universität innsbruck

Arrays

Programmiermethodik

Lukas Kaltenbrunner, Simon Priller
Universität Innsbruck

Arrays

- Zusammenfassung gleichartiger Elemente
- Im Array können primitive Datentypen oder Referenztypen verwaltet werden.
- Arrayvariablen sind in Java **Referenzvariablen** (Referenz auf Speicherplatz eines Objektes).
- Arrays haben eine fixe Länge.
  - Nach dem Anlegen kann die Länge nicht mehr verändert werden.
  - Für Container mit veränderbarer Größe existieren in Java spezielle Klassen!
  - Die Arraylänge kann durch die finale Objektvariable `length` ermittelt werden.
- Jedem Element ist ein Index vom Typ `int` zugewiesen.
  - Indexzählung beginnt bei 0!
  - Indexzählung geht bis Länge-1!
- Eindimensionale Arrays enthalten Elemente vom Elementtyp.
- Mehrdimensionale Arrays enthalten weitere Arrays!

Deklaration von Arrays

```
int array[], x, y, testArray[]; // arrays and variables
int[] array, testArray;         // arrays
```

- Deklaration
  - Es werden nur die Referenzvariablen angelegt.
  - Es wird noch kein Speicherplatz reserviert.

- Best practice
  - Deklarationen aufteilen
  - Möglichst nur eine Deklaration pro Zeile (Lesbarkeit)

Anlegen von Arrays – ohne Initialisierung

```
// Anlegen bei Deklaration
int[] array = new int[10];
```

```
// Späteres Anlegen
int[] array;
…
array = new int[10];
```

- Die Größe des Arrays kann mit `array.length` abgefragt werden.
- Werte der Elemente werden zunächst mit Standard-Werten belegt (z.B. 0 bei Integer-Array).

Anlegen von Arrays – mit Initialisierung

```
int[] array = {3, 4, 5, 6, 7};

// or

int[] array;
…
array = new int[]{3, 4, 5, 6, 7};
```

Das Array hat die Länge 5 und enthält die Elemente 3, 4, 5, 6, 7.

Arrayindex

- Arrayelemente werden über einen Index angesprochen.
- Der Index muss nicht unbedingt eine Konstante sein, er kann auch ein beliebiger Ausdruck sein.
- Der Typ des Indexausdrucks muss aber `int` sein (und kann daher auch `short`, `byte` oder `char` sein).
- Der Indexbereich ist 0 .. Länge des Arrays-1.

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

- Zugriff
  - Indexwert in eckigen Klammern (z.B. array[3])
- Indexwert muss gültig sein!
  - **Sonst gibt es einen Laufzeitfehler, d.h. das Programm wird sofort abgebrochen!**

Beispiel (main-Argumente)

```java
public class PrintArgumentsApplication {
    public static void main(String[] args) {
        for (int i = 0; i < args.length; ++i) {
            System.out.println(i + ": " + args[i]);
        }
    }
}
```

```
java PrintArgumentsApplication hello again
```

```
Ausgabe:
0: hello
1: again
```

Zuweisung von Arrays

- Arrayvariable ist eine Referenz auf das eigentliche Array.
- Einer Arrayvariable vom Typ *x* kann immer nur ein Array vom Typ *x* zugewiesen werden.
- Die Länge kann aber variieren.
- Bei der Zuweisung wird nur die Referenz (Adresse) kopiert, **nicht** der Inhalt!

```
int[] x = new int[10];
int[] y = x;
y[1] = 7;
System.out.println(y[0] + " " + x[0]);
System.out.println(y[1] + " " + x[1]);
```

```
Ausgabe:
0 0
7 7
```

Freigabe des Arrayspeichers

- Freigabe des Speicherplatzes erfolgt durch das System.
  - Speicher muss nicht explizit freigeben werden.
  - Nicht benutzter Speicher wird zur Laufzeit automatisch eingesammelt und wieder freigegeben.
  - Garbage-Collector (siehe Vorlesung über JVM).
- **Voraussetzung: Array wird nicht mehr referenziert!**
- Näher erklärt in Vorlesung über JVM

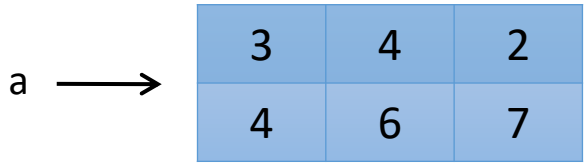
Mehrdimensionale Arrays

- Arrays können beliebig viele Dimensionen haben.

