen* Typs.

Definition Struktur

Struktur $=_{df}$ **Tupel über einer Menge von Daten (*verschiedenen* Typs)**

***Strukturen* sind *strukturierte* Datentypen, die für ein Programm eine Einheit bilden und Daten *verschiedenen* Typs zusammenfassen. Die zusammengefassten Daten sind die *Strukturelemente*.**

Strukturen werden generell in *zwei* Schritten festgelegt: Zunächst werden ein **Strukturtyp** und anschließend eine **Struktur** von diesem *Strukturtyp* deklariert.

4.6.1 Festlegen eines Strukturtyps als Klasse

In Java verwendet man eine **Klasse** zur Modellierung eines *Strukturtyps*. Die in einer Klasse zusammengefassten Variablen *verschiedenen Typs* werden als **Attribute** bzw. **Instanzvariablen** bezeichnet. Sie nehmen die *Daten (Eigenschaften) einer Struktur* auf.

Deklaration einer Klasse

public class *Klassenname* **{** *Deklaration* **. . . }**

public legt fest, dass die Klasse eine öffentliche, eine für jeden zugreifbare Klasse ist. Die *Deklarationen* legen die in der Klasse zusammengefassten Datentypen fest.

Beispiel „Datum“

Ein Datum besteht aus drei Werten, dem Tag, dem Monat und das Jahr. Man könnte es als `int` - Feld der Länge 3 auffassen. Dann muss man sich aber merken, dass die 0. Komponente der Tag, die erste Komponente der Monat und die 2. Komponente das Jahr darstellen. Als Klasse bezeichnet man die Attribute mit ihrem Namen, so dass die Daten leichter lesbar werden.

Datum.java

```
public class Datum
{
  int tag;
  int monat;
  int jahr;
}
```

Klasse Datum

Datum	
tag:	int
monat:	int
jahr:	int

Eine Klasse wird in einer *eigenen Datei* mit dem Namen der Klasse abgespeichert.

4.6.2 Festlegen einer Struktur – eines Objekts einer Klasse

Betrachten wir jetzt ein konkretes Datum, z. B. den Geburtstag von **Gottfried Wilhelm Leibniz** am 1. 7. 1646. Man benötigt eine Variable, mit der man das Datum verarbeiten kann. Solche Variablen zur Klasse nennt man auch **Objekte** dieser Klasse.

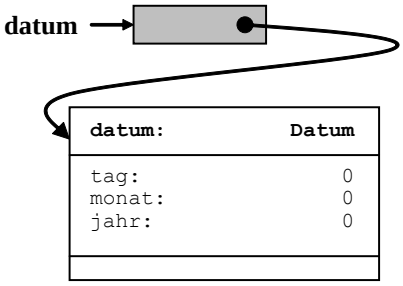
Analog den Feldern werden Objekte in zwei Schritten erzeugt:

Deklaration eines Objektes einer Klasse
Dem Compiler wird mitgeteilt, dass es sich um eine Referenzvariable handelt und Speicher für eine Adresse benötigt wird.

Klassenname Objektname **;**
```
Datum datum;
```

Definition eines Objektes einer Klasse
Der notwendige Speicherplatz wird mit Hilfe **new**-Operator bereitgestellt.

Objektname **=** **new** *Klassenname* **( ) ;**
```
datum = new Datum();
```

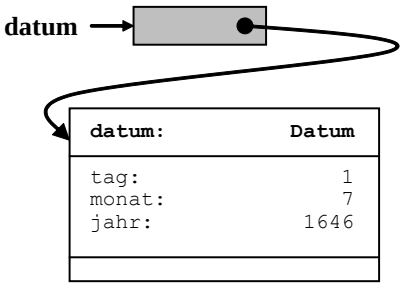


Deklaration und Definition eines Objektes einer Klasse

```
Datum datum = new Datum();
```

Zugriff auf die Attribute
Der **Zugriff** auf die einzelnen Instanzvariablen erfolgt durch den **Punkt-Operator**.

Objektname **.** *Attribut*
```
datum.tag   = 1;
datum.monat = 7;
datum.jahr  = 1646;
```



Explizite Anfangswertzuweisung (Initialisierung)
```
Datum datum = { 1, 7, 1646}
```

Objekt der Klasse Datum

Beispiel
Ein Datum soll eingegeben werden, auf Korrektheit überprüft und wieder in einer üblichen Form ausgegeben werden. Dazu ist ein Objekt Datum zu erzeugen und anschließend dessen Attributen kontrolliert einzugeben und auszulesen.

DatumsEingabe.java (Grobstruktur)
```
public class DatumsEingabe
{
  public static void main( String[] args)
  {
    // Datum
    // Datumseingabe
```

```
      // Jahr  (1600 .. 3000)
      // Monat (1 .. 12)
      // Tag   (1 .. Anzahl Tage im Monat)
    // Datumsausgabe
  }
}
```

DatumsEingabe.java

```
// DatumsEingabe.java                              MM 2009
import Tools.IO.*;                              // Eingaben

/**
 * Eingabe eines Datums,
 * Gregorianischer Kalender gilt seit 1582.
 */
public class DatumsEingabe
{
/**
 * Eingabe eines Datums,
 * Ueberpruefen der Korrektheit,
 * Ausgabe des Datum.
 */
  public static void main( String[] args)
  {
    System.out.print( "Datumseingabe");

    // Datum
    Datum datum = new Datum();

    // Datumseingabe
    do                                              // Jahr
    {
      datum.jahr =
      IOTools.readInteger( "Jahr (1600 .. 3000): ");
    } while( datum.jahr < 1600 || datum.jahr > 3000);

    do                                             // Monat
    {
      datum.monat =
      IOTools.readInteger( "Monat (1 .. 12): ");
    } while( datum.monat < 1 || datum.monat > 12);

    boolean schaltJahr = false;            // Schaltjahr
    int cc = datum.jahr / 100;
    int jj = datum.jahr % 100;
    if( jj == 0) schaltJahr = cc % 4 == 0);
    else  schaltJahr = ((jj % 4) == 0);

    int anzahl = 0;          // Anzahl der Tage im Monat
    switch( datum.monat)
    {
```

```
        case 1: case 3: case 5: case 7: case 8: case 10:
        case 12: anzahl = 31; break;

        case 4: case 6: case 9:
        case 11: anzahl = 30; break;

        case 2: if( schaltJahr) anzahl = 29;
                else anzahl = 28;
      }

      do                                                    // Tag
      {
        datum.tag =
        IOTools.readInteger( "Tag (1 .. " + anzahl + "): ");
      } while( datum.tag < 1 || datum.tag > anzahl);

      // Datumsausgabe
       System.out.println
       ( datum.tag + "." + datum.monat + "." + datum.jahr);
    }
  }
```

Da man Datumseingaben immer überprüfen muss, wäre es günstig, die Datumsüberprüfung mit der Klasse `Datum` fest zu verbinden. Diese Möglichkeit ergibt sich mit einem neuen *Programmierparadigma*, der **objektorientierten Programmierung**.