

### Einführung: Operatoren

---

- verbinden Variablen und Konstanten zu Ausdrücken
  - unäre Operatoren: 1 Operand
  - binäre Operatoren: 2 Operanden
  - ternäre Operatoren: 3 Operanden
- jeder Operator besitzt „Priorität“ => es herrscht eine Rangfolge unter den Operatoren

### Der Zuweisungsoperator

---

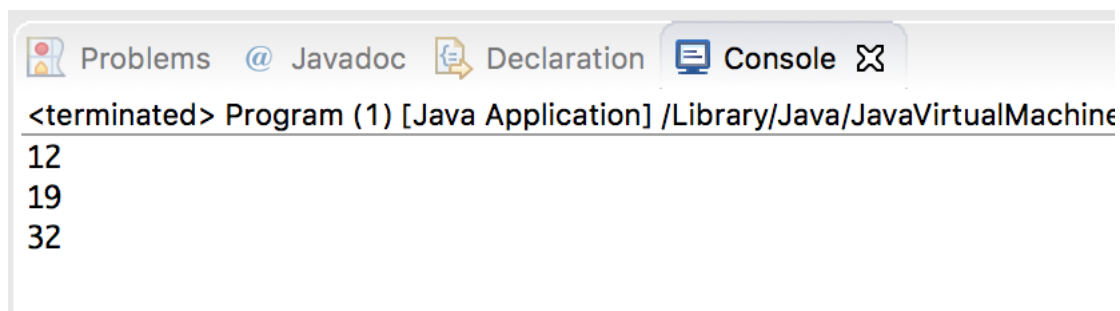
- Zuweisungsoperator: =
- Zuweisungsoperator arbeitet von rechts nach links
- => Operand rechts wird dem Operand links zugewiesen
- Typ des rechten Operanden muss zum Typ des linken Operanden passen

```
int result1 = 12;
```

```
int result2 = 12 + 7;
```

```
int result3 = (12 + 4) * 2;
```

```
System.out.println(result1);  
System.out.println(result2);  
System.out.println(result3);
```



## Die kombinierten Zuweisungsoperatoren

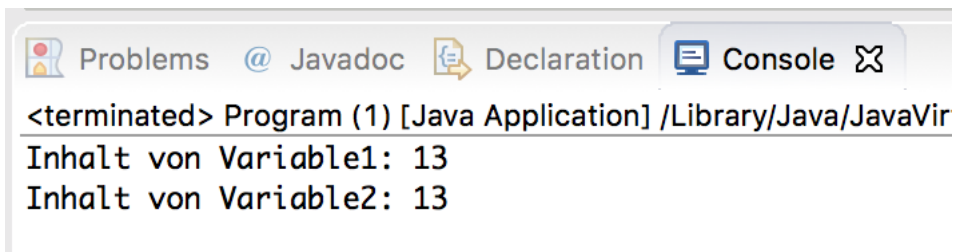
---

-verkürzte Schreibweise, die zu gleichem Ergebnis führt (siehe Beispiel)

```
int variable1 = 3;  
int variable2 = 3;
```

```
variable1 = variable1 + 10;  
variable2 += 10;
```

```
System.out.println("Inhalt von Variable1: " + variable1);  
System.out.println("Inhalt von Variable2: " + variable2);
```



-mit allen Arithmetischen Operatoren möglich (+, -, \*, /, %)

## Arithmetische Operatoren

---

-Grundrechenarten

### Übersicht Arithmetische Operatoren

Operator	Beschreibung	Beispiel
-	Vorzeichen -	zahl = -3;
+	Vorzeichen +	zahl = 3; (oder: zahl = +3;)
+	Addition	zahl = 3 + 4;
-	Subtraktion	zahl = 3 - 4;
*	Multiplikation	zahl = 3 * 4;
/	Division	zahl = 3 / 4;
%	Modulo Operator (Rest der nach Division übrig bleibt)	zahl = 32 % 3; (Ergebnis: 2)

## Der Inkrement und Dekrement Operator

---

-unäre Operatoren

-erhöht bzw. verringert den Wert einer Variable um eine Einheit

-Syntax des Inkrement Operators: ++

-Syntax des Dekrement Operators: --

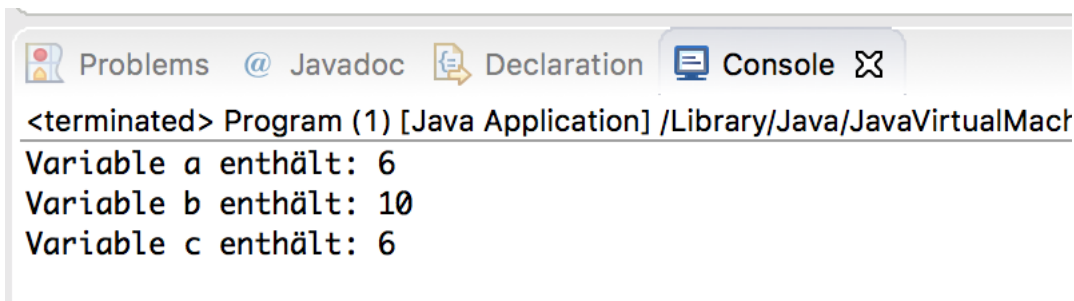
**Achtung:** Es gibt für jeden der beiden Operatoren jeweils 2 mögliche Varianten

