gds2 Gottlieb-Daimler-Schule 2
Technisches Schulzentrum Sindelfingen
mit Abteilung Akademie für Datenverarbeitung

Inhaltsverzeichnis

1 Begriffsbestimmung

Ein Array oder Feld ist eine lineare Anordnung von Variablen mit gleichem Datentyp.

- Es hat einen **Namen** und einen **Datentyp**. Der Datentyp des Arrays gibt an, welche Variablen im Array gespeichert werden dürfen.
- Arrays werden durch eckige Klammern gekennzeichnet.
- Die einzelnen Variablen im Array werden über den **Namen des Arrays** und einen numerischen **Index** angesprochen. Der Index beginnt bei 0.
- In Java werden Arrays als Objekte betrachtet. Dies hat Auswirkungen auf die Arbeit mit Arrays:
  - neue Arrays werden zur Laufzeit über den Operator **new** erzeugt
  - Array-Variablen sind Referenzvariablen
  - Arrays haben Methoden und Attribute
  - Die **Anzahl der Werte**, die in einem Array gespeichert werden können, **muss bei der Erzeugung festgelegt werden** und kann später nicht mehr verändert werden (=semi-dynamisch). Die Anzahl ist durch das finale Objektattribut `ArrayName.length` im Programm abrufbar.

2 Umgang mit Arrays

2.1 Deklaration und Initialisierung

Die Deklaration eines Arrays in Java erfolgt in zwei Schritten:

1. ***Deklaration einer Array-Variablen*** (Referenzvariable)
   erfolgt durch Anhängen von eckigen Klammern an den Datentyp:
   ```
   int [] aiIntArray;
   double [] adDoubleArray;
   Kunde [] aKunden;
   ```
   Das eigentliche Array und seine Größe brauchen bei der Deklaration noch nicht erzeugt zu werden. Das kann später erfolgen.

2. ***Erzeugen eines Arrays und Zuweisung an die Array-Variable***:

   ```
   aiIntArray = new int [10]; // Array mit Platz für 10 int-Werten
   adDoubleArray = new double [20]; // Platz für 20 double-Werte
   aKunden = new Kunde[100]; // Platz für 100 Referenzen auf
                             // Kunden-Objekte
   ```

   Alternativ: Deklaration und Erzeugung in einem Schritt:

   ```
   int[] aiIntArray = new int [10];
   ```

   Alternativ: Deklaration und direkte Initialisierung mit Werten
   Alternativ dazu kann ein Array auch beim Anlegen mit Werten initialisiert werden:

   ```
   int [] aiZahlen = {7,2,6,4,5}; // Array mit Platz für 5 int-Werte
   ```

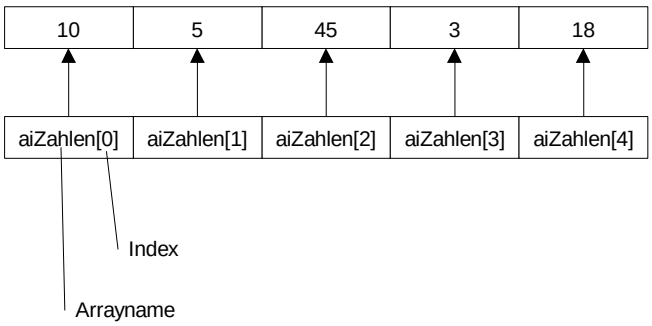
   Zu beachten:
   - der new-Operator ist nicht notwendig

2.2 Zugriff auf Array-Elemente

Der Zugriff auf die Array-Elemente erfolgt über einen Index. Dieser nummeriert die *n* Elemente aufsteigend durch, beginnend bei 0, also von 0 bis n-1. Der Index wird in eckige Klammern geschrieben:

Beispiel:

```
int[] aiZahlen;             // Deklaration der Arrayvariablen
aiZahlen    = new int [5]; // Erzeugung des Arrays
aiZahlen[0] = 10;           // Zugriff auf die Arrayelemente
aiZahlen[1] = 5;
aiZahlen[2] = 45;
aiZahlen[3] = 3;
aiZahlen[4] = 18;
```



Der Array-Index muss vom Typ int sein (wg. automatischer Typkonvertierung auch short, byte und char möglich). Jedes Array hat eine Objektattribut `length`, das die Anzahl seiner Elemente enthält. Die Einhaltung der Grenzen des Arrays wird zur Laufzeit automatisch überwacht. Indexausdrücke kleiner als 0 und größer gleich `length` erzeugen einen Laufzeitfehler ("ArrayIndexOutOfBoundsException", siehe Informationsblatt Fehlerbehandlung/Exceptions; folgt später).

