

Programmieren I

Operatoren und Ausdrücke



Heusch 6.2
Ratz 4.4.2

Institut für Automation und angewandte Informatik

```
final List<String> allResults = new ArrayList<String>();  
final Map<String, Integer> typeWordResultCount = new HashMap<String, Integer>();  
final Map<String, Integer> typePoints = new HashMap<String, Integer>();  
evaluation.put(type, typePoints);  
  
for (final Sheet sheet : this.sheets) {  
    final String sheetResult = sheet.getPlayerInput(type);  
    if (sheetResult.startsWith(start) && this.isValidWord(sheetResult, type)) {  
        validWordCountForType++;  
        allResults.add(sheetResult);  
    }  
}
```

Ausdrücke, Arten von Operatoren

- Ein **Ausdruck** ist eine *Folge von Operatoren und Operanden*, welche z.B. die Berechnung eines Wertes festlegt.
- Wenn ein Ausdruck *ausgewertet* wird, liefert er als Ergebnis einen Wert (*Resultat*).
- Mögliche *Operanden* sind z.B. Variablen und Konstanten.
- **Arten von Operatoren:**
 - Arithmetische Operatoren
 - Inkrement- und Dekrement-Operator
 - Vergleichsoperatoren
 - Logische Operatoren
 - Bitlogische Operatoren
 - Schiebeoperatoren (nicht behandelt!)
 - Zuweisungsoperatoren
 - Weitere Operatoren

Kategorisierung von Operatoren (1)

■ ...nach Anzahl der Operanden

Anzahl der Operanden	Bezeichnung	Beispiel
1	Unärer Operator	<code>i++</code>
2	Binärer Operator	<code>a + b</code>
3	Ternärer Operator	<code>(a==0) ? "Null" : "Nicht Null"</code>

Kategorisierung von Operatoren (2)

- ...nach Priorität (Vorrang)
 - höchste Priorität: 1
 - niedrigste Priorität: 13

- ...nach Assoziativität (Reihenfolge der Auswertung bei gleicher Priorität)
 - rechts (d.h. von rechts nach links)
 - links (d.h. von links nach rechts)

Reihenfolge der Ausführung von Operationen

- Treten *in einem Ausdruck mehrere Operatoren* auf, gelten für die Reihenfolge der Ausführung der Operationen standardmäßig folgende Regeln:
 - Bei Operatoren mit *unterschiedlichem Rang* werden Operationen mit *höherem Rang vor* Operationen mit *niedерem Rang* durchgeführt.
Beispiel: $a + b * c$ wird ausgewertet wie $a + (b * c)$
 $a + b <= c$ wird ausgewertet wie $(a + b) <= c$
 - Bei Operatoren mit *gleichem Rang* regelt die *Assoziativität* die Reihenfolge der Ausführung („von links nach rechts“ oder „von rechts nach links“).
Beispiel: $a + b - c$ wird ausgewertet wie $(a + b) - c$
 $a = b = c$ wird ausgewertet wie $a = (b = c)$
- Ausdrücke innerhalb von Klammern () werden vorrangig ausgewertet. D.h. durch *Klammerung* kann eine nicht-standardmäßige Reihenfolge der Auswertung erreicht werden.

Typisierung von Operatoren

Typ	Bezeichnung	Java-Datentypen
Integraler Typ ("ganze Zahlen")	I	byte, short, int, long, (char)
Nummerischer Typ	N	I und float, double
Logischer Typ	L	boolean
Primitiver Typ	P	N und L
Referenz	R	Klassen und Arrays (inkl. String)
Alle Typen	A	

Arithmetische Operatoren (1)

Operator	Name	Priorität	Typisierung	Operanden	Assoziativität
+	Unäres Plus Vorzeichen	1	N	1	rechts
-	Unäres Minus Vorz.-Umkehr	1	N	1	rechts
+	Addition	3	N, N	2	links
-	Subtraktion	3	N, N	2	links
*	Multiplikation	2	N, N	2	links
/	Division	2	N, N	2	links
%	Rest (Modulo)	2	N, N	2	links

Resultattyp: N (numerischer Typ)

Typisierung: Typ(en) der Operanden

Arithmetische Operatoren (2)

- Wenn *beide Operanden ganzzahlig* (d.h. von einem Ganzzahl-Datentyp) sind, ist auch das *Resultat ganzzahlig*; ansonsten ist das Resultat eine Gleitpunktzahl.
- Der Ausdruck ***a / b*** liefert für zwei *ganzzahlige Operanden a* und *b* nur den *Ganzzahlanteil* des Ergebnisses.
- Der Ausdruck ***a % b*** liefert den *Divisionsrest* der ganzzahligen Division.

