**OPERATORS**

**Operator** in java is a symbol that is used to perform operations.

**TYPES OF OPERATORS**

* Arithmetic Operators
* Unary Operators
* Relational Operators
* Logical Operators
* Bitwise Operators
* Assignment Operators
* Conditional Operator
* Instanceof Operator

**Arithmetic Operators**

Java arithmetic operators are used to perform addition, subtraction, multiplication, and division(+,-,\*,/,%)

**EXAMPLE**

public class arithop {

public static void main(String[] args) {

//Variables Definition and Initialization

int number1 = 12, number2 = 4;

//Addition Operation

int sum = number1 + number2;

System.out.println("Sum is: " + sum);

//Subtraction Operation

int dif = number1 - number2;

System.out.println("Difference is : " + dif);

//Multiplication Operation

int mul = number1 \* number2;

System.out.println("Multiplied value is : " + mul);

//Division Operation

int div = number1 / number2;

System.out.println("Quotient is : " + div);

//Modulus Operation

int rem = number1 % number2;

System.out.println("Remainder is : " + rem);

}

}

**OUTPUT**

Sum is: 16

Difference is : 8

Multiplied value is : 48

Quotient is : 3

Remainder is : 0

**Unary Operators**

In Java, **unary arithmetic operators** are used to increasing or decreasing the value of an operand. Increment operator adds 1 to the value of a variable, whereas the decrement operator decreases a value.

**EXAMPLE**

public class unaryop {

public static void main(String[] args) {

**int r = 6;**

**System.out.println("r=: " + r++);**

**System.out.println("r=: " + r);**

int x = 6;

System.out.println("x=: " + x--);

System.out.println("x=: " + x);

int y = 6;

System.out.println("y=: " + ++y);

int p = 6;

System.out.println("p=: " + --p);

}

}

**OUTPUT**

r=: 6

r=: 7

x=: 6

x=: 5

y=: 7

p=: 5

**Relational Operators**

The **Java Relational operators** compare between operands and determine the relationship between them.

Operator Meaning

== Is equal to

!= Is not equal to

> Greater than

< Less than

>= Greater than or equal to

<= Less than or equal to

**EXAMPLE**

public class relatiop {

public static void main(String[] args) {

//Variables Definition and Initialization

int num1 = 12, num2 = 4;

//is equal to

System.out.println("num1 == num2 = " + (num1 == num2) );

//is not equal to

System.out.println("num1 != num2 = " + (num1 != num2) );

//Greater than

System.out.println("num1 > num2 = " + (num1 > num2) );

//Less than

System.out.println("num1 < num2 = " + (num1 < num2) );

//Greater than or equal to

System.out.println("num1 >= num2 = " + (num1 >= num2) );

//Less than or equal to

System.out.println("num1 <= num2 = " + (num1 <= num2) );

}

}

**OUTPUT**

num1 == num2 = false

num1 != num2 = true

num1 > num2 = true

num1 < num2 = false

num1 >= num2 = true

num1 <= num2 = false

**LOGICAL OPERATOR**

The **Java Logical Operators** work on the Boolean operand. It's also called Boolean logical operators. It operates on two Boolean values, which return Boolean values as a result.

|  |  |  |
| --- | --- | --- |
| Operator | Meaning | Work |
| && | Logical AND | If both operands are true then only "logical AND operator" evaluate true. |
| || | Logical OR | The logical OR operator is only evaluated as true when one of its operands evaluates true. If either or both expressions evaluate to true, then the result is true. |
| ! | Logical Not | Logical NOT is a Unary Operator, it operates on single operands. It reverses the value of operands, if the value is true, then it gives false, and if it is false, then it gives true. |

**EXAMPLE**

public class logicalop {

public static void main(String[] args) {

//Variables Definition and Initialization

boolean bool1 = true, bool2 = false;

//Logical AND

System.out.println("bool1 && bool2 = " + (bool1 && bool2));

//Logical OR

System.out.println("bool1 || bool2 = " + (bool1 | bool2) );

//Logical Not

System.out.println("!(bool1 && bool2) = " + !(bool1 && bool2));

}

}

**OUTPUT**

bool1 && bool2 = false

bool1 || bool2 = true

!(bool1 && bool2) = true

**BITWISE OPERATORS**

The **Java Bitwise Operators** allow access and modification of a particular bit inside a section of the data. It can be applied to integer types and bytes, and cannot be applied to float and double.

