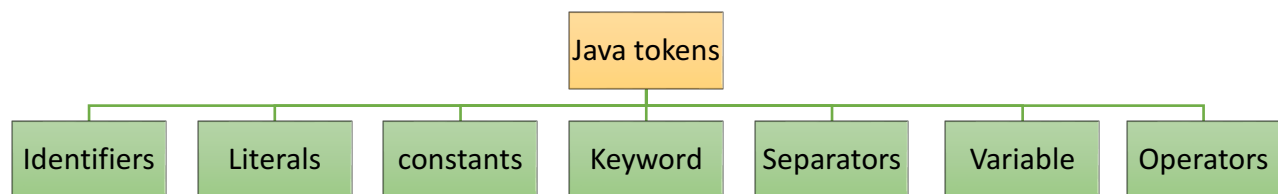


## ❖ Java Tokens

- The smallest individual units in a program are known as **Tokens**.
- A Java program is basically a collection of classes. Java language has seven types of tokens.
- A token has a specific meaning interpreted by the compiler based on the context of its appearance. It may be a single character or a group of characters.
- Tokens supported in Java include keywords, variables, constants, special characters, operator, identifier, separators and literals.



### Dig - Java Token

#### 11.1. Identifier :

Identifier are programmer - designed tokens. They are used for naming classes, methods, variables, objects, labels, packages, interfaces in a program.

An identifier follow the following rules :

1. They can have alphabets, digits and underscore (\_) and dollar (\$) sign characters.
2. They must not begin with a digit.
3. Uppercase and lowercase letters are distinct.
4. They can be any length.

## 11.2. Literals :

*Literals in java are a sequence of character (digits, letters and other characters) that represent constant values to be store in variable.*

- *Java specifies five major types of literals*

- 1. Integer Literals.*
- 2. Integer Literals.*
- 3. Floating point literals.*
- 4. Character Literals.*
- 5. String Literals.*
- 6. Boolean Literals.*

## 11.3. Constant :

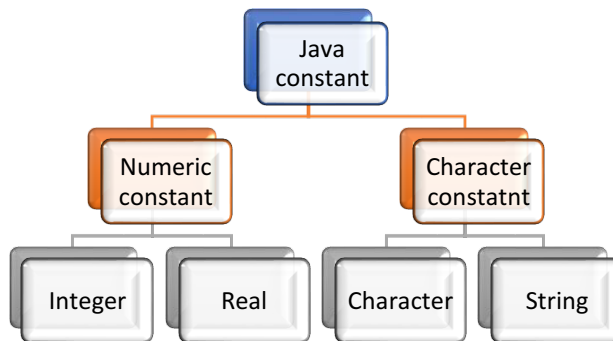
*Constants in java refers to the fixed values that do not change during the execution of program*

- *A constant in Java is used to map an exact and unchanging value to a variable name.*
- *Constants are used in programming to make code a bit more robust and human readable.*
- *Constant in Java can be declared by using the keyword final. The value of a constant cannot change throughout the program.*
- *The keyword final indicates that once you assign some value to a variable, the value does not change throughout the program.*
- *In Java, by convention, constants are variables generally declared in uppercase only.*

*For example: final int MAX=100;*

- *In the above example, MAX is an integer constant and its fixed value is 100.*

- Java supports several types of constants. They are



**1. Integer constant :** Integer constant refers to sequence of numbers without decimal point. There are three types of integer constants

(i) *Decimal integers:* They consist of combination digits 0 through 9 with or without negative sign.

Example: 10 5 -245 0

(ii) *Octal integer:* They consist of combination of digits 0 through 7 with leading zero.

Example: 074 0 034

(iii) *Hexadecimal integer:* They consist of combination of digits 0 through 9 and letter a through f or A through F with leading 0x or 0X.

Example: 0X7A4 0Xa 0xA4

**2. Real constants:** Real constant refers to sequence of numbers with decimal point. They can be represented in two forms decimal notation and exponential notation.

(i) *Decimal notation:* The decimal notation has three parts the integer part, the decimal point and the fractional part and can be used to represent single precision numbers.

