Programmieren I

Dr. Klaus Höppner

Hochschule Darmstadt – Wintersemester 2009/2010

Zuweisungsoperator

- Die Zuweisung von Werten an Variablen erfolgt mit dem Operator =.
- Beispiele:

```
a = 1;
b = a;
x = 3.1415;
```

- In Java liefert eine Zuweisung das Ergebnis als Rückgabewert zurück. Daher ist auch folgendes möglich:

```
a = b = 1;
```

Binäre arithmetische Operatoren

- Java kennt die folgenden Operatoren für das Rechnen mit zwei Zahlen:

Operator	Rechenart
+	Addition
-	Subtraktion
*	Multiplikation
/	Division
%	Modulo

- Hierbei gilt: Punktrechnung vor Strichrechnung.

Implizite Typumwandlung

Java erlaubt, mit Zahlen unterschiedlicher Datentypen zu rechnen.

Es gilt zum Beispiel:

int + long → long
*float * double → double*
int / float → float
int - char → int

Aber Achtung:

int / int → int

Zusammengesetzte Zuweisungsoperatoren

Häufig werden in Programmen Zuweisungen wie

```
summe = summe + wert;
x = x * 10.0;
n = n % 2;
```

benutzt.

Hierfür gibt es als Kurzform die zusammengesetzten Operatoren +=, -=, *=, /= und %=.
Hierbei ist »*var op= wert*« jeweils identisch mit *var = var op wert*

Unäre arithmetische Operatoren

Neben den Vorzeichenoperatoren (+ und −) gibt es bei Java noch eine besondere Art von Operatoren:
Die Inkrements- bzw. Dekrementsoperatoren.

Dies sind:

`Operator`	Rechenoperation
`++`	Erhöhung um 1
`--`	Erniedrigung um 1

Steht der Operator vor der Variablen (Präfix-Notation, z. B. `++i`), so ist der Rückgabewert der Variablenwert *nach* dem Inkrement bzw. Dekrement. Steht der Operator nach der Variablen (Postfix-Notation, z. B. `i--`), so ist der Rückgabewert der Variablen *vor* dem Inkrement bzw. Dekrement.

Vorrangtabelle für Operatoren

`++`, `--`, + (Vorzeichen), − (Vorzeichen)
`*`, `/`, `%`
+ (Addition), − (Subtraktion)
`=`, `+=`, `-=`, `*=`, `/=`, `%=`

Bis jetzt wurden behandelt:

- Deklaration von Variablen
- Rechnen mit Variablen und Werten

Zu einem sinnvollen Programm gehört aber in der Regel, dass auf bestimmte Bedingungen (z. B., dass ein Variable einen gewissen Wert hat) reagiert werden kann. Dies setzt *Kontrollstrukturen* im Programm voraus:
Verzweigungen und Schleifen

Die if-Anweisung (Verzweigung)

Eine if-Anweisung sieht in ihrer einfachsten Form wie folgt aus:

```
if ( Bedingung ) {
  Anweisungen
}
```

Beispiel:

```
if (i>0) {
    System.out.println("i ist positiv");
}
```


Die Bedingung ist dabei ein Ausdruck, der einen

boolean-Wert: **true** oder **false**

repräsentiert.
Beispiele:

```
i == 1
(i > 0) && (i<=10)
(d < 0) || (i != 5)
```

Besteht der Anweisungsblock aus einer einzelnen Anweisung, so können die geschweiften Klammern entfallen.

Der else-Teil

Neben einem Anweisungsblock, der ausgeführt werden soll, wenn die Bedingung erfüllt ist, kann man auch einen else-Block angeben, der ausgeführt wird, wenn die Bedingung nicht erfüllt ist:

```
if ( Bedingung ) {
  Wahr-Teil
} else {
  Unwahr-Teil
}
```

Beispiel

```
BufferedReader in = new BufferedReader(
        new InputStreamReader(System.in)
    );
System.out.print("Alter eingeben: ");
int alter = Integer.parseInt(in.readLine());
if ( alter>18 ) {
    System.out.println("Erwachsener mit Alter "+
                       String.valueOf(alter));
} else {
    System.out.println("Noch nicht erwachsen");
}
```


Verschachtelte `if`-Anweisungen

`if`-Anweisungen können auch verschachtelt werden. Besondere Aufmerksamkeit gilt hier dem `else`-Teil. Dieser bezieht sich auf die letzte offene `if`-Anweisung.