Beispiel zur Bearbeitung eines Arrays mithilfe einer Schleife: „Notenschnitt berechnen“

```
public class ArrayEinstieg_v2
{
	/**
	 * @param args
	 * Kurzbeschreibung: Notenschnitt berechnen
	 */
	public static void main(String[] args)
	{
		double dNote;
		double [] adNoten = new double [100];
		int iAnzahl = 0;
		double dSchnitt = 0;

		do
		{
			dNote = Eingabe.getDouble("Note" + (iAnzahl + 1) + " = ");
			if (dNote > 0)
			{
				adNoten[iAnzahl] = dNote;
				dSchnitt = dSchnitt + dNote;
				iAnzahl++;
			}
		} while (dNote > 0 && iAnzahl < adNoten.length);

		if (iAnzahl > 0)
		{
			dSchnitt = dSchnitt / iAnzahl;

			System.out.print("Ihre Noten: " );

			for (int i = 0; i < iAnzahl; i++)
			{
				System.out.print(adNoten[i] + " ");
			}
			System.out.printf("ergeben den \nNotendurchnitt = %4.2f\n",
					dSchnitt);
		}
	}
}
```

2.3 Arrays von Objekten

Arrays können auch Referenztypen aufnehmen. Deklaration und Initialisierung sind genauso wie bei einfachen Datentypen.

Beispiel:
```
Kunde [] alleKunden = new Kunde [10];
```

Erzeugt ein Array mit Platz für 10 Referenzen auf Kunden-Objekte.

Achtung:
Da die eigentlichen Array-Elemente in diesem Fall Referenzvariablen sind, ist in dem Array nur Platz für 10 *Referenzen* auf die Kundenobjekte vorhanden. Die Objekte selbst müssen erst noch erzeugt und ihre Referenz dem jeweiligen Arrayelement zugewiesen werden:

```
alleKunden[0] = new Kunde();
alleKunden[1] = new Kunde();
```
etc.

Der Zugriff auf die Kundenobjekte kann jetzt über die Arrayelemente indiziert erfolgen

Beispiel:
```
alleKunden[0].setName("Maier");
alleKunden[1].setName("Müller");
```

etc.

2.4 Arbeiten mit Arrays

In den Standardklassen System und Arrays gibt es viele Methoden, um typische Operationen mit Arrays auszuführen.

2.4.1 Kopieren

Muss ein Array kopiert werden, so kann dies in einer for-Schleife geschehen oder es können Methoden aus Standardklassen benutzt werden:
z.B.
`System.arraycopy (...)`

- kann ganze Arrays oder Teile davon kopieren
- Ziel kann das gleiche oder ein anderes Array sein
- Das Ziel-Array muss bereits existieren und die notwendige Länge haben, wird also nicht in der Methode erzeugt
- die Bereiche dürfen sich überlappen

```
import java.util.*;
public class ArrayTestKopie1
{
  public static void main(String[] args)
  {
     int [] aiQuelle ={19,2,13,4,55,26,87,110};
     int [] aiZiel   = new int[aiQuelle.length];

     System.arraycopy(aiQuelle,0, aiZiel,0,aiQuelle.length);
     System.out.println(Arrays.toString(aiZiel));
  }
}
```

Ähnliches leisten: `Arrays.copyOf (...)` und `Arrays.copyOfRange(...)`

```
import java.util.*;
public class ArrayTestKopie2
{
  public static void main(String[] args)
  {
     int [] aiQuelle ={19,2,13,4,55,26,87,110};
     int [] aiZiel;  // Das Array wird in der Methode erzeugt

     aiZiel = Arrays.copyOf(aiQuelle, aiQuelle.length);
     System.out.println(Arrays.toString(aiZiel));
     aiZiel = Arrays.copyOfRange(aiQuelle, 0, 5);
     System.out.println(Arrays.toString(aiZiel));

  }
}
Ausgabe:
[19, 2, 13, 4, 55, 26, 87, 110]
[19, 2, 13, 4, 55]
```

2.4.2 Suchen

Auf Arrays lassen sich einfach verschiedene Suchalgorithmen anwenden, z.B. die sequentielle Suche oder die binäre Suche.

