Programmiertechnik
Operatoren, Kommentare, Ein-/Ausgabe

Prof. Dr. Oliver Haase

Was sind Operatoren?

- Ein ***Operator*** ist eine in die Programmiersprache eingebaute ***Funktion***, die auf einem oder mehreren ***Operanden*** arbeitet.
- Er dient dazu, aus Werten neue, ***komplexere*** Werte zu berechnen.
- Die Anwendung eines Operators auf ein oder mehrere Operanden heißt ***Operation***.
- Die ***Stelligkeit*** eines Operators gibt die Anzahl der Operanden an. In Java gibt es:
  - einstellige (*unäre, monadische*) Operatoren
  - zweistellige (*binäre, dyadische*) Operatoren
  - dreistellige (*ternäre, triadische*) Operatoren

Was sind Operatoren?

- Jede Operation liefert einen ***Ergebniswert***.
- *Beispiele*:
  - Addition: `a + b`
  - Division: `zaehler / 3.7`
  - logische Verneinung: `!isEmpty`
  - logisches oder: `isEmpty | gefunden`
  - Vergleich: `i < 17`
- Operatoren sind ***typisiert***, d.h. erwarten Operanden eines bestimmten Typs → '+' erwartet numerische Operanden, '|' erwartet boolsche Operanden.

Operatorengruppen

Die Java-Operatoren können in die folgenden Gruppen eingeteilt werden:

- arithmetische Operatoren
- logische Operatoren
- Vergleichsoperatoren
- Inkrement- und Dekrementoperatoren
- Bitoperatoren (*nicht behandelt*)
- Zuweisungsoperatoren
- Ternäroperator
- →*Objekt- und Klassenoperatoren*

Die letzte Gruppe wird später behandelt.

Arithmetische Operatoren

- Operanden müssen
  - numerisch sein, d.h. vom Typ **`byte`**, **`short`**, **`int`**, **`long`**, **`float`**, **`double`**
  - oder vom Typ **`char`** (vorzeichenlose 2-Byte-Ganzzahl).
- Ergebnistyp (sowie Art der Arithmetik) wird nach folgenden Regeln (ausgewertet von 1 bis 4) bestimmt:
  1. ein double-Operand:
     - ✓ anderer Operand wird ggf. nach **`double`** konvertiert
     - ✓ Gleitkomma-Arithmetik
     - ✓ Ergebnis **`double`**
  2. ein float-Operand:
     - ✓ anderer Operand wird ggf. nach **`float`** konvertiert
     - ✓ Gleitkomma-Arithmetik
     - ✓ Ergebnis **`float`**

Arithmetische Operatoren

3. ein long-Operand:
   - ✓ anderer Operand wird ggf. nach **`long`** konvertiert
   - ✓ Ganzzahl-Arithmetik
   - ✓ Ergebnis **`long`**
4. sonst:
   - ✓ beide Operand werden ggf. nach **`int`** konvertiert
   - ✓ Ganzzahl-Arithmetik
   - ✓ Ergebnis **`int`**

Operanden werden auf höchsten vorkommenden Typ angepasst, zumindest aber nach int konvertiert.

Arithmetische Operatoren

Operator	*Erläuterung*
`x + y`	Addition
`x - y`	Subtraktion
`x * y`	Multiplikation
`x / y`	Ganzzahl- oder Gleitkomma-Division, *typabhängig*: • Ganzzahl: 7 / 4 = 1 • Gleitkomma: 7.0 / 4.0 = 1.75
`x % y`	Modulo, d.h. Rest der Ganzzahldivision `x / y` • Beispiel: 7 % 4 = 3
`-x`	Negation, einstelliges Minus

Logische Operatoren

- Operanden müssen vom Typ **`boolean`** sein
- Ergebnis ist wieder vom Typ **`boolean`**

Operator	***Erläuterung***
`a & b`	logisches Und, Konjunktion, 'Λ'
`a && b`	logisches Und, aber bedingte Auswertung
`a | b`	logisches Oder, Disjunktion, 'V'
`a || b`	logisches Oder, aber bedingte Auswertung
`!`	Negation

Wahrheitstafeln

- Die logischen Operatoren `&`, `|` und `!` sind in der üblichen Weise definiert:

a	b	a & b	a | b	!a
false	false	false	false	true
false	true	false	true	true
true	false	false	true	false
true	true	true	true	false

- *Merke*: Im Gegensatz zum häufigen natürlichsprachigen Gebrauch ist das logische Oder ein ***inklusives Oder***, d.h. `true | true` → `true`.