Example: 2.15 -7.8 0.78

(ii) *Exponential notation: The exponential notation is used to represent double precision numbers*

*Example: 0.97e4 12e-2 -1.2E-4*

**3. Character constant:** *A single character enclosed within single quotes is called character constant.*

*Example: '5' 'E' '\$'*

**4. String Constants:** *A sequence of characters enclosed within double quotes is called string constant.*

*Example: "Hii" "Java programming" "2021"*

**5. Back slash character constant:**

*They are special character constants that are used in output methods.*

*Example:*

*'\b' - Back space.*

*'\n' - New line.*

*'\t' - Horizontal.*

*'\r' - Carriage return.*

## **11.4. Keyword:**

- *Keywords are also known as reserved words, whose meaning is already begin defined into java libraries.*
- *Keyword cannot be used as variable names as they have special meaning into the java class libraries.*

If	switch	Else	case
Continue	default	For	break
While	do	Short	byte
Int	long	Float	double
Boolean	char	Try	catch
Throw	throws	Finally	class
Import	package	Interface	extends
Implements	void	Return	new
This	super	Instanceof	enum
Assert	goto	Const	public
private	protected	Abstract	final
Synchronize	volatile	Native	transient
Strictfp	null	True	false
Static			

## 11.5. Separators :

*Separators are symbols used to indicate where groups of code are divided and arranged the basically define the shape and function of our code.*

*1. **Paranthesis ( )** :- It is used to enclose parameter in method definition and invocation, also used for defining precedence in expression, containing expression for flow control surrounding.*

*2. **Braces { }** :- Used to contain the values of automatically initialized arrays and to define a block of code for classes methods and local scopes*

*3. **Brackets [ ]** :- Used to declare array types and for dereferencing array values.*

4. *Semicolon (;) :- Used to separate statement .*

5. *Comma (,) :- Used to separate consecutive identifies in a variable declaration also used chain statements together inside a for statement.*

6. *Period (.) :- Used to separate packages and classes also used to separate a variable or method from a reference variable.*

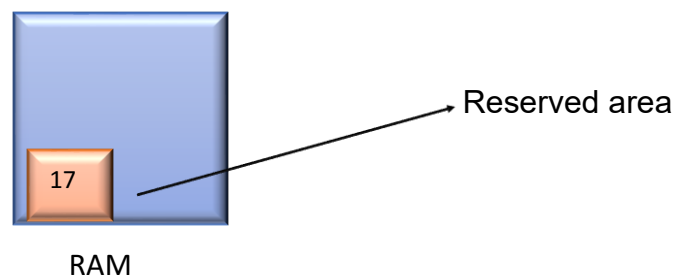
## 11.6. Variable :

- *Variables are nothing but reserved memory locations to store values. This means that when we create a variable, we reserve some space in memory.*

- *Variables are nothing but reserved memory locations to store values. This means that when we create a variable, we reserve some space in memory.*

- *A variable is a name of location where the data is stored when a program executes, (See Diagram).*

*For example: int data=17; //Here data is variable and 17 is value.*



*Dig : Variable*

- *Three types of Variable :*

*i] Local Variable.*

*ii] Static Variable.*

*iii] Non-Static Variable / Instance Variable.*

### **11.6.1 Rules of Declaring Variable :**

- 1. A variable name must begin with a letter*
- 2. They must not begin with a digit.*
- 3. Uppercase and lowercase are distinct this means that the variable total as not same as TOTAL.*
- 4. White space is not allowed.*
- 5. Underscore (\_) are allowed.*
- 6. Variable name are Case-Sensitive.*

### **11.6.2 Declaration of Variable :**

- *A **variable** is an identifier that denotes a storage location used to store a data value.*
- *A variable declaration is a statement that reserves a named memory location.*
- *In Java, all variables must be declared before they can be used.*
- *The basic form (syntax) of a variable declaration is shown here.*

*Syntax :*

***datatype variable\_name;***

- *The datatype determines what values it can hold and what operations can be performed on it.*

*See the following example for variables declaration:*

1] `int num;` //represents that num is a variable that can store value of int type.

