

Operators :

-----  
It is a symbol which describes that how a calculation will be performed on operands.

Types Of Operators :

-----  
1) Arithmetic Operator (Binary Operator)

2) Unary Operators

3) Assignment Operator

4) Relational Operator

5) Logical Operators

6) Boolean Operators

7) Bitwise Operators

8) Ternary Operator

9) Member Operator

10) new Operator

11) instanceof Operator

-----  
Arithmetic Operator OR Binary Operator :

-----  
It is known as Arithmetic Operator OR Binary Operator because it works with minimum two operands.

Ex:- +, -, \*, / and % (Modula Or Modulus Operator)

-----  
//Arithmetic Operator  
// Addition operator to join two Strings working as String concatenation oprtr  
public class Test1  
{  
 public static void main(String[] args)  
 {  
 String s1 = "Welcome to";  
 String s2 = " Java ";  
 String s3 = s1 + s2;  
 System.out.println("String after concatenation :" + s3);  
 }  
}

-----  
How to read the value from the user/keyboard (Accepting the data from client)

-----  
In order to read the data from the client or keyboard, java software people has provided a predefined class called Sc

It is available from java 5v.

-----  
static variables of System class :

-----  
System is a predefined class which contains 3 static variables.

System.out :- It is used to print normal message on the screen.

System.err :- It is used to print error message on the screen.

System.in :- It is used to take input from the user.(Attaching the keyboard with System resource)

How to create the Object for Scanner class :

```
Scanner sc = new Scanner(System.in); //Taking the input from the user
```

Scanner class provides various methods :

String next() :- Used to read a single word.

String nextLine() :- Used to read complete line or multiple Words.

byte nextByte() :- Used to read byte value

short nextShort() :- Used to read short value

int nextInt() :- Used to read integer value

float nextFloat() :- Used to read float value

double nextDouble() :- Used to read double value

boolean nextBoolean() :- Used to read boolean value.

char next().charAt(0) :- Used to read a character

```
//WAP to read your name from the keyboard
import java.util.*;
public class Test2
{
    public static void main(String [] args)
    {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter your Name :");
        String name = sc.next(); //will read single word
        System.out.println("Your name is :" + name);
    }
}
```

27-10-2023

```
//WAP to read your name from the keyboard
import java.util.Scanner;
public class ReadCompleteName
{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter your Name :");
        String name = sc.nextLine();
        System.out.println("Your Name is :" + name);
    }
}
```

```

//Reading Employee data
import java.util.Scanner;
public class EmployeeData
{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter Employee Id :");
        int id = sc.nextInt();

        System.out.print("Enter Employee Name :");
        String name = sc.nextLine(); //Buffer Problem
        name = sc.nextLine();
        System.out.println("Employee Id is :" + id);
        System.out.println("Employee Name is :" + name);
    }
}

-----//Arithmetic Operator (+, -, *, /, %)
//Reverse of a 3 digit number
import java.util.*;
class Test3
{
    public static void main(String[] args)
    {
        System.out.print("Enter a three digit number :");
        Scanner sc = new Scanner(System.in);

        int num = sc.nextInt(); //num = 567

        int rem = num % 10; //rem = 7
        System.out.print("The Reverse is :" + rem); //The reverse is :765

        num = num / 10; //num = 56
        rem = num % 10; //rem = 6
        System.out.print(rem);

        num = num / 10; //num = 5
        System.out.println(num);
    }
}

```

#### -----Unary Operator :

The operator which works upon single operand is called Unary Operator. In java we have so many unary operators which are as follows:

- 1) Unary minus operator (-)
- 2) Increment Operator (++)
- 3) Decrement Operator (--)

```

/*Unary Operators (Acts on only one operand)
//Unary minus Operator
class Test4
{
    public static void main(String[] args)