■ Beispiele:

Sei

```
int a = 1;
```

Ausdruck	Resultat
$a + 3$	4
$5.5 - 3.5$	2.0
$5 - 3.0$	2.0
$5 - a$	4
$3 * 1.5$	4.5
$a / 2$	0
$1.0 / 2.0$	0.5
$5 \% 2$	1

Inkrement- und Dekrement-Operatoren (1)

Operator	Name	Priorität	Typisierung	Operanden	Assoziativität
++	Inkrement	1	N	1	rechts
--	Dekrement	1	N	1	rechts

Der Operator kann vor oder nach dem Operand stehen (Präfix- bzw. Postfixform).

Beispiele:

```
i++
++i
```

Resultattyp: N (numerisch)

Inkrement- und Dekrement-Operatoren (2)

- Wenn der *Ausdruck allein* steht (eigene Anweisung), ist die Wirkung der Postfix- und Präfixform *gleich*.
- Wenn der resultierende Wert in einem Ausdruck *unmittelbar weiterverwendet* wird, ist die Wirkung *unterschiedlich*:
 - Falls ++ bzw. -- **vor** der Variablen steht: Die Variable wird *zuerst inkrementiert bzw. dekrementiert* und dann bei der Ermittlung des Werts des Ausdrucks verwendet.
 - Falls ++ bzw. -- **nach** der Variablen steht: Der *alte Wert* der Variablen wird in der Ermittlung des Werts des Ausdrucks verwendet.
Nach dieser Wertermittlung des Ausdrucks wird die Variable *inkrementiert bzw. dekrementiert*.
 - Zwei Beispiele:

```
int n = 0;  
int x;  
x = ++n;
```

Werte danach:

n: 1, x: 1

```
int n = 0;  
int x;  
x = n++;
```

x: 0, n: 1

Vergleichsoperatoren (1)

Operator	Name	Priorität	Typisierung	Operanden	Assoziativität
==	Gleichheit	6	A, A	2	links
!=	Ungleichheit	6	A, A	2	links
<	Kleiner	5	N, N	2	links
<=	Kleiner gleich	5	N, N	2	links
>	Größer	5	N, N	2	links
>=	Größer gleich	5	N, N	2	links

Resultattyp: $L(\text{boolean})$

Vergleichsoperatoren (2)

■ Beispiel:

Mit der Deklaration

```
int a = 2, b = 1;
```

ergeben die folgenden Ausdrücke die Resultate:

Ausdruck	Resultat
<code>a == 2</code>	<code>true</code>
<code>b != 1</code>	<code>false</code>
<code>a < b</code>	<code>false</code>
<code>3.0 >= 3.0</code>	<code>true</code>

Logische Operatoren (1)

Operator	Name	Priorität	Typisierung	Operanden	Assoziativität
!	Nicht (Negation)	1	\mathbb{L}	1	rechts
&	Und	7	\mathbb{L}, \mathbb{L}	2	links
&&	Und verkürzt	10	\mathbb{L}, \mathbb{L}	2	links
	Oder	9	\mathbb{L}, \mathbb{L}	2	links
	Oder verkürzt	11	\mathbb{L}, \mathbb{L}	2	links
^	Exklusives Oder	8	\mathbb{L}, \mathbb{L}	2	links

- Bei den "verkürzten" Operatoren && und || wird der 2. Operand nur dann ausgewertet, wenn das Ergebnis der Operation nicht schon nach dem 1. Operanden fest steht.
- Resultattyp: \mathbb{L} (boolean)

Logische Operatoren (2)

- Ergebnisse der logischen Operationen (`boolean a, b;`):

a	b	! a	a & b	a b	a ^ b
false	false	true	false	false	false
false	true	true	false	true	true
true	false	false	false	true	true
true	true	false	true	true	false

- Beispiele:

Sei vereinbart: `boolean x = true, y = false;`

Ausdruck	Resultat
<code>x && true</code>	true
<code>y x</code>	true
<code>!x</code>	false

Bitlogische Operatoren (1)

Operator	Name	Priorität	Typisierung	Operanden	Assoziativität
\sim	Komplement	1	\mathbb{I}	1	rechts
$\&$	Bitw. Und	7	\mathbb{I}, \mathbb{I}	2	links
$ $	Bitw. Oder	9	\mathbb{I}, \mathbb{I}	2	links
\wedge	Bitw. Exkl. Oder	8	\mathbb{I}, \mathbb{I}	2	links

Resultattyp: \mathbb{I} (Integral)

Bitlogische Operatoren (2)

■ Bit-Operationen (a, b: jeweils ein Bit)

a	b	! a	a & b	a b	a ^ b
0	0	1	0	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	1	1	0