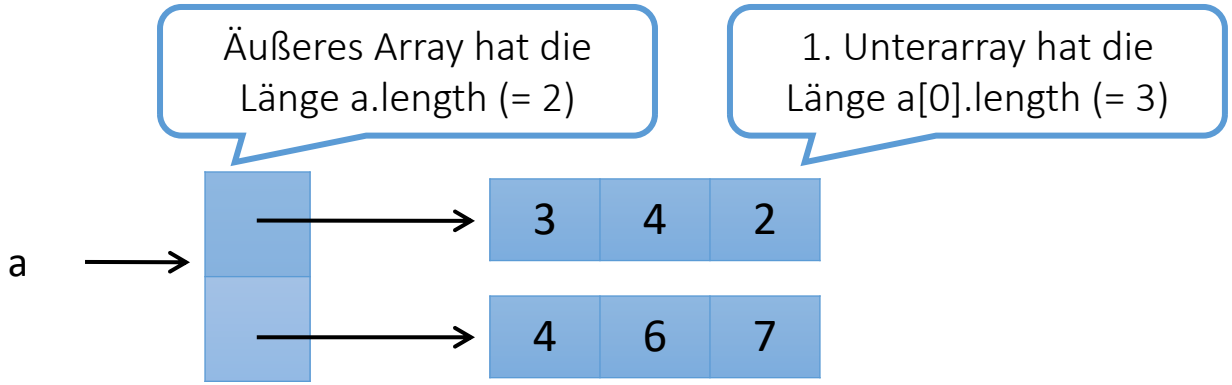
```
// Example: 2-dimensional array – matrix
int[][] matrix1 = new int[3][3];
int[][] matrix2 = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
// access
int x = matrix2[1][2];
```

- Konzeptionelle Sicht:

```
int[][] a = {{3, 4, 2}, {4, 6, 7}};
```



- „Realität“ in Java:



Weitere Aspekte (1)

- Einzelne Dimensionen können offen bleiben (nur am Ende) und später initialisiert werden.

```
int[][] gameBoard = new int[4][];
for (int i = 0; i < gameBoard.length; ++i) {
    gameBoard[i] = new int[4];
}
```

- Es können auch nicht rechteckige Arrays (jagged arrays) erzeugt werden:

```
int[][] irregularGameBoard = new int[2][];
for (int i = 0; i < irregularGameBoard.length; ++i) {
    irregularGameBoard[i] = new int[i + 3];
}
```

Weitere Aspekte (2)

- `Object` ist die direkte Oberklasse von jedem Array-Typ.
- Mit den Operatoren == und != wird überprüft, ob zwei Variablen auf das selbe Array-Objekt verweisen.

```
int[] values1 = {1, 5, 3};
int[] values2 = {1, 5, 3};
System.out.println(values1 == values2); // false
```

- Die erweiterte for-Schleife kann für das Iterieren über die Komponenten eines Arrays verwendet werden.

```
public class PrintArgumentsApplicationForeach {
    public static void main(String[] args) {
        for (String arg : args) {
            System.out.println(arg);
        }
    }
}
```

Favor For-Each over For

```java
for (int i = 0; i < args.length; ++i) {
    System.out.println(args[i]);
}
```

```java
for (String arg : args) {
    System.out.println(arg);
}
```

- Vorher:
  - Indexvariable wird nur für den Zugriff auf das nächste Element verwendet, sonst keine Funktion.
  - Indexvariable kann jederzeit verändert werden
  - Abbruchbedingung könnte falsch gewählt werden (IndexOutOfBounds-Exception)
- Nachher:
  - Jedes Element in der Datenstruktur wird verarbeitet.
  - Keinerlei Manipulationsmöglichkeiten

