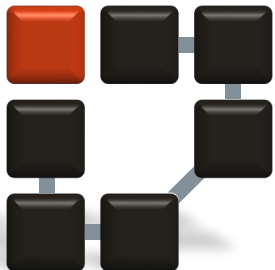


Informatik 1 für Nebenfachstudierende Grundmodul

Java – Operatoren und Ausdrücke

Kai-Steffen Hielscher
Folienversion: 9. Dezember 2019



Informatik 7
Rechnernetze und
Kommunikationssysteme



FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Inhaltsübersicht

- Kapitel 3 - Java
 - Einführung
 - Lexikalische Struktur
 - Datentypen und Variablen
 - **Operatoren und Ausdrücke**
 - Anweisungen und Ablaufsteuerung

Operatoren und Ausdrücke

- Einführung Operatoren und Ausdrücke
- Arithmetische Operatoren
- Bitoperatoren
- Vergleichsoperatoren
- Logische Operatoren
- Bedingungsoperator

Operatoren

- Operatoren verknüpfen Operanden (Werte) miteinander
 - einstellige (unäre, monadische) Operatoren haben einen Operanden
 - Präfix-Notation
Operator Operand
 - Postfix-Notation
Operand Operator
 - zweistellige (binäre, dyadische) Operatoren haben zwei Operanden
 - Infix-Notation
Operand1 Operator Operand2
 - dreistellige (ternäre, triadische) Operatoren haben drei Operanden
Operand1 ? Operand2 : Operand3
 - **Ergebnis: Resultat der Verknüpfung als Wert**
 - **elementare Ausdrücke**

Operatoren

- **Komplexe Ausdrücke** durch Zusammensetzung, mehrfache Anwendung von Operatoren, z.B.

```
double d = 3.0 / 4.0 - 7.0 / (8.0 * 17.0) / 2.0
```

- Probleme

- Assoziativität der Abarbeitung: Reihenfolge
- Priorität der Operationen: Rangfolge

- In Programmiersprachen ähnlich gelöst wie im gewöhnlichen Rechnen

- festgelegte Hierarchie von Operatoren
- festgelegte Assoziativität von Operatoren
- Punkt- vor Strichrechnung wird automatisch erkannt
- Klammerregeln entsprechen den uns geläufigen, d.h. Klammern werden von innen heraus aufgelöst

- Bsp. von oben ergibt mit Klammern:

```
((3.0 / 4.0) - ((7.0 / (8.0 * 17.0)) / 2.0))
```

Arithmetische Operatoren

■ Addition

- $a + b$
- bei Strings: Konkatenation (Aneinanderhängen)

■ Subtraktion

- $a - b$

■ Multiplikation

- $a * b$

■ Division

- a / b
- welche Division verwendet wird, hängt vom Datentyp der Operanden ab
 - Ganzzahldivision bei ganzzahligen Datentypen
 - Fließkommadivision, wenn mindestens ein Operand Fließkomma-Typ ist

■ Modulo-Operation

- $a \% b$
- Divisionsrest bei Ganzzahldivision von a durch b

Arithmetische Operatoren

■ Typ des Ergebnisses

1. einer der Operanden `double`: Ergebnis ist `double`, falls anderer Operand kein `double`, wird er implizit nach `double` konvertiert, dann wird die Operation ausgeführt
2. einer der Operanden `float`: Ergebnis ist `float`, falls anderer Operand kein `float`, wird er implizit nach `float` konvertiert, dann wird die Operation ausgeführt
3. einer der Operanden `long`: Ergebnis ist `long`, falls anderer Operand kein `long`, wird er implizit nach `long` konvertiert, dann wird die Operation ausgeführt
4. einer der Operanden `int`: Ergebnis ist `int`, falls anderer Operand kein `int`, wird er implizit nach `int` konvertiert, dann wird die Operation ausgeführt

■ gegebenenfalls explizite Typumwandlung durch den Programmierer nötig

Arithmetische Operatoren

- einstelliger Identitätsoperator
 - `+a`
- einstelliger Negationsoperator, liefert negativen Wert
 - `-a`
- einstelliger Inkrement-Operator
 - erhöht den Wert um eins
 - Postfix
 - ◆ `a++`
 - Präfix (leicht unterschiedliche Bedeutung)
 - ◆ `++a`
- einstelliger Dekrement-Operator
 - vermindert den Wert um eins
 - Postfix
 - ◆ `a--`
 - Präfix (leicht unterschiedliche Bedeutung)
 - ◆ `--a`

Arithmetische Operatoren

■ Zuweisungsoperator

- in den Beispielen schon verwendet
- weist einer Variable einen Wert zu

- `a = 5;`

- auch mehreren Variablen gleichzeitig, rechtsassoziativ (von rechts nach links)

- `a = b = c = 5;`

■ Kombination von Zuweisung und anderen arithmetischen Operatoren

- `a = a + b;` als `a += b;`
- `a = a - b;` als `a -= b;`
- `a = a * b;` als `a *= b;`
- `a = a / b;` als `a /= b;`
- `a = a % b;` als `a %= b;`
- auch für binäre Operatoren definiert.

