Objektorientierte Programmierung und Modellierung

Einführung, Teil 2

24.10.17 - WS17/18

Hakan Aksu

Raumänderung beachten

Dienstag 14:15-15:45 Uhr

in M001 (anstatt K101)

Ausfalltermine

- 31.10 Vorlesung - fällt aus
- 30./31.10 Übungen - fallen aus
- 01./02.11 Praktika - fallen aus
- 02.11 Vorlesung - findet statt - WICHTIG!

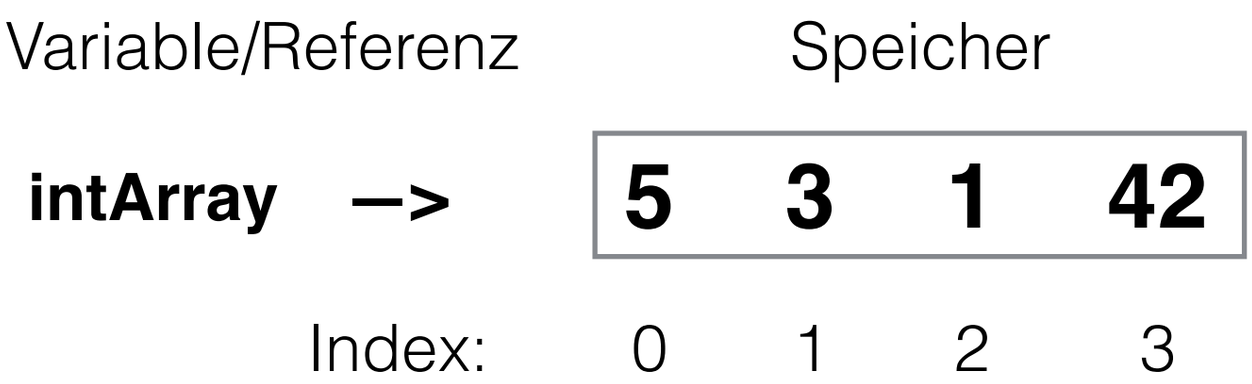
Agenda

- Arrays
- Bedingte Anweisung/Verzweigung (IF)
- Schleifen/Wiederholung (WHILE / FOR)

Arrays

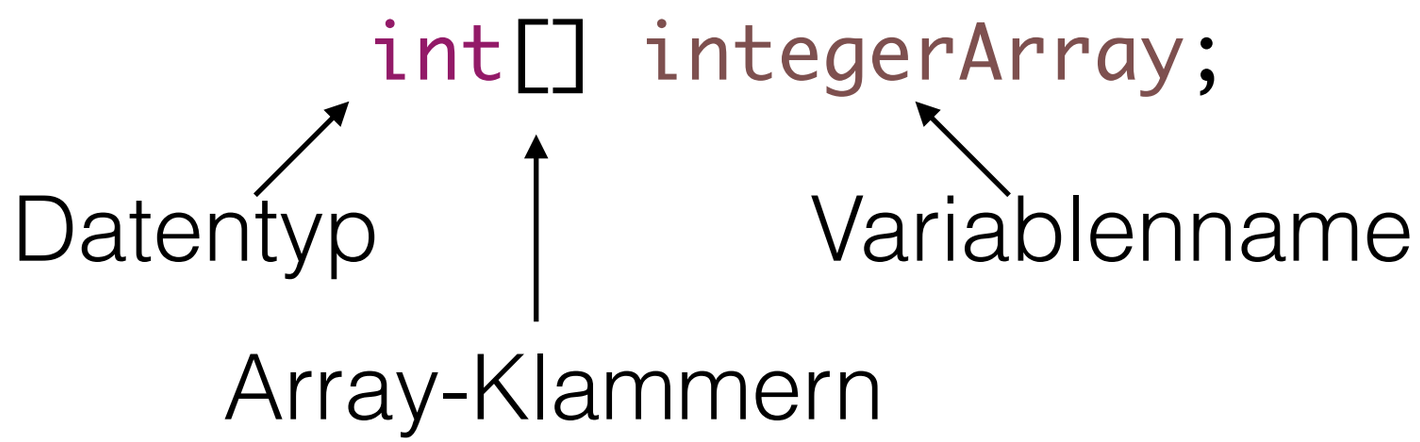
Was sind Arrays?

