**OPERATORS IN JAVA**

Miracle Techies:

Samyuktha Sudhakaran

Athulya Vijayan

Sona Sudhakaran

Naznin Shafeek

Java provides many types of operators which can be used according to the need. They are classified based on the functionality they provide.

Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations on variables and data. For example a+b;

Here, the + operator is used to add two variables a and b. Similarly, there are various other arithmetic operators in Java like

+ — Addition

— Subtraction

\* — Multiplication

/ — Division

% — Modulo operation

Example program:

class Main {

public static void main(String[] args) {

// declare variables

int a = 12, b = 5;

// addition operator

System.out.println("a + b = " + (a + b));

// subtraction operator

System.out.println("a - b = " + (a - b));

// multiplication operator

System.out.println("a \* b = " + (a \* b));

// division operator

System.out.println("a / b = " + (a / b));

// modulo operator

System.out.println("a % b = " + (a % b));

}

}

Output:

a+b=17

a-b=7

a\*b=60

a/b=2

a%b=2

**Unary Operators**

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

There are many types of unary operators in java. They are as follows:

– : Unary minus, used for negating the values.

+ : Unary plus indicates the positive value (numbers are positive without this, however). It performs an automatic conversion to int when the type of its operand is the byte, char, or short. This is called unary numeric promotion.

++ : Increment operator, used for incrementing the value by 1. There are two varieties of increment operators.

Post-Increment: Value is first used for computing the result and then incremented.

Pre-Increment: Value is incremented first, and then the result is computed.

– – : Decrement operator, used for decrementing the value by 1. There are two varieties of decrement operators.

Post-decrement: Value is first used for computing the result and then decremented.

Pre-Decrement: Value is decremented first, and then the result is computed.

Logical not operator, used for inverting a boolean value

Example Program:

public class OperatorExample{

public static void main(String args[]){

int x=10;

System.out.println(x++);//10 (11)

System.out.println(++x);//12

System.out.println(x--);//12 (11)

System.out.println(--x);//10

}}

Output:

10

12

12

10

**Shift Operators**

**Java Left Shift Operator-**

The Java left shift operator << is used to shift all of the bits in a value to the left side of a specified number of times.

**Java Left Shift Operator Example:**

public class OperatorExample{

public static void main(String args[]){

System.out.println(10<<2);//10\*2^2=10\*4=40

System.out.println(10<<3);//10\*2^3=10\*8=80

System.out.println(20<<2);//20\*2^2=20\*4=80

System.out.println(15<<4);//15\*2^4=15\*16=240

}}

Output:

40

80

80

240

**Java Right Shift Operator-**

The Java right shift operator >> is used to move the value of the left operand to right by the number of bits specified by the right operand.

**Java Right Shift Operator Example:**

public OperatorExample{

public static void main(String args[]){

System.out.println(10>>2);//10/2^2=10/4=2

System.out.println(20>>2);//20/2^2=20/4=5

System.out.println(20>>3);//20/2^3=20/8=2

}}

Output:

2

5

2

**Relational Operators**

Java Relational Operators are a bunch of binary operators used to check for relations between two operands, including equality, greater than, less than, etc. They return a boolean result after the comparison and are extensively used in looping statements as well as conditional if-else statements and so on. The general format of representing relational operator is:

Syntax:

variable1 relation\_operator variable2

Java has 6 relational operators.

== is the equality operator. This returns true if both the operands are referring to the same object, otherwise false.

!= is for non-equality operator. It returns true if both the operands are referring to the different objects, otherwise false.

< is less than operator.

> is greater than operator.

<= is less than or equal to operator.

>= is greater than or equal to operator.

Example program:

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

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

}

}

Output:

a IS 7 AND b is 11

False

True

False

True

False

True

**Ternary Operator**

Java ternary operator is the only conditional operator that takes three operands. It’s a one- liner replacement for the if-then-else statement and is used a lot in Java programming. We can use the ternary operator in place of if-else conditions or even switch conditions using nested ternary operators. Although it follows the same algorithm as of if-else statement, the conditional operator takes less space and helps to write the if-else statements in the shortest way possible.

