Operators in Java

Operators are special symbols that perform specific operations on one, two, or three operands, and then return a result

Types of Operator in Java

1) Basic Arithmetic Operators
2) Assignment Operators
3) Auto-increment and Auto-decrement Operators
4) Logical Operators
5) Comparison (relational) operators
6) Bitwise Operators
7) Ternary Operator

1) Basic Arithmetic Operators
Basic arithmetic operators are: +, -, *, /, %
+ is for addition.
– is for subtraction.
* is for multiplication.
/ is for division.
% is for modulo.

Note: Modulo operator returns remainder, for example 10 % 5 would return 0

```
public class ArithmeticOperatorDemo {
  public static void main(String args[]) {
    int num1 = 100;
    int num2 = 20;
    System.out.println("num1 + num2: " + (num1 + num2) );
    System.out.println("num1 - num2: " + (num1 - num2) );
    System.out.println("num1 * num2: " + (num1 * num2) );
    System.out.println("num1 / num2: " + (num1 / num2) );
    System.out.println("num1 % num2: " + (num1 % num2) );
  }
}
```

2) Assignment Operators
Assignments operators in java are: =, +=, -=, *=, /=, %=

num2 = num1 would assign value of variable num1 to the variable.
num2+=num1 is equal to num2 = num2+num1
num2-=num1 is equal to num2 = num2-num1
num2*=num1 is equal to num2 = num2*num1
num2/=num1 is equal to num2 = num2/num1

num2%=num1 is equal to num2 = num2%num1

```
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);
  }
}
```

3) Auto-increment and Auto-decrement Operators

++ and —

num++ is equivalent to num=num+1;
num—- is equivalent to num=num-1;

Example of Auto-increment and Auto-decrement Operators

```
public class AutoOperatorDemo {
  public static void main(String args[]){
    int num1=100;
    int num2=200;
    num1++;
    num2--;
    System.out.println("num1++ is: "+num1);
    System.out.println("num2-- is: "+num2);
  }
}
```

4) Logical Operators

Logical Operators are used with binary variables. They are mainly used in conditional statements and loops for evaluating a condition.

Logical operators in java are: &&, ||, !

Let's say we have two boolean variables b1 and b2.

b1&&b2 will return true if both b1 and b2 are true else it would return false.

b1||b2 will return false if both b1 and b2 are false else it would return true.

!b1 would return the opposite of b1, that means it would be true if b1 is false and it would return false if b1 is true.

Example of Logical Operators

```
public class LogicalOperatorDemo {
  public static void main(String args[]) {
    boolean b1 = true;
    boolean b2 = false;

    System.out.println("b1 && b2: " + (b1&&b2));
    System.out.println("b1 || b2: " + (b1||b2));
    System.out.println("!(b1 && b2): " + !(b1&&b2));
  }
}
```

5) Comparison(Relational) operators
We have six relational operators in Java: ==, !=, >, <, >=, <=

== returns true if both the left side and right side are equal
!= returns true if left side is not equal to the right side of operator.
> returns true if left side is greater than right.
< returns true if left side is less than right side.
>= returns true if left side is greater than or equal to right side.
<= returns true if left side is less than or equal to right side.

Example of Relational operators

```
public class RelationalOperatorDemo {
  public static void main(String args[]) {
    int num1 = 10;
    int num2 = 50;
    if (num1==num2) {
        System.out.println("num1 and num2 are equal");
    }
    else{
```

```
        System.out.println("num1 and num2 are not equal");
    }

    if( num1 != num2 ){
        System.out.println("num1 and num2 are not equal");
    }
    else{
        System.out.println("num1 and num2 are equal");
    }

    if( num1 > num2 ){
        System.out.println("num1 is greater than num2");
    }
    else{
        System.out.println("num1 is not greater than num2");
    }

    if( num1 >= num2 ){
        System.out.println("num1 is greater than or equal to num2");
    }
    else{
        System.out.println("num1 is less than num2");
    }

    if( num1 < num2 ){
        System.out.println("num1 is less than num2");
    }
    else{
        System.out.println("num1 is not less than num2");
    }

    if( num1 <= num2){
        System.out.println("num1 is less than or equal to num2");
    }
    else{
        System.out.println("num1 is greater than num2");
    }
  }
}
```

6) Bitwise Operators

There are six bitwise Operators: &, |, ^, ~, <<, >>

```
num1 = 11; /* equal to 00001011*/
num2 = 22; /* equal to 00010110 */
```

Bitwise operator performs bit by bit processing.
num1 & num2 compares corresponding bits of num1 and num2 and generates 1 if both bits are equal, else it returns 0. In our case it would return: 2 which is 00000010 because in the binary form of num1 and num2 only second last bits are matching.

