CS204:Object Oriented Programming Concepts

Sachchida Nand Chaurasia

Assistant Professor

Department of Computer Science

Banaras Hindu University

Varanasi

Email id: snchaurasia@bhu.ac.in, sachchidanand.mca07@gmail.com

January 21, 2022

Java Operator I

✓ Operators are symbols that perform operations on variables and values.

✓ For example, + is an operator used for addition, while * is also an operator used for multiplication.

► Operators in Java can be classified into 6 types:

1. Arithmetic Operators
2. Assignment Operators
3. Relational Operators
4. Logical Operators

Java Operator II

5. Unary Operators
6. Bitwise Operators

Java Operator III

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)

Example:

Java Operator IV

```
a + b;
a - b;
a * b;
a / b;
a % b;
```

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

Java Operator V

```
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 VI

Output:

```
a + b = 17
a - b = 7
a * b = 60
a / b = 2
a % b = 2
```

Java Operator VII

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 VIII

```
class Main
{
    public static void main(String[] args)
    {
        // create variables
        int a = 4;
        int var;
        // assign value using =
        var = a;
        System.out.println("var using =: " + var);
        // assign value using =+
        var += a;
        System.out.println("var using +=: " + var);
```

Java Operator IX

```
        // assign value using =*
        var *= a;
        System.out.println("var using *=: " + var);
    }
}
```

Output:

```
var using =: 4
var using +=: 8
var using *=: 32
```

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

Java Operator X

```java
// check is a is less than b
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 XI

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

Java Operator XII

```
public class Main
{
    public static void main(String[] args)
    {
        // create variables
        int a = 7, b = 11;
        // value of a and b
        System.out.println("a is " + a + " and b is " + b);
        // == operator
        System.out.println(a == b); // false
        // != operator
        System.out.println(a != b); // true
        // > operator
```

Java Operator XIII

```
        System.out.println(a > b); // false
        // < operator
        System.out.println(a < b); // true
        // >= operator
        System.out.println(a >= b); // false
        // <= operator
        System.out.println(a <= b); // true
    }
}
```

Java Operator XIV

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 XV

```java
public class Main
{
    public static void main(String[] args)
    {
        // && operator
        System.out.println((5 > 3) && (8 > 5)); // true
        System.out.println((5 > 3) && (8 < 5)); // false
        // || operator
        System.out.println((5 < 3) || (8 > 5)); // true
        System.out.println((5 > 3) || (8 < 5)); // true
        System.out.println((5 < 3) || (8 < 5)); // false
        // ! operator
        System.out.println!(5 == 3)); // true
```

Java Operator XVI

```
        System.out.println(!(5 > 3)); // false
    }
}
```

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.

Java Operator XVII

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

```java
int num = 5;
// increase num by 1
++num;
```

Java Operator XVIII

```java
public class Main
{
    public static void main(String[] args)
    {
        // declare variables
        int a = 12, b = 12;
        int result1, result2;
        // original value
        System.out.println("Value of a: " + a);
        // increment operator
        result1 = ++a;
        System.out.println("After increment: " + result1);
        System.out.println("Value of b: " + b);
```

Java Operator XIX

```
        // decrement operator
        result2 = - -b;
        System.out.println("After decrement: " + result2);
    }
}
```

Output:

```
Value of a: 12
After increment: 13
Value of b: 12
After decrement: 11
```

Java Operator XX

```java
public class Operator
{
    public static void main(String[] args)
    {
        int var1 = 5, var2 = 5;
        // var1 is displayed
        // Then, var1 is increased to 6.
        System.out.println(var1++);
        // var2 is increased to 6
        // Then, var2 is displayed
        System.out.println(++var2);
    }
}
```

Java Operator XXI

Output: ?

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

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

Java Operator XXII

Operator	Description
~	Bitwise Complement
«	Left Shift
»	Right Shift
»>	Unsigned Right Shift
&	Bitwise AND
^	Bitwise exclusive OR