- Eine Liste mit einer vordefinierten Länge
- z.B. eine Integer-Liste mit der Länge 4



Deklarieren von Arrays

- typische/empfohlene Schreibweise:



- weitere/nicht übliche Schreibweisen:

```
int []a;
int b[];
```

Arrays initialisieren

- Erzeugung von Array-Objekten

```
int[] integerArray = new int[3];
```

Länge des Arrays

- Es wurden noch keine Werte zugewiesen! Zuweisung durch indexbasierten Zugriff:

```
integerArray[0] = 1;
integerArray[1] = 5;
integerArray[2] = 8;
```

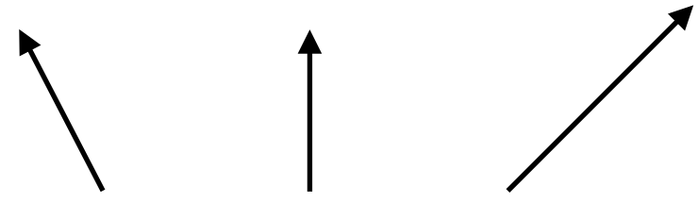
Index

zugewiesener Wert

Arrays initialisieren

- Alternative Initialisierung:

```
String[] namen = {"Hans", "Peter", "Schmidt"};
```



Es entsteht ein String-Array der Länge 3
mit den Werten “Hans“,“Peter“ und “Schmidt“

Zugriff auf Arrays

- Zugriff auf einzelne Werte

```
int[] integerArray = {1, 5, 8};
int summe = integerArray[0] + integerArray[1];
```

summe ist 6

indexbasierter Zugriff

Länge von Arrays

- Nachdem die Länge einmal festgelegt wurde, kann sie nicht mehr geändert werden.

- Die Länge erhält man über „*length*“

```
int[] integerArray = {1, 5, 8};
int l = integerArray.length;
```

Liefert die Länge des Arrays als Integer-Wert

Array als Datentyp

- Arrays sind **Referenztypen**
- Variablen enthalten Referenzen/Speicheradressen auf Array-Objekte

primitiver Datentyp **vs** **Referenztyp**

```
int integerValue = 5;
```

Die Variable
enthält den Wert 5

```
int[] integerArray = {1, 5, 8};
```

Die Variable enthält die
Speicheradresse zum Array-Objekt

Klonen von Variablen

```
int a = 5;
int b = a;
b = 1;
```

```
int[] integerArray = {1, 5, 8};
int[] tmpArray = integerArray;
tmpArray[0] = 2;
```

Was sind
‘a‘ und ’b’
bzw. ’integerArray’ und ’tmpArray’
nach dem Ausführen?

Klonen von Variablen

```
int a = 5;
int b = a;
b = 1;
```

a = 5
b = 1

```
int[] integerArray = {1, 5, 8};
int[] tmpArray = integerArray;
tmpArray[0] = 2;
```

integerArray = {2, 5, 8}
tmpArray = {2, 5, 8}

Warum?

Klonen von Variablen

```
int a = 5;
int b = a;
b = 1;
```

‘b’ wird der Wert von ’a’ zugewiesen.
—> Der Wert wird geklont.

```
int[] integerArray = {1, 5, 8};
int[] tmpArray = integerArray;
tmpArray[0] = 2;
```

Es wird die Speicheradresse geklont.
—> Man arbeitet auf den selben Werten.
—>Änderungen am Speicher/Inhalt haben globale Auswirkungen.

Wie klont man Array-Inhalte?

