Arrays

In Java werden Datenstrukturen, die null oder mehr Elemente enthalten können, als Kollektionen (Collections) bezeichnet. Ein Array ist eine Kollektion (Collection), die eine **feste Größe** hat und deren `Elemente alle vom gleichen Typ` sein müssen. Elemente können einem Array zugewiesen oder aus ihm mithilfe eines **Index** abgerufen werden. Java-Arrays verwenden eine nullbasierte Indizierung: Der Index des ersten Elements ist 0, der des zweiten Elements ist 1, usw.

Hier ist die vollständige Syntax zur Initialisierung eines Arrays mit dem Namen `arrayName`:

```
type[] arrayName = new type[size];
```

`type` legt den Datentyp der Elemente im Array fest, das kann ein primitiver Datentyp (z.B. int) oder eine Klasse (z.B. String) sein – auch Datentypen, die aus eigenen Klassen abgeleitet sind, sind selbstverständlich erlaubt. Die beiden eckigen Klammern nach dem Typ (`[]`) legen fest, dass hier ein Array definiert werden soll.

`size` ist die Größe des Arrays - das ist die Anzahl der Elemente, die dieses Array aufnehmen kann. Die Größe kann später nicht mehr geändert werden. Nach der Erstellung des Arrays werden die Elemente auf ihre Standardwerte initialisiert (typischerweise 0, false oder null).

Beispiele:

- `int[] zahlen = new int[22];` - Ein Array mit dem Variablen-Namen `zahlen`, das 22 Integer Zahlen speichern kann.
- `double[] kommazahlen = new double[8];` - Ein Array mit dem Variablen-Namen `kommazahlen`, das 8 Zahlen vom Typ "double" speichern kann.
- `String[] saetze = new String[5];` - Ein Array mit dem Namen `saetze`, das 5 Zeichenketten speichern kann.
- `Auto[] fuhrpark = new Auto[200];` - Ein Array mit dem Namen `fuhrpark`, das 200 Objekte des Typs `Auto` speichern kann. Die Klasse `Auto` muss man natürlich selbst definieren, damit das funktioniert.

Arrays können auch mithilfe einer **Kurznotation** definiert werden, die es ermöglicht, das Array gleichzeitig zu erstellen und zu initialisieren:

```
// Die beiden folgenden Zeilen machen dasselbe:
// Sie deklarieren ein Array, das ganze Zahlen speichern kann
// und initialisieren die Element direkt als 4, 9 und 7
int[] dreiZahlenV1 = new int[] { 4, 9, 7 };
int[] dreiZahlenV2 = { 4, 9, 7 };
```

Die Länge des Arrays wird durch die Initialisierung mit drei Elementen auf 3 festgelegt und kann, wie immer bei Arrays, anschließend nicht mehr geändert werden.

```
int[] zahlenarray = {11,32,42,2,4};
```

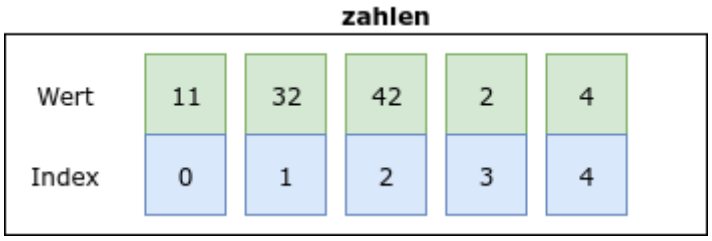
Ein Array kann man als Instanzen einer spezielle Klasse verstehen. Array werden also als Objekte behandelt und müssen durch den Operator new instanziiert werden. Die Array-Klasse bringt spezielle Methoden und Operationen mit, auf die man beim Umgang mit Arrays zurückgreifen kann, z.B liefert die Methode `length`, die Länge des Arrays zurück:

```
zahlen.length // in unserem Fall: 5
```

1)

Zugriff auf Array-Elemente

Jedes Element eines Array hat einen Wert und einen Index. **Die Zählung des Index beginnt immer bei Null**. Für unser Beispiel-Array sieht das also folgendermaßen aus:



Über den Index eines Elements, kann man auf dessen Wert zugreifen:

```
zahlen[1] // hier: 32
zahlen[4] // hier: 4
```

Mit einer entsprechenden Wertzuweisung kann auf diese Art auch der Wert eines Arrayelements gesetzt werden:

```
zahlen[1] = 33;
zahlen[4] = 78;
```

Der Zugriff auf einen Index, der außerhalb der gültigen Indizes des Arrays liegt, führt zu einer `IndexOutOfBoundsException`.

Alle Array-Elemente sequentiell verarbeiten

Häufig möchte man alle Elemente eines Array der Reihe nach anschauen und möglichweise etwas mit den gespeichert Werten machen, man spricht davon, dass man "über das Array iteriert".