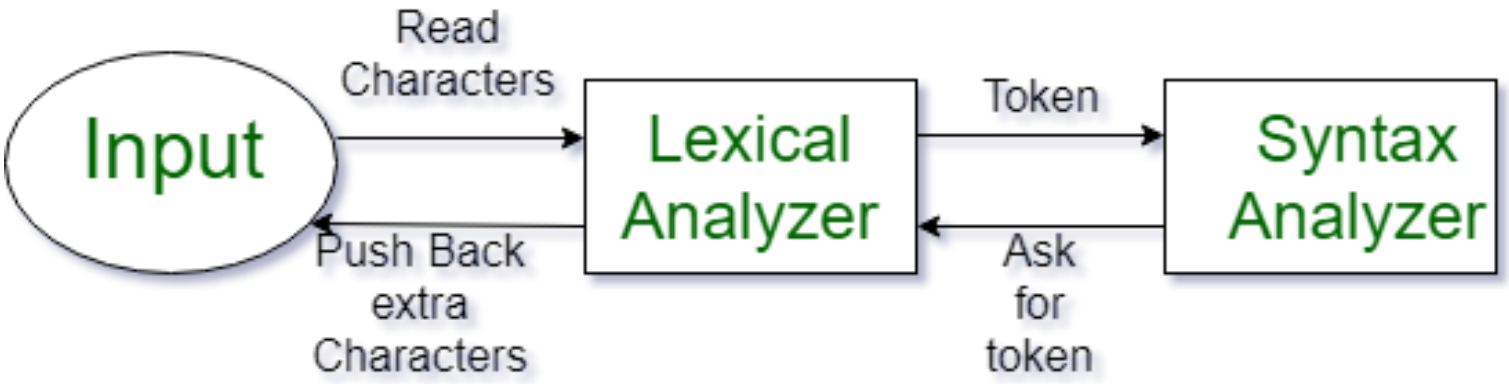
Introduction of Lexical Analyzer:

✓ 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:

- Type token (id, number, real, . . .)
- Punctuation tokens (IF, void, return, . . .)
- Alphabetic tokens (keywords)
- Keywords; Examples-for, while, if etc.

C Operator Precedence and Associativity IV

- Identifier; Examples-Variable name, function name, etc.
- Operators; Examples '+', '++', '-' etc.
- Separators; Examples ',' ';' etc

Example of Non-Tokens:

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

C Operator Precedence and Associativity V

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.

C Operator Precedence and Associativity VI

4. Remove comments.
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–

C Operator Precedence and Associativity VII

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

For example, consider the program:

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

Another example:

C Operator Precedence and Associativity VIII

```
int main()
{
      // 2 variables
      int a, b;
      a = 10;
     return 0;
}
--------------------------
Tokens:
int
main
(
)
```

C Operator Precedence and Associativity IX

```
{
    int
    a
    ,
    b
    ;
    a
    =
    10
    ;
    return
    0
    ;
```

C Operator Precedence and Associativity X

```
}
```

```
//How many tokens?

printf("BSCPMKSMK");
```

```
int main()
{
      int a = 10, b = 20;
      printf("sum is :%d",a+b);
      return 0;
}
-----------
//How many tokens?
```

C Operator Precedence and Associativity XI

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

C Operator Precedence and Associativity XII

- 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:

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

C Operator Precedence and Associativity XIII

```
 //Count number of tokens :

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

✓ Identification valid a token.

```
inta=10;
int a=10;
----------------------
//Number of valid tokens: ????
```

Lvalues and Rvalues in C:

There are two kinds of expressions in C -

lvalue - 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).

C Operator Precedence and Associativity XV

–> In any assignment statement "lvalue" must have the capacity to hold the data

–> 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.

C Operator Precedence and Associativity XVI

–> 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;

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

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

C Operator Precedence and Associativity XVII

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

C Operator Precedence and Associativity XVIII

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 XIX

```
// declare a, b an object of type 'int'
int a = 1, b;
a + 1 = b; // Error, left expression is  is not variable(a + 1)
// declare pointer variable 'p', and 'q'
int *p, *q; // *p, *q are lvalue
*p = 1; // valid l-value assignment
// below is invalid - "p + 2" is not an l-value  p + 2 = 18;
q = p + 5; // valid - "p + 5" is an r-value
// Below is valid - dereferencing pointer expression gives an l-value
*(p + 2) = 18;
p = &b;
int arr[20]; // arr[12] is an lvalue; equivalent  to *(arr+12)
```

C Operator Precedence and Associativity XX

```
// Note: arr itself is also an lvalue
struct S
{
     int m;

}
;
struct S obj; // obj and obj.m are lvalues
// ptr-> is an lvalue; equivalent to (*ptr).m
// Note: ptr and *ptr are also lvalues
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.

```
int a=1;
int b=4;
int c;
//expression
c= a + b;
// Which one is correct
(c=a) + b or
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 Division Modulus	Left to right
11	+ -	Addition Subtraction	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

Java Operators Precedence and Associativity V

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

a+=b
a+=b
a*=b
a=a+b
a=a+b

Java Operators Precedence and Associativity VI

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