Klonen von Variablen

```
int[] integerArray = {1, 5, 8};
int[] tmpArray = integerArray;
tmpArray[0] = 2;
```

1. Möglichkeit

Erstellen eines neuen Arrays und kopieren einzelner Werte —> Umständlich

2. Möglichkeit

Verwendung der clone()-Methode

```
int[] integerArray = {1, 5, 8};
int[] tmpArray = integerArray.clone();
tmpArray[0] = 2;
```

integerArray = {1, 5, 8}
tmpArray = {2, 5, 8}

Mehrdimensionale Arrays

- Arrays von Arrays

3x2 Array (rechteckig)
(int[][] mArray = new int[3][2];)

```
int[][] mArray = { {10, 20}, {30, 40}, {50, 60} };
int[][] mArray2 = { {10, 20}, {30, 40, 50}, {60} };
```

zwei-Dimensionale-Arrays

nicht rechteckig

Mehrdimensionale Arrays

```
int[][] mArray = { {10, 20}, {30, 40}, {50, 60} };
```

m n	0	1	2
0	10	30	50
1	20	40	60

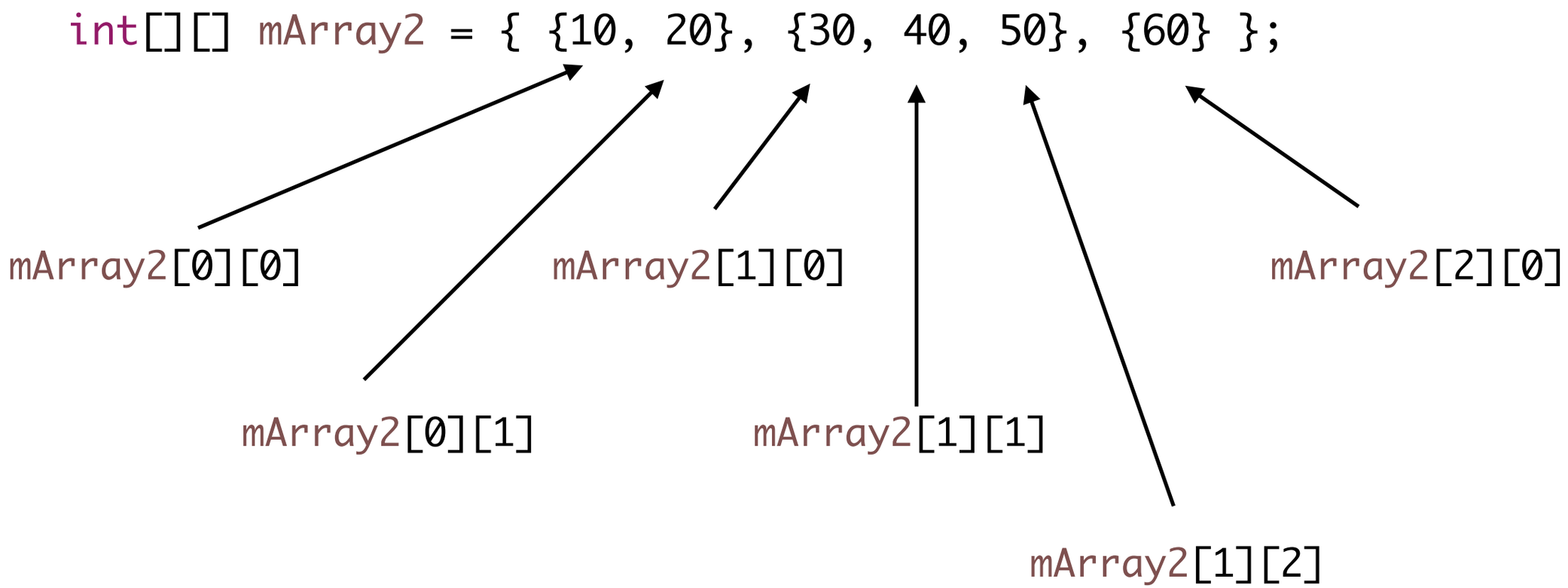
Zugriff:

`mArray[m][n]`

Beispiel:

