Einstieg in die Informatik mit Java

Ausdrücke

Gerd Bohlender

Institut für Angewandte und Numerische Mathematik

Gliederung

1. Die wichtigsten arithmetischen Ausdrücke
   - Arithmetische Operatoren
   - Inkrement– und Dekrementoperatoren
   - Zuweisungsoperator
   - Mathematische Standardfunktionen
   - Vergleichsoperatoren
2. Kombinierte Zuweisungsoperatoren
3. Logische Operatoren
4. Weitere Operatoren
5. Klassifizierung von Operatoren
6. Priorität der Operatoren
7. Typumwandlungen

Gliederung

1. Die wichtigsten arithmetischen Ausdrücke
   - Arithmetische Operatoren
   - Inkrement– und Dekrementoperatoren
   - Zuweisungsoperator
   - Mathematische Standardfunktionen
   - Vergleichsoperatoren
2. Kombinierte Zuweisungsoperatoren
3. Logische Operatoren
4. Weitere Operatoren
5. Klassifizierung von Operatoren
6. Priorität der Operatoren
7. Typumwandlungen

Arithmetische Operatoren

einstellig:	+ −
zweistellig:	+ − * / %

Achtung

Sind beide Operanden ganzzahlig, so handelt es sich bei `/` (vgl. Pascal: `div`) um die ganzzahlige Division, d. h. Nachkommastellen werden abgeschnitten! Dies gilt auch bei einem negativen Ergebnis!

Beispiel

```
a = 5.0/3.0;   // ergibt a = 1.666
a = 5.0/3;     // ebenso
a = 5/3.0;     // ebenso
b = 5/3;       // ergibt b = 1
c = 5/3.;      // und was ergibt dies?
```

Rest bei der Division

Der Rest bei der Division zweier ganzer Zahlen (vgl. Pascal: `mod`) kann mit Hilfe des Operators % bestimmt werden. Es gilt: `a % b = a - (a/b)*b`.

Beispiele

```
c = 5 % 3;      // ergibt c = 2
d = (-5)/3;     // ergibt d = -1
e = (-5) % 3;   // ergibt e = -2
```

Rest bei der Division

Achtung

Der „Restoperator“ kann auch auf Gleitkommazahlen angewandt werden!

Beispiel

```
f = 3.5 % 1.1;   // ergibt f = 0.2
```

Inkrement- und Dekrementoperatoren

- Der Inkrementoperator `++` erhöht den Operanden um eins (vgl. Pascal: `succ`, `inc`).
- Der Dekrementoperator `--` vermindert den Operanden um eins (vgl. Pascal: `pred`, `dec`).

Präfix– und Postfixvarianten

- Beide Operatoren können sowohl vor als auch nach dem Operanden stehen. Ihre Wirkung auf den Operanden ist dabei gleich, allerdings unterscheiden sich die beiden Varianten im Wert des Ausdrucks:
- Bei der Präfixvariante wird zuerst der Wert des Operanden um eins verändert (erhöht bzw. vermindert) und erst danach der Wert des Ausdrucks zugewiesen,
- bei der Postfixvariante geschieht dies in genau umgekehrter Reihenfolge.

Beispiel

```
i = 1; g = ++i;   // ergibt g = 2 und i = 2.
i = 1; h = i++;   // ergibt h = 1 und i = 2.
```

Zuweisungsoperator

Syntax

Variable = Ausdruck

Der Zuweisungsoperator = steht zwischen einer Variablen (linke Seite) und einem Ausdruck (rechte Seite). Der Ausdruck wird ausgewertet und der Wert anschließend der Variablen zugewiesen. Da es sich um einen Operator handelt, erhält der gesamte Ausdruck ebenfalls den ermittelten Wert.

Achtung

Bei der Zuweisung handelt sich im Gegensatz zu Pascal um einen Operator!
Mehrfachzuweisungen sind erlaubt (`a = b = 1;`)!

Mathematische Standardfunktionen

Mathematische Standardfunktionen sind in Java in der Klasse

`java.lang.Math`

enthalten. Diese Klasse braucht nicht zu importiert werden, sie wird standardmäßig geladen. Allerdings muss bei der Verwendung von Methoden und Konstanten jeweils der Klassenname `Math` vorangestellt werden.