2] `String name;` //represents that name is a variable that can store string value.

3] `boolean light` //represents that bol is a variable that can take Boolean value (true/false);

### 11.6.3. Initialization of Variable :

- An initialization is an assignment made when you declare a variable.
- You can assign a value to a variable at the declaration time by using an assignment operator (=).
- The assignment operator is the equal sign (=); any value to the right of the equal sign is assigned to the variable on the left of the equal sign.

For example :

1] `int Roll_No= 10;` // This line declares Roll\_No as an int variable which holds value "10".

2] `boolean bol = true;` // This line declares bol as boolean variable which is set to the value "true".

### 11.6.4. Scope of Variables :

- Scope refers to the lifetime and accessibility of a variable.
- The scope of a variable defines the section of the code in which the variable is accessible or visible.
- The lifetime of a variable refers to how long the variable exists before it is destroyed.



## 11.6.5. Types of Variables

### 7. Local Variables

*If any variable is defined inside the method will be treated as local variable.*

**Syntax:**

```
return_type method_Name()
{
    Datatype var = value;    // local variable
    -----
    -----
}
```

*Local variable must be initialized every formal parameter by default will be treated as **Local variable***

**Syntax:**

```
return_type method_Name(datatype var)
{
    -----
    (formal parameter can handle
    dynamic values)
}
```

- *The life of local variable is within the same method is allocated for the local variable at the time of calling the method and memory is deallocated once the comes outside the method.*
- *The scope of the variable is within the same method i.e., whose values can be accessed inside the same method only.*
- *In real time it is highly re-commented to define local variable if that wants to be used inside a single method of a class.*

- *For example :*

```
public class Local_Variable
{
    public static void main(String[] args)
    {
        int a=10;
        System.out.println(a);
        int _b=20;
        System.out.println(_b);
        int $c=30;
        System.out.println($c);
        int d2=40;
        System.out.println(d2);
    }
}
```

### **8. Static Variables**

- *If any variable defines inside the class and outside the methods using static keyword is known as **Static variable**.*
- *The life of static variable define is until the **class** is available in the memory that means for this variable memory allocated at the time of loading of class in the main memory and memory is deallocated once the class is unloaded from the main memory.*

#### **Note -**



*If real time class is anywhere in the class.*

*The scope of the static variable is anywhere in the class.*

**Syntax -**

```
class ClassName
{
    Static datatype variable; // Static variable
    Void f1(){
        -----
        ----- }
    Void f2(){
        -----
        -----}
}
```

*In the above syntax static variable can be declare or initialize but it is re-commented to initialize.*

**Note -**



*In real time it highly re-commented to static variables whenever a single memory copy wants to be maintained for a class.*

*Single memory copy is required we want to handle common data for every end user.*

⇒ *For static variable memory is allocated only once at the time loading of class even multiple objects are created for that class.*

⇒ *Static variables can be accessed with following three ways:*

*(a) with class name.*

*(b) with reference variable which contains null.*

*(c) with reference variable which contains object.*

*for Example*

1. Use Static variable & write java program.



```
public class Emp
{
    private static double salary;
    // salary variable is a private static variable
    public static final String DEPARTMENT = "Monthly";
    // DEPARTMENT is a constant
    public static void main(String args[])
    {
        salary = 30000;
        System.out.println(DEPARTMENT+" average salary:" +salary);
    }
}
```

2. Static variable example.



```
public class Static_Variable{
    static int a=4; // static variable
    int d=10;
    public static void main(String[] args){
        Static_Variable ref =new Static_Variable();
        System.out.println(ref.b);
        System.out.println(Static_Variable.a);
        System.out.println(ref.d);
    }
}
```

## 2. Non-Static Variables / Instance variable

- If any variable defines inside the class and outside the methods without static keyword is known as **Non-Static variable**.
- The life of static variable define is until the **object** is available that means for non-static variable memory allocated whenever object is created and memory is deallocated once the object is destroy.
- The scope of non-static variable is anywhere the class.
- Instance variables can also be called as non-static variables.

