

Java Operator I

► Operators in Java can be classified into 6 types:

- ① Arithmetic Operators
- ② Assignment Operators
- ③ Relational Operators
- ④ Logical Operators
- ⑤ Unary Operators
- ⑥ Bitwise Operators

Java Operator II

- ① Java Arithmetic Operators: Arithmetic operators are used to perform arithmetic operations on variables and data.

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo Operation (Remainder after division)

Java Operator III

Example:

```
1 a + b;  
2 a - b;  
3 a * b;  
4 a / b;  
5 a % b;
```

```
1 public class Main  
2 {  
3     public static void main(String [] args)  
4     {  
5         // declare variables  
6         int a = 12, b = 5;  
7         // addition operator
```

Java Operator IV

```
8   System.out.println ("a + b = " + (a + b));  
9   // subtraction operator  
10  System.out.println ("a - b = " + (a - b));  
11  // multiplication operator  
12  System.out.println ("a * b = " + (a * b));  
13  // division operator  
14  System.out.println ("a / b = " + (a / b));  
15  // modulo operator  
16  System.out.println ("a % b = " + (a % b));  
17 }  
18 }
```

Java Operator V

Output:

```
1 a + b = 17
2 a - b = 7
3 a * b = 60
4 a / b = 2
5 a % b = 2
```

Java Operator VI

- ② Java Assignment Operators: Assignment operators are used in Java to assign values to variables.

Operator	Example	Equivalent to
=	a = b;	a = b;
+=	a += b;	a = a + b;
-=	a -= b;	a = a - b;
*=	a *= b;	a = a * b;
/=	a /= b;	a = a / b;
%=	a %= b;	a = a % b;

Java Operator VII

```
1 class Main
2 {
3     public static void main(String [] args)
4     {
5         // create variables
6         int a = 4;
7         int var;
8         // assign value using =
9         var = a;
10        System.out.println ("var using =: " + var);
11        // assign value using +=
12        var += a;
13        System.out.println ("var using +=: " + var);
14        // assign value using *=
```

Java Operator VIII

```
15     var *= a;  
16     System.out.println ("var using *=: " + var);  
17 }  
18 }
```

Output:

```
1 var using =: 4  
2 var using +=: 8  
3 var using *=: 32
```

Java Operator IX

- ③ Java Relational Operators: Relational operators are used to check the relationship between two operands.

```
1 // check is a is less than b  
2 a < b;
```

Here, < operator is the relational operator. It checks if a is less than b or not.

It returns either true or false.

Java Operator X

Operator	Description	Example
<code>==</code>	Is Equal To	<code>3 == 5</code> returns False
<code>!=</code>	Not Equal To	<code>3 != 5</code> returns True
<code>></code>	Greater Than	<code>3 > 5</code> returns False
<code><</code>	Less Than	<code>3 < 5</code> returns True
<code>>=</code>	Greater Than or Equal To	<code>3 >= 5</code> returns False
<code><=</code>	Less Than or Equal To	<code>3 <= 5</code> returns False

Java Operator XI

```
1 public class Main
2 {
3     public static void main(String [] args)
4     {
5         // create variables
6         int a = 7, b = 11;
7         // value of a and b
8         System.out.println ("a is " + a + " and b is " + b);
9         // == operator
10        System.out.println (a == b); // false
11        // != operator
12        System.out.println (a != b); // true
13        // > operator
14        System.out.println (a > b); // false
```

Java Operator XII

```
15 // < operator
16 System.out.println (a < b); // true
17 // >= operator
18 System.out.println (a >= b); // false
19 // <= operator
20 System.out.println (a <= b); // true
21 }
22 }
```

Java Operator XIII

- ④ Java Logical Operators: Logical operators are used to check whether an expression is **True** or **False**. They are used in decision making.

Operator	Example	Meaning
&& (Logical AND)	expression1 && expression2	True only if both expression1 and expression2 are True
(Logical OR)	expression1 expression2	True if either expression1 or expression2 is True
! (Logical NOT)	!expression	True if expression is False and vice versa

Java Operator XIV

```
1 public class Main
2 {
3     public static void main(String [] args)
4     {
5         // && operator
6         System.out.println ((5 > 3) && (8 > 5)); // true
7         System.out.println ((5 > 3) && (8 < 5)); // false
8         // || operator
9         System.out.println ((5 < 3) || (8 > 5)); // true
10        System.out.println ((5 > 3) || (8 < 5)); // true
11        System.out.println ((5 < 3) || (8 < 5)); // false
12        // ! operator
13        System.out.println (!(5 == 3)); // true
14        System.out.println (!(5 > 3)); // false
```

Java Operator XV

```
15     }
16 }
```

Java Operator XVI

- 5 Java Unary Operators: Unary operators are used with only one operand. For example, `++` is a unary operator that increases the value of a variable by 1. That is, `++5` will return 6.

