Operationen mit Wertdatentypen

Häufig benötigte Grundfunktionalitäten werden in Java (wie in vielen anderen Programmiersprachen) mittels vordefinierter Operationen zur Verfügung gestellt, die durch bestimmte Symbole (Operatoren) gekennzeichnet sind. Manche Operatoren haben mehrere Bedeutungen, die sich an der Art und Reihenfolge der Operanden orientieren. (z.B. der +-Operator bei Strings). Im Gegensatz zu C++ können Java-Operatoren nicht mit eigener Bedeutung versehen (überschrieben) werden. Man unterscheidet zwischen unären (einseitigen) und binären Operationen.

```
a       +       b

Operand Operator Operand
```

Beispiel binäre Operation

```
! a

Operator Operand
```

Beispiel unäre Operation

Es gibt zahlreiche Operationen, von denen in der Regel meist nur die folgenden häufig genutzt werden.

- Arithmetische (mathematische) Operationen
- Relationale (Vergleichs-) Operationen
- Logische Operationen
- Zuweisungsoperationen
- Inkrement- / Dekrement-Operationen

Bei zusammengesetzten Operationen kann die Priorität durch Klammerung verändert werden. Klammern haben immer die höchste Priorität:

1. Arithmetische Operationen:

```
Addition          +
Subtraktion       -
Multiplikation    *
Division          /
Modulus           %
```

Die arithmetischen Operatoren entsprechen den in der Mathematik verwendeten Funktionen. Es gilt auch hier die Punkt-vor-Strichregel.

2. Relationale Operatoren (Vergleichsoperationen)

Die relationalen Operatoren vergleichen zwei Operanden (Zahlen oder Ausdrücke) miteinander und ergeben den Wahrheitswert **`true`** oder **`false`**.

```
Gleich                 ==
Ungleich               !=
Kleiner als            <
Kleiner oder gleich    <=
Größer                 >
Größer oder gleich     >=
```

3. Logische Operatoren

Mit den logischen Operatoren können (logische) Werte und Ausdrücke miteinander verknüpft werden. Hierzu wird der Wahrheitswert des einen Operanden berechnet und mit dem Wahrheitswert des andern Operanden verknüpft. Das Ergebnis ist wiederum ein Wahrheitswert.

```
&&    (UND)   TRUE, wenn beide Operanden TRUE sind

||    (ODER)  TRUE, wenn mindestens ein Operand TRUE

!     (NOT)   negiert den Wahrheitswert eines Ausdrucks
```

Hinweis: Die einstelligen Operatoren & | ^ lassen sich nur auf *Zahlenwerte* beziehen. Sie bedeuten eine bitweise Maskierungs-Operation, deren Ergebnis immer ein Zahlenwert und kein Wahrheitswert ist.

Beispiel:

```
class Operatoren{

    public static void main(String[] args) {

        int a = 10; // binär: IOIO
        int b = 14; // binär: IIIO

        System.out.println(""+ (a & b));  // IOIO
        System.out.println(""+ (a | b));  // IIIO
        System.out.println(""+ (a ^ b));  // OIOO
    }
}
```

Ausgabe:
10
14
4

4. Zuweisungsoperatoren

Bei einer Zuweisung mit = muss der linke Wert (Lvalue) eine Variable sein, der rechte (Rvalue) kann eine Konstante oder eine Variable sein. Die Zuweisung verläuft immer von rechts nach links. Es können mehrere Zuweisungen in einer Zeile stehen:

```
int a, b, c, d;
a = b = c = d = 13;
```

Zusammengesetzte Zuweisungsoperatoren haben dieselbe niedrige Priorität wie die einfache Zuweisung.

```
=          x = y ;
+ =        x = x + y ;
- =        x = x – y ;
* =        x = x * y ;
/ =        x = x / y ; // Vorsicht !
%=         x = x % y ;
```

5. Inkrement / Dekrement - Operatoren

Diese Operatoren sind vom unären Typ und erhöhen bzw. erniedrigen den Wert des Operanden genau um EINS. Sie können auch bei Gleitkommazahlen eingesetzt werden.

`a++;` `++a;`	**entspricht:**	`a = a + 1;`
`a--;` `--a;`	**entspricht:**	`a = a – 1;`

In-/Dekrementoperatoren können dem Operanden voran- oder nachgestellt werden (Präfix / Postfix). Die Unterschiede in der Auswirkung zeigen sich aber erst dann, wenn die Operation in eine Funktion oder einen Ausdruck *eingebettet* ist - z.B. `println( )` .

Beispiel:

```
class Inkrement1 {

     public static void main(String args [] ) {

          int i = 3;

          System.out.println( ++i );

          // Ausgabe: 4 !
     }
}

class Inkrement2 {

     public static void main(String args [] ) {

          int i = 3;

          System.out.println( i++ );

          // Ausgabe: 3 !
     }
}
```