Java Operators

Precedence	Operators	Description	Usage Details or Example	Associativity
1	`[]`	Array index	Refers to an item within an array – either to obtain its value or to use it as an assignment statement target. For example, `myData[n]` refers to element `n` of array `myData`.	left-to-right
	`()`	Expression grouping	Forces an explicit order of execution. For example, in the expression `x * (y + z)`, the parentheses force evaluation of `y + z` before multiplication by `x`.	
	`.`	Class or instance member access	Refers to a field or method (specified on the right side of the `.` operator) of an object instance or class (specified on the left side). For example, `myObject.x` refers to field `x` of object reference `myObject`, while `myObject.aMethod()` refers to method `aMethod` of object reference `myObject`.	
2	`++`, `--`	Postfix increment, postfix decrement	Increases or decreases the operand's value by 1, taking its previous value as the expression value. For example, `n++` increases the value of `n` by `1`, *after* its value is used in the enclosing expression.	(none)
3	`+`, `-`	Unary plus (arithmetic identity), unary minus (arithmetic negation)	Maintains or reverses the sign of a numeric value. `-x` has the same value as `(0 - x)`, while `+x` has the same value as `(0 + x)`.	right-to-left
	`++`, `--`	Prefix increment, prefix decrement	Increases or decreases the operand's value by 1, taking its updated value as the expression value. For example, `--n` decreases the value of `n` by `1`, *before* its value is used in an enclosing expression.	
	`!`	Boolean NOT (negation)	Inverts a Boolean value. For example, if `conditionA` is a `boolean` with the value `true`, the the value of `!conditionA` is `false`. If If `conditionB` is `false`, then `!conditionB` is `true`.	
	`~`	Bitwise NOT (one's complement)	Inverts the bits (binary digits) of an integral value. For example, if `b` is a `byte` value with a base-2 representation of `10101010`, then the value of `~b`, expressed in base-2 notation, is `01010101`.	
4	`()`	Cast to type	Instructs the compiler to treat a value of one type as another. For example, `(byte) 812` casts the `int` value `812` to a `byte`, by ignoring all but the right-most byte of the value, resulting in a value of `44`.	right-to-left
	`new`	Object creation	Creates an instance of the specified class or array type. For example, `new StringBuilder()` creates an instance of the `java.lang.StringBuilder` class. (The no-parameter constructor of `StringBuilder` is then invoked to perform initialization.)	
5	`*`, `/`, `%`	Multiplication, division, modulo operation (remainder)	Evaluates the specified multiplication or division operation. For example, `5 * 3` gives a value of `15`, while `5 / 3` gives a value of `1` (integer division gives the integer quotient, rounded towards zero), and `5 % 3` gives a value of `2` (the remainder).	left-to-right
6	`+`, `-`	Addition, subtraction	Adds or subtracts pairs of values. If `x` and `y` are numeric values, `x + y` gives the arithmetic sum of `x` and `y` (after any necessary widening), and `x – y` gives the arithmetic difference.	left-to-right
	`+`	String concatenation	If either of `s` and `t` are `String` instances, then `s + t` converts the other to a `String` (if necessary), and gives the concatenation of the two. For example `"ab" + 12` evaluates to `"ab12"`.	
7	`<<`, `>>`, `>>>`	Left shift, signed right shift, unsigned right shift	Shifts the bits of an integral value to the right or left. For example, given an `int x` with a value of `-10` (expressed as a base-2 literal as `0b11111111_11111111_11111111_11110110`), then `x >> 1` evaluates to `-5` (or `0b11111111_11111111_11111111_11111011`; note the preservation of the sign bit), while `x >>> 1` evaluates to `2_147_483_643` (`0b01111111_11111111_11111111_11111011`), and `x << 1` is `-20` (`0b11111111 11111111 11111111 11101100`).	left-to-right
8	`<`, `<=`, `>`, `>=`	Less than, less than or equal to, greater than, greater than or equal to	Given numeric values `x` and `y`, the expression `x < y` evaluates to `true` if `x` is less then `y`, and `false` otherwise. `x <= y` evaluates to `true` if `x` is less than or equal to `y`, and `false` otherwise. Conversely, the value of `x > y` is `true` if (and only if) `y < x`, while `x >= y` is `true` if (and only if) `y <= x`.	(none)
	`instanceof`	Type (class or interface) instance test	Tests an object reference against a specified class or instance. For example, if `objectA` is an object reference, and `ClassB` is a class, then `objectA instanceof ClassB` evaluates to `true` if `objectA` is an instance of `ClassB` or a subclass (direct or indirect) of `ClassB`, and `false` otherwise.	
9	`==`, `!=`	Value equality, inequality	Given primitive values `x` and `y`, the expression `x == y` has the value `true` if `x` and `y` have the exact same values, and `false` otherwise. Inversely, `x != y` has the value `true` if `x` and `y` do not have the exact same values, and `false` otherwise. (Beware of equality comparisons between floating-point values!)	left-to-right
	`==`, `!=`	Reference (identity) equality, inequality	Given object references `s` and `t`, the expression `s == t` has the value `true` if both `s` and `t` refer to the same object instance, and `false` otherwise. Inversely, `s != t` has the value `true` if `s` and `t` do not refer to the same object instance, and `false` otherwise. Note that 2 object instances with the same contents will usually still have distinct identities.	