Operator	Meaning
<code>+</code>	Unary plus: not necessary to use since numbers are positive without using it
<code>-</code>	Unary minus: inverts the sign of an expression
<code>++</code>	Increment operator: increments value by 1
<code>--</code>	Decrement operator: decrements value by 1
<code>!</code>	Logical complement operator: inverts the value of a boolean

Java Operator XVII

```
1 int num = 5;
2 // increase num by 1
3 ++num;

1 public class Main
2 {
3     public static void main(String [] args)
4     {
5         // declare variables
6         int a = 12, b = 12;
7         int result1 , result2 ;
8         // original value
9         System.out. println ("Value of a: " + a);
10        // increment operator
```

Java Operator XVIII

```
11     result1 = ++a;
12     System.out.println ("After increment: " + result1 );
13     System.out.println ("Value of b: " + b);
14     // decrement operator
15     result2 = --b;
16     System.out.println ("After decrement: " + result2 );
17 }
18 }
```

Output:

```
1 Value of a: 12
2 After increment: 13
3 Value of b: 12
4 After decrement: 11
```

Java Operator XIX

```
1 public class Operator
2 {
3     public static void main(String [] args)
4     {
5         int var1 = 5, var2 = 5;
6         // var1 is displayed
7         // Then, var1 is increased to 6.
8         System.out.println (var1++);
9         // var2 is increased to 6, then, var2 is displayed
10        System.out.println (++var2);
11    }
12 }
```

Output: ?

Java Operator XX

- ⑥ Java Bitwise Operators: Bitwise operators in Java are used to perform operations on individual bits.

```
1 Bitwise complement Operation of 35
2 35 = 00100011 (In Binary)
3 ~ 00100011
4 _____
5 11011100 = 220 (In decimal)
6 Here, ~ is a bitwise operator . It inverts the value of each bit (0 to 1 and 1 to 0) .
```

Java Operator XXI

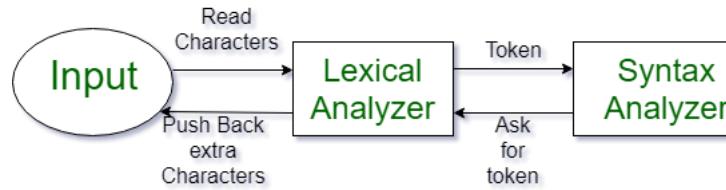
Operator	Description
<code>~</code>	Bitwise Complement
<code><<</code>	Left Shift
<code>>></code>	Right Shift
<code>>>></code>	Unsigned Right Shift
<code>&</code>	Bitwise AND
<code>^</code>	Bitwise exclusive OR

C Operator Precedence and Associativity I

Introduction of Lexical Analyzer:

- ✓ Lexical Analysis is the first phase of the compiler also known as a scanner.
It converts the High level input program into a sequence of Tokens.
- ✓ The lexical analyzer is the part of the compiler that detects the token of the program and sends it to the syntax analyzer.
- ✓ Token is the smallest entity of the code, it is either a keyword, identifier, constant, string literal, symbol. Examples of different types of tokens in C.

C Operator Precedence and Associativity II



What is a token?

A lexical token is a sequence of characters that can be treated as a unit in the grammar of the programming languages.

Example of tokens:

C Operator Precedence and Associativity III

- Type token (id, number, real, . . .)
- Punctuation tokens (IF, void, return, . . .)
- Alphabetic tokens (keywords)
- Keywords; Examples-for, while, if etc.
- Identifier; Examples-Variable name, function name, etc.
- Operators; Examples '+', '++', '-' etc.
- Separators; Examples ',', ';' etc

Example of Non-Tokens:

1 Comments, preprocessor directive, macros, blanks, tabs, newline, etc.

C Operator Precedence and Associativity IV

Lexeme: The sequence of characters matched by a pattern to form the corresponding token or a sequence of input characters that comprises a single token is called a lexeme. eg- “float”, “abs_zero_Kelvin”, “=”, “-”, “273”, “;” .