Die **Standard-Klasse Arrays** stellt zahlreiche Klassenmethoden für den Umgang mit Arrays von einfachen Datentypen zur Verfügung.

```
public static int binarySearch(int []a, int key);
Beispiel:
import java.util.*;
public class ArrayTestSuchen
{
  public static void main(String[] args)
  {
     int [] aiZahlen ={1,2,3,4,5,6,7,10};
     System.out.println(Arrays.binarySearch(aiZahlen,2));
  }
}
Ausgabe: 1
```

Hinweise:

- **Achtung:** `binarySearch` liefert nur dann ein definiertes Ergebnis, wenn das Array vorher sortiert wurde.
- Als Rückgabe liefert die Methode den Index des gesuchten Wertes (im Bsp. Wert = 2, Index = 1). Ist der gesuchte Wert mehrfach vorhanden, ist nicht definiert, von welchem Exemplar der Index zurückgegeben wird.
  Ist der gesuchte Wert nicht im Array enthalten, so ist der Rückgabewert negativ. (Siehe Java Doku)
- Da die binäre Suche eine Sortierung voraussetzt, ist sie nur dann effizient, wenn viele Suchvorgänge geplant sind, da „Sortieren“ ein aufwendiger Vorgang ist.

2.4.3 Sortieren von Arrays mit Elementen primitiver Datentypen

Für alle primitiven Datentypen bietet die Klasse Arrays entsprechende Klassenmethoden zum Sortieren an.

Beispiel:

```
import java.util.*;
public class ArrayTestSortieren
{
  public static void main(String[] args)
  {
     int [] a = {19,2,13,4,55,26,87,110};

     System.out.println(Arrays.toString(a));
     Arrays.sort(a);
     System.out.println(Arrays.toString(a));

  }
}
```

Ausgabe:
```
[19, 2, 13, 4, 55, 26, 87, 110]
[2, 4, 13, 19, 26, 55, 87, 110]
```

Da Arrays Referenztypen sind, wird bei einer Parameterübergabe jeweils die Referenz auf das Array übergeben. Somit kann der Inhalt des übergebenen Arrays sortiert werden und es ist keine Rückgabe notwendig.

2.4.4 Sortieren von Arrays mit Elementen von Referenztypen

Beim Sortieren von Arrays deren Elemente Objekte einer Klassen sind, stellt sich die Frage, nach welchem Sortierkriterium die Sortierung erfolgen soll. Enthält ein Array zum Beispiel Objekte der Klasse Kunde als Element, so ist nicht generell klar, nach welchem Objektattribut die Sortierung erfolgen soll. Die Sortierung könnte alphabetisch nach dem Nachnamen, dem Vornamen, der Adresse usw. erfolgen. Ebenso gut könnte die Sortierung aber auch numerisch nach der Gesamtbestellsumme, dem Rabattsatz oder dem Zahlungsziel erfolgen.

Bei primitiven Datentypen ist der Fall klar, die Sortierung erfolgt anhand der numerischen Ordnung (<, >, =). Bei Strings wird die lexikalische Reihenfolge durch die bereits bekannt `compareTo` Methode festgelegt (Signatur: `int compareTo(String s2)`; siehe 16_OOP_Standardklassen).

Zum Sortieren von Arrays deren Elemente Objekte einer Klassen sind, muss der Klasse eine eigene `compareTo` Objektmethode hinzugefügt werden (Signatur: `int compareTo(Klasse objekt2)`.

Beispiel für die Klasse Kunde, wenn nach Bestellsumme sortiert werden soll:

```
public class Kunde implements Comparable<Kunde>
{
	// Anfang Attribute
	private String sName;
	private String sVorname;
	private double dGesamtBestellsumme;
...

@Override
	public int compareTo(Kunde andererKunde)
	{
		if (this.dGesamtBestellsumme > andererKunde.dGesamtBestellsumme)
			return 1;
		else
			if (this.dGesamtBestellsumme < andererKunde.dGesamtBestellsumme)
				return -1;
			else
				return 0;
	}
...
}
```

Damit die `Arrays.sort()` Methode für das Kunden Array funktioniert, muss die Klasse Kunde ein sogenanntes Interface implementieren (`implements Comparable<Kunde>` Zu Interfaces

in Java siehe später). Außerdem muss die Methode `compareTo` mit der entsprechenden Signatur zur Klasse hinzugefügt werden. Diese Methode definiert die gewünschte Ordnung der Objekte der Klasse. Sie vergleicht den Inhalt des aktuellen Objekts (`this`) mit dem zweiten Objekt (hier `andererKunde`); ist `andererKunde` „größer“, so ist das Ergebnis < 0, bei „gleichen“ Kunden = 0, sonst > 0.