```
2+3*5;
(2+3)*5=25 or
2+(3*5)=17
```

2+3*5

2+(3*5)

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.**

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

10/2*5

/ *

x -

10*2/5

10/2*5

10*2/5

Java Operators Precedence and Associativity VIII

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

```
int val = fun();
|int| |val| |=| |fun| |(| |)| |; |

// if suppose = operator is having greater precedence then,
    fun will belong to =
    operator and therefore it will be treated as a variable.
int (val = fun)();
```

val = fun()

int (val = fun)()

Java Operators Precedence and Associativity IX

```
// = operator is having less less precedence as compared to ()
    therefore, ()
    belongs to fun and will be treated as a function.
int val = (fun());
```

```
//Which function will be called first.
int main()
{
    int a;
    a = MCA() + MSC();
    printf("\n%d",a);
    return 0;
}
```

Java Operators Precedence and Associativity X

```
int MCA()
{
    printf("MCA");
    return 1;
}

int MSC()
{
    printf("MSC");
    return 1;
}
---------------------
Output: ???
```

Java Operators Precedence and Associativity XI

```
23
24 Answer: MCAMSC2 or MSCMCA2.
25 // It is not defined whether MCA() will be called first or
       whether MSC() will be called. Behaviour is undefined and
       output is complier dependent.
26
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.
```

▸ 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.

Java Operators Precedence and Associativity XIV

- The - - operator works in a similar way like the ++ operator except it decreases the value by 1.

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

Example:

(a+b)++; Error ✓

++(a+b); Error. ✓

(a+b)++

a+b = a+b+1

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

Java Operators Precedence and Associativity XV

capability to store data). Because (a+b) is rvalue. (a+b) is an expression or you can say it is value not an operator.

```
int main()
{
        int x = 1;
        int y=0;
       x=y++;
      // (x=y) = (x=y) +1;
       scanf("%d",&y);
        printf("%d\n%d",x,y);
      // return 0;
}
```

x = 0
y = 1

x = y++
x = (y++)

x = (y=y+1)
x = y = (y+1)

Java Operators Precedence and Associativity XVI

✓ Unary operator must be associated with a valid operand.

```
public class Precedence
{
    public static void main(String[] args)
    {
        int a = 10, b = 5, c = 1;
        System.out.println(a+++b);
        System.out.println(a +++ b);
        System.out.println(a++ + b);
        System.out.println(a +++b);
        System.out.println(a + ++b);
        System.out.println(a+ ++b);
    }
```

a++ -

a=a+1+b) 15 11 5

b 5

2 5

14 5

14 6

21 14 7

Java Operators Precedence and Associativity XVII

```
13 }
```

Java Operators Precedence and Associativity XVIII

```
a+++b;
//Valid tokens in line number 5:
|a| |++| |+| |b| |; |

//Make valid syntax for post-increment and pre-increment
// Unary operator must be associated with a valid operand.
//++ will be associated with a
a++
+
b
--------------
a++ + b;
```

Java Operators Precedence and Associativity XIX

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

B. O.
=
a
-
++
c
-
++
b

10 - 8 = 2

r = a - (++c) - (++b)

r = 10 - 2 - 6

r = a - ++c - ++b

Java Operators Precedence and Associativity XX

```
result = a-++c-++b;
//Valid tokens in line number 7:
|result|, |=|, |a|, |-|, |++|, |-|, |++| and |b|

//Make valid syntax for post-increment
```

```
public class Precedence
{
    public static void main(String[] args)
    {
        int a = 10, b = 15, c = 20, d=25;
        //int a = 17, b = 15, c = 20, d=25;
        if(a<= b == d > c)
```

a<=b==d>c

a<=b==(d>c)

True

True

a <=

b ==

d

Java Operators Precedence and Associativity XXI

```
        {
            System.out.println("TRUE");
        }
        else
        {
            System.out.println("FALSE");
        }
    }
}
```

a<=b

Java Operators Precedence and Associativity XXII

```
|a| |<| |=| |b| |=| |=| |d| |>| |c|
OR
|a| |<=| |b| |==| |d| |>| |c|

|<=| --> Precedence 9
|==| --> Precedence 8
|>| --> Precedence 9

((a<=b) == (d>c))
(1 == 1)
```