How Lexical Analyzer functions

- ① It always matches the longest character sequence.
- ② Tokenization i.e. Dividing the program into valid tokens.
- ③ Remove white space characters.
- ④ Remove comments.

C Operator Precedence and Associativity V

- ⑤ It also provides help in generating error messages by providing row numbers and column numbers.
 - ✓ The lexical analyzer(scanner) identifies the error with the help of the automation machine and the grammar of the given language on which it is based like C, C++, and gives row number and column number of the error.
- Suppose we pass a statement through lexical analyzer–

```
1 a = b + c ; //It will generate token sequence like this:  
2 id= id + id; //Where each id refers to it's variable in the symbol table  
               referencing all details
```

For example, consider the program:

C Operator Precedence and Associativity VI

```
1 int a=5; // int a = 5 ; just for understanding.  
2 Tokens:  
3 |int| |a| |=| |5| |;
```

Another example:

```
1 int main()  
2 {  
3     // 2 variables  
4     int a, b;  
5     a = 10;  
6     return 0;  
7 }  
-----  
8 Tokens:
```

C Operator Precedence and Associativity VII

```
10 int
11 main
12 (
13 )
14 {
15     int
16     a
17     ,
18     b
19     ;
20     a
21     =
22     10
23     ;
```

C Operator Precedence and Associativity VIII

```
24     return
25     0
26     ;
27 }
```

```
1 //How many tokens?
2
3 printf("BSCPMKSMK");
```

C Operator Precedence and Associativity IX

```
1 int main()
2 {
3     int a = 10, b = 20;
4     printf("sum is :%d",a+b);
5     return 0;
6 }
7 -----
8 //How many tokens?
```

```
1 int max(int i);
2 -----
3 //Count number of tokens :
```

C Operator Precedence and Associativity X

- Lexical analyzer first read int and finds it to be valid and accepts as token
- max is read by it and found to be a valid function name after reading (
- int is also a token , then again i as another token and finally ;

Answer: Total number of tokens 7:

```
1 |int| |max| |(| |int| |i| |)| |;
```

```
1 //Count number of tokens :
```

```
2  
3 printf("i = %d, &i = %x", i, &i);
```

C Operator Precedence and Associativity XI

✓ Identification valid a token.

```
1 inta=10;
2 int a=10;
3 -----
4 //Number of valid tokens: ????
```

C Operator Precedence and Associativity XII

Lvalues and Rvalues in C:

There are two kinds of expressions in C -

Ivalue - Expressions that refer to a *memory location* are called "lvalue" expressions. An lvalue may appear as either the left-hand or right-hand side of an assignment operator (=).

✓ lvalue often represents as identifier.

lvalue(left value): simply means an object that has an identifiable location in memory (i.e. having an address).

→ In any assignment statement "lvalue" must have the capacity to hold the data.

C Operator Precedence and Associativity XIII

→ lvalue **must be a variable** because they have the capability to store the data.

→ lvalue cannot be a function, expression ($a+b$) or a constant (like 3, 4 etc).

→ rvalue(right value): simply means an object that has no identifiable location in memory.

→ Anything which is capable of returning a constant expression or value.

→ Expression like ($a+b$) will return some constant value.

For example: $a++$; is equivalent to $a = a+1$; here we have both lvalue and rvalue. before $=$, a is lvalue and after $=$, $a+1$ is rvalue.

Take our example $a=b++$; convert it into normal expression $a=b=b + 1$;

C Operator Precedence and Associativity XIV

Take our example $(a=b)++$; convert it into normal expression $(a=b) = (a=b) + 1;$

```
1 int g = 20; // valid statement
2 10 = 20; // invalid statement; would generate compile-time error.

1 // declare a an object of type 'int'
2 int a;
3 // a is an expression referring to an 'int' object as l-value
4 a = 1;
5 int b = a; // Ok, as l-value can appear on right
6 // Switch the operand around '=' operator
7 9 = a;
8 // Compilation error: as assignment is trying to change the value of assignment
   operator
```

C Operator Precedence and Associativity XV

rvalue - The term rvalue refers to a *data value* that is stored at some address in memory.

- ✓ An rvalue is an expression that cannot have a value assigned to it which means an rvalue may appear on the right-hand side but not on the left-hand side of an assignment (=).
- ✓ Variables are lvalues and so they may appear on the left-hand side of an assignment.
- ✓ Numeric literals are rvalues and so they may not be assigned and cannot appear on the left-hand side.

