



Java Einführung

Operatoren

Kapitel 2 und 3

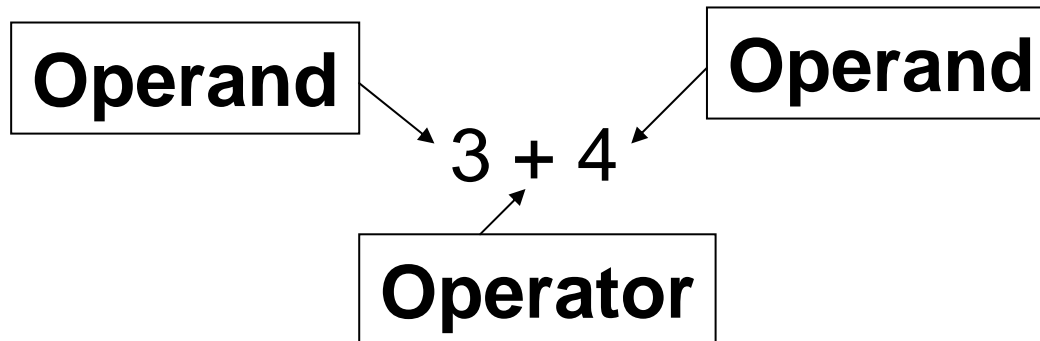
Inhalt dieser Einheit



- Operatoren (unär, binär, ternär)
- Rangfolge der Operatoren
- Zuweisungsoperatoren
- Vergleichsoperatoren
- Logische Operatoren

Operatoren

- Abhängig vom Datentyp können bestimmte Operationen vorgenommen werden
- Operatoren haben ein oder mehrere Operanden



- Die Ausführung der Operationen erfolgt nach einer festgelegten Reihenfolge (d.h. einige Operatoren haben Vorrang vor andern)

Operatoren

- **unäre** Operatoren (1 Operand):
Bsp. ++ (erhöht den Wert des Operanden um 1, `var++`) oder logisches Nicht (!)
- **binäre** Operatoren (2 Operanden):
Bsp. + (Addition der beiden Operanden, `var1 + var2`)
- **ternärer** Operator (3 Operanden):
?: als einziger ternärer Operator in Java erlaubt die verkürzte Implementierung einer if-else-Verzweigung
`(i < j) ? "true" : "false"`

Operatoren im Überblick (Auszug) 1

=	Zuweisung	
+ -	Addition/ Subtraktion	
* /	Multiplikation/ Division	
++ --	Inkrement/ Dekrement	Erhöht oder erniedrigt den Wert der Variablen um 1
%	Modulo	Liefert den Rest einer ganzzahligen Division
(i<j)?"ja":"nein"	if-then-else	Einzigster ternärer Operator in Java. Kurzform der IF-Schleife

Operatoren im Überblick

(Auszug) 2

<code><=</code> <code><</code> <code>>=</code> <code>></code>	Vergleich	Der Vergleich zweier Werte liefert <code>true</code> oder <code>false</code> zurück.
<code>==</code> <code>!=</code>	Vergleich (gleich, ungleich)	Der Vergleich zweier Werte liefert <code>true</code> oder <code>false</code> zurück.
<code>!</code>	logisches NICHT	Negiert den Wahrheitswert einer Aussage
<code>&&</code>	logisches UND	Verknüpft zwei Aussagen. Liefert <code>true</code> , wenn beide Aussagen <code>true</code> sind.
<code> </code>	logisches ODER	Verknüpft zwei Aussagen. Liefert <code>true</code> , wenn <i>eine</i> der beiden Aussagen <code>true</code> sind.

Rangfolge der Operatoren in Java

hoch	. [] ()
	++ -- ! ~ instanceof
	new (data type)
	* / %
	+ -
	<< >> >>>
	<>
	== !=
	&
	^
	&&
	? :
	+= -= *= /= &= ^=
&= = <<= >>= >>>=	
niedrig	

- Die Operatoren oben in der Tabelle werden zuerst ausgewertet.
- Operatoren in der gleichen Zeile haben die gleiche Rangfolge und werden von links nach rechts ausgewertet.

