Feel free to contact santhosh_arena@yahoo.co.in

SCJP java Programmer certification Exam And Training. This site is entirely independent of Sun Microsystems Inc, The Sun Certified Java Programmers Exam (SCJP) is the internationally recognized java certification exam. This site offers free mock Exam preparation questions, practice tests, tutorials, certification faq and sample code. This site contains a detailed tutorial organized by topic. At the end of the tutorial you can learn from your mistakes and apply your concepts on the free mock exams java certification practice tests.

Chapter 2 : Operators and assignments

The Java programming language has included five simple arithmetic operators like + (addition), - (subtraction), * (multiplication), / (division)

1. Unary operators

1.1 Increment and Decrement operators ++ --

We have postfix and prefix notation. In post-fix notation value of the variable/expression is modified after the value is taken for the execution of statement. In prefix notation, value of the variable/expression is modified before the value is taken for the execution of statement.
x = 5; y = 0; y = x++; Result will be x = 6, y = 5
x = 5; y = 0; y = ++x; Result will be x = 6, y = 6
Implicit narrowing conversion is done, when applied to byte, short or char.

1.2 Unary minus and unary plus + -

+ has no effect than to stress positivity.
- negates an expression's value. (2's complement for integral expressions)

1.3 Negation !

Inverts the value of a boolean expression.

1.4 Complement ~

Inverts the bit pattern of an integral expression. (1's complement - 0s to 1s and 1s to 0s)
Cannot be applied to non-integral types.

1.5 Cast ()

Persuades compiler to allow certain assignments. Extensive checking is done at compile and runtime to ensure type-safety.

2. Arithmetic operators - *, /, %, +, -

- Can be applied to all numeric types.
- Can be applied to only the numeric types, except '+' - it can be applied to Strings as well.
- All arithmetic operations are done at least with 'int'. (If types are smaller, promotion happens. Result will be of a type at least as wide as the wide type of operands)
- Accuracy is lost silently when arithmetic overflow/error occurs. Result is a nonsense value.
- Integer division by zero throws an exception.
- % - reduce the magnitude of LHS by the magnitude of RHS. (continuous subtraction)
- % - sign of the result entirely determined by sign of LHS
- 5 % 0 throws an ArithmeticException.
- Floating point calculations can produce NaN (square root of a negative no) or Infinity (division by zero). Float and Double wrapper classes have named constants for NaN and infinities.
- NaN's are non-ordinal for comparisons. x == Float.NaN won't work. Use Float.IsNaN(x) But equals method on wrapper objects(Double or Float) with NaN values compares Nan's correctly.
- Infinities are ordinal. X == Double.POSITIVE_INFINITY will give expected result.
- + also performs String concatenation (when any operand in an expression is a String). The language itself overloads this operator. toString method of non-String object operands are called to perform concatenation. In case of primitives, a wrapper object is created with the primitive value and toString method of that object is called. ("Vel" + 3 will work.)
- Be aware of associativity when multiple operands are involved.

```
System.out.println( 1 + 2 + "3" ); // Prints 33
System.out.println( "1" + 2 + 3 ); // Prints 123
```

3. Shift operators - <<, >>, >>>

- << performs a signed left shift. 0 bits are brought in from the right. Sign bit (MSB) is preserved. Value becomes old value * 2 ^ x where x is no of bits shifted.
- >> performs a signed right shift. Sign bit is brought in from the left. (0 if positive, 1 if negative. Value becomes old value / 2 ^ x where x is no of bits shifted. Also called arithmetic right shift.
- >>> performs an unsigned logical right shift. 0 bits are brought in from the left. This operator exists since Java doesn't provide an unsigned data type (except char). >>> changes the sign of a negative number to be positive. So don't use it with negative numbers, if you want to preserve the sign. Also don't use it with types smaller than int. (Since types smaller than int are promoted to an int before any shift operation and the result is cast down again, so the end result is unpredictable.)
- Shift operators can be applied to only integral types.
- -1 >> 1 is -1, not 0. This differs from simple division by 2. We can think of it as shift operation rounding down.
- 1 << 31 will become the minimum value that an int can represent. (Value becomes negative, after this operation, if you do a signed right shift sign bit is brought in from the left and the value remains negative.)
- Negative numbers are represented in two's complement notation. (Take one's complement and add 1 to get two's complement)
- Shift operators never shift more than the number of bits the type of result can have. (i.e. int 32, long 64) RHS

operand is reduced to RHS % x where x is no of bits in type of result.

int x;

x = x >> 33; / / Here actually what happens is x >> 1

4. Comparison operators - all return boolean type.

4.1 Ordinal comparisons - <, <=, > , >=

· Only operate on numeric types. Test the relative value of the numeric operands.

· Arithmetic promotions apply. char can be compared to float.

4.2 Object type comparison - instanceof

· Tests the class of an object at runtime. Checking is done at compile and runtime same as the cast operator.

· Returns true if the object denoted by LHS reference can be cast to RHS type.