C Operator Precedence and Associativity XVI

```
1 // declare a, b an object of type 'int'
2 int a = 1, b;
3 a + 1 = b; // Error, left expression is  is not variable(a + 1)
4 // declare pointer variable 'p', and 'q'
5 int *p, *q; // *p, *q are lvalue
6 *p = 1; // valid l-value assignment
7 // below is invalid - "p + 2" is not an l-value  p + 2 = 18;
8 q = p + 5; // valid - "p + 5" is an r-value
9 // Below is valid - dereferencing pointer expression gives an l-value
10 *(p + 2) = 18;
11 p = &b;
12 int arr[20]; // arr[12] is an lvalue; equivalent  to *(arr+12)
13 // Note: arr itself is also an lvalue
14 struct S
```

C Operator Precedence and Associativity XVII

```
15 {
16     int m;
17 }
18 ;
20 struct S obj; // obj and obj.m are lvalues
21 // ptr-> is an lvalue; equivalent to (*ptr).m
22 // Note: ptr and *ptr are also lvalues
23 struct S* ptr = &obj;
```

Java Operators Precedence and Associativity I

- ✓ Precedence of operators come into picture when in an expression we need to decide which operator will be evaluated first.
- ✓ Operator with higher precedence will be evaluated first.

```
1 int a=1;
2 int b=4;
3 int c;
4 // expression
5 c= a + b;
6 // Which one is correct
7 (c=a) + b or
8 c = (a+b)
```

Java Operators Precedence and Associativity II

***Larger number means higher precedence.**

Precedence	Operator	Type	Associativity
15	() [] . .	Parentheses Array subscript Member selection	Left to Right
14	++ --	Unary post-increment Unary post-decrement	Left to Right

Java Operators Precedence and Associativity III

13	++ -- + - ! ~ (type)	Unary pre-increment Unary pre-decrement Unary plus Unary minus Unary logical negation Unary bitwise complement Unary type cast	Right to left
12	*	Multiplication	Left to right
11	/ %	Division Modulus	Left to right

Java Operators Precedence and Associativity IV

10	<< >> >>>	Bitwise left shift Bitwise right shift with sign extension Bitwise right shift with zero extension	Left to right
9	< <= > >= instanceof	Relational less than Relational less than or equal Relational greater than Relational greater than or equal Type comparison (objects only)	Left to right
8	== !=	Relational is equal to Relational is not equal to	Left to right
7	&	Bitwise AND	Left to right
6	^	Bitwise exclusive OR	Left to right
5		Bitwise inclusive OR	Left to right

Java Operators Precedence and Associativity V

4	&&	Logical AND	Left to right
3		Logical OR	Left to right
2	? :	Ternary conditional	Right to left
1	= += -= *= /= %=%	Assignment Addition assignment Subtraction assignment Multiplication assignment Division assignment Modulus assignment	Right to left

Java Operators Precedence and Associativity VI

✓ + and * operators. * operator having greater precedence than + operator.

- 1 $2+3*5;$
- 2 $(2+3)*5=25$ or
- 3 $2+(3*5)=17$

Java Operators Precedence and Associativity VII

✓ Associativity of operators come into picture when precedence of operators are same and need to decide which operator will be evaluated first. Associativity can be either left-to-right or right-to-left.

✓ **/ and * operators. Both having same precedence, then associated will come into a picture.**

```
1 10/2*5;  
2 // if left -to-right  
3 (10/2)*5=25  
4 // if right -to-left  
5 10/(2*5)=1
```

Java Operators Precedence and Associativity VIII

- ✓ ()- parenthesis in function calls.
- ✓ Parenthesis() operator having greater precedence than assignment(=) operator.

```
1 int val =fun();  
2 | int | | val | |=| fun| |(| )| |; |  
3  
4 // if suppose = operator is having greater precedence then, fun will belong to =  
   operator and therefore it will be treated as a variable .  
5 int ( val = fun)();  
6  
7 // = operator is having less less precedence as compared to () therefore , ()  
   belongs to fun and will be treated as a function .  
8 int val = (fun());
```

Java Operators Precedence and Associativity IX

```
1 //Which function will be called first.
2 int main()
3 {
4     int a;
5     a = MCA() + MSC();
6     printf("\n%d",a);
7     return 0;
8 }
9
10 int MCA()
11 {
12     printf("MCA");
13     return 1;
14 }
```

Java Operators Precedence and Associativity X

```
15 int MSC()
16 {
17     printf("MSC");
18     return 1;
19 }
20 -----
21 Output: ???
22
23 Answer: MCAMSC2 or MSCMCA2.
24 // It is not defined whether MCA() will be called first or whether MSC() will be
25 // called. Behaviour is undefined and output is compiler dependent.
26
```

Java Operators Precedence and Associativity XI

²⁷ //Here associativity will not come into picture as we have just one operator and which function will be called first is undefined. Associativity will only work when we have more than one operators of same precedence.