```

```
{

    int x = 15;
    System.out.println(-x);
    System.out.println(-(-x));
}

-----
//Unary Operators
//Unary Pre increment Operator
class Test5
{
    public static void main(String[] args)
    {
        int x = 15;
        int y = ++x; //First increment then assignment
        System.out.println(x+":"+y);
    }
}

-----
//Unary Operators
//Unary Post increment Operator
class Test6
{
    public static void main(String[] args)
    {
        int x = 15;
        int y = x++; //First assignment then increment
        System.out.println(x+":"+y);
    }
}

-----
//Unary Operators
//Unary Pre increment Operator
class Test7
{
    public static void main(String[] args)
    {
        int x = 15;
        int y = ++15; //error
        System.out.println(y);
    }
}

-----
//Unary Operators
//Unary Pre increment Operator
class Test8
{
    public static void main(String[] args)
    {
        int x = 15;
        int y = ++(++x); //error
        System.out.println(y);
    }
}

-----
//Unary Operators
//Unary post increment Operator
class Test9
{
```

```
public static void main(String[] args)
{
    int x = 15;
    System.out.println(++x + x++);
    System.out.println(x);
    System.out.println(".....");

    int y = 15;
    System.out.println(++y + ++y);
    System.out.println(y);
}
```

-----  
Note :- Increment and decrement operator we can apply with any data type except boolean.  
-----

```
//Unary Operators
//Unary post increment Operator
class Test10
{
    public static void main(String[] args)
    {
        char ch ='A';
        ch++;
        System.out.println(ch);
    }
}
```

```
-----  
//Unary Operators
//Unary post increment Operator
class Test11
{
    public static void main(String[] args)
    {
        double d = 15.15;
        d++;
        System.out.println(d);
    }
}
```

```
-----  
//Unary Operators
//Unary Pre decrement Operator
class Test12
{
    public static void main(String[] args)
    {
        int x = 15;
        int y = --x; //First decrement then assignment
        System.out.println(x+":"+y);
    }
}
```

```
-----  
//Unary Operators
//Unary Post decrement Operator
class Test13
{
    public static void main(String[] args)
    {
```

```
    int x = 15;
    int y = x--;
    System.out.println(x+":"+y);
}
```

---

Interview Question

---

Whenever we work with Arithmetic Operator or Unary minus operator, the minimum data type required is int, So aft

```
//IQ
class Test14
{
    public static void main(String args[])
    {
        byte i = 1;
        byte j = 1;
        byte k = i + j; //error
        System.out.println(k);
    }
}
```

---

```
class Test15
{
    public static void main(String args[])
    {
        /*byte b = 6;
        b = b + 7; //error
        System.out.println(b); */

        byte b = 6;
        b += 7;//short hand operator b += 7 is equal to (b = b + 7)
        System.out.println(b);
    }
}
```

Note :- In the above program it generates error while working with Arithmetic Operator but when we change the operator Arithmetic to short hand operator then the expression result we can assign on byte data type.

---

```
class Test16
{
    public static void main(String args[])
    {
        byte b = 1;
        byte b1 = -b; //error
        System.out.print(b1);
    }
}
```

---

What is a local variable :

---

If a variable is declared inside a method body(not as a method parameter) then it is called Local / Stack/ Temporary

Ex:-

```
public void input()
{
    int y = 12;
}
```

Here in the above example y is local variable.

Local variable we can't use outside of the function or method.

A local variable must be initialized before use otherwise we will get compilation error.

We can't use any access modifier on local variable except final.

Program

```
-----  
public class Test17  
{  
    public static void main(String [] args)  
    {  
        int x ; //must be initialized before use  
        System.out.println(x);  
  
        public int y = 100;//only final is acceptable  
        System.out.println(y);  
    }  
}
```

Note :- In the above program we will get compilation error

-----  
Why we cannot use local variables outside of the method ?

-----  
In java all the methods are executed as a part of Stack Memory. Stack Memory works on the basis of LIFO (Last In First Out).

