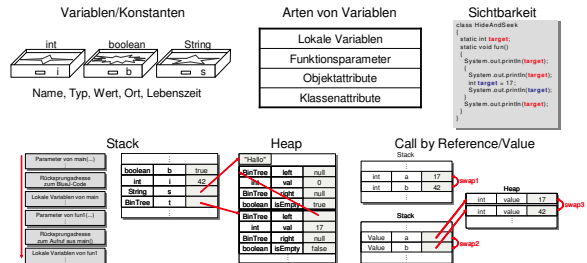


Operatoren und Ausdrücke

Thomas Röfer

Operatoren
Schreibweisen von Operatoren
Arten von Operatoren
Vorrang von Operatoren
Typanalyse von Ausdrücken

Rückblick „Variablen, Konstanten und Referenzen“



Operatoren

- Definition
 - Operatoren sind Funktionen, die anders notiert werden
 - In Java gibt es nur vordefinierte Operatoren mit festgelegter Bedeutung
 - Man kann weder neue Operatoren definieren, noch welche überladen
- Beispiel
 - Das „+“ in „a + b“ könnte einer Funktion „int javaPlusInt(int,int)“ entsprechen, die zwei Ganzzahlen addiert und das Ergebnis zurückliefert
- Typen
 - Operatoren haben einen Typ, z.B.
 - : int → int
 - + : int × int → int
 - == : int × int → boolean

Operatorschreibweisen

- Präfix
 - Operator steht vor dem Operanden
 - Z.B. unäres Minus: -a
- Postfix
 - Operator steht hinter dem Operanden
 - Z.B. Fakultät: a! (nicht in Java); Postinkrement: a++
- Infix
 - Operator steht zwischen zwei Operanden
 - Z.B. Addition: a + b
- Roundfix
 - Mehrteiliger Operator kreist seinen Operanden ein
 - Z.B. Klammern: (a + b)
- Mixfix
 - Mehrteiliger Operator steht zwischen seinen Operanden
 - Z.B. Integral: ∫ f(x,y) dx (nicht in Java); Bedingungsoperator: a > b ? a : b

Überladungen von Operatoren

- Viele Operatoren gibt es in mehreren Überladungen
 - Jede Überladung hat einen eigenen Eingabebereich
 - Jede Überladung definiert eine eigene Funktion
- Beispiel: +
 - + : int × int → int
 - + : long × long → long
 - + : float × float → float
 - + : double × double → double
 - + : String × String → String
 - + : long × String → String
 - + : String × long → String
 - + : double × String → String
 - + : String × double → String

Arten von Operatoren

- Arithmetische Operatoren
 - z.B. '+' oder '-'
- Bit-Operatoren
 - z.B. '>>' oder '&'
- Vergleichsoperatoren
 - z.B. '>' oder '<'
- Logische Operatoren
 - z.B. '&&' oder '||'
- Zuweisungsoperatoren
 - z.B. '=' oder '+='
- Sonstige
 - z.B. '(type)' oder '?.'

Arithmetische Operatoren

- ▶ **Allgemein**
 - ▶ Arithmetische Operatoren verrechnen ein oder zwei Werte zu einem neuen Wert
 - ▶ Bei zwei Operanden müssen beide einen kompatiblen Typ haben
 - ▶ Der Ergebnistyp ist identisch mit dem „größeren“ Typ der beiden Operanden
- ▶ **Operatoren**
 - ▶ Addition: $a + b$, $17 + 4$, $17 + 4.0$
"a" + "b" ("ab"), $17 + "4"$ ("174"), $"17" + 4$. ("174.0")
 - ▶ Subtraktion: $a - b$, $17 - 4$, $17 - 4.0$
 - ▶ Multiplikation: $a * b$, $17 * 4$, $17 * 4.0$
 - ▶ Division: a / b , $17 / 4$ ($= 4$), $17 / 4.0$ ($= 4.25$)
 - ▶ Modulo (Divisionsrest): $a \% b$, $17 \% 4$ ($= 1$), $17.5 \% 4.5$ ($= 4.0$)

Arithmetische Operatoren

- ▶ **Ganzzahlarithmetik**
 - ▶ Überlauf ist kein Fehler
 - ▶ Das Ergebnis wird auf die entsprechende Bit-Anzahl gekappt
 - ▶ Division durch 0 ist nicht erlaubt (erzeugt *ArithmeticException: / by zero*)
- ▶ **Fließkommaarithmetik**
 - ▶ Division durch 0 ist kein Fehler!
 - ▶ $+x / 0.0 == \text{Double.POSITIVE_INFINITY}$
 - ▶ $-x / 0.0 == \text{Double.NEGATIVE_INFINITY}$
 - ▶ $0.0 / 0.0 == \text{Double.NaN}$
 - ▶ Bei einem Überlauf ist das Ergebnis *Double.POSITIVE_INFINITY* oder *Double.NEGATIVE_INFINITY*