Erweiterte for-Schleife

- Der Compiler setzt die erweiterte for-Schleife durch eine for-Schleife um.
- Im Vergleich zur klassischen for-Schleife gibt es beim Verwenden der erweiterten for-Schleife für Arrays einige Einschränkungen:
  - Ein Start- oder Endindex kann nicht gesetzt werden.
  - Das Array wird immer vom ersten bis zum letzten Index durchlaufen.
  - Der Index wird in jeder Iteration um 1 erhöht.
  - Der Index ist nicht sichtbar und kann nicht verwendet werden.
  - Die Komponenten im Array können gelesen werden, allerdings kann der Wert nicht ersetzt werden.
- Sofern für den Anwendungsfall diese Einschränkungen zu tragen kommen, kann beispielsweise die for-Schleife Abhilfe bieten.

Kopieren von Arrays

- `System.arraycopy(src, srcPos, dest, destPos, length)`
  - Kopiert `length`-Elemente des Arrays `src` ab der Stelle `srcPos` in das Array `dest` ab der Position `destPos`.
  - Kann verwendet werden, um Elemente eines Arrays zu verschieben.
- `Arrays.copyOf(original, newLength)`
  - Erstellt ein neues Array und kopiert die ersten `newLength`-Elemente des Arrays `original` in das neue Array.
- `clone()`
  - Arrays bieten die Methode `clone()` an, um eine flache Kopie des Arrays zu erzeugen.

```
int[] original = {1, 5, 3};
int[] copy1 = original.clone();
int[] copy2 = Arrays.copyOf(original, original.length);
int[] copy3 = new int[original.length];
System.arraycopy(original, 0, copy3, 0, original.length);
```

Arrays-Klasse

- Die Klasse `Arrays` ist Teil des Java Collections Framework.
- Stellt statische Methoden für die Manipulation von Arrays bereit.
- Beispiele:
  - `binarySearch` – zum Suchen in einem sortieren Array
  - `equals` – zum Überprüfen ob zwei Arrays gleich sind
  - `fill` – zum Füllen von Arrays
  - `hashCode` – zum Berechnen eines Hash
  - `sort` – zum Sortieren der Array-Elemente
  - `toString` – zum Erzeugen einer String-Repräsentation des Array-Inhaltes
- Bei mehrdimensionalen Arrays ist bei den Methoden `equals`, `hashCode und toString` Vorsicht geboten.
- Weitere Details siehe Dokumentation

Beispiel Arrays.toString

```
String[][] array1 = {{"a", "b"}, {"c", "d"}};
String[][] array2 = {{"a", "b"}, {"c", "d"}};

System.out.println(Arrays.toString(array1));
System.out.println(Arrays.toString(array2));

System.out.println(Arrays.deepToString(array1));
System.out.println(Arrays.deepToString(array2));
```

```
Ausgabe:
[[Ljava.lang.String;@a3db808, [Ljava.lang.String;@4c56ea0f]
[[Ljava.lang.String;@87f99898, [Ljava.lang.String;@356f015d]
[[a, b], [c, d]]
[[a, b], [c, d]]
```

Beispiel Arrays.equals

```
String[][] array1 = {{"a", "b"}, {"c", "d"}};
String[][] array2 = {{"a", "b"}, {"c", "d"}};

System.out.println(Arrays.equals(array1, array2));
System.out.println(Arrays.deepEquals(array1, array2));
```

```
Ausgabe:
false
true
```

Arrays & Strings

- Weder Strings noch Arrays sind nullterminiert!
- Strings sind keine Arrays!
- Die Methode `toCharArray` der Klasse `String` kann verwendet werden, um ein `String` in ein `char-Array` umzuwandeln.

```
String text = "hello";
char[] letters = text.toCharArray();
Arrays.sort(letters);
System.out.println(letters);
```

```
Ausgabe:
ehllo
```