Schlüsselwörter, Literale

Schlüsselwörter (Keywords) sind Wörter, die nicht mehr verwendet werden dürfen, weil sie schon eine besondere Bedeutung in der Sprache haben. In Java sind dies die Wörter in der folgenden Tabelle.

`abstract`	`class`	`extends`	`implements`	`new`	`static`	`throws`
`boolean`	`const`	`final`	`import`	`package`	`strictfp`	`transient`
`break`	`continue`	`finally`	`instanceof`	`private`	`super`	`try`
`byte`	`default`	`float`	`int`	`protected`	`switch`	`void`
`case`	`do`	`for`	`interface`	`public`	`synchronized`	`volatile`
`catch`	`double`	`goto`	`long`	`return`	`this`	`widefp`
`char`	`else`	`if`	`native`	`short`	`throw`	`while`

Literale sind das, was man hinschreibt, wenn man Werte direkt eingibt, also Strings in Anführungszeichen, Zahlen, die Wahrheitswerte `true` und `false`, sowie das Symbol für die Nichtzuordnung von Objekten `null`.

Ausdrücke (Expressions)

Ausdrücke entstehen durch Aneinanderreihung von Operatoren mit ihren Operanden. Alle Operatoren mit Prioritäten (P) sind in folgender Tabelle zusammengefasst.

Die erste Spalte gibt neben der Priorität auch die Assoziativität der Operatoren an. Diese ist wichtig, wenn ein Operand zwischen mehreren Operatoren gleicher Priorität steht.

← steht für „linksassoziativ“ und bedeutet, dass der Operand stärker links gebunden ist als rechts. So wird `a+b-c` genau so ausgewertet wie `(a+b)-c`, weil `b` zwischen zwei gleich starken Operatoren steht, die aber beide ← sind.

`mannschaft.name[17]` hingegen wird eben nicht abwegig als `(mannschaft.name)[17]` interpretiert (17. Mannschaftsname), sondern als `mannschaft.(name[17])`, wie man es vernünftigerweise erwartet. Es wäre sehr unpraktisch, wenn man hier Klammern setzen müsste. Da die beiden Operatoren '`.`' und '`[`' die Assoziativität → haben, muss man es nicht.

Bei Operatoren verschiedener Priorität treten diese Probleme natürlich nicht auf. Der stärkere siegt.

P	Bedeutung	Beispiel	Operanden
→15	Klammern ()	`3*(2+4)`	beliebig
	Arrayzugriff []	`name[3*i]`	`int`
	Punktoperator	`name.length`	beliebig
	Postfix-Operatoren	`i++; i--`	arithmetisch
→14	Prefix-Operatoren	`++i; --i`	arithmetisch
	Vorzeichen	`-5`	arithmetisch
	bitweise Negation ~	`~i`	`int`
	logische Negation !	`leer=!isFull(pool)`	`boolean`
→13	Typkonvertierung	`(int)3.14`	beliebig
	Instantiierung	`new Double(1.2)`	`Object`
←12	Multiplikation	`2*3.7`	arithmetisch
	Division	`2/3.7`	arithmetisch
	Divisonsrest	`zwei=12%5`	arithmetisch
←11	Additon	`2+3.7`	arithmetisch
	Subtraktion	`2-3.7`	arithmetisch
	Verkettung	`name="Ha"+"ns"`	`String`
←10	Shift links	`zwanzig=5<<2`	`int`
	Shift rechts (Vorz. rein)	`minuszwei=-5>>2`	`int`
	Shift rechts (0 rein)	`eins=-1>>>31`	`int`
←9	kleiner, kleinergleich?	`nein=3<3; ja=3<=3`	arithmetisch
	größer, größergleich?	`ja=4>3; nein=3>=4`	arithmetisch
	Typvergleich	`ja=bmw instanceof Auto`	`Object`
←8	identische Werte?	`ja=2.3==2.3`	primitiv
	verschiedene Werte?	`nein=2.3!=2.3`	primitiv
	dasselbe Objekt?	`ja=bmw==bmw`	`Object`
	verschiedenes Objekt?	`nein=bmw!=bmw`	`Object`
←7	bitweises AND	`fünf=0xd5 & 0x0f`	`int`
	logisches AND	`ja=1<2 & 3==3`	`boolean`
←6	bitweises XOR	`fünf=0x08 ^ 0x0d`	`int`
	logisches XOR	`ja=1<2 ^ 1==2`	`boolean`
←5	bitweises OR	`sieben=6|3`	`int`
	logisches OR	`ja=1<2 | 1==2`	`boolean`
←4	abkürzendes AND	`nein=2<1 && nixmehr()`	`boolean`
←3	abkürzendes OR	`ja=1<2 || nixmehr()`	`boolean`
→2	bedingte Zuweisung;	`zwei=5<6?2:3; drei=5==6?2:3`	`boolean`, beliebig
→1	(Operation und	`*=  /=  +=  -=`	Variable, arithmetisch
	anschließende	`%=  <<=  >>= >>>=`	Variable, `int`
	effiziente)	`&=  ^=  |=`	Variable, `int`/`boolean`
	Zuweisung	`x=2+a`	beliebig