- **Syntax -**

```
class ClassName
{
    datatype variable; // Non-Static variable
    Void f1(){
        .....
        ..... }
    Void f2(){
        .....
        .....}
}
```

**Note -**

*In real time it highly re-commented to static variables whenever it separate memory copy wants to be maintain for every new object.*

- *For non-static variable memory is allocated multiple time for every new object.*
- **For Example**

1] **Write example of Non-Static variable.**



```
public class Static_Variable
{
    int b=40; // non- static variable
    int d=10;
    public static void main(String[] args)
    {
        Static_Variable ref =new Static_Variable();
        System.out.println(ref.b);
        System.out.println(ref.d);
    }
}
```

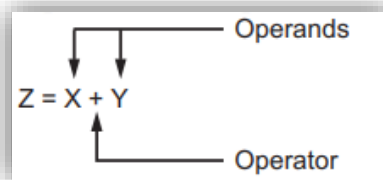
## 2] Create java program to use Non- static variable.



```
public class Employee
{
    // this instance variable is visible for any child class.
    public String name;
    // salary variable is visible in Employee class only.
    private double salary;
    // The name variable is assigned in the constructor.
    public Employee (String empName)
    {
        name = empName;
    }
    // The salary variable is assigned a value.
    public void setSalary(double empSal) {
        salary = empSal;
    }
    // This method prints the employee details.
    public void printEmp()
    {
        System.out.println("Name: " + name );
        System.out.println("Salary:" + salary);
    }
    public static void main(String args[])
    {
        Employee empOne = new Employee("Satish");
        empOne.setSalary(30000);
        empOne.printEmp();
    }
}
```

## 11.7. Operators :

- *An operator is a symbol that takes one or more arguments and operators on them to produce a result.*
- *Operators are used in programs to manipulate data and variable.*
- *The data items that the operators act upon are called operands. An operator is a symbol or a keyword that specifies an operation to be performed.*



### Dig- Concept of operands and operators

- If an operator acts on a single variable, it is called unary operator, if an operator on two variables, it is called binary operator, and if an operator acts on three variables, then it is called ternary operator.

#### ▪ 11.7.1. Types of Operator :

##### 1. Arithmetic Operators :

- Arithmetic operators can operate on any built-in numeric data type of java. We cannot use these operators on Boolean types
- Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra.
- Arithmetic operators are used to perform some mathematical operations like addition, subtraction, multiplication, division, and modulo (or remainder). These are generally performed on an expression or operands.
- According to number of operands required for an operator, they are classified as Unary (one operand), binary (two operands) and ternary (three operands) operators.

- *Java has five operators*

Operator	Description
<b>+</b> (Addition)	Adds values on either side of the operator.
<b>-</b> (Subtraction)	Subtracts right hand operand from left hand operand.
<b>*</b> (Multiplication)	Multiplies values on either side of the operator
<b>/</b> (Division)	Divides left hand operand by right hand operand.
<b>%</b> (Modulus)	Divides left hand operand by right hand operand and returns remainder.

*For Example* ➔

1. **Add the two number.**



```
class Addition
{
    public static void main (String []args)
    {
        int a=7,b=10,c;
        c=a+b;
        System.out.println("Add the a+b:"+c);
    }
}
```

2. **Write a java program to use a Arithmetic operators.**



```
public class Arithmetic_Operators
{
    public static void main(String[] args)
    {
        int s=10, d=7;
        System.out.println("s:"+s);
        System.out.println("d:"+d);
        System.out.println("s+d:"+s+d);
        System.out.println("s-d:"+s-d);
        System.out.println("s*d:"+s*d);
        System.out.println("s/d:"+s/d);
        System.out.println("s%d:"+s%d);
    }
}
```



## 2. Relational Operators :

• When we want to compare values of two variables for various means, relational operators are used. After comparing these values, we may take several decisions in the program.

• Relational operators can be called as comparison operators.

