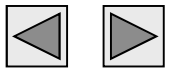
Softwareentwicklung

Programmierung mit Java

Teil 2



Die Klasse String (1)

Zweck der Klasse *String*: Speicherung und Bearbeitung von Zeichenketten (Wörter, Sätze)

Erzeugung eines Strings:

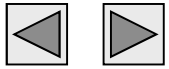
```
String s1 = "Zeichenkette";
```

oder

```
String s2 = new String("Zeichenkette");
```

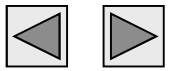
oder

```
char[] charArray = {'A', 'B', 'C'};
String s3 = new String(charArray);
```



Die Klasse String (2)

- Operator +: Konkatenation von Zeichenketten
  - `String s1 = “ABC“ + variable + “DEF“;`
- Nützliche Methoden der Klasse String:
  - `“Geoinformation“.length() liefert 14`
  - `“  ABCD  “.trim() liefert “ABCD“`
  - `"Geoinformation".substring(3,7) liefert "info“`
  - `"Institution".indexOf("ti") liefert 3`
- Erweiterung: Klasse StringBuffer

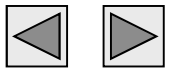


Arbeiten mit Zeichenketten

- Beispiel:

```
String s1 = "Das ist das Haus";
String s2 = "Nikolaus.";
String s3 = s1 + " vom " + s2;
//in s3 steht:"Das ist das Haus vom Nikolaus.";
String s4 = s3.substring(12,16);
// in s4 steht: "Haus"
```

- Strings sind konstant (können nicht geändert werden)
  - sie müssen nicht mit new erzeugt werden
  - Stringvariablen können jedoch geändert werden, z.B. s1 = "abc"; in obigem Beispiel.
    (Anmerkung: dabei wird nicht der alte String umgeändert, sondern ein neuer String im Speicher erzeugt, auf den s1 dann verweist.)



Operator + für Zeichenketten

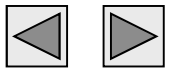
- Beispiel:
  - **statt:**

```
System.out.print("Die Temperatur ");
System.out.print(fahrenheitIn);
System.out.print(" entspricht in Celsius ");
System.out.print(celsiusOut);
System.out.println(".")
```

  - **einfacher:**

```
System.out.println("Die Temperatur " + fahrenheitIn +
        " entspricht in Celsius " + celsiusOut + ".");
```

- Der Operator "+" verkettet zwei Zeichenketten, wenn mindestens ein Operand eine Zeichenkette ist.
- Was gibt System.out.print(4 + 7 + "x") aus? 11x
- Was gibt System.out.print("x" + 4 + 7) aus? x47

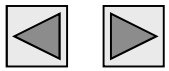


Arrays

Index	Wert
0	45
1	-117
2	12
3	0
4	999

Integer-Array mit fünf Elementen

- Bisher: primitive Datentypen (int, double, ...)
- Arrays als erster nicht-primitiver Datentyp
- Alle Werte haben denselben Typ (z.B. int)
- Zugriff über Index:
```
my_array[2] = 12;
```
- Indizes fangen mit 0 an



Arbeiten mit Array-Variablen (1)

3 Schritte: (nur Schritt 2 ist wirklich neu)

1. Deklaration

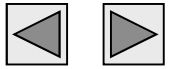
```
int[] my_array;
// Array von Integerzahlen
```

2. ***Instantiierung***

```
my_array = new int[3];
// Erschafft Array mit 3 Elem.
```

3. Initialisierung

```
my_array[0] = 17;
my_array[1] = 0;
my_array[2] = -4*my_array[0];
```



Arbeiten mit Array-Variablen (2)

Alternative:

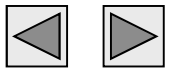
- `int[] my_array = new int[3];`

  oder

- **Deklaration** mit ***impliziter Erschaffung*** und **Initialisierung**:

  ```
  int[] my_array = {45, -117, 12, 0, 999};
  ```

  erschafft ein 5-elementiges Array und initialisiert es mit den aufgezählten Werten. (Der Operator **new** wird nicht benötigt.)



