Operators

Operators

Operators can be divided into the following groups:

- Arithmetic
- Bitwise
- Relational
- Logical
- instanceof operator

Operators

Operator	Category	Precedence
Unary Operator	postfix	expression++ expression--
	prefix	++expression --expression +expression -expression ~!
Arithmetic Operator	multiplication	* / %
	addition	+ -
Shift Operator	shift	<< >> >>>
Relational Operator	comparison	< > <= >= instanceof
	equality	== !=
Bitwise Operator	bitwise AND	&
	bitwise exclusive OR	^
	bitwise inclusive OR	|
Logical Operator	logical AND	&&
	logical OR	||
Ternary Operator	ternary	? :
Assignment Operator	assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Operators

- The operands of the arithmetic operators must be of a numeric type. It can not be applied on boolean types.
- Arithmetic operators can be used with char type as char type in Java is essentially a subset of int.
- Integer division does not have any fractional component.
- +=, -=, *=, /=, %= are known as arithmetic assignment operators.

Operators

- var = var op expression

  can be written as

  var op = expression

- Advantage of arithmetic assignment operators
  - Saves time in typing
  - Implemented more efficiently by the Java run-time system.

Operators

- Increment & decrement Operators

  x++;

  x = y++;

  x = --y;

- Unary operators precedes an operand.

  int a = 5;

  int b = -a;

  sop(b) ; // -5

Operators

Bitwise Operators

- Java uses encoding known as 2's complement.
- High order bit determines the sign of an integer.
- The bitwise operators are applied to each individual bit within each operand.

Bitwise NOT (~)

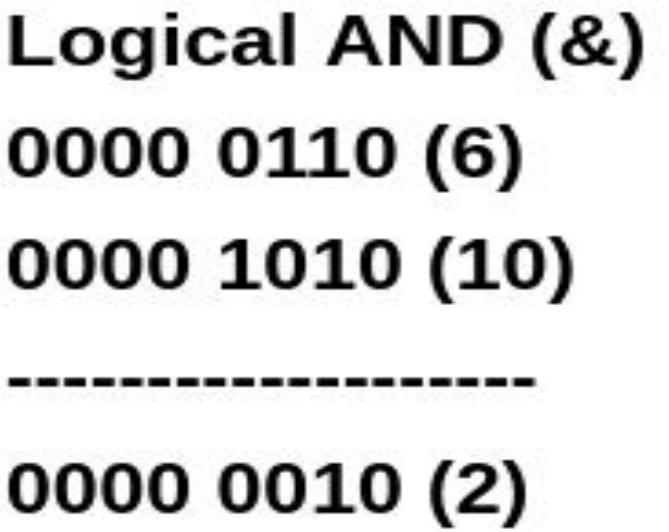
- Inverts all bit of the operand (1's complement)

Operators

Bitwise Logical Operators

A	B	A | B	A & B	A ^ B	~A
0	0	0	0	0	1
1	0	1	0	1	0
0	1	1	0	1	1
1	1	1	1	0	0

Operators



Similarly, you can see the result of bitwise logical or, xor and not operator.

Operators

Left Shift Operator (<<)

- For each shift left, the high-order bit is shifted out (and lost), and a zero is brought in on the right.

  Ex. a = 6 (0000 0110)

  a << 1 (0000 1100) // 12

- Left shift gives multiplication by 2.

Operators

Right Shift Operator (>>)

- For each shift right, the top (leftmost) bits exposed by the right shift are filled in with the previous contents of the top bit. This is called sign extensions and serves to preserve the sign of negative number when it is shifted right.

  Ex. a = -6 (1111 1010)

  a >> 1 (1111 1101) // -3

- Right shift divides the value by two and discards the remainder.

Operators

Unsigned Right Shift Operator (>>>)

- For each unsigned shift right, the top (leftmost) bits exposed by the right shift are filled by zero, no matter what its initial value was. I.e. it always shifts zeroes into the high order bit.
- Ex. a = -1 (1111111 1111111 1111111 1111111)

  a >>> 24 (00000000 00000000 00000000 1111111)   // 255

- As smaller values are automatically promoted to int in expressions so while doing shifting, it is actually 32 bit value which is actually being shifted.
- a = a >>> 24 can be written as a >>>= 24

Operators

Relational Operators

- **== , != , > , < , >= , <=**
- Outcome of these operators is always a boolean value

  Ex. boolean c = a < b;

- In Java, true and false are non numeric values which do not relate to zero or nonzero. For example, **if (1) { }** statement is wrong and will give compilation error.

Operators

Boolean Logical Operators

Operator	Result
&	Logical AND
|	Logical OR
^	Logical XOR (exclusive OR)
||	Short-circuit OR
&&	Short-circuit AND
!	Logical unary NOT
&=	AND assignment
|=	OR assignment
^=	XOR assignment
==	Equal to
!=	Not equal to
?:	Ternary if-then-else

A	B	A | B	A & B	A ^ B	!A
False	False	False	False	False	True
True	False	True	False	True	False
False	True	True	False	True	True
True	True	True	True	False	False

Operators

Boolean Logical Operators

- The logical boolean operators &, | , ^ , ! operates in the same way as they operate on the bits of an integer.
- If && and | | is used rather than & and | , Java will not bother to evalluate the right-hand operand when the outcome of the expression can be determined by the left operand alone.
- If (d != 0 && num / d > 5) { } // No risk of causing a run time exception.
- If (x == 0 & y++ < 10) { d = 10; } // Value of y and d, if x = 1 and y = 5 ??

Operators

Parentheses, Assignment and ? Operator

- Chain of assignment. Ex. x = y = z = 100; // Set x, y and z to 100.
- Ternary Operator => Expression1 ? expression2 : Expression3
- Use parentheses raises the precedence of the operations and sometimes clarify the meaning of an expression.

a >> b + 2; // equals to a >> (b + 2) i.e. first add then shift

(a >> b) +2 ;