Operatoren in Java

1 Allgemeines

Operatoren werden üblicherweise in Ausdrücken verwendet. Ein Ausdruck besteht immer aus einem Operator und mindestens einem Operanden, auf die der Operator angewendet wird. Jeder Ausdruck hat einen Ergebniswert, der durch die Anwendung des Operators auf die Operanden entsteht. Der Datentyp des Ergebniswerts bestimmt sich durch die Datentypen der Operanden.

Man unterscheidet einstellige, zweistellige und dreistellige Operatoren, je nachdem, wie viele Operanden erwartet werden.

Ein Ausdruck wird gemäß der vorgegebenen Vorrangs- und Ausführungsregeln (z.B. "Punkt vor Strich") ausgewertet. Diese lassen sich durch explizite Klammerung beeinflussen. Auf die Details soll hier nicht eingegangen werden (Die verbindlichen Regeln finden sich in der jeweils aktuellen “Java Language Specification” auf www.oracle.com)

2 Arithmetische Operatoren

Arithmetische Operatoren erwarten numerische Operanden[1].
Java kennt die üblichen Operatoren für die Grundrechenarten Addition, Subtraktion, Multiplikation, Division und Restwert (Modulo). Sie sind zweistellig und lassen sich auf alle numerischen Datentypen anwenden.

Operator	Bezeichnung	Bedeutung
+	Summe	a + b ergibt die Summe von a und b *)
-	Differenz	a - b ergibt die Differenz aus a und b
*	Produkt	a * b ergibt das Produkt aus a und b
/	Quotient	a / b ergibt den Quotienten von a und b
%	Restwert	a % b ergibt den Rest der Ganzzahldivision von a durch b. (Auch auf Fließkommazahlen anwendbar.)

*) Der Operator + kann auch zur Stringverkettung benutzt werden. Siehe dazu Abschnitt 6.3.

Achtung: Bei der Division ist eine Besonderheit zu beachten:
Das Ergebnis bei der Anwendung des Divisionsoperators hängt vom Datentyp der Operanden ab.

Beispiele:

```
double dErgebnis = 0;

dErgebnis = 2 / 4;      // dErgebnis wird Wert der Wert 0 zugewiesen. Da beide Operanden
                        // vom Datentyp Ganzzahl (int) sind, wird eine Ganzzahldivision
                        // durchgeführt
dErgebnis = 2.0 / 4;    // dErgebnis wird Wert der Wert 0.5 zugewiesen. Da ein Operand
                        // von einem Gleitpunkt-Datentyp ist (double), wird eine
                        // Gleitpunktdivision durchgeführt
```

[1] Da der Datentyp char ebenfalls ein Ganzzahltyp ist, kann mit char-Werten auch gerechnet werden. Beispiel: ('D' - 'A') ergibt 3, da 'D' an 3. Position hinter dem 'A' im Unicode liegt.

Einstellige Operatoren für Vorzeichen:

Operator	Bezeichnung	Bedeutung
+	positives Vorzeichen	+n ist gleichbedeutend mit n
-	negatives Vorzeichen	-n kehrt das Vorzeichen von n um

Einstellige Operatoren für Inkrement- und Dekrement

Die Operatoren ++ und -- haben ein besonderes Verhalten, da sie bei der Auswertung in einem Ausdruck nicht nur einen Ergebniswert liefern, sondern gleichzeitig auch den Operanden selbst verändern. Der Inkrementoperator ++ erhöht den Operanden um 1, der Dekrementoperator -- erniedrigt ihn um 1. Für beide gibt es ein Prä- und eine Postvariante mit folgender Bedeutung:

Operator	Bezeichnung	Bedeutung
++	Präinkrement	++a liefert a+1 und erhöht a um 1
++	Postinkrement	a++ liefert a und erhöht a um 1
--	Prädekrement	--a liefert a-1 und erniedrigt a um 1
--	Postdekrement	a-- liefert a und erniedrigt a um 1

Beispiele:

```
int iErgebnis = 0;
int iZahl = 3;
++iZahl;                    // Präinkrement; iZahl hat den Wert 4
iZahl = 3;
iZahl++;                    // Postfix; iZahl hat den Wert 4
                            // kein Unterschied. Aber:
iZahl = 3;
iErgebnis = 2 * ++iZahl;    // ergibt iErgebnis = 8, iZahl = 4
iZahl = 3;
iErgebnis = 2 * iZahl++;    // ergibt iErgebnis = 6, iZahl = 4
```