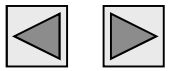
Hinweise zu Arrays (1)

- Größe eines Arrays kann erst zur Laufzeit feststehen:

```
int[] my_array;
int i = ...;        //zur Laufzeit berechnet
my_array = new int[i];
```

- Die Größe eines einmal instantiierten Arrays kann nicht verändert werden.
- Größe eines Arrays kann mittels **`arrayname.length`** abgefragt werden:

```
int laenge = my_array.length;
```



Hinweise zu Arrays (2)

- Bsp: Ausgabe aller Elemente eines Arrays

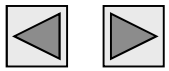
```
int[] my_array = {45, -117, 12, 0, 999};

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

- Seit Java 5 gibt es eine Abkürzung für solche Schleifen:

```
for (int aktuell : my_array) {
   System.out.println(aktuell);
}
```

Die Variable „aktuell“ nimmt bei jedem Durchlauf den Wert den nächsten Arrayelements an.
Der Datentyp von „aktuell“ muss daher vom gleichen Typ, wie das Array sein!



Beispiel zu Arrays

Programm:

```
// Bestimmung der kleinsten Zahl eines ganzzahligen Arrays

class Minimum {
   public static void main (String args[]) {
     int[] my_array = {45,12,1,13,-4,0,-23,1001};
     int minimum, i;

     minimum=my_array[0];
     for (i=0; i<my_array.length; i=i+1)
       if (my_array[i]<minimum)
         minimum=my_array[i];

     System.out.print("Die kleinste Zahl ist ");
     System.out.println(minimum);
    }
}
```

Ausgabe:

```
Die kleinste
Zahl ist -23
```

Mehrdimensionale Arrays

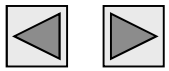
- Anwendungen
  - zweidimensional: Repräsentation von Matrizen
  - zwei- und mehrdimensional: Speicherung von Wertetabellen
- Mehrdimensionale Arrays werden durch Arrays von Arrays dargestellt
- **Syntax** zur Deklaration und Instantiierung eines n-dimensionalen Arrays:

  Typ **[ ][ ]...[ ]** ***Arrayname*** **= new** ***Typ*** **[d_1][d_2]...[d_n];**

- **Beispiel:**

```
int[][] matrix = new int[6][3];
```

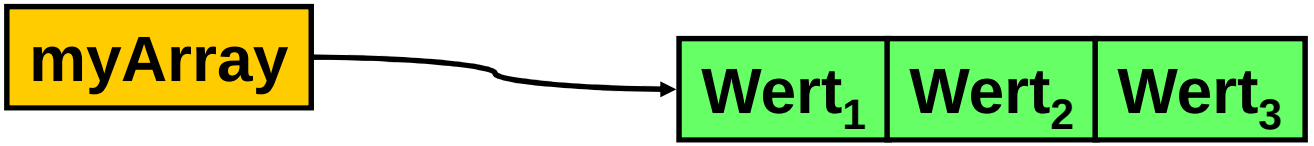
  erschafft eine zweidimensionale Matrix mit 6 Zeilen und 3 Spalten



Hinweise zu Arrays (1)

- **Arrays sind keine primitiven Datentypen**
  - Array-Variablen enthalten nicht selbst das Array, sondern verweisen auf ein Array, das durch **new** irgendwo im Hauptspeicher angelegt wird (Array-Variablen sind **Referenzen**).

    Beispiel: `int[] myArray=new int[3];`



➔ kein simpler Vergleich zweier Arrays möglich:

```
int[] a = {7,12,13,0};
int[] b = {7,12,13,0};
System.out.println(a = = b);
```

ergibt die Ausgabe **false**

```
int i = 3;
int j = 3;
System.out.println(i = = j);
```

ergibt die Ausgabe **true**

Hinweise zu Arrays (2)

- **Arrayvariablen sind Referenzen**
  - Wertzuweisung zweier Arrayvariablen kopiert nicht das Array, sondern setzt beide auf denselben Speicherbereich:
  - Bsp:

```
int[] a = {1,2,3};
int[] b = {4,5,6};
a = b;
a[0] = 77;
System.out.println(b[0]);
```