Arrays (Reihungen)

- Arrays (Reihungen) in Java: Syntax, Typisierung, Semantik.
- Wichtige Algorithmen mit Arrays
  - Arrays als Implementierung von Mengen und Listen
  - Finden von Maximum und Minimum
  - Binäre Suche (Wdh.)
  - Sortieren mit Arrays, insbes. Quicksort.
- Mehrdimensionale Arrays

Arrays: Motivation

Wir wollen 10 Preise einlesen und den niedrigsten markieren:

```
19.95
23.95
24.95
18.95 <-- niedrigster Preis
29.95
19.95
20.00
22.99
24.95
19.95
```

Alle Daten müssen eingelesen werden, bevor wir ausgeben können, daher müssen wir sie zwischenspeichern.

Dafür zehn Variablen zu verwenden wäre sehr unflexibel.

Arrays

Durch

```
double[] data = new double[10];
```

deklarieren wir ein *Array* von Double-Werten der Größe 10.

Genauer:

- Ein Array ist ein Verweis auf eine Abfolge fester Länge von Variablen des gleichen Typs, genannt *Fächer* (engl. *slots*).
- Der Typ *typ*`[]` ist der Typ der Arrays mit Einträgen vom Typ *typ*.
- Der Ausdruck `new` *typ*`[`n`]` liefert ein frisches Array vom Typ *typ* der Länge n zurück. Hier ist n ein Ausdruck vom Typ `int`.

Im Beispiel ist also `data` eine Arrayvariable, die ein frisches Array vom Typ `double` der Länge 10 enthält.

Im Rechner ist der Arrayinhalt als Block aufeinanderfolgender Speicherzellen repräsentiert. Das Array selbst ist ein Objekt bestehend aus der Länge und der Speicheradresse des Blocks.

Arrayzugriff

Durch

```
data[4] = 29.95;
```

setzen wir das Fach mit der Nummer 4 auf den Wert 29.95. Mit

```
System.out.println("Der Preis ist EUR" + data[4]);
```

können wir diesen Wert ausgeben.

`data[4]` verhält sich ganz genauso wie eine “normale” Variable vom Typ `double`.

Ist e ein Array und i ein Ausdruck des Typs `int`, so bezeichnet e`[`i`]` das Fach mit Index i des Arrays e.

Achtung: Diese Indizes beginnen bei Null. Es muss also i echt kleiner als die Länge von e sein.

Im Beispiel sind die Fächer also

```
data[0], data[1], data[2], data[3], ... , data[9]
```

Länge

Ist e ein Array, so ist e`.length` die Länge des Arrays als Integer.

Das ist nützlich, wenn man die Länge später (durch Editieren und Neucompilieren) vergrößern muss. `data.length` stellt sich dann automatisch mit um. Ein “festverdrahteter” Wert wie 10 müsste auch explizit umgeändert werden.

Die Länge wird abgerufen wie eine als `public` deklarierte Instanzvariable. Man kann sie aber im Programm nicht verändern, d.h.
`data.length = 20` ist nicht erlaubt.

Wichtiges

- Ein Array ist ein Verweis auf eine Folge von Variablen, der sog. Fächer.
- Arraytypen werden durch Anfügen von eckigen Klammern gebildet. Ein Arraytyp ist weder Objekt- noch Grunddatentyp.
- Frische Arrays werden mit `new` erzeugt; die Länge des Arrays wird in eckigen Klammern angegeben.
- Die Fächer eines Arrays werden durch Anfügen des in eckige Klammern gesetzten Index bezeichnet.
- Ein auf diese Weise bezeichnetes Fach *ist* eine Variable. Man kann ihren Wert verwenden und ihr mit `=` einen neuen Wert zuweisen.
- Arrayindizes beginnen immer bei 0.
- Die Länge eines Arrays erhält man mit `.length`

Einlesen der Daten

Wir wollen in unser Array `data` zehn Preise von der Konsole einlesen. So geht es:

```
ConsoleReader console = new ConsoleReader(System.in);
for (int i = 0; i < data.length; i++)
    data[i] = Double.parseDouble(konsole.readLine());
```

Vorsicht: Man sollte nicht `i <= data.length` schreiben, das würde zu einem Zugriffsfehler führen, da es das Fach mit Index `data.length` nicht gibt.

In Java führen Zugriffsfehler zum Programmabbruch.

In C können sie dazu führen, dass beliebige Befehle unbeabsichtigt ausgeführt werden (der gefürchtete *buffer overflow*).