3 Verbund-Zuweisungsoperatoren

Verbund-Zuweisungsoperatoren sind eine Kombination aus arithmetischen Operatoren und Zuweisung. Java kennt folgende arithmetische Verbundzuweisungsoperatoren:

Operator	Bezeichnung	Bedeutung
=	einfache Zuweisung	a = b weist a den Wert von b zu und liefert b als Ergebniswert
+=	Additionszuweisung	a += b weist a den Wert von a+b zu und liefert a+b als Ergebniswert
…	Entsprechendes gilt für die anderen zweistelligen arithmetischen und bitweisen Operatoren	

Da die Verbund-Zuweisungsoperatoren nicht wirklich notwendig sind und sie die Lesbarkeit eines Programms verschlechtern, ist ihre Verwendung nicht empfohlen.

Beispiel:

```
public static void main (String []args)
{
  int iZahl = 5;
  iZahl += 10;      // entspricht der Anweisung iZahl = iZahl + 10;
                    // iZahl hat also jetzt den Wert 15
  int iZahl2 = 20;
  iZahl2 %= iZahl; // entspricht iZahl2 = iZahl2 % iZahl;
                    // iZahl2 hat also jetzt den Wert 5
}
```

4 Vergleichsoperatoren

Mit Vergleichsoperatoren werden Ausdrücke miteinander verglichen. Das Ergebnis ist immer ein Wahrheitswert (true oder false). Sie werden meist in Bedingungsausdrücken verwendet. Vergleichsoperatoren funktionieren mit beliebigen – auch gemischten - einfachen Datentypen. Es stehen die aus der Mathematik bekannten Operatoren zur Verfügung:

Operator	Bezeichnung	Bedeutung	anwendbar auf
==	gleich	a == b ergibt true, wenn *a gleich b* ist.	alle einfachen Datentypen und alle Referenztypen *)
!=	ungleich	a != b ergibt true, wenn *a ungleich b* ist	alle einfachen Datentypen und alle Referenztypen *)
<	kleiner	a < b ergibt true, wenn *a kleiner als b* ist	alle einfachen Datentypen, außer boolean
<=	kleiner gleich	a <= b ergibt true, wenn *a kleiner oder gleich b* ist	alle einfachen Datentypen, außer boolean
>	größer	a > b ergibt true, wenn *a größer als b* ist	alle einfachen Datentypen, außer boolean
>=	größer gleich	a > b ergibt true, wenn *a größer oder gleich b* ist	alle einfachen Datentypen, außer boolean

*) bei Anwendung auf Referenztypen werden aber nur deren Referenzen (Adressen im Hauptspeicher) verglichen, und nicht der Inhalt auf den sie verweisen (siehe später).

Beispiel:

```
public static void main (String []args)
{
  boolean b;
  int iZahl = 10;
  b = iZahl > 0;    // b erhält true
}
```

Achtung: Ein Vergleich von Gleitpunkt-Werten auf Gleichheit (==) ist im Allgemeinen nicht sinnvoll, da zwei mathematisch äquivalente Ausdrücke sich numerisch trotzdem vielleicht in der 15. Stelle nach dem Komma unterscheiden können. Dies liegt an der begrenzten Genauigkeit bei numerischen Berechnungen. Die zu vergleichenden Werte sollten vorher auf eine bestimmte Genauigkeit gerundet werden.

Vergleich von Character- Werten:

Auch Character-Werte können mit einander verglichen werden. *Beispiel:*

```
char cZeichen = '5';

if (cZeichen >= '0')
{
  System.out.println(cZeichen + " kommt im Unicode nach '0'");
}
```

Da die Dezimalziffern '0' bis '9' im Unicode genau hintereinander angeordnet sind, ergibt sich eine Ordnungsrelation '0' < '1' < '2' < '3' < … < '9'
Dasselbe gilt für das Alphabet 'a' < 'b' < 'c' < 'd' < … < 'z' bzw. 'A' < 'B' < 'C' < 'D' < … < 'Z'

5 Logische Operatoren

Logische Operatoren verknüpfen boolsche Werte miteinander. Sie werden meistens bei der logischen Verknüpfung von Bedingungen verwendet.

Java kennt die logischen Operatoren UND, ODER, EXCLUSIV-ODER und NICHT. Zur Optimierung von UND und ODER steht in Java die sog. „Short-Circuit-Evaluation“ (Kurzschlussauswertung (K)) zur Verfügung: die Auswertung wird abgebrochen, sobald das Ergebnis feststeht.