Logische Operatoren

- ▶ **Allgemein**
 - ▶ Logische Operatoren verrechnen ein oder zwei Wahrheitswerte zu einem neuen Wahrheitswert
- ▶ **Operatoren**
 - ▶ Logisches Nicht: $!(age > 18) == (age <= 18)$
 - ▶ Logisches Und: $age > 18 \ \& \ age < 65$
 - ▶ Logisches Oder: $age <= 18 \ || \ age >= 65$
 - ▶ Logisches Exklusiv-Oder: $age <= 18 \ \wedge \ age >= 65$

&	false	true
false	false	false
true	false	true

und

	false	true
false	false	true
true	true	true

oder

^	false	true
false	false	true
true	true	false

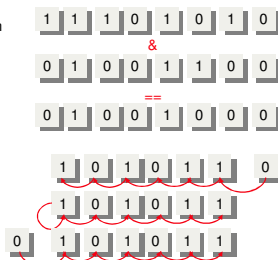
exklusiv-oder

Logische Operatoren

- ▶ **Operatoren**
 - ▶ Logisches Und: $age > 18 \ \&\& \ age < 65$
 - ▶ Logisches Oder: $age <= 18 \ || \ age >= 65$
- ▶ **Unvollständige Auswertung**
 - ▶ $if(age > 18 \ \&\& \ age < 65) \dots$ entspricht $if(age > 18)$
 $if(age < 65) \dots$
 - ▶ $if(age <= 18 \ || \ age >= 65) \dots$ entspricht $if(age > 18) \dots$
 $else \ if(age < 65) \dots$
- ▶ Nützlich z.B. für
 - ▶ $if(d != 0 \ \&\& \ 100 / d > 1)$
- ▶ oder
 - ▶ $if(d == 0 \ || \ 100 / d > 1)$

Bit-Operatoren

- ▶ **Allgemein**
 - ▶ Bit-Verknüpfungen können auf Ganzzahltypen angewendet werden
- ▶ **Verknüpfungen**
 - ▶ Und: $0xea \ \& \ 0x4c$
 - ▶ Oder: $0xea \ || \ 0x4c$
 - ▶ Exklusiv-Oder: $0xea \ \wedge \ 0x4c$
 - ▶ Nicht: $\sim 0xea$
- ▶ **Verschiebungen**
 - ▶ Links: $0x2b \ \ll \ 1$
 - ▶ Rechts: $0x2b \ \gg \ 1$
 - ▶ Vorzeichenlos rechts: $0x2b \ \ggg \ 1$



Vergleichsoperatoren

- ▶ **Allgemein**
 - ▶ Vergleichsoperatoren liefern immer einen Wahrheitswert (boolean) zurück
 - ▶ Beide Operanden müssen einen kompatiblen Typ haben
- ▶ **Numerische Operatoren**
 - ▶ Gleichheit: $age == 18$
 - ▶ Ungleichheit: $age != 18$
 - ▶ Größer: $age > 18$
 - ▶ Größer oder gleich: $age >= 18$
 - ▶ Kleiner: $age < 18$
 - ▶ Kleiner oder gleich: $age <= 18$
- ▶ **Für Objekte (z.B. Strings)**
 - ▶ Identität: $s1 == s2$ (Ist s1 dieselbe Zeichenkette wie s2?)
 - ▶ Gleichheit: $s1.equals(s2)$ (Ist kein Operator sondern eine Funktion)

Zuweisungsoperatoren

- ▶ **Allgemein**
 - ▶ Zuweisungsoperatoren weisen einen R-Wert einem L-Wert zu und liefern den R-Wert als Ergebnis
 - ▶ Sie sind die einzigen Operatoren mit Seiteneffekt
- ▶ **Zuweisung**
 - ▶ `i = 7; s = "Hallo"; a[a[i]][2][17] = 42 + i; x = y = z = 0; a[x = 7] = 1;`
- ▶ **Rechnen und Zuweisung**
 - ▶ `i += 17; a[a[1]] <<= 4; x *= x;`
 - ▶ **L-Wert op= R-Wert entspricht L-Wert = L-Wert op R-Wert, aber der L-Wert wird nur einmal ausgewertet**
- ▶ **Beispiel**
 - ▶ `a += 7` entspricht `a = a + 7`
 - ▶ `b[a++] += 7` entspricht nicht `b[a++] = b[a++] + 7`