Der folgende Code macht daher nicht das gewünschte:

```
if (i >= 0)
  if (i % 2 != 0)
    System.out.println("ungerade");
else
  System.out.println("negativ");
```


Korrekter Code:

```
if (i >= 0) {
  if (i % 2 != 0)
    System.out.println("ungerade");
} else
  System.out.println("negativ");
```

besser:

```
if (i >= 0) {
  if (i % 2 != 0) {
    System.out.println("ungerade");
  }
} else {
  System.out.println("negativ");
}
```

Die switch-Anweisung

Manchmal kommen Ketten von **if**-Anweisungen vor, die anhand diskreter möglicher Werte einer Variablen verzweigen. Beispiel:

```
if (a == wert1) {
    // Code
} else if (a == wert2) {
    // Code
} else {
    // Code
}
```


Die switch-Anweisung (Forts.)

Solch ein Code kann (bei elementaren Datentypen!) durch die **switch**-Anweisung eleganter geschrieben werden:

```
switch (a) {
case wert1:
    // Code
    break;
case wert2:
    // Code
    break;
default:
    // Code
}
```

Achtung: **break** nicht vergessen! Sonst wird auch der Code aller nachfolgenden Fälle ausgeführt.

Arten von Schleifen in Java

Java kennt drei Arten von Schleifen:

1. die `while`-Schleife,
2. die `do-while`-Schleife,
3. die `for`-Schleife

Die while-Schleife

- Eine while-Schleife hat folgende Struktur:

```
while ( Bedingung ) {
  Anweisungen
}
```

- Der Anweisungsblock wird ausgeführt, solange die Bedingung wahr ist.
- Da die Bedingung bereits *vor* der ersten Ausführung geprüft wird, kann es sein, dass der Anweisungsblock *kein* Mal ausgeführt wird.

Beispiel

```
int summe = 0;
int i = 1;

while (i<=100) {
    summe += i;
    i++;
}

System.out.println(summe);
```

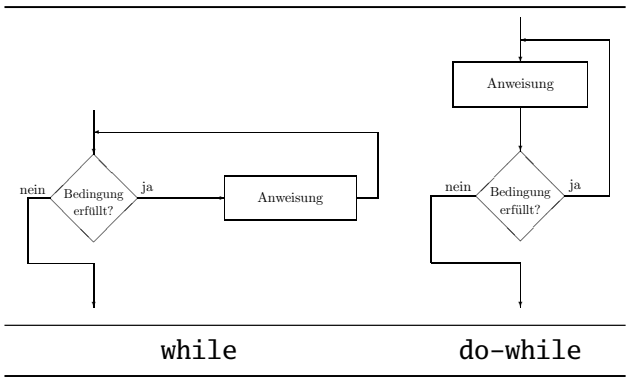

Die do-while-Schleife

- Eine do-while-Schleife hat folgende Struktur:

```
do {
  Anweisungen
} while ( Bedingung );
```

- Der Anweisungsblock wird ausgeführt, solange die Bedingung wahr ist.
- Der Anweisungsblock wird mindestens *ein* Mal ausgeführt.
- *Achtung*: Hinter der Bedingung steht ein Semikolon!

Vergleich der Ablaufpläne



while do-while

Die for-Anweisung

Die for-Anweisung hat die Struktur

```
for ( Term1 ; Term2 ; Term3 ) {
  Anweisungen
}
```

(Achtung: Zwischen den Termen befindet sich jeweils ein Semikolon).

Dies ist äquivalent zu:

```
Term1;
while (Term2) {
  Anweisungen
  Term3;
}
```


Beispiel

```
int summe = 0;
int i;

for (i=1; i<=100; i++) {
    summe += i;
}

System.out.println(summe);
```

Die erweiterte `for`-Schleife

In Java existiert zusätzlich die erweiterte Form der **`for`**-Schleife über alle Elemente eines Array (oder später einer Collection):

```
String[] woerter = {"Hallo", "Echo", "Otto"};
for (String wort : woerter) {
    System.out.println(wort);
}
```