■ Beispiel:

```
byte a = 5;           // 00000101
byte b = 66;          // 01000010
byte c = (byte)(a | b); // 01000111
```


Zuweisungsoperatoren (1)

- In Java ist auch eine *Zuweisung*, z.B. $i = i + 1$, ein Ausdruck ('Zuweisungsausdruck').
- Der wichtigste *Zuweisungsoperator* ist der '='-Operator.
- Auf der linken Seite des Operators steht ein Ausdruck, welcher ein *modifizierbares Datenobjekt* bestimmt.
- Der Ausdruck auf der rechten Seite wird *ausgewertet* und das Resultat dem Datenobjekt auf der linken Seite als Wert *zugewiesen*.
- Bei *unterschiedlichen Datentypen*
 - erfolgt eine automatische Typumwandlung des Wertes des Ausdrucks rechts in den Typ des Datenobjekts links, falls er Typ-kompatibel ist,
 - sonst Fehler (siehe "Typkompatibilität und Typkonversion").
- Zuweisungsausdrücke liefern (wie alle Ausdrücke) einen *Wert*. Der Wert eines Zuweisungsausdrucks ist der Wert des Datenobjekts links nach erfolgter Zuweisung.
- Zuweisungen können prinzipiell *überall auftreten* wo ein Ausdruck erlaubt ist.

Zuweisungsoperatoren (2)

Operator	Funktion
=	Zuweisung
+=	Zuweisung mit Addition
-=	Zuweisung mit Subtraktion
*=	Zuweisung mit Multiplikation
/=	Zuweisung mit Division
%=	Zuweisung mit Modulo
&=	Zuweisung mit UND (bitlogisch / logisch)
=	Zuweisung mit ODER (bitlogisch / logisch)
^=	Zuweisung mit EXKLUSIV-ODER (bitlogisch / logisch)

Zuweisungsoperatoren (3)

■ **Beispiele:** Sei vereinbart: `int i, k;`

■ `(i = 2) + (k = 3)` Wert des Ausdrucks: 5

■ `i = k = 1` Entspricht `i = (k = 1)`.
Wert des Ausdrucks: 1

■ `i += 5` Der Wert von `i` wird um 5 erhöht
Wert des Ausdrucks: erhöhter `i`-Wert

Weiterer Operator: Bedingungsoperator (1)

Operator	Name	Priorität	Typisierung	Operanden	Assoziativität
? :	Bedingungsoperator	12	$\mathbb{L}, \mathbb{A}, \mathbb{A}$	3	rechts

- Der Bedingungsoperator wird auch „Fragezeichenoperator“ genannt.
- Resultattyp: \mathbb{A} (Alle Typen)



Heusch 6.3.7
Ratz 4.1.7

Bedingungsoperator (2)

- Der Bedingungsoperator ist der einzige dreistellige Operator in Java:

- Syntax:

```
conditional_expression ? expression1 : expression2
```

- Der erste Operand (*conditional_expression*, *ce*) erwartet einen logischen Ausdruck, in dessen Abhängigkeit entweder der zweite (falls *ce==true*) oder dritte Operand (falls *ce==false*) zurückgeliefert wird.

- Beispiel:

```
boolean b = true;  
System.out.println(b ? 1 : 2); // Ausgabe: "1"
```

Weiteres Beispiel für Bedingungsoperator

- Ermittlung des Maximums von a und b:

```
int a = 3, b = 5, max;  
max = (a > b) ? a : b;
```

- Analoge Schreibweise mit `if` und `else`:

```
if ( a > b )  
    max = a;  
else  
    max = b;
```

Zusammenfassung: Priorität von Operatoren

Operator	Priorität	Assoziativität
++, --, !, ~, (Typ), +, - (<i>unäres Plus und Minus</i>)	1	rechts
*, /, %	2	links
+, -	3	links
<<, >>, >>>	4	
<, <=, >, >=, instanceof	5	links
==, !=	6	links
& (log./bitw. Und, vollst. Auswertung)	7	links
^ (log./bitw. Exklusiv-Oder, vollst. Ausw.)	8	links
(log./bitw. Oder, vollst. Auswertung)	9	links
&& (log. Und, Kurzauswertung)	10	links
(log. Oder, Kurzauswertung)	11	links
?: (Bedingungsoperator)	12	rechts
=, *=, /=, %=, +=, -=, <<=, >>=, >>>=, &=, ^=, =	13	rechts

Ausgabe

- In der Klasse `java.lang.System` realisiert.
- Drei Streams
 - Standardausgabe: `System.out`
 - Standardfehlerausgabe: `System.err`
 - Standardeingabe: `System.in`
- Beispiele:


```
System.out.println("Ich bin eine Zeile");  
System.out.print("Ich bin");  
System.out.println(" eine Zeile");  
System.err.println("Aaaargghh ich steerb...");  
int zahl = 4711;  
System.out.println("Die Zahl ist " + zahl);
```

Stringverkettung



Daten formatiert ausgeben

Seit Java 1.5 gibt es auch die Möglichkeit Daten formatiert auszugeben (ähnlich wie in C).