Zuweisungsoperatoren

- ▶ **Pre-Inkrement/Dekrement**
 - ▶ Verändert den Wert der Variablen und liefert diesen als Ergebnis zurück
 - ▶ `++i; ++a[5]; a[--i] = a[i];`
- ▶ **Post-Inkrement/Dekrement**
 - ▶ Verändert den Wert der Variablen, liefert aber den alten Wert als Ergebnis zurück
 - ▶ `i++; a[5]++; a[i--] = a[i];`
- ▶ **Beispiel 1**

```
int i = 0;           int j = 0;
a[i++] = 17;        b[++] = 42;
a[i++] = 18;        b[++] = 43;
a[0] == 17, a[1] == 18    b[1] == 42, b[2] == 43
```
- ▶ **Beispiel 2**

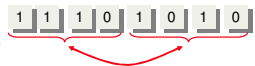
```
class Counter
{
    int counter;
    int preinkrement() {counter = counter + 1; return counter;}
    int postinkrement() {int old = counter; counter = counter + 1; return old;}
}
```

Sonstige Operatoren

- ▶ **If-Else**
 - ▶ *bedingung ? ausdruck : ausdruck*
 - ▶ z.B. `a = b == 0 ? 0 : 100 / b;`
`r = r >= Math.PI ? r - 2 * Math.PI : r < -Math.PI ? r + 2 * PI : r;`
- ▶ **new**
 - ▶ `new` erzeugt neue Objekte oder Arrays
 - ▶ z.B. `int[] a = new int[10];`
`FileInputStream stream = new FileInputStream("aFile");`
- ▶ **Typumwandlung**
 - ▶ `int a = (int) 3.1415`
- ▶ **Test auf Klassentyp**
 - ▶ `Factorial f = new Factorial();`
 - ▶ `bool b = f instanceof Factorial;`

Vorrang von Operatoren

- ▶ **Bindungskraft (Präzedenz)**
 - ▶ Was wird zuerst ausgewertet, wenn verschiedene Operatoren in einem Ausdruck verwendet werden?
 - ▶ z.B. Punkt- vor Strichrechnung: `1 + 2 * 3 = 1 + (2 * 3) = 7`
- ▶ **Assoziativität**
 - ▶ Was wird zuerst ausgewertet, wenn Operatoren gleicher Bindungsstärke in einem Ausdruck verwendet werden?
 - ▶ z.B. links bei Minus: `3 - 2 - 1 = (3 - 2) - 1 = 0`
- ▶ **Beispiel**
 - ▶ `a = a >> 4 & 0x0f | a << 4 & 0xf0`
 - ▶ `a = ((a >> 4) & 0x0f) | ((a << 4) & 0xf0)`



Vorrang von Operatoren

- | | Asso. | Beschreibung |
|--|--------|--|
| ▶ <code>..()</code> | links | Funktionsaufruf und Array-Index |
| ▶ <code>-,+,-,+,+,-</code> | rechts | Vorzeichen, logisches Nicht, arithm. Nicht, In-, Dekrement |
| ▶ <code>new.(typ)</code> | rechts | Objekterzeugung und Typumwandlung |
| ▶ <code>*,/,%</code> | links | Multiplikation, Division, Modulo |
| ▶ <code>+,-</code> | links | Addition und Subtraktion |
| ▶ <code><<,>>,>>></code> | links | Links- und Rechtsverschiebung |
| ▶ <code><,<=,>,>=,instanceof</code> | links | Kleiner, Kleiner-Gleich, Größer, Größer-Gleich |
| ▶ <code>==,!=</code> | links | Gleichheit und Ungleichheit |
| ▶ <code>&</code> | links | arithmetisches Und |
| ▶ <code>^</code> | links | arithmetisches Exklusiv Oder |
| ▶ <code> </code> | links | arithmetisches Oder |
| ▶ <code>&&</code> | links | logisches Und |
| ▶ <code> </code> | links | logisches Oder |
| ▶ <code>?:</code> | links | arithmetisches If-Else |
| ▶ <code>=,+=,-=,*=,/=,%=,^=</code> | rechts | Zuweisung und Rechenzuweisung |
| ▶ <code>&=, =,<<=,>>=,>>>=</code> | rechts | Rechenzuweisung |

