Precedence and Associativity of Operators in Java

There can be more than one operator in a certain expression in java. So the interpreter needs to decide which operators are to be evaluated first and then the next set of operators.

For example **(1+9) * (a-(9*i)) + 10**

This can be related to the BODMAS rule in school level mathematics that we learnt.

So Precedence and Associativity of operators will help us to evaluate such long expressions easily.

PRECEDENCE

Precedence is the property of the operator that decides which operator will be executed before another when there are more than one operators in an expression, for example: **1 + 2 * 5** . here what will be the order of the evaluation of the expression? will it be **1+2** then **3 * 5** or **2 * 5** and then **1 + 10** ? The two cases give us two results, one is 15 and the other is 11. So Which operator will be executed first depends on the precedence of the operator. In this case the ***** operator will be executed first and the answer will 11.

ASSOCIATIVITY

Another case can arise where all the operators are similar, in that case what will be the order of operation of the operators will be? This is the case where we use the second property of java, the associativity of the operators. Consider the following statement: **a = b = c = d = e = f = g = 100** Now in what way, will the assigning of statements work? Associativity y tells us the direction of execution of operation whether we are going to go from *left to right or right to left* . So **=** has the associativity from *right to left.* so in this case *g* will be assigned to 100 and then it will follow like this: **(a = (b = (c = (d = (e = (f = (g = 100)))))))**

The full list of associativity and precedence of the operators are given below:

Collected from http://cs-fundamentals.com/java-programming/java-operators-precedence-and-associativity.php#eval-order.

Precedence	Operator	Description	Associativity
1	[] () .	array index method call member access	Left -> Right
2	++ -- + - ~ !	pre or postfix increment pre or postfix decrement unary plus, minus bitwise NOT logical NOT	Right -> Left
3	(type cast) new	type cast object creation	Right -> Left
4	* / %	multiplication division modulus (remainder)	Left -> Right
5	+ - +	addition, subtraction string concatenation	Left -> Right
6	<< >> >>>	left shift signed right shift unsigned or zero-fill right shift	Left -> Right
7	< <= > >= instanceof	less than less than or equal to greater than greater than or equal to reference test	Left -> Right
8	== !=	equal to not equal to	Left -> Right
9	&	bitwise AND	Left -> Right
10	^	bitwise XOR	Left -> Right
11	|	bitwise OR	Left -> Right
12	&&	logical AND	Left -> Right
13	||	logical OR	Left -> Right
14	? :	conditional (ternary)	Right -> Left
15	= += -= *= /= %= &= ^= |= <<= >>= >>>=	assignment and short hand assignment operators	Right -> Left