Arithmetische Operatoren

- Zuweisung in Kombination mit Postfix- und Präfix-Inkrement
 - bei Präfix erst Inkrementieren, dann Zuweisung
 - `a = 1;`
`b = ++a;`
 - `a` hat nun den Wert 2, `b` hat den Wert 2
 - bei Postfix erst Zuweisung, dann Inkrementieren
 - `a = 1;`
`b = a++;`
 - `a` hat nun den Wert 2, `b` hat den Wert 1

Bitoperatoren

- für ganzzahlige Operatoren
- operiert stellenweise auf der Bitdarstellung
- relevant u.a. für hardwarenahe Programmierung
- Beispiel
 - `a = 0b10101010;`
`b = 0b00001111;`
- bitweise Negation (einstellig)
 - `c = ~a;`
`// c = 0b01010101`
- bitweises Und
 - `c = a & b;`
`// c = 0b00001010;`
- bitweises Oder
 - `c = a | b;`
`// c = 0b10101111;`
- bitweises XOR
 - `c = a ^ b;`
`// c = 0b10100101;`

Bitoperatoren

- Schiebeoperatoren
- im Beispiel
 - `a = 0b10101010;`
- Verschieben nach links um bestimmte Anzahl von Stellen (Bits), auffüllen mit 0
 - `c = a << 1;`
`// c = 0b01010100`
- Verschieben nach rechts um bestimmte Anzahl von Stellen (Bits), auffüllen mit höchstem Bit
 - `c = a >> 1;`
`// c = 0b11010101`
- Verschieben nach rechts um bestimmte Anzahl von Stellen (Bits), auffüllen mit 0
 - `c = a >>> 1;`
`// c = 0b01010101`

Bitoperatoren

- Kombination mit Zuweisungsoperator

- `&=`, `|=`, `^=`, `<<=`, `>>=`, `>>>=`

- Beispiel:

- `a = 0b10101010;`
`b = 0b00001111;`
`a &= b;`
`// jetzt: a = 0b00001010`
`a |= b;`
`// jetzt: a = 0b10101111`
`a >>>= 2;`
`// jetzt: a = 0b00101010`

Vergleichsoperatoren

- binäre Operatoren (zwei Operanden)
- vergleicht die Operanden
- Wert des Ergebnisses immer ein **boolescher Wert**, Typ `boolean`, Wert `true` oder `false`
- daher auch **boolesche** Operatoren
- meist nur für primitive Datentypen sinnvoll anwendbar
- für Referenzdatentypen spezielle Methoden für Vergleiche
- mehrere Vergleichsoperatoren können durch **logische Operatoren** zu komplexen logischen Ausdrücken verknüpft werden

Vergleichsoperatoren

- Gleichheit

- $a == b$

- Ungleichheit

- $a != b$

- größer als

- $a > b$

- größer oder gleich

- $a >= b$

- kleiner als

- $a < b$

- kleiner oder gleich

- $a <= b$

Logische Operatoren

- alle Operanden sind boolesche Werte
- logisches Und
 - `a & b`
- logisches Und, zweiter Operand (hier: `b`) wird nur ausgewertet, wenn Ergebnis nicht schon nach Auswertung des ersten Operanden (hier: `a`) feststeht
 - `a && b`
- logisches Oder
 - `a | b`
- logisches Oder, zweiter Operand (hier: `b`) wird nur ausgewertet, wenn Ergebnis nicht schon nach Auswertung des ersten Operanden (hier: `a`) feststeht
 - `a || b`

Logische Operatoren

- logisches exklusives Oder (XOR)
 - $a \wedge b$
- einstellige logische Negation
 - $\neg a$

Bedingungsoperator

- dreistellig

Operand1 ? Operand2 : Operand3

- Operand1: boolescher Wert, Vergleich

- Wert des Ergebnisses

- Operand2, wenn Operand1 zu `true` evaluiert
- ansonsten Operand3

- Operand2 und Operand3: gleicher Typ

- numerischer Datentyp
- boolescher Wert
- String

- Beispiel:

`(a == 10) ? "a hat den Wert 10" : "a ist nicht 10";`