Beispiele

Sinusfunktion: `Math.sin(x)`, Konstante π: `Math.PI`

Seit Java 5 geht auch

```
import static java.lang.Math.*;
```

danach kann `Math.` weggelassen werden.

Tabelle der Standardfunktionen in java.lang.Math

Methode	Funktion	Argumenttyp	Ergebnistyp
`abs(x)`	Absolutbetrag	`int`	`int`
		`long`	`long`
		`float`	`float`
		`double`	`double`
`acos(x)`	Arkuskosinus	`double`	`double`
`asin(x)`	Arkussinus	`double`	`double`
`atan(x)`	Arkustangens	`double`	`double`
`atan2(x,y)`	Konvertiert kartesische	`x`: `double`	`double`
	in polare Koordinaten.	`y`: `double`	
`cos(x)`	Kosinus	`double`	`double`
`ceil(x)`	Nächste grössere ganze Zahl	`double`	`double`
`exp(x)`	Exponentialfunktion	`double`	`double`
`floor(x)`	Nächste kleinere ganze Zahl	`double`	`double`
`log(x)`	Natürlicher Logarithmus	`double`	`double`
`max(x,y)`	Maximum	`int`	`int`
		`long`	`long`
		`float`	`float`
		`double`	`double`
`min(x,y)`	Minimum	`int`	`int`
		`long`	`long`
		`float`	`float`
		`double`	`double`
`pow(x,y)`	`x` hoch `y`	`x`: `double`	`double`
		`y`: `double`	
`random()`	G. v. Zufallszahl aus $[0, 1)$		`double`

Tabelle der Standardfunktionen, Fortsetzung

Methode	Funktion	Argumenttyp	Ergebnistyp
`rint(x)`	Nächste ganze Zahl	`double`	`double`
`round(x)`	Rundet zur	`float`	`int`
	nächsten ganzen Zahl	`double`	`long`
`sin(x)`	Sinus	`double`	`double`
`sqrt(x)`	Wurzel	`double`	`double`
`tan(x)`	Tangens	`double`	`double`
`toDegrees(x)`	Konvertiert Bogen– in Gradmaße	`double`	`double`
`toRadians(x)`	Konvertiert Grad– in Bogenmaße	`double`	`double`

Vergleichsoperatoren

Vergleichsoperatoren liefern boolsche Werte als Ergebnistyp zurück.

Operator	**Bedeutung**
==	gleich
!=	ungleich
<=	kleiner gleich
>=	größer gleich
<	kleiner
>	größer

Gliederung

1. Die wichtigsten arithmetischen Ausdrücke
   - Arithmetische Operatoren
   - Inkrement– und Dekrementoperatoren
   - Zuweisungsoperator
   - Mathematische Standardfunktionen
   - Vergleichsoperatoren
2. Kombinierte Zuweisungsoperatoren
3. Logische Operatoren
4. Weitere Operatoren
5. Klassifizierung von Operatoren
6. Priorität der Operatoren
7. Typumwandlungen

Kombinierte Zuweisungsoperatoren

Kombinierte Zuweisungsoperatoren verkürzen die Schreibweise von Zuweisungen der Art:

```
a = a ○ b;
```

Steht also links wie rechts vom Zuweisungsoperator = dieselbe Variable, kann der Ausdruck auch verkürzt geschrieben werden als:

```
a ○= b;
```

Der Zuweisungsoperator lässt sich mit den folgenden Operatoren kombinieren:

`+=`	`-=`	`*=`	`/=`
`&=`	`|=`	`^=`	`%=`
`<<=`	`>>=`	`>>>=`	

Kombinierte Zuweisungsoperatoren, Beispiele

Beispiel

```
g=1;
h=1;
g  = g + 5; // ergibt g = 6
h += 5;     // ergibt h = 6
```

Achtung

Die Wirkung auf `g` und `h` ist die gleiche, die Anzahl der Auswertungen von `g` und `h` sind dagegen nicht gleich. `g` wird hier zweimal ausgewertet, `h` hingegen nur einmal (wichtig bei Nebeneffekten)!