|  |  |  |
| --- | --- | --- |
| Operator | Meaning | Work |
| & | Binary AND Operator | There are two types of AND operators in Java: the logical && and the binary &. Binary & operator work very much the same as logical && operators works, except it works with two bits instead of two expressions. The "Binary AND operator" returns 1 if both operands are equal to 1. |
| | | Binary OR Operator | Like "AND operators ", Java has two different "OR" operators: the logical ||and the binary |. Binary | Operator work similar to logical || operators works, except it, works with two bits instead of two expressions. The "Binary OR operator" returns 1 if one of its operands evaluates as 1. if either or both operands evaluate to 1, the result is 1. |
| ^ | Binary XOR Operator | It stands for "exclusive OR" and means "one or the other", but not both. The "Binary XOR operator" returns 1 if and only if exactly one of its operands is 1. If both operands are 1, or both are 0, then the result is 0. |
| ~ | Binary Complement Operator |  |
| << | Binary Left Shift Operator |  |
| >> | Binary Right Shift Operator |  |
| >>> | Shift right zero fill operator |  |

**EXAMPLE**

public class bitwiseop {

public static void main(String[] args) {

//Variables Definition and Initialization

int num1 = 30, num2 = 6, num3 =0;

//Bitwise AND

System.out.println("num1 & num2 = " + (num1 & num2));

//Bitwise OR

System.out.println("num1 | num2 = " + (num1 | num2) );

//Bitwise XOR

System.out.println("num1 ^ num2 = " + (num1 ^ num2) );

//Binary Complement Operator

System.out.println("~num1 = " + ~num1 );

//Binary Left Shift Operator

num3 = num1 << 2;

System.out.println("num1 << 1 = " + num3 );

//Binary Right Shift Operator

num3 = num1 >> 2;

System.out.println("num1 >> 1 = " + num3 );

//Shift right zero fill operator

num3 = num1 >>> 2;

System.out.println("num1 >>> 1 = " + num3 );

}

}

**OUTPUT**

num1 & num2 = 6

num1 | num2 = 30

num1 ^ num2 = 24

~num1 = -31

num1 << 1 = 120

num1 >> 1 = 7

num1 >>> 1 = 7

**ASSIGNMENT OPERATORS**

The **Java Assignment Operators** are used when you want to assign a value to the expression. The assignment operator denoted by the single equal sign **=**

|  |  |
| --- | --- |
| Operator | Meaning |
| += | Increments then assigns |
| -= | Decrements then assigns |
| \*= | Multiplies then assigns |
| /= | Divides then assigns |
| %= | Modulus then assigns |
| <<= | Binary Left Shift  and assigns |
| >>= | Binary Right Shift and assigns |
| >>>= | Shift right zero fill and assigns |
| &= | Binary AND assigns |
| ^= | Binary exclusive OR and assigns |
| |= | Binary inclusive OR and assigns |

**EXAMPLE**

public class AssignmentOperatorDemo {

public static void main(String args[]) {

int num1 = 10;

int num2 = 20;

num2 = num1;

System.out.println("= Output: "+num2);

num2 += num1;

System.out.println("+= Output: "+num2);

num2 -= num1;

System.out.println("-= Output: "+num2);

num2 \*= num1;

System.out.println("\*= Output: "+num2);

num2 /= num1;

System.out.println("/= Output: "+num2);

num2 %= num1;

System.out.println("%= Output: "+num2);

}

}

**O/P**

= Output: 10

+= Output: 20

-= Output: 10

\*= Output: 100

/= Output: 10

%= Output: 0

**CONDITIONAL OPERATOR**

The Java Conditional Operator selects one of two expressions for evaluation, which is based on the value of the first operands. It is also called **ternary operator**because it takes three arguments.

**EXAMPLE**

public class condiop {

public static void main(String[] args) {

String out;

int a = 6, b = 12;

out = *a==*b ? "Yes":"No";

}

}

**OUTPUT**

Ans: No

**Instanceof Operator**

The **Java instanceof Operator** is used to determining whether this object belongs to this particular (class or subclass or interface) or not.

**EXAMPLE**

class Simple1{

public static void main(String args[]){

Simple1 s=new Simple1();

System.out.println(s instanceof Simple1);//true

}

}

**OUTPUT**

true