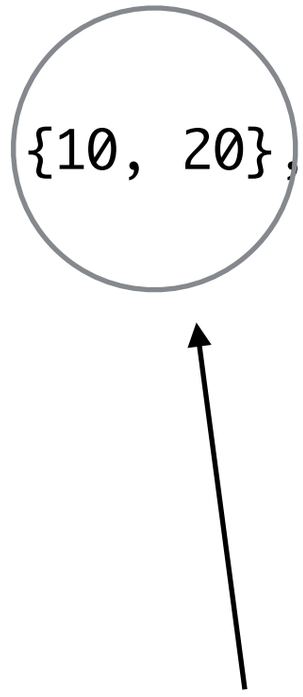
`mArray[1][0] -> 30`

Mehrdimensionale Arrays



Mehrdimensionale Arrays

```
int[][] mArray2 = { {10, 20}, {30, 40, 50}, {60} };
```



```
int[] tmp = mArray2[0].clone();
```

Vergleich von Arrays

Gegeben sei:

```
int[] integerArray = {1, 5, 8};
int[] integerArray2 = {1, 5, 8};
int[] integerArray3 = integerArray;
```

Mit dem Vergleichsoperator ‘==‘
und der objekteigenen equals-Methode
wird die Referenz (Speicheradresse) verglichen:

```
   integerArray == integerArray2    -> liefert false
integerArray.equals(integerArray2)   -> liefert false
   integerArray == integerArray3    -> liefert true
integerArray2.equals(integerArray2)  -> liefert true
```

Mit der equals-Methode aus der Arrays-Klasse
werden Inhalte von Arrays verglichen

```
Arrays.equals(integerArray, integerArray2)  -> liefert true
```

Bedingte Anweisung (IF)

IF-ELSE-Abfragen

Wenn man einen Code unter bestimmten Bedingungen ausführen will, verwendet man IF-Abfragen.

Boolescher Ausdruck


```
if (condition) {
```

Auszuführender Block, wenn “condition“ erfüllt (true) ist.

```
} else {
```

Auszuführender Block, wenn “condition“ nicht erfüllt (false) ist.

```
}
```

Der Else-Teil ist optional

IF-ELSE-Abfragen

- Beispiele:

```
String s;

// Benutzereingabe

if (s.length() > 5) {
   //Code
} else {
   //Code
}
```

```
String s;

// Benutzereingabe

if (s.length() > 5) {
   //Code
}
//Code
```

Unterschied?

Schleifen (WHILE / FOR)

Schleifen

- while-Schleife
- do-while-Schleife
- for-Schleife
- for-each-Schleife (erweiterte for-Schleife)

WHILE-Schleife

Wenn man einen Code unter bestimmten Bedingungen mehrmals ausführen will, verwendet man WHILE-Schleifen.

Boolescher Ausdruck

```
while (condition) {

}
```

Auszuführender Block.

VOR jedem Ausführen wird die “condition“ erneut geprüft!

WHILE-Schleife

- Beispiel:

```
int value = 1;
int faktor = //Benutzereingabe
while (value < 1000) {
   value *= faktor;
}
```

DO-WHILE-Schleife

Wenn man einen Code einmal und unter bestimmten Bedingungen weitere Male ausführen will, verwendet man WHILE-Schleifen.

Boolescher Ausdruck

Auszuführender Block.

NACH jedem Ausführen wird die “condition“ erneut geprüft!

```
do {

} while (condition);
```

DO-WHILE-Schleife

- Beispiel

```
int value = 1;
int faktor = //Benutzereingabe
do {
   value *= faktor;
} while (value < 1000);
```

FOR-Schleife

Wenn man einen Code mit einer sich verändernden (Zähl-)Variabel bis zum erfüllen einer Bedingung mehrmals ausführen will,
verwendet man FOR-Schleifen.

Deklarieren und Initialisierung der Laufvariablen (Anfangswert)

Boolescher Ausdruck (Im Normalfall: In Abhängigkeit von der Laufvariablen)

```
for (init; condition; next) {

}
```

Auszuführender Block.

Nach jedem Ausführen wird die “condition“ erneut geprüft!