Gliederung

1. Die wichtigsten arithmetischen Ausdrücke
   - Arithmetische Operatoren
   - Inkrement– und Dekrementoperatoren
   - Zuweisungsoperator
   - Mathematische Standardfunktionen
   - Vergleichsoperatoren
2. Kombinierte Zuweisungsoperatoren
3. Logische Operatoren
4. Weitere Operatoren
5. Klassifizierung von Operatoren
6. Priorität der Operatoren
7. Typumwandlungen

Logische Operatoren

Operator	Bedeutung
`&` `&&`	*Logisches Und* Wahr, falls beide Operanden wahr sind.
`|` `||`	*Logisches Oder* Wahr, falls mindestens einer der Operanden wahr ist.
`!`	*Logische Negation* Wahr, falls der Operand falsch ist.
`~`	*Exklusives Oder* Wahr, falls die Operanden verschieden sind.

Kurzschlussoperatoren

Die Operatoren && und || heißen *Kurzschlussoperatoren*. Sie unterscheiden sich von den Operatoren & und | durch die Art der Auswertung des zugrundeliegenden Ausdrucks:
Kann aus der linken Seite schon das Ergebnis bestimmt werden, so wird die rechte Seite nicht mehr ausgewertet.

Beispiel

```
int i, j;
if ((j!=0) && (i/j < 5))   // i/j wird bei j=0
                           // nicht ausgewertet!
if ((j!=0) & (i/j < 5))    // i/j wird bei j=0
                           // ausgewertet! Fehler!
```

Gliederung

1. Die wichtigsten arithmetischen Ausdrücke
   - Arithmetische Operatoren
   - Inkrement– und Dekrementoperatoren
   - Zuweisungsoperator
   - Mathematische Standardfunktionen
   - Vergleichsoperatoren
2. Kombinierte Zuweisungsoperatoren
3. Logische Operatoren
4. Weitere Operatoren
5. Klassifizierung von Operatoren
6. Priorität der Operatoren
7. Typumwandlungen

Weitere Operatoren

Operator	Bedeutung
`a?b:c`	*Konditionaloperator*: Verkürzte Form für `if ... else ...` Ist `a` wahr, so ist das Ergebnis `b`, sonst `c`.
`&  |  ^  ~`	*Bitoperatoren*: Bitweises Und, Oder, exklusives Oder, bitweise Negation
`a << b`	*Schiebeoperator*: `a` wird um `b` Bits nach links verschoben.
`a >> b`	*Schiebeoperator*: `a` wird um `b` Bits nach rechts verschoben, dabei wird links mit einem Vorzeichenbit aufgefüllt.
`a >>> b`	*Schiebeoperator*: `a` wird um `b` Bits nach rechts verschoben, dabei wird mit Nullen aufgefüllt.

Achtung

Grund für die Existenz des Operators >>> ist, dass es in Java keine `unsigned`–Typen gibt!

Gliederung

1. Die wichtigsten arithmetischen Ausdrücke
   - Arithmetische Operatoren
   - Inkrement– und Dekrementoperatoren
   - Zuweisungsoperator
   - Mathematische Standardfunktionen
   - Vergleichsoperatoren
2. Kombinierte Zuweisungsoperatoren
3. Logische Operatoren
4. Weitere Operatoren
5. Klassifizierung von Operatoren
6. Priorität der Operatoren
7. Typumwandlungen

Klassifizierung von Operatoren

Operatoren werden anhand der folgenden Kriterien klassifiziert:

(1) Anzahl der Operanden (*unär*, *binär*, *ternär*),

(2) Priorität der Operatoren,

(3) Assoziativität: Auswertung erfolgt in Normalfall von links nach rechts, bei unären Operatoren und Zuweisungen von rechts nach links,

(4) Präfix– oder Postfixvariante (nur für ++ und $--$ relevant).

Reihenfolge der Auswertung der Operanden

Achtung

Operanden werden im Gegensatz zu Operatoren immer von links nach rechts ausgewertet!

Beispiel

```
int k=2, l=3, m=4;
k = l = m;          // k,l,m haben alle den Wert 4.
```

Der obige Ausdruck wird wie folgt ausgewertet:

(1) `k`,
(2) `l`,
(3) `m`,
(4) rechte Zuweisung,
(5) linke Zuweisung.

Achtung

Reihenfolge der Auswertungen ist bei Seiteneffekten bedeutsam!