Operator	Meaning	Explanation	Example	Result
==	Equal to	This operator is used for check equality	17 == 17	True
!=	Not equal to	This operator is used for check inequality	10 != 10	False
>	Greater than	This operator is used for check greater value.	17 > 10	False
<	Less than	This operator is used for check less value.	7 < 10	True
>=	Greater than equal to	This operator is used for check greater as well as equality	17 >= 10	True
<=	Less than equal to	This operator is used for check less as well as equality.	17 <= 10	False

For Example ➔

### 1. Program for implementation of Relational operator.

⇒

```
public class Relational_Operator
{
    public static void main(String[] args)
    {
        int d=7, s=10, h=17;
        System.out.println("*** Greater than number (>) ***");
        System.out.println("d > s:"+(d>s));
        System.out.println("s > d:"+(s>d));
    }
}
```

```
        System.out.println("s > h:"+(s>h));
        System.out.println("h > d:"+(h>d));
        System.out.println();
    System.out.println("***Greater than equal to number (>=)***");
        System.out.println("d >= s:"+(d>=s));
        System.out.println("s >= d:"+(s>=d));
        System.out.println("s >= h:"+(s>=h));
        System.out.println("h >= d:"+(h>=d));
        System.out.println("");
    System.out.println("*** Less than number (<)***");
        System.out.println("d < s:"+(d<s));
        System.out.println("s < d:"+(s<d));
        System.out.println("s < h:"+(s<h));
        System.out.println("h < d:"+(h<d));
        System.out.println("");
    System.out.println("***Less than equal to number (<=)***");
        System.out.println("d <= s:"+(d<=s));
        System.out.println("s <= d:"+(s<=d));
        System.out.println("s <= h:"+(s<=h));
        System.out.println("h <= d:"+(h<=d));
        System.out.println("");
    System.out.println("***Equal to (==)***");
        System.out.println("d == s:"+(d==s));
        System.out.println("s == h :"+(s==h));
        System.out.println("h == d:"+(h==d));
        System.out.println("");
    System.out.println("***Not equal to (!=)***");
        System.out.println("d != s:"+(d!=s));
        System.out.println("s != h:"+(s!=h));
        System.out.println("h != d:"+(h!=d));
    }
}
```

#### Note →



*Relational operators are mostly used in decision making statements such as if, while, do-while and for to decide the course of action of running program.*

### 3. Logical Operators :

- Logical operators are used to construct compound conditions. A compound condition is a combination of several simple conditions.

Operator	Meaning	Example	Explanation
&&	And operator	If(a>b && a>c)  System.out.print("yes");	If a value is greater than b and c then only yes is displayed.
	OR operator	If(a==1    b==1)  System.out.print("yes");	If either a value is 1 or b value is 1 then yes is displayed.
!	Not operator	If(!(a==0) )  System.out.print("yes");	If a value is not equal to zero then only yes is displayed.

For example →

### 1. program demonstrating Logical Operators by using boolean data type.

⇒

```
public class Logical_Operator_1
{
    public static void main(String[] args)
    {
        boolean A = true ;
        boolean B = false ;
        System.out.println("A && B:" + (A && B));
        System.out.println("A || B : " + (A || B));
        System.out.println("!B:" + (!B));
        System.out.println("!A:" + (!A));
    }
}
```

```
        System.out.println("A^B:"+(A^B));
        System.out.println("A|B:"+(A|B));
    }
}
```

## 2. Example of Logical expression

⇒

```
public class Logical_Operator_2 //class Emp
{
    public static void main (String[] args)
    {
        String name= "Deepanjali";
        int age =24;
        int salary = 30000;
        if (age <= 30 && salary >=20000 )
        {
            System.out.println("Qualified");
        }
        else
        {
            System.out.println("Not Qualified");
        }
    }
}
```

## 4. Assignment Operators (Java shorthand operator):

- *Assignment operators are used to assign the value of an expression to a variable.*
- *This operator (=) is used to store some value into a variable.*
- *The assignment operator in java is also known as **Java shorthand operator**.*
- *Syntax =*

