OPERATORS

Operators are used to manipulate primitive data types.

Java operators can be classified as unary,binary, or ternary—meaning taking one, two, or three arguments, respectively.

Java Unary Operator

The Java unary operators require only one operand. Unary operators are used to perform various operations

i.e.:

- incrementing/decrementing a value by one
- negating an expression
- inverting the value of a boolean

Java Unary Operator Example: ++ and –

```
1. class OperatorExample
2. {
3. public static void main(String args[])
4. {
5. int x=10;
6. System.out.println(x++);          //10 (11)
7. System.out.println(++x);          //12
8. System.out.println(x--);          //12 (11)
9.  System.out.println(--x);         //108.
10.}
11.}
```

Output:

```
10
12
12
10
```

Java Unary Operator Example 2: ++ and –

```
1. class OperatorExample
2. {
3. public static void main(String args[])
4. {
5.  int a=10;
6.  int b=10;
7.  System.out.println(a++ + ++a);          //10+12=22
8.  System.out.println(b++ + b++);          //10+11=21 7.
9.  }
10.}
```

Output:

```
22
21
```

Java Unary Operator Example: ~ and !

```
1. class OperatorExample{
2. public static void main(String args[]){
3.     int a=10;
4.     int b=-10;
5. boolean c=true;
6. boolean d=false;
7. System.out.println(~a);      //-11 (minus of total positive value which starts from 0)
8. System.out.println(~b);          //9 (positive of total minus, positive starts from 0)
9. System.out.println(!c);      //false (opposite of boolean value)
10.    System.out.println(!d);          //true
11.    }
12.    }
```

Output:

```
-11
9
False
true
```

A binary or ternary operator appears between its arguments.

Java operators fall into eight different categories:

1. **Assignment**
2. **Arithmetic**
3. **Relational**
4. **Logical**
5. **Bitwise**
6. **Compound assignment**
7. **Conditional**
8. **Type**

Assignment Operators	**=**
Arithmetic Operators	**- + * / % ++ --**
Relational Operators	**> < >= <= == !=**
Logical Operators	**&& || & | ! ^**
Bit wise Operator	**& | ^ >> >>>**
Compound Assignment Operators	**+= -= *= /= %= <<= >>= >>>=**
Conditional Operator	**?:**

1. Java Assignment Operator

The java assignment operator statement has the following syntax:

<*variable*> = <*expression*>

If the value already exists in the variable it is overwritten by the assignment operator (=).

Java Assignment Operator Example

```
1. class OperatorExample{
2. public static void main(String args[])
3. {
4. int a=10;
5. int b=20;
6. a+=4;                //a=a+4 (a=10+4)
7. b-=4;                //b=b-4 (b=20-4)
8. System.out.println(a);
9. System.out.println(b);
10. }
11. }
```

Output:

```
14
16
```

2. Java Arithmetic Operators

Java arithmetic operators are used to perform addition, subtraction, multiplication, and division.They act as basic mathematical operations.

Assume integer variable A holds 10 and variable B holds 20, then:

Operator	Description	Example
+	Addition - Adds values on either side of the operator	A + B will give 30
-	Subtraction - Subtracts right hand operand from left hand operand	A - B will give -10
*	Multiplication - Multiplies values on either side of the operator	A * B will give 200
/	Division - Divides left hand operand by right hand operand	B / A will give 2
%	Modulus - Divides left hand operand by right hand operand and returns remainder	B % A will give 0
++	Increment - Increases the value of operand by 1	B++ gives 21
--	Decrement - Decreases the value of operand by 1	B-- gives 19

Java Arithmetic Operator Example: Expression

```
1.  class OperatorExample
2.  {
3.  public static void main(String args[])
4.  {
5.      System.out.println(10*10/5+3-1*4/2);
6.  }
7.  }
```

Output:

```
21
```

3. Relational Operators

Relational operators in Java are used to compare 2 or more objects. Java provides six relational operators: Assume variable A holds 10 and variable B holds 20, then:

Operator	Description	Example
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not, ifvalues are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value ofright operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal tothe value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomestrue.	(A <= B) is true.

Example:

```
public RelationalOperatorsDemo( )
{
      int x = 10, y = 5;
      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 <= y : "+(x <= y));
      System.out.println("x == y : "+(x == y));
      System.out.println("x != y : "+(x != y));

 public static void main(String args[])
 {
       new RelationalOperatorsDemo();
 }
}
```

Output:

```
$java RelationalOperatorsDemo
x > y : true
x < y : false
x >= y : true
x <= y : false
x == y : false
x != y : true
```

4. Logical Operators

Logical operators return a true or false value based on the state of the Variables. Given that x and y represent boolean expressions, the boolean logical operators are defined in the Table below.