Typerweiterung

- ▶ **Automatische Typkonvertierung (Typerweiterung)**
 - ▶ `byte` → `short` → `int` → `long` → `float` → `double`
 - ▶ `char` → `int`
- ▶ **Entspricht „unsichtbaren“ Operatoren**
 - ▶ `: byte` → `short`
 - ▶ `: short` → `int`
 - ▶ `: char` → `int`
 - ▶ `: int` → `long`
 - ▶ `: long` → `float`
 - ▶ `: float` → `double`
- ▶ **Wahl der Überladung eines Operators**
 - ▶ Es wird immer die Überladung eines Operators gewählt, die die wenigsten Typerweiterungen erfordert
 - ▶ Ausnahme: konstante Ausdrücke werden nach der Berechnung wieder auf den kleinsten Typ reduziert, in dem das Ergebnis noch darstellbar ist

Typanalyse von Ausdrücken

- ▶ **Beispiel**
 - ▶ `int i; byte b;`
 - ▶ `i > b ? b * 4 : i * 2`
 - ▶ Was ist der Typ jedes Teilausdrucks?
 - ▶ Welche Typweiterungen müssen vorgenommen werden?
- ▶ **Ansatz**
 - ▶ Beginne mit dem Typ des äußersten Operators
 - ▶ Die Typen der Operanden, die noch nicht eindeutig bestimmt sind, werden durch freie Typvariablen beschrieben
 - ▶ Nach und nach werden die Typen der Operanden durch die Typen der Ausdrücke ersetzt, die als Operanden angegeben sind.
 - ▶ Bei der Ersetzung muss der Ergebnistyp des Ausdrucks gleich dem Typ des Operanden sein (Unifikation). Dadurch werden immer mehr freie Typvariablen gebunden.
 - ▶ Werden verschiedene Typen unifiziert, müssen Typweiterungen eingefügt werden.
 - ▶ Ist dies nicht möglich, ist der Ausdruck nicht typkorrekt.

Typanalyse von Ausdrücken: Beispiel

- ▶ **Typ von ? :**
 - ▶ $?: : \text{boolean} \times \alpha \times \alpha \rightarrow \alpha$
- ▶ **1. Operand ist > : $\alpha' \times \alpha'' \rightarrow \text{boolean}$**
 - ▶ $(\alpha' \times \alpha'' \rightarrow \text{boolean}) \times \alpha \times \alpha \rightarrow \alpha$
- ▶ **2. Operand ist * : $\alpha'' \times \alpha'' \rightarrow \alpha''$**
 - ▶ Ergebnistyp muss gleich sein, d.h. $\alpha'' = \alpha$
 - ▶ $(\alpha' \times \alpha' \rightarrow \text{boolean}) \times (\alpha \times \alpha \rightarrow \alpha) \times \alpha \rightarrow \alpha$
- ▶ **3. Operand ist * : $\alpha'' \times \alpha'' \rightarrow \alpha''$**
 - ▶ Ergebnistyp muss gleich sein, d.h. $\alpha'' = \alpha$
 - ▶ $(\alpha' \times \alpha' \rightarrow \text{boolean}) \times (\alpha \times \alpha \rightarrow \alpha) \times (\alpha \times \alpha \rightarrow \alpha) \rightarrow \alpha$
- ▶ **i : $\text{int} \rightarrow \alpha' = \text{int}, \text{b} : \text{byte}$**
 - ▶ Typweiterung (`byte` \rightarrow `short` \rightarrow `int`) einfügen
 - ▶ $(\text{int} \times (\text{byte} \rightarrow \text{short} \rightarrow \text{int}) \rightarrow \text{boolean}) \times (\alpha \times \alpha \rightarrow \alpha) \times (\alpha \times \alpha \rightarrow \alpha) \rightarrow \alpha$
- ▶ **i : $\text{int} \rightarrow \alpha = \text{int}, \text{b} : \text{byte}$**
 - ▶ Typweiterung (`byte` \rightarrow `short` \rightarrow `int`) einfügen
 - ▶ $(\text{int} \times (\text{byte} \rightarrow \text{short} \rightarrow \text{int}) \rightarrow \text{boolean}) \times ((\text{byte} \rightarrow \text{short} \rightarrow \text{int}) \times \text{int} \rightarrow \text{int}) \times (\text{int} \times \text{int} \rightarrow \text{int}) \rightarrow \text{int}$

- ▶ **Ausdruck**
 - ▶ `int i; byte b;`
 - ▶ `i > b ? b * 4 : i * 2`
- ▶ **Typen der Operatoren**
 - ▶ `?`: $\text{boolean} \times \alpha \times \alpha \rightarrow \alpha$
 - ▶ `>`: $\alpha \times \alpha \rightarrow \text{boolean}$,
 $\alpha \in \{\text{byte}, \dots, \text{double}\}$
 - ▶ `*`: $\alpha \times \alpha \rightarrow \alpha$,
 $\alpha \in \{\text{int}, \dots, \text{double}\}$