*variablename = expression*

*Or*

*variablename op = expression*

- **Example =**

a= a+b =a+=b

a= a-b=a-=b

**Note =>**



*1·Variable name is variable·*

*2·Expression is a valid java expression·*

*3·Op is any java operator*

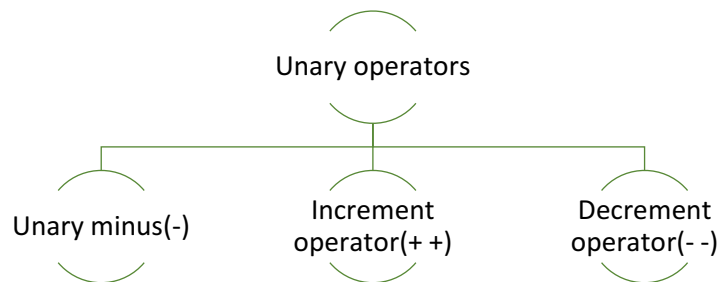
**For Example =>**

**1. program demonstrating use of java shorthand/Assignment operator**

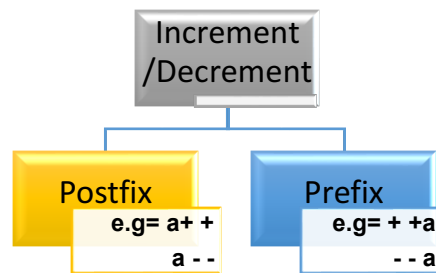


```
public class Assignment_Operator
{
    public static void main(String[] args)
    {
        int a=10, b=7,c=17,d=4;
        a+=b;
        System.out.println("a:"+a);
        b-=c;
        System.out.println("b:"+b);
        c%=2;
        System.out.println("c:"+c);
        d+= a*b;
        System.out.println("d:"+d);
    }
}
```

## 7. Decrement and Increment Operators(Unary Operator) :



### • Increment and Decrement operator




### Note →



- i)  $a++$  or  $++a$  means add the value 1 to variable  $a$ .
- ii)  $a--$  or  $--a$  means subtract the value 1 to variable.

### i) Postfix :

- In this first return value to a variable and then Increment / Decrement.
- Postfix example :

$a = 6$                        $b = a++$   
  
 return value

Output :  $a = 7$                        $b = 6$

**Note →**

*In postfix, first return the value of a to b and then increment the value of a.*

**ii) Prefix :**

- In prefix style, first Increment/ decrement the value and then return the value to the variable.*
- Prefix example :*

i)	a = 5	b = ++ a
Output =	a = 6	b = 6
ii)	a = 10	b = -- a
Output =	a = 9	b = 9

**For Example :****1. program increment & decrement Operator in expression**

```
public class Increment_Decrement_1
{
    public static void main (String []args)
    {
        int a=5,b=10,c=0;
        c= ++a +b;
        System.out.println("a:"+a);
        System.out.println("b:"+b);
        System.out.println("c:"+c);
    }
}
```

## 2. Example for increment & decrement Operator.



```
public class Increment_Decrement {  
  
    public static void main(String[] args) {  
        int a=1,b=2,c,d,e,f;  
        a++;  
        b++;  
        c=++b;  
        d=c++;  
        e=--d;  
        f=e--;  
        System.out.println("a:"+a);  
        System.out.println("b:"+b);  
        System.out.println("c:"+c);  
        System.out.println("d:"+d);  
        System.out.println("e:"+e);  
        System.out.println("f:"+f);  
    }  
}
```

## 2. Ternary Operator or Conditional Operators ( ? : )

- *This operator is called ternary because it acts on 3 variables*

*The syntax for this operator is:*

*Variable = Expression1? Expression2: Expression3;*

- *First Expression1 is evaluated. If it is true, then Expression2 value is stored into variable; otherwise Expression3 value is stored into the variable.*

- *Example : a = 10 ; b = 15;*

*X = (a > b) ? a : b;*

*Output :- x = 15*

*In this example, X will be assigned the value of b.*

*This can be archived in the following form :*

*If (a > b)*

*X = a ;*



```

else
    X = b ;