num1 | num2 compares corresponding bits of num1 and num2 and generates 1 if either bit is 1, else it returns 0. In our case it would return 31 which is 00011111

num1 ^ num2 compares corresponding bits of num1 and num2 and generates 1 if they are not equal, else it returns 0. In our example it would return 29 which is equivalent to 00011101

~num1 is a complement operator that just changes the bit from 0 to 1 and 1 to 0. In our example it would return -12 which is signed 8 bit equivalent to 11110100

num1 << 2 is left shift operator that moves the bits to the left, discards the far left bit, and assigns the rightmost bit a value of 0. In our case output is 44 which is equivalent to 00101100

Note: In the example below we are providing 2 at the right side of this shift operator that is the reason bits are moving two places to the left side. We can change this number and bits would be moved by the number of bits specified on the right side of the operator. Same applies to the right side operator.

num1 >> 2 is right shift operator that moves the bits to the right, discards the far right bit, and assigns the leftmost bit a value of 0. In our case output is 2 which is equivalent to 00000010

Example of Bitwise Operators

```
public class BitwiseOperatorDemo {
  public static void main(String args[]) {

    int num1 = 11;  /* 11 = 00001011 */
    int num2 = 22;  /* 22 = 00010110 */
    int result = 0;

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

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

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

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

    result = num1 << 2;
```

```
    System.out.println("num1 << 2: "+result); result = num1 >> 2;
    System.out.println("num1 >> 2: "+result);
  }
}
```

7) Ternary Operator

This operator evaluates a boolean expression and assign the value based on the result.

Syntax:

variable num1 = (expression) ? value if true : value if false

If the expression results true then the first value before the colon (:) is assigned to the variable num1 else the second value is assigned to the num1.

Example of Ternary Operator

```
public class TernaryOperatorDemo {
  public static void main(String args[]) {
      int num1, num2;
      num1 = 25;
      /* num1 is not equal to 10 that's why
        * the second value after colon is assigned
        * to the variable num2
        */
       num2 = (num1 == 10) ? 100: 200;
       System.out.println( "num2: "+num2);

       /* num1 is equal to 25 that's why
        * the first value is assigned
        * to the variable num2
        */
       num2 = (num1 == 25) ? 100: 200;
       System.out.println( "num2: "+num2);
  }
}
```

Java Operators Precedence and Associativity

Java operators have two properties those are precedence, and associativity. Precedence is the priority order of an operator, if there are two or more operators in an expression then the operator of highest priority will be executed first then higher, and then high. For example, in expression 1 + 2 * 5, multiplication (*) operator will be processed first and then addition. It's because multiplication has higher priority or precedence than addition.

Alternatively, you can say that when an operand is shared by two operators (2 in above example is shared by + and *) then higher priority operator picks the shared operand for processing. From

above example you would have understood the role of precedence or priority in execution of operators. But, the situation may not be as straightforward every time as it is shown in above example. What if all operators in an expression have same priority? In that case the second property associated with an operator comes into play, which is associativity. Associativity tells the direction of execution of operators that can be either left to right or right to left. For example, in expression a = b = c = 8 the assignment operator is executed from right to left that means c will be assigned by 8, then b will be assigned by c, and finally a will be assigned by b. You can parenthesize this expression as (a = (b = (c = 8))).

Note that, you can change the priority of a Java operator by enclosing the lower order priority operator in parentheses but not the associativity. For example, in expression (1 + 2) * 3 addition will be done first because parentheses has higher priority than multiplication operator.

below table presents all Java operators from highest to lowest precedence along with their associativity.

Table : Java operators - precedence chart highest to lowest

Precedence	Operator	Description	Associativity
1	[] () .	array index method call member access	Left -> Right
2	++ -- + - ~ !	pre or postfix increment pre or postfix decrement unary plus, minus bitwise NOT logical NOT	Right -> Left
3	(type cast) new	type cast object creation	Right -> Left
4	* / %	multiplication division modulus (remainder)	Left -> Right
5	+ - +	addition, subtraction string concatenation	Left -> Right
6	<< >> >>>	left shift signed right shift	Left -> Right

		unsigned or zero-fill right shift	
7	< <= > >= instanceof	less than less than or equal to greater than greater than or equal to reference test	Left -> Right
8	== !=	equal to not equal to	Left -> Right
9	&	bitwise AND	Left -> Right
10	^	bitwise XOR	Left -> Right
11	|	bitwise OR	Left -> Right
12	&&	logical AND	Left -> Right
13	||	logical OR	Left -> Right
14	? :	conditional (ternary)	Right -> Left
15	= += -= *= /= %= &= ^= |= <<= >>= >>>=	assignment and short hand assignment operators	Right -> Left