2nd Day of Learning Java

Expression

Expression is the combination of operands(variables), operators, method calls or parenthesis.

Expression has two parts : Operand & Operator.

e.g.

- a + b
- String name = “Anil” (not expression , this is statement because it declares a variable and assigns a value to it but does not directly produce a value.)

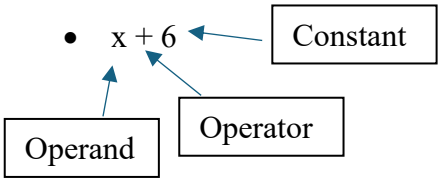
Operators in Java

Operator is a symbol or keyword that tells compiler to perform specific mathematical or logical operations whereas operand is a variable or constant that operators act upon.

In general, the operator is a character that represents action. For Example, + is the arithmetic operator that represents addition.

e.g.

- x + 6



Here, + is the operator which acts on these two operands x (variable) and 6 (constant) and produces the desired output.

Types of Operators in Java

On the Basis of Operand

There are three (3) types of operators based on the number of operands used to perform an operation.

1. **Unary Operator**
   The operator that acts on a single operand is known as the Unary Operator. For Example, increment & decrement operator
   e.g. a++, --b

2. **Binary Operator**
   The operator that acts on two operand is known as the Binary Operator.

e.g. a+b, x >= y

3. **Ternary Operator**
   The operator that acts on three operand is known as the Ternary Operator.
   e.g. (x > y) ? x : y;

On the Basis of Symbols and Named

1. **Symbolic Operator**
   a) Arithmetic Operator (+, - , *, /, %)
   b) Relational Operator (Comparison Operator) (<, >, <=, >=, ==, !=)
   c) Logical Operator (&&, ||, !)
   d) Assignment Operator (=, +=, -=, *=, /=, %=)
   e) Increment & Decrement Operator (++, --)
   f) Conditional Operator (?:)
   g) Bitwise Operator (&, !, ^, ~)
   h) Shift Operator (<<, >>, >>>)
2. **Named Operator**
   a) Instanceof Operator

Arithmetic Operator

Arithmetic operators are used to performing fundamental arithmetic operations such as Addition, Subtraction, Multiplication, Division, and Remainder.

e.g. a+b , b % c

Relational Operator

Relational operators are mainly used to create conditions in decision-making statements. The result is always of a Boolean type i.e. either true or false.

e.g. a < b, x == y

Logical Operator

used to perform logical operations, typically with Boolean (true/false) values. These operators are often used in conditional statements like if, while, and for.

Assignment Operator

Assignment operator is used to assign the value on its right to the operand on its left. We can categorize assignment operator into 3 categories.

1. **Simple Assignment**

   To store or assign a value into a variable: int x = 10;

To store a value of a variable into another variable: int y = x;

2. **Compound Assignment**
   a shorthand way of performing operations on variables and then assigning the result back to the same variable. Instead of writing out the full expression, we combine the operator with the = symbol.
   e.g. int x = 5;
   x += 3; (x = x + 3)

3. **Assignment as Expression**

   int x = a + b – 9;

   Here, the expression a + b – 9 is evaluated first, then its result is stored in the variable x.

Increment / Decrement Operator

The operator which is used to increase the value of the variable (operand) by one is called increment operator.

The operator which is used to decrease the value of the variable (operand) by one is called decrement operator.

There are two types of increment and decrement operators, i.e. prefix and postfix.

e.g. ++a or a++ --b or b--

Conditional Operator

Syntax: datatype variable_name = (expression)? Value if true : value if false;

a = 10;

b = 3;

e.g. int result = (a > b)? a : b;

Bitwise Operator

An operator that acts on individual bits (0 or 1) of the operands is called bitwise operator. They are often used in low-level programming tasks like bit manipulation, data encoding, and system programming.

It acts only on integer types such as byte, short, int and long. Bitwise operator in java cannot be applied to float and double data types.

Types of Bitwise Operators

Bitwise AND (&)

Compares each bit of two numbers and sets the result bit to 1 if both corresponding bits are 1.

Example:

Input:

- **Number 1**: 0010 (Decimal: 2)
- **Number 2**: 0011 (Decimal: 3)

We compare each bit position **one by one**:

0010

& 0011

0010

- Compare the **first bit (leftmost)**: 0 & 0 = 0
- Compare the **second bit**: 0 & 0 = 0
- Compare the **third bit**: 1 & 1 = 1
- Compare the **fourth bit (rightmost)**: 0 & 1 = 0

Result:

- The result is 0010 in binary, which is **2** in decimal.

Bitwise OR (|)

Compare each bit of two numbers and sets the result bit to 1 if at least one of the corresponding bits is 1.

Example:

- 5 | 3 (Binary: 0101 | 0011 = 0111)
- Output: 7

Bitwise XOR (^)

Returns 1 if and only if one of the operands is 1. However, if both the operands are 0 or if both are 1, then the result is 0.

(Compares each bit of two numbers and sets the result bit to 1 if the corresponding bits are different.)

Example:

- 5 ^ 3 (Binary: 0101 ^ 0011 = 0110)
- Output: 6

Bitwise Complement (~)

Flips all the bits of a number (0 becomes 1, and 1 becomes 0). Bitwise Complement is a unary operator.

Example:

- ~0010 = 1101

---

*Note: The bitwise complement of any integer **N** is equal to **- (N + 1)**.*

Consider, an integer is 35. As per above rule the bitwise complement is –(35 +1) = -36.

35 in Binary: 00100011

~ 00100011

= 11011100

The binary number 11011100 is equivalent to 220 in decimal. But The most significant bit (MSB) (leftmost bit) is 1, which indicates it's a **negative number**.

---

*Note: However, we cannot directly convert the result into decimal and get the desired output. This is because the binary result **11011100** is also equivalent to **-36**.*

Compute the 2's complement of 36:

36 = 00100100 (In Binary)

1's complement = 11011011

2's complement:

11011011

\+ 1

= 11011100

The 2's complement of 36 (i.e. -36) is 11011100 which is equivalent to the bitwise complement of 35.

So, the bitwise complement of 35 is -(35 + 1) = -36.

Left Shift (<<)

The left shift (<<) operator shifts the bits of the number to the left by the specified number of positions, filling the rightmost bits with zeros. It is equivalent to multiplying the number by 2^n, where n is the number of bit positions shifted.

Example:

- 5 << 2 (Binary: 00000101 << 2 = 00010100) (Two Zeros appended to the right side)
- Output: 20
- n = 2. So, 5 * 2^2 = 5 * 4 = 20

Right Shift (>>) / Signed Right Shift

The right shift operator (>>) works by shifting the binary representation of the number to the right and filling the leftmost positions with the sign bit. It is equivalent to dividing the number by 2^n (where n is the number of shifts)

- If the sign bit is 0 then it represents a positive number. In other word, if the number is positive, the leftmost position is filled with 0.
- If the sign bit is 1 then it represents a negative number. In other word, if the number is negative, the leftmost position is filled with 1.

Example:

- 10 >> 2 (In binary, 00001010>>2 = 0010 – Two bits removed from the right, and zeros added to the left.)
- Output: 2
- n = 2. So, 10 / 2^2 = 10 * 4 = 2

Unsigned Right Shift (>>>) / Zero fill right shift

Shifts all bits to the right and fills the leftmost bits with 0 regardless of the sign of the number.

Instanceof Operator

The instanceof operator in Java is used to check whether an object is an instance of a specific class or implements a particular interface.

Syntax: object instanceof ClassName