Idiom für die Arrayverarbeitung

Merke: Das “Durcharbeiten” eines Arrays erfolgt meist so:

```
for (int i = 0; i < a.length; i++) {
    Bearbeiten des Faches mit Index i
}
```

Vorsicht mit Arrayvariablen

```
double[] data;
System.out.println(data[5]);
```

ist falsch. `data[5]` wurde ja nicht initialisiert.

```
double[] data;
data[5] = 7;
System.out.println(data[5]);
```

ist aber auch falsch!

“new” nicht vergessen

Der Grund ist, dass die Variable `data` selber noch gar nicht initialisiert wurde.

Man muss so einer Variable erst ein Array (= Verweis auf Folge von Variablen) zuweisen. Normalerweise macht man das mit `new`:

```
double data;
data = new double[10];
```

Man kann aber auch einen anderen Arrayausdruck zuweisen, z.B.

```
double[] kopie = data;
```

Dann aber Vorsicht mit Aliasing:

```
data[5] = 7; kopie[5] = 8; // data[5] ist jetzt 8
```

Arrays kopieren

Um eine wirkliche Kopie von `data` zu erhalten, macht man folgendes:

```
double[] kopie = new double[data.length];
for (int i = 0; i < data.length; i++)
    kopie[i] = data[i];
```

oder kürzer mit der Methode `System.arrayCopy` (siehe Doku).

Arrays variabler Länge

Oft weiß man nicht von vornherein, wie groß ein Array sein muss.

Beispiel: Benutzer gibt der Reihe nach Preise ein und hört mit 0 auf.

Man kann dann ein sehr großes Array bilden und es nur teilweise füllen.

Eine zusätzliche `int` Variable gibt an, bis wohin man gefüllt hat.

Niedrigste Preise

```
public class Preise {
    public static void main(String[] args) {
        final int DATA_LENGTH = 1000;
        BufferedReader konsole = ...;
        double[] data = new double[DATA_LENGTH];
        int dataSize = 0;
        boolean done = false;

        System.out.println("Geben Sie die Preise ein, beenden mit

        while (!done) {
            double price = console.readDouble();
            if (price == 0) // Eingabeende
                done = true;
            else if (dataSize < data.length) {
                data[dataSize] = price;
                dataSize++;
```

Niedrigste Preise

```
        } else { // Array voll
            System.out.println("Das Array ist voll.");
            done = true;
        }       }
    if (dataSize > 0) {
        double lowest = data[0];
        int lowestNo = 0;
        for (int i = 1; i < dataSize; i++)
            if (data[i] < lowest) {
                lowest = data[i];
                lowestNo = i;
            }
        for (int i = 0; i < dataSize; i++) {
            System.out.print(data[i]);
            if (i == lowestNo)
                System.out.print(" <-- niedrigster Preis");
            System.out.println(); }}}}
```

Arrays als Methodenparameter

Eine Array kann als Parameter übergeben werden:

```
public static double mittelwert(double[] zahlen) {
   if (zahlen.length == 0) return 0.0;
   double summe = 0;
   for (int i = 0; i < zahlen.length; i++)
       summe = summe + zahlen[i];
   return summe / zahlen.length;
}
```

Man kann nun etwa `mittelwert(data)` aufrufen. Wert ist der Mittelwert von `data`.

Arrays als Methodenparameter

Es wird nur das Array übergeben, d.h. der Verweis auf das Array. Man kann daher (zuweilen unerwünschte) Seiteneffekte erhalten:

```
public static double f(double[] zahlen) {
    if (zahlen.length == 0) return 23;
    else {
       zahlen[0] = 27;
       return 23;
}
```

Der Aufruf `f(data)` hat stets den Wert 23, setzt aber gleichzeitig `data[0]` auf 27.

Arrays als Rückgabewerte

Ein Arraytyp kann auch als Rückgabewert in Erscheinung treten.

Hier ist eine Methode, die die Eckpunkte eines regelmässigen n-Ecks zurückgibt:

```
static Point[] nEck(int n, Point zentrum, double radius)
/* Gibt die Eckpunkte eines regelmaessigen Polygons mit n Ec
zentrum und Radius radius aus */
{ Point[] result = new Point[n];
  /* Formeln mit sin, cos geloescht.*/
  return result;
}
```

Finden eines Wertes

Man möchte wissen, ob ein Preis ≤ 1000 ist:

```
boolean gefunden = false;
for (i = 0; i < data.length; i++)
   gefunden = gefunden || (data[i] <= 1000.);
```

jetzt ist `gefunden` true genau dann, wenn `data` einen Eintrag ≤ 100 hat.