Java Operators Precedence and Associativity XII

► Increment ++ and Decrement -- Operator as Prefix and Postfix

- ✓ Precedence of Postfix increment/Decrement operator is greater than Prefix increment/Decrement.
- ✓ Associativity of Postfix is also different from Prefix. Associativity of postfix operators is from left-to-right and that of prefix operators is from right-to-left.
- ✓ Operators with some precedence have same associativity as well.

Java Operators Precedence and Associativity XIII

- If you use the `++` operator as prefix like: `++var`. The value of var is incremented by 1 then, it returns the value. or means first increment then assign it to another variable.
- If you use the `++` operator as postfix like: `var++`. The original value of var is returned first then, var is incremented by 1. or means first assign it to another variable then increment.
- The `--` operator works in a similar way like the `++` operator except it decreases the value by 1.

Java Operators Precedence and Associativity XIV

- ✓ you cannot use rvalue before or after increment/decrement operator.

Example:

(a+b)++; Error

++(a+b); Error.

Error: lvalue required as increment operator(compiler is expecting a variable as an increment operand but we are providing an expression (a+b) which does not have the capability to store data). Because (a+b) is rvalue. (a+b) is an expression or you can say it is value not an operator.

Java Operators Precedence and Associativity XV

```
1 int main()
2 {
3     int x = 1;
4     int y=0;
5     x=y++;
6     // (x=y) = (x=y) +1;
7     scanf("%d", &y);
8     printf("%d\n%d", x, y);
9     // return 0;
10 }
```

- ✓ Unary operator must be associated with a valid operand.

Java Operators Precedence and Associativity XVI

```
1 public class Precedence
2 {
3     public static void main(String [] args)
4     {
5         int a = 10, b = 5, c = 1;
6         System.out.println (a+++b);
7         System.out.println (a +++ b);
8         System.out.println (a++ + b);
9         System.out.println (a +++b);
10        System.out.println (a + ++b);
11        System.out.println (a+ ++b);
12    }
13 }
```

Java Operators Precedence and Associativity XVII

```
1 a+++b;  
2 // Valid tokens in line number 5:  
3 |a| |++| |+| |b| |; |  
4  
5 // Make valid syntax for post-increment and pre-increment  
6 // Unary operator must be associated with a valid operand.  
7 // ++ will be associated with a  
8 a++  
9 +  
10 b  
11 -----  
12 a++ + b;
```

Java Operators Precedence and Associativity XVIII

```
1 public class Precedence
2 {
3     public static void main(String [] args)
4     {
5         int a = 10, b = 5, c = 1, result ;
6         result = a--++c---b;
7         System.out. println ( result );
8     }
9 }
```

Java Operators Precedence and Associativity XIX

```
1 result = a-++c-++b;  
2 // Valid tokens in line number 7:  
3 | result |, |=|, |a|, |-|, |++|, |--|, |++| and |b|  
4  
5 // Make valid syntax for post-increment
```

```
1 public class Precedence  
2 {  
3     public static void main(String [] args)  
4     {  
5         int a = 10, b = 15, c = 20, d=25;  
6         // int a = 17, b = 15, c = 20, d=25;  
7         if(a<= b == d > c)  
8     }
```

Java Operators Precedence and Associativity XX

```
9      System.out.println ("TRUE");
10 }
11 else
12 {
13     System.out.println ("FALSE");
14 }
15 }
16 }
```

Java Operators Precedence and Associativity XXI

```
1 |a| |<| |=| |b| |=| |=| |d| |>| |c|
2 OR
3 |a| |<=| |b| |==| |d| |>| |c|
4
5 |<=| --> Precedence 9
6 |==| --> Precedence 8
7 |>| --> Precedence 9
8
9 ((a<=b) == (d>c))
10 (1 == 1)
11
```