Beim Ausdruck $y = 6 + 4/2$ wissen Sie jetzt anhand der Tabelle, dass die Division vor der Addition ausgewertet wird, deshalb ergibt sich für y ein Wert von 8.

Verkürzte Schreibweise

- `x+=4;` `// x=x+4;`
- `x*=10;` `// x=x*10;`
- `++a;` `// a=a+1;`
- `--a;` `// a=a-1;`

Achtung: `a++` hat eine andere Funktionsweise als `++a`

- `m=n++;` ist wie `m=n;` `n++;`
- `m=++n;` ist wie `n++;` `m=n;`

Der Zuweisungsoperator

(primitive Datentypen) 1

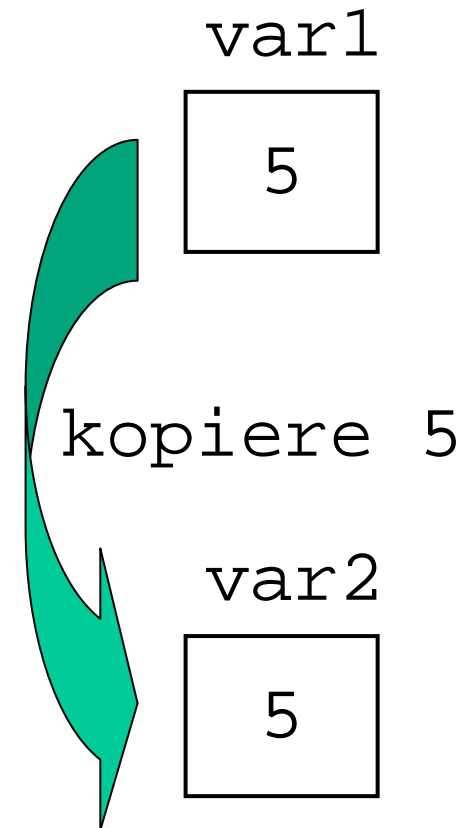
- Operator: =
- Syntax:
Operand1 = Operand2
Weist dem Operanden auf der linken Seite den Wert des Operanden auf der rechten Seite zu.
- Beispiele:
 - Variableninitialisierung: `int i = 2;`
 - Arithmetische Operationen: `i = i + 2;`

Der Zuweisungsoperator (primitive Datentypen) 2

Primitive Datentypen:

Zuweisung des Wertes
entspricht der Kopie des
Wertes in die neue
Variable.

```
var2 = var1;
```



Der Zuweisungsoperator (primitive Datentypen) 3

```
...  
int i1 = 19; // i1 = 19  
int i2 = 47; // i2 = 47  
i1 = i2; // i1 = 47  
i1 = 27+13; // i1 = 40  
i2 = 88; // i2 = 88  
...
```

Der Vergleichsoperator

(Primitive Datentypen)

...

```
int i1 = 19;
```

```
int i2 = 19;
```

```
boolean b = i1 == i2; // b ist true
```

```
boolean c = i1 <= 9; // c ist false
```

...

Logische Operatoren

(Primitive Datentypen)

Logisches UND &&

Logischen ODER || (2x AltGr <)

Negation !

```
int x = 10;
```

```
boolean norm = (x >= 9 && x <= 11);
```

```
boolean result = (0 <= x && x <= 10  
|| 100 <= x && x <= 110);
```

Logische Operatoren

b1	b2	NOT !b1	AND b1 && b2	OR b1 b2
true	true	false	true	true
true	false	false	false	true
false	true	true	false	true
false	false	true	false	false

Wahrheitstabelle

Nach dieser Einheit sollten Sie ...

- Zuweisungsoperator, Vergleichsoperator und logische Operatoren kennen.
- Die Reihenfolge ihrer Ausführung kennen.