Example for Ternary operator

1. public class OperatorExample{

2. public static void main(String args[]){

3. int a=2;

4. int b=5;

5. int min=(a<b)?a:b;

6. System.out.println(min);

7. }}

Output : 2

**Assignment Operators**

These operators are used to assign values to a variable. The left side operand of the assignment operator is a variable, and the right side operand of the assignment operator is a value. The value on the right side must be of the same data type of the operand on the left side. Otherwise, the compiler will raise an error. This means that the assignment operators have right to left associativity, i.e., the value given on the right-hand side of the operator is assigned to the variable on the left. Therefore, the right-hand side value must be declared before using it or should be a constant. The general format of the assignment operator is,;

Variable operator value;

Types of Assignment Operators in Java

The Assignment Operator is generally of two types. They are:

1. Simple Assignment Operator: The Simple Assignment Operator is used with the “=” sign where the left side consists of the operand and the right side consists of a value. The value of the right side must be of the same data type that has been defined on the left side.

2. Compound Assignment Operator: The Compound Operator is used where +,-,\*, and / is used along with the = operator.

Example for Assignment operators

public class OperatorExample{

public static void main(String args[]){

int a=10;

int b=20;

a+=4;//a=a+4 (a=10+4)

b-=4;//b=b-4 (b=20-4)

System.out.println(a);

System.out.println(b);

}}

Output : 14 16

**Logical Operators**

Logical Operators are used to check whether an expression is true or false.They are used in decision making.

1.Logical ‘AND’ Operator (&&)

* Returns true when both the conditions are satisfied or are true.
* If even one of the two yields false, the operator results false.

## **2.Logical ‘OR’ Operator (||)**

## returns true when one of the two conditions is satisfied or is true

## If even one of the two yields true, the operator results true.

## To make the result false, both the constraints need to return false.

## **3.Logical ‘NOT’ Operator (!)**

## If the condition is false, the operation returns true and when the condition is true, the operation returns false.

## Example:

**a = 10, b = 20, c = 30**

**For AND operator:**

Condition 1: c > a

Condition 2: c > b

**Output:** True [Both Conditions are true]

**For OR Operator:**

Condition 1: c > a

Condition 2: c > b

**Output:** True [One of the Condition if true]

**For NOT Operator:**

Condition 1: c > a

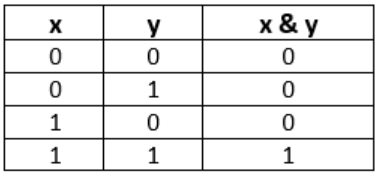
Condition 2: c > b

**Output:** False [Because the result was true and NOT operator did it's opposite]

**Bitwise Operators**

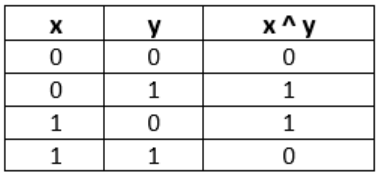
## Bitwise operators in Java are used to perform operations on individual bits.

### 1.Bitwise AND (&)

It is a binary operator denoted by the symbol **&**. It returns 1 if and only if both bits are 1else returns 0. 

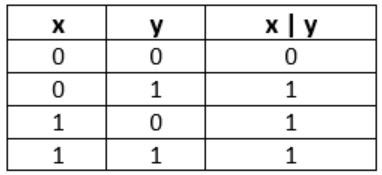
2.Bitwise exclusive OR (^)

### It is a binary operator denoted by the symbol **^** (pronounced as caret). It returns 0 if both bits are the same, else returns 1.



### 3.Bitwise inclusive OR (|)

### It is a binary operator denoted by the symbol **|** (pronounced as a pipe). It returns 1 if either of the bit is 1, else returns 0.



### 4.Bitwise Complement (~)

### It is a unary operator denoted by the symbol **~** (pronounced as the tilde). It returns the inverse or complement of the bit. It makes every 0 a 1 and every 1 a 0.

### 5.Bit Shift Operators

### Shift operator is used in shifting the bits either right or left. We can use shift operators if we divide or multiply any number by 2.

### Eg: if a=10

a>>2; //shifts two bits

a>>4; //shifts 4 bits

**Example of Bitwise operators**:

i**nt** x = 9, y = 8;

// bitwise and

// 1001 & 1000 = 1000 = 8

System.out.println("x & y = " + (x & y));  //8

//bitwise XOR

// 1001 ^ 1000 = 0001 = 1

System.out.println("x ^ y = " + (x ^ y));   //1

// bitwise inclusive OR

// 1001 | 1000 = 1001 = 9

System.out.println("x | y = " + (x | y)); //9

// bitwise compliment

// ~0010= 1101 = -3

System.out.println("~x = " + (~x));  //-3

**Java operator precedence**