X	Y	!x	x & y x && y	x | y x || y	x ^ y
True	True	False	true	true	False
True	False	False	false	true	True
False	True	True	false	true	True
False	False	True	false	false	false

Example:

```
public class LogicalOperatorsDemo
{
   public LogicalOperatorsDemo()



   {
      boolean x = true;
      boolean y = false;
      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 || y: " + (x || y));
      System.out.println("x ^ y : " + (x ^ y));
      System.out.println("!x : " + (!x));
   }
public static void main(String args[])
{
new LogicalOperatorsDemo();
}
}
```

Output:

```
$java LogicalOperatorsDemo
      x & y : false
      x && y : false
      x | y : true
      x || y: true
      x ^ y : true
      !x : false
```

5. Bitwise Operators

Java provides Bit wise operators to manipulate the contents of variables at the bit level. The result of applying bitwise operators between two corresponding bits in the operandsis shown in the Table below.

A	B	~A	A & B	A | B	A ^ B
1	**1**	0	1	1	0
1	**0**	0	0	1	1
0	**1**	1	0	1	1
0	**0**	1	0	0	0

```
public class Test
{
public static void main(String args[])
{
    int a = 60; /* 60 = 0011 1100 */
```

```
    int b = 13;    /* 13 = 0000 1101 */int c = 0;
    c = a & b;     /* 12 = 0000 1100 */
    System.out.println("a & b = " + c );
    c = a | b;     /* 61 = 0011 1101 */



    System.out.println("a | b = " + c );
    c = a ^ b;     /* 49 = 0011 0001 */
    System.out.println("a ^ b = " + c );
    c = ~a;        /*-61 = 1100 0011 */
    System.out.println("~a = " + c );
    c = a << 2;    /* 240 = 1111 0000 */
    System.out.println("a << 2 = " + c );
    c = a >> 2;    /* 215 = 1111 */
    System.out.println("a >> 2 = " + c );
    c = a >>> 2;   /* 215 = 0000 1111 */
    System.out.println("a >>> 2 = " + c );
}
}
```

Output:

$java Test

```
a & b = 12
a | b = 61
a ^ b = 49
~a = -61
a << 2 = 240
a >> 2  = 15
a >>> 2 = 15
```

6. Compound Assignment operators

The compound operators perform shortcuts in common programming operations. Java has eleven compound assignment operators.

Syntax: argument1 **operator** = argument2.

Java Assignment Operator Example

```
1. class OperatorExample
2. {
3. public static void main(String[] args)
4. {
5. int a=10;
6.  a+=3;     //10+3
7. System.out.println(a);
8.  a-=4;     //13-4
```

```
9.  System.out.println(a);
10. a*=2;     //9*2
11. System.out.println(a);
12. a/=2;    //18/2


13. System.out.println(a);12.
14. }
15. }
```

Output:

```
13
9
18
9
```

7. Conditional Operators

The Conditional operator is the only ternary (operator takes three arguments) operator in Java. The operator evaluates the first argument and, if true, evaluates the second argument.

If the first argument evaluates to false, then the third argument is evaluated. The conditional operator is the expression equivalent of the if-else statement.

The conditional expression can be nested and the conditional operator associates from right to left: **(a?b?c?d:e:f:g) evaluates as (a?(b?(c?d:e):f):g)**

Example:

```
public class TernaryOperatorsDemo {

        public TernaryOperatorsDemo() {
                int x = 10, y = 12, z = 0;
                z = x > y ? x : y;
                System.out.println("z : " + z);
        }
        public static void main(String args[]) {
                new TernaryOperatorsDemo();
        }
}
```

Output:

```
$java TernaryOperatorsDemo
z : 12
```

8. instanceof Operator:

This operator is used only for object reference variables. The operator checks whether the object is of a particular type(class type or interface type). instanceof operator is written as:

(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. Following is the

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 );
   }
}
```

This would produce the following result:

True

OPERATOR PRECEDENCE:

The order in which operators are applied is known as precedence. Operators with a higher precedence are applied before operators with a lower precedence.

The operator precedence order of Java is shown below. Operators at the top of the table are applied before operators lower down in the table.

If two operators have the same precedence, they are applied in the order they appear in a statement. That is, from left to right. You can use parentheses to override the default precedence.

Category	Operator	Associativity
Postfix	() [] . (dot operator)	Left to right
Unary	++ - - ! ~	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	>> >>> <<	Left to right
Relational	> >= < <=	Left to right

Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right

Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= |=	Right to left
Comma	,	Left to right

Example:

In an operation such as,

result = 4 + 5 * 3

First (5 * 3) is evaluated and the result is added to 4 giving the Final Result value as 19. Note that _*‘ takes higher precedence than _+‘ according to chart shown above. This kindof precedence of one operator over another applies to all the operators.