2.4.5 Löschen

Zum „Entfernen“ einzelner Einträge aus einem Array, stehen in den oben benutzten Standardklassen keine Methoden zur Verfügung. Hierfür muss man auf einen eigenen Algorithmus zurückgreifen, der alle nachfolgenden Elemente nach vorne verschiebt.

Alternativ kann man aber andere Datenstrukturen verwenden, wie die Klasse ArrayList, die im Zusammenhang mit Collections besprochen wird (siehe Informationsblatt zu Collections, folgt später).

2.4.6 „For-each“ Schleife[1] (Iterationsschleife) für Arrays (und Collection-Klassen)

Seit Java 5 gibt es zusätzlich zu den bereits bekannten Schleifenstrukturen eine spezielle For-Schleifen-Syntax um ein Array sequentiell von „vorne nach hinten“ zu bearbeiten.

Diese Schleifenart ist nur dann sinnvoll anzuwenden, wenn man **alle Elemente** eines Arrays nacheinander verarbeiten möchte. Für den Fall, dass man nur einen Teilbereich eines Arrays durchlaufen möchte oder in umgekehrter Reihenfolge, so muss man wieder auf die „normale“ Zählschleife zurückgreifen.

Beispiel:

```
public static void main(String[] args)
{
      char[] acListe = {'H', 'a', 'l', 'l', 'o'};
      for (char c : acListe)
      {
            System.out.print(c);
      }
}
```

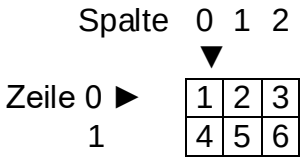
Ausgabe: „`Hallo`“

[1] „For-each“ von engl. „Für jedes (Element)“; in einigen Programmiersprachen wird statt dem Schlüsselwort `for` das Schlüsselwort `foreach` verwendet. Dieser Schleifentyp ist auch für sogenannte Collection Klassen anwendbar (siehe später)

2.5 Mehrdimensionale Arrays

Mehrdimensionale Arrays werden erzeugt, indem 2 oder mehr Klammernpaare angegeben werden, wobei selten mehr als 3 Dimensionen angewendet werden.
Ein 2-dimensionales Array entspricht einer Tabelle, ein 3-dimensionales Array einem Würfel.
Beispiel:
2-dimensionales Array mit 2 Zeilen und 3 Spalten:

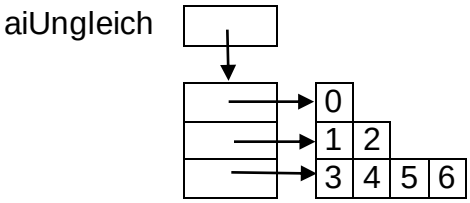
```
int [][] aiTabelle = new int [2][3];
aiTabelle[0][0] = 1;
aiTabelle[0][1] = 2;
aiTabelle[0][2] = 3;
aiTabelle[1][0] = 4;
aiTabelle[1][1] = 5;
aiTabelle[1][2] = 6;
```



Auch hier ist literale Initialisierung möglich. Die Zuweisung wird dann von links oben nach rechts unten vorgenommen.

Mehrdimensionale Arrays werden als geschachtelte Arrays gespeichert. Das heißt: das Array der ersten Dimension enthält Verweise auf die Arrays der zweiten Dimension („Array of Arrays“). Dadurch sind auch nicht-rechteckige Arrays möglich:

```
int [][] aiUngleich = {{0},
                       {1,2},
                       {3,4,5,6}
                      };
```



Die Länge der einzelnen Zeilen kann jeweils über `length` ermittelt werden.

```
int iZeile, iSpalte;
for (iZeile = 0; iZeile < aiUngleich.length; iZeile++)
{
  for (iSpalte = 0; iSpalte < aiUngleich[iZeile].length; iSpalte ++)
  {
    System.out.printf("%2d", aiUngleich[iZeile][iSpalte]);
  }
  System.out.println();
}
// Alternative mit "for each"
for (int[] aArray : aiUngleich)
{
     for (int i : aArray)
     {
          System.out.printf("%2d", i);
     }
     System.out.println();
}
```

Ausgabe:

```
 0
 1  2
 3  4  5  6
 0
 1  2
 3  4  5  6
```