Einführung in die Programmierung mit Java

Teil 6: Arrays

Martin Hofmann Steffen Jost

LFE Theoretische Informatik, Institut für Informatik,
Ludwig-Maximilians Universität, München

21. November 2017

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

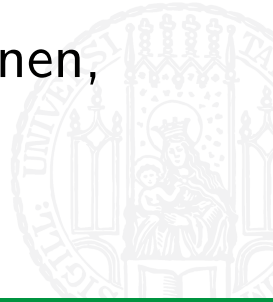
Teil 6: Arrays

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 der Länge *n* zurück, mit Einträgen des Typs *typ*. Hier ist *n* ein Ausdruck vom Typ `int`.

Durch

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

In diesem 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:

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

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
}
```

Besondere For-Schleife

Will man auf die Fächer nur lesend zugreifen, so kann man die folgende Notation verwenden:

```
for (double x : a) {c}
```

ist äquivalent zu

```
for (int i = 0; i < a.length; i++) {
    double x = a[i];
    c
}
```

Zur Initialisierung von Arrays oder zu anderer schreibender Bearbeitung ist diese Notation wertlos:

```
for (double x : a) {x = 3.141;}
```

lässt `a` unverändert und hat auch sonst keine sichtbare Wirkung.

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;
        Scanner konsole = new Scanner(System.in);
        double[] data = new double[DATA_LENGTH];
        int dataSize = 0;
        boolean done = false;

        System.out.println("Geben Sie die Preise ein,  \
                           beenden mit 0");
```

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

Ein 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 Ecken, Zentrum 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 ≤ 1000 hat.
Man möchte wissen, wieviele Preise ≤ 1000 sind (hier zur Abwechslung mit der neuen Notation):

```
int count = 0;
for (double fach : data)
   if (fach <= 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 {
   /* Instanzvariablen hier ausnahmsweise public */
   public String modell;
   public double preis;
   public double psZahl;
   /* Methoden und Konstruktoren */
}

Auto[] liste = ...
```

Verändernder Zugriff mit der neuen For-Schleife

Folgender Code erhöht alle Preise um 3%:

```
for(Auto auto : liste)
    auto.preis = auto.preis * 1.03;
```

Hier wird `auto` der Reihe nach an die *Werte* der Fächer von `liste` gebunden. Mithilfe dieser Werte, die ja Objektverweise sind, kann man dann *schreibend* auf die Objekte selbst zugreifen.

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 */
}
```

Vergrössern eines Arrays

Wird ein teilweise gefülltes Array zu klein, so kann man es in ein größeres neu allokiertes Array umkopieren.
Es ist üblich, das neue Array von jeweils doppelter Größe zu wählen.

Die Klasse ArrayList

Die vordefinierte Klasse `java.util.ArrayList` stellt Arrays *variabler Größe* mit Methoden zum Einfügen, Löschen, etc. bereit.
Einschränkung: in einer `ArrayList` können nur Objekte stehen, keine Werte von Grunddatentypen wie `int, double, boolean`.
Will man `int`egers in einer `ArrayList` verwalten, so muss die Wrapperklasse `Integer` verwendet werden. Seit Java 1.5 werden `int`s automatisch in `Integer`s konvertiert. Auto Boxing/Unboxing

Die Klasse `ArrayList` kann also mit dem Typ der Einträge parametrisiert werden: `ArrayList<Auto>` ist die Klasse der ArrayListen, welche `Auto`-Objekte enthalten.
Anwendungsbeispiel: Definition einer Klasse `Polygon`, die eine Liste von Punkten enthält repräsentiert durch `ArrayList<Point>`. Zeichnen im GraphicsWindow, Berechnung der Fläche.

Die Klasse ArrayList

- Der Typ der Elemente einer `ArrayList` wird in spitze Klammern gefasst: `ArrayList<Point>`
  Dies soll uns momentan reichen, weiteres zu diesem Thema folgt dann in Kapitel 12. "generische Klassen"
- `ArrayList` ist in Java mittlerweile die bessere Wahl im Vergleich zu den allgemeineren, einfachen Arrays;
  z.B. benötigen Arrays zur Laufzeit noch Typprüfungen.
- `for`-Schleifen Notation kann genauso benutzt werden:

```
ArrayList<Point> al = new ArrayList<Point>();
al.add(new Point(1,2));
al.add(new Point(3,4));
for (Point p : al) {
    System.out.print(p);
}
```

Schreibt: "`java.awt.Point[x=1,y=2]java.awt.Point[x=3,y=4]`

Arrays vs. ArrayList

```
double[] ary = { 1.2, 3.4 };                    // Erstellen
                                                //  & Initialisieren
double d  = ary[0];                             // Zelle lesen
ary[1] = 6.9;                                   // Zelle schreiben
System.out.println("Size:" + ary.length);      // "Size:2"
System.out.println(ary);                        // "[D@135fbaa4"
```

Klassische Arrays vermeiden und besser `ArrayList` verwenden:

```
List<Double> ary= new ArrayList<Double>();  // Erstellen
Collections.addAll(ary, 1.2, 3.4);          // Initialisieren
Double d = ary.get(0);                      // Zelle lesen
ary.set(1,6.9);                             // Zelle schreiben
System.out.println("Size:" + ary.size());   // "Size:2"
System.out.println(ary);                    // "[1.2, 6.9]"
```

Beachte:

1. Andere Initialisierung
2. `Double` statt `double`
3. `size()` statt `length`

Vorteile:

1. `toString()` geht nun
2. `add(index,wert)` möglich
3. `remove` möglich

Arrays vs ArrayList

Klassische Arrays sind in Java primär aus historischen Gründen enthalten. `ArrayList` hat zahlreiche Vorteile:

- Größenänderung einfacher
- Bessere Typsicherheit späteres Kapitel „Generics“
- Bessere Wartbarkeit des Codes, da leichter auszutauschen gegen andere Datenstruktur

Nachteile sind:

- Aufwändigere Syntax durch reguläre Methodenaufrufe, z.B. `ary.set(7,8);` statt `ary[7]=8`
- Elemente müssen Objekte einer Klasse sein, d.h. `Double` statt `double`, `Integer` statt `int`, `Boolean` statt `bool`, etc.
- `ArrayList` ist immer 1-dimensional; möglich aber meist umständlich: `ArrayList<ArrayList<Double>>`

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);
```

Zusammenfassung Arrays

- Ein Array ist eine Folge fester Länge von Variablen ein und desselben Typs.
- Arrays werden verwendet um Mengen und Listen von Daten zu repräentieren.
- Mit For-Schleifen kann der Reihe nach auf die Fächer eines Arrays zugegriffen werden. Für lesenden Zugriff auf Arrayfächer gibt es eine besondere Kurzform der For-Schleife
- Teilweise gefüllte Arrays repräsentieren Listen variabler Größe. Durch Neuallokieren und Umkopieren können diese auch scheinbar beliebig vergrößert werden.
- Die Klasse `ArrayList` stellt diese Funktionalität zur Verfügung.
- Zweidimensionale Arrays sind Arrays, deren Fächer selbst wieder Arrays sind.