```

➤ For example ->

### 1. Use of conditional operators find the maximum of 2 numbers.

⇒

```

public class Conditional_Operator_Max
{
    public static void main(String[] args)
    {
        int a = 10, b = 5, ans;
        ans = (a > b) ? a : b;
        System.out.println("Maximum of two no is: " + ans);
    }
}

```

### 2. Use of Conditional operators & find minimum of two numbers

⇒

```

import java.util.Scanner;
public class Conditional_Operator_Min
{
    public static void main(String[] args)
    {
        int a, b, ans;
        Scanner sn = new Scanner(System.in);
        a = sn.nextInt();
        b = sn.nextInt();
        ans = (a < b) ? a : b;
        System.out.println("Minimum of two number is: " + ans);
    }
}

```

### 6. Instanceof Operator :

- The instanceof operator compares an object to a specified type.
- The instanceof operator is used to test whether the object is an instance of the specified type (class or subclass or interface).

- The instanceof operator is also known as type comparison operator because it compares the instance with type. It returns either true or false. If we apply the instanceof operator with any variable that have null value, it returns false.

- **Syntax:**

(Object reference variable) instanceof (class/interface type)

- If the object referred by the variable on the left side of the operator passes the IS-A check for the class/interface type on the right side, then the result will be true.

- **For Example:**

```
public class Test
{
    public static void main(String args[])
    {
        String name = "James";
        // following will return true since name is type
        of String
        boolean result = name instanceof String;
        System.out.println( result );
    }
}
```

**Output:**

True

## 7. Bitwise Operator :

- In order to manipulate the data at the bit level, the bitwise operators are provided in Java.

- These operators are used for testing the bits as well as shifting them to left or right etc.

*These can be applied to integer types only.*

• Bitwise operator works on bits (0 or 1) and perform bit by bit operation. Assume if  $x = 60$ ; and  $y = 13$ ; Now in binary format they will be as follows:

$x = 0011\ 1100$

$y = 0000\ 1101$

-----

$x \& y = 0000\ 1100$

$x | y = 0011\ 1101$

$x \wedge y = 0011\ 0001$

$\sim x = 1100\ 0011$

Operator	Meaning	Explanation	Example
&	Bitwise And	Multiplies the individual bits of operands	(A & B) will give 12 which is 0000 1100
	Bitwise OR	Adds the individual bits of operands	(A   B) will give 61 which is 0011 1101
^	Bitwise XOR	It copies the bit if it is set in one operand but not both.	A ^ B) will give 49 which is 0011 0001
<<	Binary Left shift	Shifts the bits of the number towards left a specified number of positions	A << 2 will give 240 which is 1111 0000
>>	Binary Right shift	Shifts the bits of the number towards right a specified number of positions and also preserves the sign bit	A >> 2 will give 15 which is 1111
>>>	Zero fill right shift	Shifts the bits of the number towards right a specified number of positions and it stores 0 (Zero) in the sign bit	A >>> 2 will give 15 which is 0000 1111
~	Binary Complement	Gives the complement form of a given number by changing 0's as 1's and vice versa	(~A) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number

➤ *For example*

**1. Write a program to perform Bitwise operations**  
**//Bitwise Operations**

⇒

```
class Bits{
    public static void main(String args[]){
        byte x,y;
        x=10;
        y=11;
        System.out.println ("~x="+(~x));
        System.out.println ("x & y="+(x&y));
        System.out.println ("x | y="+(x|y));
        System.out.println ("x ^ y="+(x^y));
        System.out.println ("x<<2="+(x<<2));
        System.out.println ("x>>2="+(x>>2));
        System.out.println ("x>>>2="+(x>>>2));
    }
}
```

**2. Given that a= 64 , I=a<<2 write a program to find the value of I.**

⇒

```
public class Bitwise_Operator
{
    public static void main(String[] args)
    {
        int a = 64, I;
        I=a<<2;
        System.out.println("a="+a);
        System.out.println("The value of I
                           is :"+I);
    }
}
```