Once the method execution is over, local variables are also deleted from stack frame so we cannot use local variable outside of the method.

```
class StackMemory  
{  
    public static void main(String[] args)  
    {  
        System.out.println("Main method started..");  
        m1();  
        System.out.println("Main method ended..");  
    }  
    public static void m1()  
    {  
        System.out.println("m1 method started..");  
        m2();  
        System.out.println("m1 method ended..");  
    }  
    public static void m2()  
    {  
        int x = 100;  
        System.out.println("I am m2 method!!!" + x);  
    }  
}
```

-----  
28-10-2023

```
-----  
/*Program on Assignment Operator  
class Test18  
{  
    public static void main(String args[])  
    {
```

```

int x = 5, y = 3;
System.out.println("x = " + x);
System.out.println("y = " + y);
 $\square$ 
x %= y;      //short hand operator x = x % y
System.out.println("x = " + x);
 $\square$ 
 $\square$ 
}

```

What is BLC class :

BLC stands for Business Logic class. A BLC class never contains main method. It is used to write the logic only.

What is ELC class :

ELC stands for Executable Logic class. An ELC class always contains main method. It is called ELC class because the e

Note :- WE SHOULD ALWAYS TAKE OUR JAVA CLASSES IN A SEPARATE FILE OTHERWISE THE RE-USABILITY OF THE CLA

Here we have 2 files :

Circle.java(BLC)

```

package com.ravi.blc_elc;
/*
Find the area of circle. Accept the radius value from the user
if the radius is zero or negative then return -1.
*/

```

```

//BLC
public class Circle
{
    public static double getAreaOfCircle(int radius)
    {
        if(radius <=0)
        {
            return -1;
        }
        else
        {
            final double PI = 3.14;
            double areaOfCircle = PI * radius * radius;
            return areaOfCircle;
        }
    }
}

```

AreaOfCircle.java(ELC)

```

package com.ravi.blc_elc;

import java.util.Scanner;

//ELC
public class AreaOfCircle
{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the radius of the Circle :");
    }
}

```

```

    int radius = sc.nextInt();
    ...
    double areaOfCircle = Circle.getAreaOfCircle(radius);
    System.out.println("Area of Circle is :" + areaOfCircle);
    sc.close();
}

}

```

#### Relational Operator :-

These operators are used to compare the values. The return type is boolean. We have total 6 Relational Operators.

- 1) > (Greater than)
- 2) < (Less than)
- 3) >= (Greater than or equal to)
- 4) <= (Less than or equal to)
- 5) == (double equal to)
- 6) != (Not equal to )

```

/*Program on relational operator(6 Operators)
class Test19
{
    public static void main(String args[])
    {
        int a = 10;
        int b = 20;
        System.out.println("a == b : " + (a == b) ); //false
        System.out.println("a != b : " + (a != b) ); //true
        System.out.println("a > b : " + (a > b) ); //false
        System.out.println("a < b : " + (a < b) ); //true
        System.out.println("b >= a : " + (b >= a) ); //true
        System.out.println("b <= a : " + (b <= a) ); //false
    }
}

```

#### If condition :

It is decision making statement. It is used to test a boolean expression. The expression must return boolean type.

```

//Program to check a number is 0 or +ve or -ve
import java.util.Scanner;
class Test20
{
    public static void main(String args[])
    {
        Scanner sc = new Scanner(System.in);
        System.out.print("Please enter a Number :");

        int num = sc.nextInt();
        if(num == 0)
            System.out.println("It is zero");

        else if(num>0)
            System.out.println(num+" is positive");
    }
}