Operator	Bezeichnung	Bedeutung
&&	UND (mit K)	a && b ergibt true, wenn sowohl a als auch b wahr sind. Ist a bereits falsch wird false zurückgegeben und b nicht mehr ausgewertet.
||	ODER (mit K)	a || b ergibt true, wenn mindestens einer der beiden Ausdrücke a oder b wahr ist. Ist a bereits wahr wird true zurückgegeben und b nicht mehr ausgewertet.
&	UND (ohne K)	a & b ergibt true, wenn sowohl a als auch b wahr sind. Beide Teilausdrücke werden ausgewertet.
|	ODER (ohne K)	a | b ergibt true, wenn mindestens einer der beiden Ausdrücke a oder b wahr ist. Beide Teilausdrücke werden ausgewertet.
!	NICHT	!a ergibt false, wenn a wahr ist, und true, wenn a falsch ist.
^	EXCLUSIV-ODER	a ^ b ergibt true, wenn genau einer der beiden Ausdrücke a oder b wahr ist.

Beispiel:

```
public static void main (String []args)
{
  boolean b;
  int iZahl = 10;
  b = (iZahl > 0) && (iZahl < 20);  // b erhält true
}
```

6 Sonstige Operatoren

6.1 Bedingungsoperator "? :"

Der Bedingungsoperator ist der einzige dreistellige Operator, d.h. er verarbeitet drei Operanden.

Allgemeine Form:

ergebnis = <logischer_Ausdruck> **?** <Ausdruck1> **:** <Ausdruck2>;

Ergibt der <logischer_Ausdruck> den Wert <wahr>, so erhält ergebnis den Wert von <Ausdruck1>, sonst den Wert von <Ausdruck2>.

Der Datentyp des Ergebniswerts entspricht dem wertebereichsmäßig größeren der beiden Ausdrücke <Ausdruck1> und <Ausdruck2>.

Beispiel:

```
public static void main (String []args)
{
  int iZahl = 10, iErgebnis;
  iErgebnis = iZahl > 20 ? iZahl * 10 : iZahl; // iErgebnis erhält 10
}
```

6.2 Typecast-Operator

Mit dem Typecast-Operator können explizite Datentyp-Umwandlungen vorgenommen werden. Der Zieldatentyp wird dabei einem Ausdruck in Klammern vorangestellt. Der Ausdruck

(<Datentyp>) <Ausdruck>;

wandelt den eigentlichen Datentyp von <Ausdruck> in den geklammerten <Datentyp> um. Hierbei gelten jedoch bestimmte Regeln, die im Zusammenhang mit dem Thema "Rechnen in Java" besprochen werden. Siehe dazu Merkblatt "Rechnen in Java".

Beispiel:

```
public static void main (String []args)
{
  int iZahl;
  double dZahl = 10.9;
  iZahl = (int) dZahl;
  /*
  * iZahl erhält 10, da ein int keine Nachkommastellen kennt
  */
}
```

6.3 String-Verkettung "+"

Der +-Operator kann neben der Addition von numerischen Operanden auch zur Stringverkettung benutzt werden.

Ist mindestens einer der beiden Operanden im Ausdruck a + b ein String, so wird der gesamte Ausdruck als Stringverkettung ausgeführt, also a und b werden aneinandergehängt und nicht addiert. Der Rückgabewert von a + b ist ein String.

Beispiel:

```
public static void main (String []args)
{
  int iZahl = 5;
  System.out.println("iZahl = " + iZahl);
  /*
   * ergibt als Ausgabe: iZahl = 5
   * vor der Stringverkettung wird die Ganzzahl iZahl implizit in
   * den String "5" konvertiert.
   */
}
```

Vorsicht ist geboten, wenn sowohl Stringverkettung als auch Addition in einem Ausdruck verwendet werden sollen. Hierbei sind die Vorrang- und Ausführungsregeln zu beachten:

Beispiel:

```
public static void main (String []args)
{
  int iZahl = 5, iZahl2 = 10;
  System.out.println("Summe = " + iZahl + iZahl2);
  /*
   * ergibt als Ausgabe: Summe = 510 ,
   * da der gesamte Ausdruck von links nach rechts ausgewertet wird.
   * Soll die Summe ausgegeben werden muss geklammert werden:
   */
  System.out.println("Summe = " + (iZahl + iZahl2));
}
```