 Ab Java 1.5

Beispiele für die Ausgabe mit `printf`:

```
System.out.printf( "Hello %s. Missed call from %s.\n", "Ulli", "Tanja" );
int i = 123;
System.out.printf( "%d| %d|\n" , i, -i);           // |123| |-123|
System.out.printf( "%5d| %5d|\n" , i, -i);         // | 123| | -123|
System.out.printf( "%-5d| %-5d|\n" , i, -i);       // |123  | | -123  |

double d = 1234.5678;
System.out.printf( "%10f| %10f|\n" , d, -d);       // |1234,567800| |-1234,567800|
System.out.printf( "%10.2f| %10.2f|\n" , d, -d);   // | 1234,57| | -1234,57|
System.out.printf( "%010.2f| %010.2f|\n", d, -d); // |0001234,57| |-001234,57|
```

Formatstring
(1. Argument)

Formatspezifizierer
(beginnen mit %)

Übersicht zur Formatierung mit `printf`

Umwandlungs- zeichen	Argument- Kategorie	Ergebnis-String
<code>%d</code>	Ganzzahl	Dezimale Ganzzahl mit Vorzeichen
<code>%o</code>	Ganzzahl	Oktale Ganzzahl ohne Vorzeichen
<code>%x, %X</code>	Ganzzahl	Hexadezimalschreibweise ohne Vorzeichen in Klein- bzw. Großschreibung
<code>%f</code>	Gleitkommazahl	Gleitkommazahl in Standard-Notation
<code>%e, %E</code>	Gleitkommazahl	Gleitkommazahl in Exponentialzahldarstellung (<code>'e'</code> in Groß- bzw. Kleinschreibung)
<code>%g, %G</code>	Gleitkommazahl	Wie <code>%f</code> oder <code>%e</code> , jedoch abhängig vom Wert (Exponen- tialzahldarstellung wird benutzt, falls der Exponent größer als die Genauigkeit oder kleiner als 4 ist)
<code>%b</code>	boolean	<code>false</code> oder <code>true</code>
<code>%c</code>	Zeichen	Unicode-Zeichen
<code>%s</code>	String	String
<code>%%</code>		Prozentzeichen
<code>%n</code>		Neue Zeile
<code>%t, %T</code>	date/time	Datums-/Zeitangabe, s. Klasse <code>java.util.Formatter</code>

Eingabe von Daten: Die Klasse „Scanner“

- Für den Zweck der Eingabe von primitiven Datentypen und Strings gibt es die Klasse `java.util.Scanner`.
- Beispiel:

```
java.util.Scanner scan = new java.util.Scanner(System.in);  
int i = scan.nextInt();
```

- Achtung: Die Eingaben erfolgen „lokalisiert“, d.h. wenn die Konfiguration der JVM auf deutsch eingestellt ist, gilt für die korrekte Eingabe von Gleitkommazahlen das Komma als Trennzeichen!

Beispiele für die Eingabe (1)

```
public class Input {  
  
    public static void main(String[] args) {  
  
        java.util.Scanner scan = new java.util.Scanner(System.in);  
  
        System.out.print("Please enter an integer number: ");  
        int i = scan.nextInt();  
  
        System.out.print("Please enter a floating point number: ");  
        double d = scan.nextDouble();  
  
        System.out.print("Please enter a string: ");  
        String word = scan.next(); // nächster Token  
  
        System.out.println("INT:      " + i);  
        System.out.println("DOUBLE:  " + d);  
        System.out.println("STRING:  " + word);  
    }  
  
}
```

Beispiele für die Eingabe (2)

```
public class InputWrapper {  
  
    public static void main(String[] args) {  
  
        // Nun erst String einlesen und diesen dann konvertieren  
  
        java.util.Scanner scan = new java.util.Scanner(System.in);  
        String input;  
        System.out.print("Please enter an integer number: ");  
        input = scan.next();           // Einlesen als String  
        int i = Integer.parseInt(input); // Konvertierung in int  
  
        System.out.print("Please enter a floating point number: ");  
        input = scan.next();           // Einlesen als String  
        double d = Double.parseDouble(input); // Konvert. in double  
  
        System.out.println("INT:      " + i);  
        System.out.println("DOUBLE:   " + d);  
    }  
}
```

Bei der Konvertierung gibt es (im Gegensatz zum Scanner)
keine Lokalisierung → Dezimalpunkt