```

```

    }else
    System.out.println(num+" is negative");

    sc.close(); //To close Scanner resource
}

-----
/*program to calculate telephone bill
For 100 free call rental = 360
For 101 - 250, 1 Rs per call
For 251 - unlimited , 1.2 Rs per call
*/
import java.util.*;
class Test21
{
public static void main(String args[])
{
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter current Reading :");
    int curr_read = sc.nextInt();

    System.out.print("Enter Previous Reading :");
    int prev_read = sc.nextInt();

    int nc = curr_read - prev_read; if(curr_read > prev_read)
    System.out.println("Your Number of call for this month is :" + nc);

    double bill = 0.0;
    if (nc <=100)
    {
        bill = 360;
    }
    else if(nc<=250)
    {
        bill = 360 + (nc-100)*1.0;
    }
    else if(nc >250)
    {
        bill = 360 + 150 + (nc-250)*1.2;
    }
    System.out.println("The bill is :" +bill);
}
}
-----
```

Nested if:

If an 'if condition' is placed inside another if condition then it is called Nested if.

In nested if condition, we have one outer if and one inner if condition, the inner if condition will only execute when

```

if(condition) //Outer if condition
{
    if(condition) //inner if condition
    {
    }
    else //inner else
    {
    }
}
else //outer else
{
```

```

}

-----
```

```

//Nested if
//big among three number
class Test22
{
public static void main(String args[])
{
    int a =15;
    int b =12;
    int c =18;

    int big=0;

    if(a>b) // (Outer if condition)
    {
        if(a>c) //Nested If Block (inner if)
        {
            big=a;
        }
        else
        {
            big=c;
        }
    }
    else //already confirmed b is greater than a
    {
        if(b>c)
        {
            big=b;
        }
        else
        {
            big=c;
        }
    }
    System.out.println("The big number is :" +big);
}
}
```

Note :- In the above program to find out the biggest number among three number we need to take the help of nested if. The code Logical Operator came into the picture.

Logical Operator :-

It is used to combine or join the multiple conditions into a single statement.

It is also known as short-Circuit logical operator.

In Java we have 3 logical Operators

- 1) && (AND Logical Operator)
- 2) || (OR Logical Operator)
- 3) ! (NOT Logical Operator)

&& :- All the conditions must be true. if the first expression is false it will not check right side expressions.

|| :- Among multiple conditions, at least one condition must be true. if the first expression is true it will not check right side expressions.

! :- It is an inverter, it makes true as a false and false as a true.

Note :- The && and || operator only works with boolean operand so the following code will not compile.

```

if(5 && 6)
{
```

```
}
```

---

```
/*Program on Logical Operator (AND, OR, Not Operator)
//Biggest number among 3 numbers
```

```
class Test23
{
public static void main(String args[])
{
    java.util.Scanner sc = new java.util.Scanner(java.lang.System.in);
    System.out.print("Enter the value of a :");
    int a = sc.nextInt();
    System.out.print("Enter the value of b :");
    int b = sc.nextInt();
    System.out.print("Enter the value of c :");
    int c = sc.nextInt();
    int big =0;

    if(a>b && a>c)
        big = a;
    else if(b>a && b>c)
        big = b;
    else
        big = c;
    System.out.println("The big number is :" +big);
}
```

---

```
//OR Operator (At least one condition must be true)
```

```
class Test24
{
public static void main(String args[])
{
    int a=10;
    int b=5;
    int c=20;
    System.out.println(a>b || a<c); //true
    System.out.println(b>c || a>c); //false
}
```

---

```
// !Operator (not Operator works like an Inverter)
```

```
class Test25
{
    public static void main(String args[])
{
    System.out.println(!true);
}
```

---

```
Boolean Operators :
```

---

```
Boolean Operators work with boolean values that is true and false. It is used to perform boolean logic upon two bo
```

It is also known as non short circuit. There are two non short circuit logical operators.