### Beispiel für das Präinkrement

```
int a = 5;
int b = 10;
int c = 0;

//Präinkrement
c = ++a;

System.out.println("Variable a enthält: " + a);
System.out.println("Variable b enthält: " + b);
System.out.println("Variable c enthält: " + c);
```



The screenshot shows an IDE console window with the following tabs: Problems, Javadoc, Declaration, and Console. The console output is as follows:

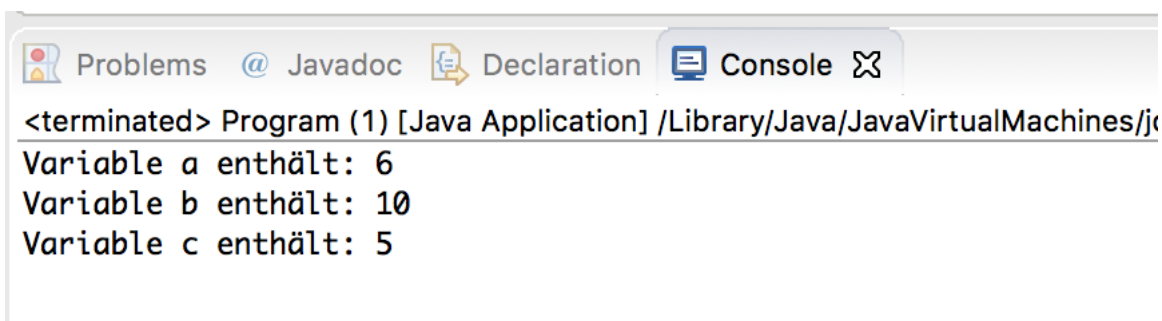
```
<terminated> Program (1) [Java Application] /Library/Java/JavaVirtualMach
Variable a enthält: 6
Variable b enthält: 10
Variable c enthält: 6
```

### Beispiel für das Postinkrement

```
int a = 5;
int b = 10;
int c = 0;

//Postinkrement
c = a++;

System.out.println("Variable a enthält: " + a);
System.out.println("Variable b enthält: " + b);
System.out.println("Variable c enthält: " + c);
```



The screenshot shows an IDE console window with the following tabs: Problems, Javadoc, Declaration, and Console. The console output is as follows:

```
<terminated> Program (1) [Java Application] /Library/Java/JavaVirtualMachines/jc
Variable a enthält: 6
Variable b enthält: 10
Variable c enthält: 5
```

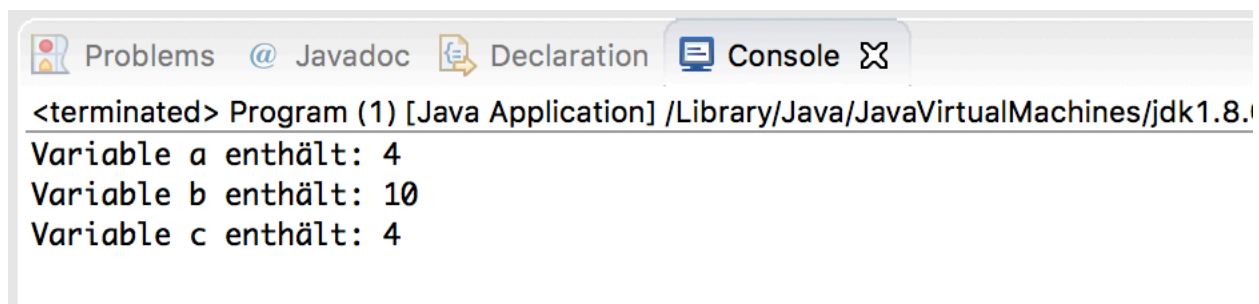
## Beispiel für das Prädekrement

```
int a = 5;
int b = 10;
int c = 0;
```

## //Prädekrement

```
c = --a;
```

```
System.out.println("Variable a enthält: " + a);
System.out.println("Variable b enthält: " + b);
System.out.println("Variable c enthält: " + c);
```