10	`&`	Bitwise AND	Given the integral values `x` and `y`, the expression `x & y` gives an integer result in which all bits that are set in both `x` and `y` are set in the result, and all other bits in the results are not set. For example, the value of `10 & 3` is `2`; that is, `0x00001010 & 0x00000011` gives the value `0x00000010`.	left-to-right
	`&`	Fully evaluated Boolean AND	Given the Boolean primitive values `b` and `c`, the expression `b & c` will evaluate both unconditionally, resulting in the value `true` if both `b` and `c` are `true`, and `false` otherwise.	
11	`^`	Bitwise XOR (exclusive OR)	Given the integral values `x` and `y`, the expression `x ^ y` gives an integer result in which all bits that are set in either `x` or `y`, *but not both*, are set in the result, and all other bits in the result are not set. For example, the value of `10 ^ 3` is `9`; that is, `0x00001010 ^ 0x00000011` gives the value `0x00001001`.	left-to-right
	`^`	Boolean XOR (exclusive OR)	Given the Boolean primitive values `b` and `c`, the expression `b ^ c` will result in the value `true` if either `b` and `c`, *but not both*, is `true`, and `false` otherwise.	
12	`|`	Bitwise OR	Given the integral values `x` and `y`, the expression `x | y` gives an integer result in which all bits that are set in either (or both) of `x` or `y` are set in the result, and all other bits in the result are not set. For example, the value of `10 | 3` is `11`; that is, `0x00001010 ^ 0x00000011` gives the value `0x00001011`.	left-to-right
	`|`	Fully evaluated Boolean OR	Given the Boolean primitive values `b` and `c`, the expression `b | c` will evaluate both unconditionally, resulting in the value `true` if either `b` and `c` (or both) are `true`, and `false` otherwise.	
13	`&&`	Short-circuit Boolean AND	Given the Boolean primitive values `b` and `c`, the expression `b && c` will evaluate `b` and (if `b` is `true`) `c`, and give the value `true` if both `b` and `c` are `true`, and `false` otherwise.	left-to-right
14	`||`	Short-circuit Boolean OR	Given the Boolean primitive values `b` and `c`, the expression `b || c` will evaluate `b` and (if `b` is `false`) `c`, and give the value `true` if either `b` and `c` (or both) are `true`, and `false` otherwise.	left-to-right
15	`? :`	Ternary (conditional) evaluation	Given the Boolean primitive value `b`, and the values `m` and `n`, the expression `b ? m : n` will evaluate `b` and give the value `m` if `b` is true, and the value `n` if `b` is `false`.	right-to-left
16	`=`	Assignment	The expression `x = y` assigns the value of `y` to the variable `x` (which could be a local variable, a static field of a class, a non-static field of an object instance, or an array element reference), and gives as a result the value that was assigned. For example, `x = 10` assigns the value `10` to the variable `x`, and gives `10` as the value of the expression.	right-to-left
	`*=`, `/=`, `%=`	Assignment, augmented with multiplication, division, modular division	Augmented (compound) assignment combines assignment with an additional operation, resulting in an update to the value of some variable, field, or array element. The expression `a op= b`, where *`op`* is an operator valid for the operands `a` and `b`, is equivalent to `a = a op b`. For example, `x += 2` is equivalent to `x = x + 2`. As with regular assignment, the value assigned becomes the resulting value of the expression.	
	`+=`, `-=`	Assignment, augmented with addition, subtraction		
	`+=`	Assignment, augmented with string concatenation		
	`&=`, `^=`, `|=`	Assignment, augmented with bitwise or Boolean AND, XOR, OR		
	`<<=`, `>>=`, `>>>=`	Assignment, augmented with left shift, signed right shift, unsigned right shift		

General Notes

- Expressions are evaluated in ascending precedence order, and from left to right. However, the associativity of a precedence group specifies how a chain of successive operators in that group are evaluated. For example, the assignment operator has right-to-left associativity, so in the expression `a = b = c`, the value of `c` is first assigned to `b`, then the value of that assignment expression (`c`, in this case) is assigned to `a`.
- Numeric primitives (including `char`) are widened according to the following sequence:
    - If either operand is a `double`, then the other is widened to a `double` (if it's not already a `double`);
    - Otherwise, if either operand is a `float`, then the other is widened to a `float` (if it's not already a `float`).
    - Otherwise; if either operand is a `long`, then the other is widened to a `long` (if it's not already a `long`);
    - Otherwise, both operands (or the single operand of a unary operator) are widened to `int` (if they aren't already that type). This can sometimes lead to counter-intuitive results; for example, the sum of 2 `byte` values cannot be assigned to a `byte` variable, unless the addition result is cast to a `byte`.
- Usually, the assignment, augmented assignment, and postfix/prefix increment/decrement operators are used for their side effects. That is, they are used primarily to modify the value of variables, rather than as sub-expressions within enclosing expressions. Some organizations strongly discourage (or even forbid) the use of such expressions as sub-expressions.