Auszuführende Zuweisung nach jedem Schleifendurchlauf

FOR-Schleife

- Beispiele:

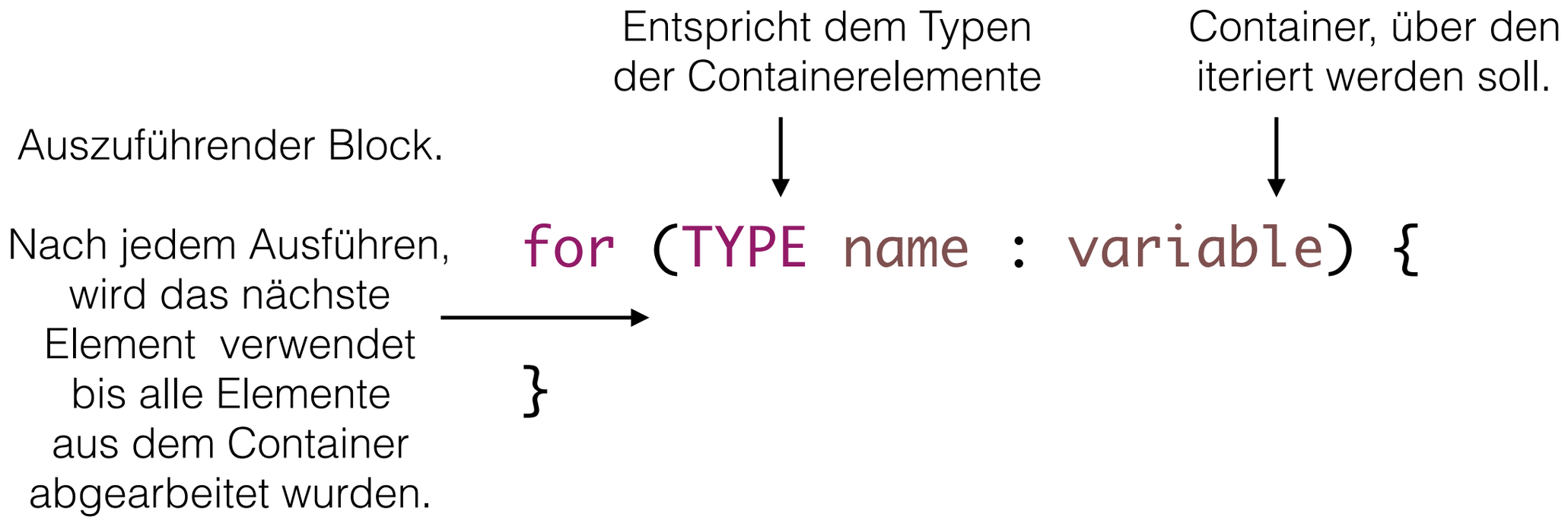
```
int[] integerArray = {1,3,5};
for (int i = 0; i < integerArray.length; i++) {
        System.out.println(integerArray[i]);
}
```

```
int summe = 0;
for (int i = 100; i > 0; i--)
        summe += i;
```

```
int summe = 0;
for (int i = 1; i < 100; i += 2)
        summe += i;
```

FOR-EACH-Schleife

Wenn man über einen Container (z.B. Listen oder Arrays) iterieren will verwendet man FOR-EACH-Schleifen.



FOR-EACH-Schleife

- Beispiel

```
int[] integerArray = {2,4,6,8};
for (int integerValue : integerArray) {
    System.out.println(integerValue);
}
```

Quiz

```
String[] sArray = {„Ich","bin","Nummer", 5};
```

Ist das möglich?

Quiz

```
array = {{{1,2},{6,4,2},{1,1,3}},{{8}},{{2},{1,9,77}}};
```

Welche Dimension?

Wie greife ich auf die 77 zu?

Quiz

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

}
```

```
for (int i : iArray) {

}
```

Was sind Vor- und Nachteile?

Quiz

```
while (true) {
  //Code
}
```

Was passiert hier?

Quiz

```
public static void main(String[] args) {
   int[] iArray = {1,2,3};
   arrayIncrement(iArray);
   for (int i = 0; i < iArray.length; i++) {
      System.out.println(iArray[i]);
   }
}

public static void arrayIncrement(int[] intArray) {
   for (int i = 0; i < intArray.length; i++) {
      intArray[i]++;
   }
}
```

Was wird ausgegeben?