· LHS should be an object reference expression, variable or an array reference.

· RHS should be a class (abstract classes are fine), an interface or an array type, castable to LHS object reference. Compiler error if LHS & RHS are unrelated.

· Can't use java.lang.Class or its String name as RHS.

· Returns true if LHS is a class or subclass of RHS class

· Returns true if LHS implements RHS interface.

· Returns true if LHS is an array reference and of type RHS.

· x instanceof Component[ ] - legal.

· x instanceof [ ] - illegal. Can't test for 'any array of any type'

· Returns false if LHS is null, no exceptions are thrown.

· If x instanceof Y is not allowed by compiler, then Y y = (Y) x is not a valid cast expression. If x instanceof Y is allowed and returns false, the above cast is valid but throws a ClassCastException at runtime. If x instanceof Y returns true, the above cast is valid and runs fine.

4.3 Equality comparisons - ==, !=

· For primitives it's a straightforward value comparison. (promotions apply)

· For object references, this doesn't make much sense. Use equals method for meaningful comparisons. (Make sure that the class implements equals in a meaningful way, like for X.equals(Y) to be true, Y instance of X must be true as well)

· For String literals, == will return true, this is because of compiler optimization.

5. Bit-wise operators - &, ^, |

· Operate on numeric and boolean operands.

· & - AND operator, both bits must be 1 to produce 1.

· | - OR operator, any one bit can be 1 to produce 1.

· ^ - XOR operator, any one bit can be 1, but not both, to produce 1.

- In case of booleans true is 1, false is 0.
- Can't cast any other type to boolean.

6. Short-circuit logical operators - &&, ||

- Operate only on boolean types.
- RHS might not be evaluated (hence the name short-circuit), if the result can be determined only by looking at LHS.
- false && X is always false.
- true || X is always true.
- RHS is evaluated only if the result is not certain from the LHS.
- That's why there's no logical XOR operator. Both bits need to be known to calculate the result.
- Short-circuiting doesn't change the result of the operation. But side effects might be changed. (i.e. some statements in RHS might not be executed, if short-circuit happens. Be careful)

7. Ternary operator

- Format a = x ? b : c ;
- x should be a boolean expression.
- Based on x, either b or c is evaluated. Both are never evaluated.
- b will be assigned to a if x is true, else c is assigned to a.
- b and c should be assignment compatible to a.
- b and c are made identical during the operation according to promotions.

8. Assignment operators.

- Simple assignment =.
- op= calculate and assign operators extended assignment operators.
- *=, /=, %=, +=, -=
- x += y means x = x + y. But x is evaluated only once. Be aware.
- Assignment of reference variables copies the reference value, not the object body.
- Assignment has value, value of LHS after assignment. So a = b = c = 0 is legal. c = 0 is executed first, and the value of the assignment (0) assigned to b, then the value of that assignment (again 0) is assigned to a.
- Extended assignment operators do an implicit cast. (Useful when applied to byte, short or char)

```
byte b = 10;
b = b + 10; // Won't compile, explicit cast required since the expression evaluates to an int
b += 10; // OK, += does an implicit cast from int to byte
```

9. General

- In Java, No overflow or underflow of integers happens. i.e. The values wrap around. Adding 1 to the maximum int value results in the minimum value.
- Always keep in mind that operands are evaluated from left to right, and the operations are executed in the order of precedence and associativity.
- Unary Postfix operators and all binary operators (except assignment operators) have left to right assoiciativity.

· All unary operators (except postfix operators), assignment operators, ternary operator, object creation and cast operators have right to left assoiciativity.

· Inspect the following code.

```
public class Precedence {
final public static void main(String args[]) {
        int i = 0;
        i = i++;
        i = i++;
        i = i++;
        System.out.println(i); // prints 0, since = operator has the lowest precedence.
        int array[] = new int[5];
        int index = 0;
        array[index] = index = 3; // 1st element gets assigned to 3, not the 4th element
        for (int c = 0; c < array.length; c++)
                System.out.println(array[c]);
                System.out.println("index is " + index); // prints 3
        }
}
```

-

Type of Operators	Operators	Associativity
Postfix operators	[] . (parameters) ++ --	Left to Right
Prefix Unary operators	++ -- + - ~ !	Right to Left
Object creation and cast	new (type)	Right to Left
Multiplication/Division/Modulus	* / %	Left to Right
Addition/Subtraction	+ -	Left to Right
Shift	>> >>> <<	Left to Right
Relational	< <= > >= instanceof	Left to Right
Equality	== !=	Left to Right
Bit-wise/Boolean AND	&	Left to Right
Bit-wise/Boolean XOR	^	Left to Right
Bit-wise/Boolean OR	|	Left to Right
Logical AND (Short-circuit or Conditional)	&&	Left to Right

Logical OR (Short-circuit or Conditional)	**||**	**Left to Right**
Ternary	**? :**	**Right to Left**
Assignment	**= += -= *= /= %= <<= >>= >>>= &= ^= |=**	**Right to Left**