Gliederung

1. Die wichtigsten arithmetischen Ausdrücke
   - Arithmetische Operatoren
   - Inkrement– und Dekrementoperatoren
   - Zuweisungsoperator
   - Mathematische Standardfunktionen
   - Vergleichsoperatoren
2. Kombinierte Zuweisungsoperatoren
3. Logische Operatoren
4. Weitere Operatoren
5. Klassifizierung von Operatoren
6. Priorität der Operatoren
7. Typumwandlungen

Priorität der Operatoren

Priorität	Operator	Assoz.	Bedeutung
15	`.`, `[]`, `()`	L	Komponentenzugriff bei Klassen, Feldern und Methodenaufruf
14	`++`, `--`, `+`, `-`, `!`, `~` `new`	R	Unäre Operatoren Instanzbildung
13	(*Typ*)	R	Explizite Typkonvertierung
12	`*`, `/`, `%`	L	Multiplikative Operatoren
11	`+`, `-`	L	Additive Operatoren
10	`<<`, `>>`, `>>>`	L	Schiebeoperatoren
9	`<`,`>`,`<=`, `>=` `instanceof`	L	Vergleichsoperatoren
8	`==`, `!=`	L	Vergleichsoperatoren
7	`&`	L	Und–Operator (bitweise, logisch)
6	`^`	L	Exklusives Oder (bitweise, logisch)
5	`|`	L	Oder–Operator (bitweise, logisch)
4	`&&`	L	(Logisches Und)–Operator (Kurzschluß–Auswertung)
3	`||`	L	(Logisches Oder)–Operator (Kurzschluß–Auswertung)
2	`?:`	R	Konditionaloperator
1	`=`, `+=`, `*=`, usw.	R	Zuweisungsoperatoren

Reihenfolge der Auswertung

- Die höchste Priorität wird mit der Zahl 15 gekennzeichnet, die niederste mit der Zahl 1.
- Operatoren mit gleicher Priorität werden von links nach rechts („Assoz.“ = L = linksassoziativ) bzw. von rechts nach links („Assoz.“ = R = rechtsassoziativ) ausgewertet.
- Außerdem kann die Reihenfolge der Operationen immer durch Klammern `()` festgelegt werden.

Gliederung

1. Die wichtigsten arithmetischen Ausdrücke
   - Arithmetische Operatoren
   - Inkrement– und Dekrementoperatoren
   - Zuweisungsoperator
   - Mathematische Standardfunktionen
   - Vergleichsoperatoren
2. Kombinierte Zuweisungsoperatoren
3. Logische Operatoren
4. Weitere Operatoren
5. Klassifizierung von Operatoren
6. Priorität der Operatoren
7. Typumwandlungen

Typumwandlungen, implizit

(a) Umwandlung in „größere“ Typen geht automatisch.

`byte` → `short` → `int` → `long` → `float` → `double`

`char` → `int`

Achtung

Von `long` nach `float` können Rundungsfehler auftreten!

Typumwandlungen, explizit

(b) Umwandlungen in „kleinere“ Typen können zu Fehlern führen und müssen deshalb explizit dem Kompiler mitgeteilt werden. Dies ist mit Hilfe des Cast–Operators möglich:

`double` $\longrightarrow$ `float` $\longrightarrow$ `long` $\longrightarrow$ `int` $\longrightarrow$ `short` $\longrightarrow$ `byte`

`int` $\longrightarrow$ `char` $\longrightarrow$ `byte`

`short` $\longleftrightarrow$ `char`

Syntax

(Typname) Ausdruck

Typumwandlungen, Beispiel

Beispiel

```
double d;
int i = (int) d;
```

Typumwandlungen von/nach boolean

Achtung

Es existieren keine Konversionen von oder nach `boolean`, allerdings ist folgendes möglich:

Beispiel

```
boolean b, int i;
b = i != 0;
// wandelt Null in false, Eins in true um.

i = b?1:0;
// wandelt false in Null, true in Eins um.
```

Verwendete Datentypen

Aus Effizienzgründen wird nicht in den Datentypen `byte`, `short` und `char` gerechnet, sondern in `int`.
Bei gemischten Ausdrücken wird im größeren Typ gerechnet.

Beispiel

```
byte a, b;
byte c = (byte) (a + b);
// Cast, da (a+b) vom Typ int
```