Hier bieten sich zunächst **zwei Möglichkeiten** an:

Wenn man die volle Kontrolle haben möchte über welche Elemente des Arrays man iteriert, verwendet man am besten eine Zählschleife (for-Schleife). Bei dieser Art, die Array Elemente

auszulesen, hat mal stets auch den Wert des entsprechenden Index vorliegen:

```
char[] vowels = { 'a', 'e', 'i', 'o', 'u' };

for (int i = 0; i < 3; i++) {
    // Output the vowel
    System.out.print(i + "::" + vowels[i] + " ");
}

// 0::a 1::e 2::i
```

Um ein Array auf diese Weise vollständig zu durchlaufen, muss der Kopf der for-Schleife so aussehen:

```
for (int i = 0; i < meinArray.length; i++) {
  ...
}
```

Der Umstand, dass ein Array auch eine Java-"Collection" ist, bedeutet, dass du neben dem direkten Zugriff auf Werte über den Index auch über alle Werte iterieren kannst, indem du eine "for-each"-Schleife verwendest:

```
char[] vowels = { 'a', 'e', 'i', 'o', 'u' };

for(char vowel: vowels) {
    // Output the vowel
    System.out.print(vowel);
}

// => aeiou
```

Vogelbeobachtung

Du bist ein begeisterter Vogelbeobachter und führst Buch darüber, wie viele Vögel in den letzten sieben Tagen deinen Garten besucht haben.

Die Zahl der beobachteten Vögel speicherst du in einem Array, ein befreundeter Programmierer hat dir bereits eine Codevorlage zur Verfügung gestellt 🔧 Fix Me!

(A1)

Passe die Methode `getNumbersOfLast7Days()` so an, dass diese die Beobachtungszahlen für die vergangenen 7 Tage zurück gibt. Die Zahlen waren wie folgt:

Vor 6 Tagen (Sa)	Vor 5 Tagen (So)	Vor 4 Tagen (Mo)	Vor 3 Tagen (Di)	Vorgestern (Mi)	Gestern (Do)	Heute (Fr)
12	7	0	6	15	8	7

Mache dir klar, dass die Methode als Rückgabetyp tatsächlich ein Array aus Zahlen festlegt.

Für die weiteren Aufgaben gilt, dass das Array `birdsPerDay` die Zahlen für vergangenen 7 Tage enthält.

(A2)

Implementiere die Methode `Birdwatching.getToday()`, um zurückzugeben, wie viele Vögel heute deinen Garten besucht haben. Die Vogelzählungen sind im Array `birdsPerDay` nach Tagen geordnet, wobei das erste Element die Anzahl des ältesten Tages ist und das letzte Element die Anzahl für heute darstellt.

(A3)

Implementiere die Methode `Birdwatching.incrementTodaysCount()`, um die Anzahl der Vögel von heute um eins zu erhöhen.

(A4)

Implementiere die Methode `Birdwatching.getCountForFirstDays()`, die die Anzahl der Vögel zurückgibt, die seit Beginn der Woche deinen Garten besucht haben, aber die Zählung auf die angegebene Anzahl von Tagen ab dem Beginn der Woche begrenzt.

—

(A2)

Erweitere das Programm aus (A1) so, dass für alle Elemente des Arrays eine Zeile wie die folgende ausgegeben wird:

```
Das Array-Element mit dem Index 0 hat den Wert 11
Das Array-Element mit dem Index 1 ...
Das Array-Element mit dem Index 2 ...
```

Verwende dazu eine Zählschleife (for-Schleife). Welches ist der Startwert, welches der Endwert, den diese Schleife haben muss?

(A3)

Schreibe eine Methode, die den Wert des dritten Elements des Arrays ausgibt.

(A4)

Verändere im Schleifenkopf der for-Schleife aus Aufgabe 2 den Vergleichsoperator, indem du ihn umkehrst (> statt <). Erläutere die Ausgabe.

(A5)

Erzeuge ein Array vom Datentyp String, das die Wochentage enthält. Lasse ebenfalls direkt bei der Instanziierung auf der Konsole die Länge des Arrays ausgeben. Beim Aufrufen von printArray() sollen nun auch die Wochentage ausgegeben werden.

---

arrays.odp	96.8 KiB	08.11.2022 14:39
arrays.pdf	100.8 KiB	08.11.2022 14:40
robin-6906521_640.jpg	27.9 KiB	09.10.2024 18:36

1)

Java verfügt auch über Klassen, mit denen man Objektlisten und Arrays effizienter verwalten kann, fürs Erste begnügen wir uns mal mit dem Weg "zu Fuß"

From:
https://www.info-bw.de/ -

Permanent link:
https://www.info-bw.de/faecher:informatik:oberstufe:java:algorithmen:arrays:definition:start?rev=1728495191

Last update: **09.10.2024 17:33**