Operator Precedence in the Java™ Programming Language

handout for CS 302 by Will Benton (willb@cs)

Operator precedence defines the order in which various operators are evaluated. (In fact, you may remember "order of operations" from secondary school algebra.)

As an example, let's say we have the following line of Java code:

```
int x = 4 + 3 * 5;
```

The variable `x` gets the value of evaluating the expression `4 + 3 * 5`. There are a couple of ways to evaluate that expression, though: We can either perform the addition first or perform the multiplication first.

By choosing which operation to perform first, we are actually choosing between two different expressions:

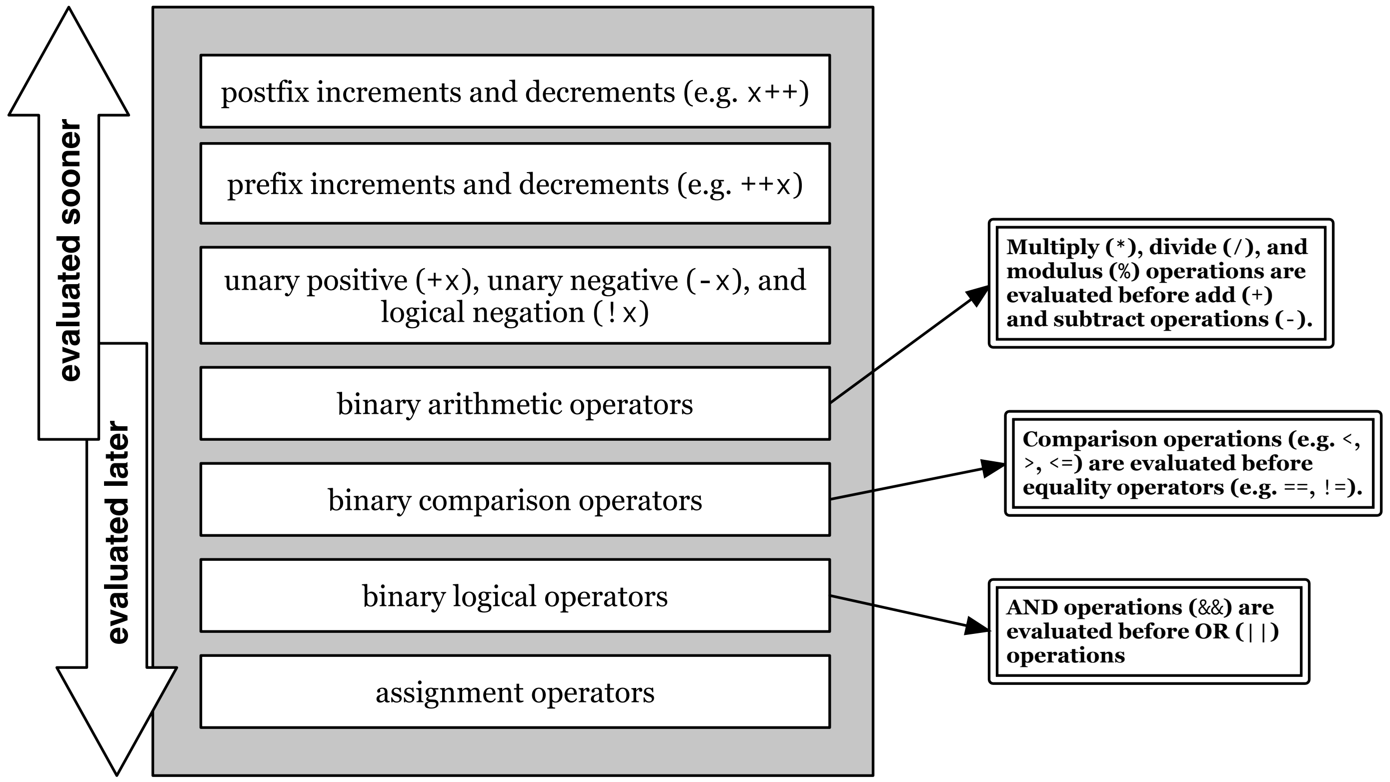
```
1. (4 + 3) * 5 == 35
2. 4 + (3 * 5) == 19
```

In the absence of parentheses, which choice is appropriate? Programming languages answer this question by defining precedence levels for each operator, indicating which is to be performed first. In the case of Java, multiplication takes precedence over addition; therefore, `x` will get the value `19`.

For arithmetic expressions, multiplication and division are evaluated before addition and subtraction, just like in mathematics. Of course, just as you might in a math class, you can always parenthesize Java expressions to indicate which are to be evaluated first.

Sensible use of parentheses will make your programs easier to read even if your expressions all use the standard evaluation order.

This sheet shows the operator precedences for the Java operators you'll be using most frequently in CS 302. On the reverse of this sheet is a chart of the precedence levels for every operator in the Java language, provided in case you're curious!


evaluated sooner
evaluated later
postfix increments and decrements (e.g. x++)
prefix increments and decrements (e.g. ++x)
unary positive (+x), unary negative (-x), and logical negation (!x)
binary arithmetic operators
binary comparison operators
binary logical operators
assignment operators
Multiply (*), divide (/), and modulus (%) operations are evaluated before add (+) and subtract operations (-).
Comparison operations (e.g. <, >, <=) are evaluated before equality operators (e.g. ==, !=).
AND operations (&&) are evaluated before OR (||) operations

`[] . ,`	subscript, member selection, comma (only in `for` loop headers)	**left-associative**
x++ *x*-- ~*x*	postfix increment, postfix decrement, bitwise negation	
++*x* --*x* +*x* -*x* !*x*	prefix increment, prefix decrement, unary positive, unary negative, logical negation	**right-associative**
(*X*) new *X*	typecasting, object creation	
`* / %`	multiplication, division, modulus	**left-associative**
x+*y* *x*-*y* *x*+"*x*"	addition, subtraction, string concatenation	
`<< >> >>>`	bitwise shift	
`< <= > >=`	comparison	
`instanceof`	runtime type compatibility	
`== !=`	equality and inequality	
`&`	bitwise AND	
`^`	bitwise XOR	
`|`	bitwise OR	
`&&`	logical AND	
`||`	logical OR	
x ? *y* : *z*	ternary (conditional)	**right-associative**
`=` `+= -= *= /= %=` `<<= >>= >>>=` `&= ^= |=`	assignment and compound assignment	

higher precedence

This chart (c) 2005-2007 William C. Benton.
sources:
Chan, Patrick. *The Java Developers Almanac 1.4.* Addison-Wesley, 2002.
Gosling, James, et al. *The Java Language Specification, 3e.* Addison-Wesley, 2005.

lower precedence

Note: operators with the same precedence level in an expression are evaluated based on their associativity. For example, left-associative operators group from left to right. Therefore, the expression `x * y % z` is equivalent to `(x * y) % z`.