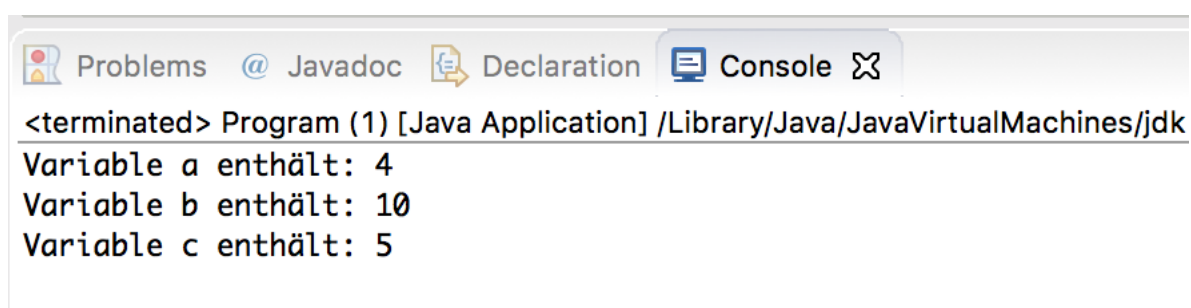
## Beispiel für das Postdekrement

```
int a = 5;
int b = 10;
int c = 0;
```

```
//Postdekrement
```

```
c = a--;
```

```
System.out.println("Variable a enthält: " + a);
System.out.println("Variable b enthält: " + b);
System.out.println("Variable c enthält: " + c);
```



## Die Vergleichsoperatoren

---

-geben immer einen booleschen Rückgabewert zurück

### Übersicht Vergleichsoperatoren

Operator	Beschreibung	Beispiel
<	kleiner als	1 < 6 (true)
>	größer als	1 > 6 (false)
<=	kleiner als oder gleich	4 <= 4 (true)    3 <= 4 (true)
>=	größer als oder gleich	4 >= 4 (true)    6 >= 4 (true)
==	gleich	4 == 4 (true) 3 == 4 (false)
!=	ungleich	4 != 4 (false) 3 != 4 (true)

## Logische Operatoren

---

- geben immer einen booleschen Rückgabewert zurück
- werden meist dazu verwendet, um mehrere Vergleiche zueinander in Beziehung zu setzen

### Übersicht Logische Operatoren

Operator	Beschreibung	Beispiel
<code>  </code>	logisches ODER (kurz: OR)	<code>(a &lt; b)    (c == d)</code>
<code>&amp;&amp;</code>	logisches UND (kurz: AND)	<code>(a &lt; b) &amp;&amp; (c == d)</code>
<code>!</code>	logisches NICHT (kurz: NOT)	<code>!b</code>

## Bitweise Operatoren

-ermöglichen Bitmanipulationen

=> einzelne Bits können gezielt geändert werden

-im Beispiel wird mit Hexadezimalzahlen gearbeitet => falls du nicht weißt was das ist, einfach in das BONUS-Modul Zahlensysteme gehen

### Übersicht Bitweise Operatoren

Operator	Beschreibung	Beispiel
<<	Linksshift (alle Bits nach links schieben)	$x \ll 1$ ; $x \ll 4$ ;
>>	Rechtsshift (alle Bits nach rechts schieben)	$x \gg 1$ ; $x \gg 4$ ;
&	UND-Operator (kurz: AND)	$2 \& 3$
	ODER-Operator (kurz: OR)	$2   3$
^	exklusiv-ODER-Operator (kurz: XOR)	$2 \wedge 3$

### Bitmanipulationen: UND-Operator

(Einzelne Bits auf "0" setzen, während die anderen Bits unverändert bleiben)

```
int a = 6;  
int b = a & 0xFD;
```

#### Nebenrechnung:

(6)	0000 0110
& (0xFD)	1111 1101
<hr/>	
	0000 0100
	➡ 4

#### Funktion der Maske:

Alle Bits die in der Maske "0" sind, werden auf "0" gesetzt.

Alle Bits die in der Maske auf "1" sind bleiben **unverändert**.



## Bitmanipulationen: ODER-Operator

(Einzelne Bits auf "1" setzen, während die anderen Bits unverändert bleiben)

```
int a = 6;  
int b = a | 0x01;
```

**Nebenrechnung:**

(6)	0000 0110
(0x01)	0000 0001
<hr/>	
	0000 0111
	→ 7

### Funktion der Maske:

Alle Bits die in der Maske "1" sind, werden auf "1" gesetzt.  
Alle Bits die in der Maske auf "0" sind bleiben **unverändert**.

## Bitmanipulationen: Exklusiv-ODER-Operator

(Bits werden invertiert)

```
int a = 6;  
int b = a ^ 0x0F;
```

**Nebenrechnung:**

(6)	0000 0110
^ (0x0F)	0000 1111
<hr/>	
	0000 1001
	→ 9

### Funktion der Maske:

Alle Bits die in der Maske "1" sind, werden invertiert.  
Alle Bits die in der Maske auf "0" sind bleiben **unverändert**.

## Operatoren Prioritäten (Rangfolge)

---

- regelt welcher Operator Vorrang hat
- da Rangfolge die Reihenfolge regelt, kann man auf sehr viel Klammersetzung verzichten
- man bekommt schnell ein Gefühl für die Rangfolge

**Nützlicher Link zur Liste mit Rangfolge:** [https://de.wikibooks.org/wiki/Java\\_Standard:\\_Operatoren](https://de.wikibooks.org/wiki/Java_Standard:_Operatoren)