Man möchte wissen, wieviele Preise ≤ 1000 sind:

```
int count = 0;
for (i = 0; i < data.length; i++)
   if (data[i] <= 1000.) count++;
```

Jetzt ist `count` gleich der Anzahl derjenigen ≤ 1000.

Löschen eines Wertes

Man möchte den Eintrag an der Stelle `pos` aus einem teilweise gefüllten Array löschen.

Falls die Ordnung keine Rolle spielt:

```
data[pos] = data[dataSize-1];
dataSize = dataSize - 1;
```

Falls die Ordnung beibehalten werden muss:

```
for (int i = pos; i < dataSize - 1; i++)
   data[i] = data[i+1];
dataSize = dataSize - 1;
```

Einfügen eines Elements

. . . an der Stelle pos unter Beibehaltung der Ordnung:

```
for (int i = dataSize; i > pos; i--)
    data[i] = data[i-1];
data[pos] = neuerWert;
dataSize = dataSize + 1;
```

Man muss sich immer wieder sehr genau klar machen, was hier passiert.

In der Anwendung muss man natürlich sicherstellen, dass `pos` nicht ausserhalb der Grenzen liegt.

Arrays von Objekten

Hat man mehrere gleichlange Datensätze, so bietet es sich an, sie als ein einziges Array, dessen Einträge Objekte sind, zu repräsentieren:

Beispiel: Eine Liste von Automodellen, die Liste der zugehörigen Preise, die Liste der zugehörigen PS-Zahlen.

```
public class Auto {
   private String modell;
   private double preis;
   private double psZahl;
   /* Methoden und Konstruktoren */
}

Auto[] liste = ...
```

Arrays als Instanzvariablen

Arrays können auch als Instanzvariablen eines Objektes in Erscheinung treten.

Es empfiehlt sich, dann im Konstruktor auch gleich ein frisches Array zu erzeugen.

Beispiel:

```
public class Polygon {
  private int n; // Zahl der Ecken
  private Point[] ecken; // Liste der Ecken
/* Methoden und Konstruktoren */
}
```

Zweidimensionale Arrays

Die Einträge eines Arrays können wieder Arrays sein. Das gibt ein zweidimensionales Array.

```
int[][] einmaleins = new int[10][10];
for (int i = 0; i < 10; i++)
    for (int j = 0; j < 10; j++)
        einmaleins[i][j] = (i+1) * (j+1);
```

Object

Es gibt den Typ `Object`.

Ein Objekt einer beliebigen Klasse wird bei Bedarf auf Typ `Object` automatisch konvertiert.

Ein Ausdruck des Typs `Object` kann durch Vorschalten von `(`*typ*`)` auf Typ *typ* konvertiert werden.

Hier muss *typ* eine Klasse sein, also z.B. `Bankkonto` und kein primitiver Typ wie `double`.

War der Ausdruck in Wirklichkeit gar kein Objekt der Klasse *typ*, dann wird das Programm abgebrochen.

Künstliches Beispiel

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

a[0] = new Point(3,4);
a[1] = new Bankkonto();
```

Jetzt wäre `(Point)a[0]` ein Ausdruck vom Typ `Point`, dessen Wert der soeben frisch erzeugte Punkt (3,4) ist.

Der Ausdruck `(Point)a[1]` ist auch vom Typ `Point`. Der Compiler akzeptiert ihn, aber der Versuch ihn auszuwerten führt zu Programmabbruch.

Übung

- Was ist hier falsch?

```
int[] v = new int[10];
for (int i = 1; i <= 10; i++) v[i] = i*i;
```

- Beispiele für sinnvolle Methode, die
  - einen Arrayparameter vom Typ `int[]` hat, der modifiziert wird.
  - einen Arrayparam. vom Typ `int[]` hat, der nicht modifiziert wird.
  - die ein Array vom Typ `int[]` zurückgibt.
- Man schreibe eine Schleife, die ein Array wie folgt besetzt: 1 4 9 16 25 36 49 64 81 100
- Man konstruiere ein 2D-Array von Booleans, das schachbrettartig vorbesetzt ist.
- Überprüfen, ob zwei Arrays dieselben Elemente in derselben Reihenfolge enthalten.