Bedingte Auswertung

- Die Operatoren **`&&`** und **`||`** entsprechen den Operatoren **`&`** bzw. **`|`**, außer dass der 2. Operand nicht ausgewertet wird wenn das Ergebnis nach der Auswertung des 1. Operanden schon feststeht:
  - **`a && b`**: **`a`** = **`false`** → Ergebnis kann nur **`false`** sein!
  - **`a || b`**: **`a`** = **`true`** → Ergebnis kann nur **`true`** sein!
- Bedingte Auswertung (*lazy evaluation*)
  - spart Auswertungszeit
  - vermeidet Auswertung von Programmteilen (→wird später erklärt)

Vergleichsoperatoren

- Die Operatoren <, <=, >, >= arbeiten auf numerischen Werten sowie dem Typ **`char`** (vorzeichenlose 2-Byte-Ganzzahl).
- Für die Typanpassung gelten dieselben Regeln wie für arithmetische Operatoren

Operator	***Erläuterung***
`a < b`	ist a kleiner als b?
`a <= b`	ist a kleiner oder gleich b?
`a > b`	ist a größer als b?
`a >= b`	ist a größer oder gleich b?

Vergleichsoperatoren

- Die Operatoren `==` und `!=` arbeiten auf
  - numerischen Werten sowie dem Typ **`char`**
  - *→Objektreferenzen*
  - **`boolean`**

Operator	***Erläuterung***
`a == b`	ist a gleich b?
`a != b`	ist a ungleich b?

Inkrement- und Dekrementoperatoren

- Die Operatoren arbeiten auf numerischen Typen sowie **`char`**, und liefern ein Ergebnis vom selben Typ.

Operator	*Erläuterung*
`x++`	*Postinkrement*; gibt den Wert x zurück und erhöht ihn danach um 1.
`x--`	*Postdekrement*; gibt den Wert x zurück und verringert ihn danach um 1.
`++x`	*Präinkrement*; erhöht den Wert x um 1 und gibt ihn danach zurück.
`--x`	*Prädekrement*; verringert den Wert x um 1 und gibt ihn danach zurück.

Zuweisungsoperatoren

- Neben der einfachen Zuweisung '**`x = y`**' gibt es eine Reihe abkürzender Zuweisungen der Form '**`x <op>= y`**', die denselben Effekt haben wie '**`x = x <op> y`**'.
- Der einfache Zuweisungsoperator '=' arbeitet auf numerischen Werten, **`char`** und →Objektreferenzen.
- Alle andere Zuweisungsoperatoren arbeiten nur auf numerischen Werten und **`char`**.

Zuweisungsoperatoren

Operator	*Bedeutung / äquivalent zu*
`x = y`	Weist der Variablen `x` den Wert `y` zu.
`x += y`	`x = x + y`
`x -= y`	`x = x - y`
`x *= y`	`x = x * y`
`x /= y`	`x = x / y`
`x %= y`	`x = x % y`

Ternäroperator

- In Java gibt es nur einen ternären, d.h. dreistelligen Operator, den sogenannten *Ternäroperator*.
- Er hat die Form '**`<Bedingung>? x : y`**'
- Bedeutung: Wenn *Bedingung* erfüllt, liefere x als Ergebnis, sonst liefere y als Ergebnis.
- *Beispiel*:

```
int i = (j < 10)? 47: 11;
```

Auswertungsreihenfolge

- Ein Ausdruck kann mehrere Operatoren hintereinander enhalten
  → *in welcher Reihenfolge wird ausgewertet?*
- Dafür gibt es 2 Arten von Regeln:
  - ***Prioritäten***: geben an, welche Operatoren Vorrang vor anderen haben
    → *"Punkt vor Strich"*
  - ***Assoziativität***: bestimmt die Auswertungsreihenfolge für Operatoren gleicher Priorität.
- *Merke*: Die gewünschte Auswertungsreihenfolge kann immer per Hand durch geeignete Klammern erreicht werden!