& boolean AND operator (All condions must be true but if first expression is false still it will check all right side exp

```

| boolean OR operator (At least one condition must be true but if the first condition is true still it will check all right)

-----
```

```

/* Boolean Operators
/*
& boolean AND operator
| boolean OR operator

*/
//Works with boolean values

class Test26
{
    public static void main(String[] args)
    {
        int z = 5;
        if(++z > 5 | ++z > 6) //Logical OR
        {
            z++;
        }
        System.out.println(z); //7

        System.out.println(".....");

        z = 5;
        if(++z > 5 | ++z > 6) //Boolean OR
        {
            z++;
        }
        System.out.println(z); //8
    }
}

-----
```

```

class Test27
{
    public static void main(String[] args)
    {
        int z = 5;
        if(++z > 6 & ++z > 6)
        {
            z++;
        }
        System.out.println(z);
    }
}
```

-----  
30-OCT-23

-----  
Bitwise Operator :-

-----  
In order to work with binary bits java software people has provided Bitwise operator. It also contains 3 operators  
& (Bitwise AND) :- Returns true if both the inputs are true.

| (Bitwise OR) :- Returns false if both the inputs are false

^ (Bitwise X-OR) :- Returns true if both the arguments are opposite to each other.

```

//Bitwise Operator
class Test28
```

```
{
public static void main(String[] args)
{
    System.out.println(true & true); //true
    System.out.println(false | true); //true
    System.out.println(true ^ true); //true

    System.out.println(6 & 7); //6
    System.out.println(6 | 7); //7
    System.out.println(6 ^ 7); //1
}
-----
```

Bitwise complement operator :

-> It will not work with boolean literal.

```
//Bitwise Complement Operator
public class Test29
{
    public static void main(String args[])
{
    //System.out.println(~ true); Invalid
    System.out.println(~ -8);
}
}
```

-----  
Ternary Operator OR Conditional Operator :

The ternary operator (?) consists of three operands. It is used to evaluate boolean expressions. The operator decided the size of if-else condition.

```
//Ternary Operator OR Conditional Operator
public class Test30
{
    public static void main(String args[])
{
    int a = 60;
    int b = 59;
    int max = 0;

    max=(a>b)?a:b; //Type casting
    System.out.println("Max number is :" +max);

}
}
```

-----  
public class Test
{
 public static void main(String [] args)
{
 char ch = 'A';
 float i = 12;
 System.out.println(false?i:ch); //65.0 //type casting
 System.out.println(true?ch:i); //65.0 //type casting

}
}

```
    }
-----
```

Member access Operator Or Dot Operator :

-----

It is used to access the member of the class so whenever we want to call the member of the class (fields + methods)

We can directly call any static method and static variable from the main method with the help of class name , here o

If static variable or static method is present in the same class where main method is available then we can directly call another class then to call those static members of the class, class name is required.

```
class Welcome
{
    static int x = 100;

    public static void access()
    {
        System.out.println(x);
    }
}

public class Test
{
    public static void main(String [] args)
    {
        System.out.println(Welcome.x);
        Welcome.access();
    }
}
```

-----

/\* new Operator

This Operator is used to create Object. If the member of the class (field + method) is static, object is not required. we

On the other hand if the member of the class (variables + method) is not declared as static then it is called non-static object is required.

Program :

```
class Welcome
{
    int x = 100; //instance variable (Non-static field)

    public void access() //instance method
    {
        System.out.println(x);
    }
}

public class Test
{
    public static void main(String [] args)
    {
        Welcome w = new Welcome();
        System.out.println(w.x);
        w.access();
    }
}
```

---

instanceof operator :-

- 1) This Operator will return true/false
- 2) It is used to check a reference variable is holding the particular/corresponding type of Object or not.
- 3) It is also a keyword.
- 4) In between the object reference and class name , we must have some kind of relation (assignment relation) otherwise it will give error.

```
/* instanceof operator
public class Test
{
    public static void main(String[] args)
    {
        String str = "India";

        if(str instanceof String)
        {
            System.out.println("str is pointing to String object");
        }

        Integer i = 45;
        if(i instanceof Integer)
        {
            System.out.println("i is pointing to Integer object");
        }

        Double d = 90.67;
        if(d instanceof Number) //IS-A relation between Double and Number class
        {
            System.out.println("d is pointing to Double object");
        }
    }
}
```