Prioritäten

Bezeichnung	*Operator*	*Priorität*
→Komponentenzugriff bei Klassen	`.`	15
→Komponentenzugriff bei Feldern	`[]`	15
→Methodenaufruf	`()`	15
unäre Operatoren	`++, --, -,!`	14
explizite Typkonvertierung	`(type)`	13
multiplikative Operatoren	`*, /, %`	12
additive Operatoren	`+, -`	11
Vergleichsoperatoren	`<, >, <=, >=`	9
Gleichheit/Ungleichheit	`==, !=`	8
logisches Und	`&&`	4
logisches Oder	`||`	3
Ternäroperator	`?:`	2
Zuweisungsoperatoren	`=, +=, -=, *=, …`	1

Assoziativität

- Regelt Auswertungsreihenfolge für Operatoren *gleicher Priorität*
- Alle Zuweisungsoperatoren werden von rechts nach links ausgewertet.
- Das ermöglicht Ausdrücke der Form: '`a = b = c`'; das bedeutet:

  'b ← c, dann a ← b'
- Alle anderen Operatoren werden von links nach rechts ausgewertet.
- *Beispiel*: `a + b - c` entspricht `(a + b) -c` ,

  und nicht `a + (b -c)`

Kommentare

Kommentare

- dienen dazu, Programme zu erklären
- werden vom Compiler ignoriert, d.h. *erzeugen keinen Bytecode*!
- helfen nicht nur anderen, das Programm zu verstehen, sondern auch dem Autoren selbst.
- sollten möglichst *häufig* und *sinnvoll* verwendet werden.

In Java gibt es Kommentarzeichen

- für einzeilige Kommentare (Kommentar endet beim Zeilenumbruch)

```
// Kommentarzeile 1
// Kommentarzeile 2
```

- für mehrzeilige Kommentare (explizites Kommentarende-Zeichen)

```
/* Kommentarzeile 1
   Kommentarzeile 2
   kommentarzeile 3 */
```

Bildschirmausgabe

- **`System.out.println(<Ausgabe>);`** schreibt **`<Ausgabe>`** auf den Bildschirm, gefolgt von einem Zeilenumbruch (*'print line'*).
- **`System.out.print(<Ausgabe>);`** schreibt **`<Ausgabe>`** auf den Bildschirm, ***ohne*** Zeilenumbruch (*'print'*).
- **`<Ausgabe>`** kann eine Zeichenkette in Anführungsstrichen sein, z.B. **`"Hallo Welt"`**.
- Die folgenden Anweisungen

```
System.out.println("Hallo Welt");
System.out.print("Mein erstes ");
System.out.prinln("Java-Programm");
```

erzeugen die Bildschirmausgabe:

```
Hallo Welt
Mein erstes Java-Programm
```

Bildschirmausgabe

- Zeichenketten können mit Hilfe des '+'-Operators zusammengesetzt (*konkateniert*) werden. Damit können z.B. zu lange Zeilen im Programmtext vermieden werden. Die folgende Anweisung

```
System.out.println("Hallo Welt. Mein erstes " +
                   "Java-Programm.");
```

erzeugt die Ausgabe:

```
Hallo Welt. Mein erstes Java-Programm.
```

- *Jede Anweisung (nicht nur Bildschirmausgabe) muss mit einem Semikolon abgeschlossen sein!*

Bildschirmausgabe

- Neben Zeichenketten können auch Werte von Variablen und Ausdrücken, sowie Literale ausgegeben werden. Die Anweisungen

```
int i = 3;
System.out.println(i + 2);
```

erzeugen die Ausgabe:

```
5
```

- Zeichenketten und Werte können konkateniert werden. Die Anweisungen

```
int i = 3;
System.out.println("i: " + i);
```

erzeugen die Ausgabe:

```
i: 3
```

Tastatureingabe

Folgender Code fordert Benutzer zu Tastatureingabe auf, liest eingegebene Zeichenkette in String ein und gibt diesen auf Bildschirm aus:

```
java.util.Scanner scanner = new java.util.Scanner(System.in);
System.out.print("Text eingeben: ");
String text = scanner.next();
System.out.println("eingegebener Text: " + text);
```

Beispiel:

1
```
Text eingeben:
```

2
```
Text eingeben:hello
```

3
```
Text eingeben:hello
eingegebener Text: hello
```

Tastatureingabe

Es können auch Ganzzahlen, Gleitkommazahlen und Boolsche Werte eingelesen werden:

```
java.util.Scanner scanner = new java.util.Scanner(System.in);

System.out.print("Ganzzahl eingeben: ");
int zahl = scanner.nextInt();
System.out.println("eingegebene Ganzzahl: " + zahl);

System.out.print("Boolschen Wert eingeben: ");
boolean boolWert = scanner.nextBoolean();
System.out.println("eingegebener Boolscher Wert: " + boolWert);
```