Java - Basic Operators

Java provides a rich set of operators to manipulate variables. We can divide all the Java operators into the following groups −

- Arithmetic Operators
- Relational Operators
- Bitwise Operators
- Logical Operators
- Assignment Operators
- Misc Operators

The Arithmetic Operators

Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra. The following table lists the arithmetic operators −

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

```
public class Test {


  public static void main(String args[]) {

    int a = 10;

    int b = 20;

    int c = 25;

    int d = 25;


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

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

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

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

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

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

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

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


      // Check the difference in d++ and ++d

      System.out.println("d++   = " +  (d++) );

      System.out.println("++d   = " +  (++d) );

   }

}
```

Output

```
a + b = 30
a - b = -10
a * b = 200
b / a = 2
b % a = 0
c % a = 5
a++   = 10
b--   = 11
d++   = 25
++d   = 27
```

The Relational Operators

There are following relational operators supported by Java language.

Assume variable A holds 10 and variable B holds 20, then −

Operator	Description	Example
== (equal to)	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!= (not equal to)	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.

> (greater than)	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.
< (less than)	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>= (greater than or equal to)	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<= (less than or equal to)	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

```
public class Test {

   public static void main(String args[]) {
      int a = 10;
      int b = 20;

      System.out.println("a == b = " + (a == b) );
      System.out.println("a != b = " + (a != b) );
      System.out.println("a > b = " + (a > b) );
      System.out.println("a < b = " + (a < b) );
      System.out.println("b >= a = " + (b >= a) );
      System.out.println("b <= a = " + (b <= a) );
   }
}
```

Output

```
a == b = false
a != b = true
a > b = false
a < b = true
b >= a = true
b <= a = false
```

The Bitwise Operators

Java defines several bitwise operators, which can be applied to the integer types, long, int, short, char, and byte.

Bitwise operator works on bits and performs bit-by-bit operation. Assume if a = 60 and b = 13; now in binary format they will be as follows −

a = 0011 1100

b = 0000 1101

----

a&b = 0000 1100

a|b = 0011 1101

a^b = 0011 0001

~a = 1100 0011

The following table lists the bitwise operators −

Assume integer variable A holds 60 and variable B holds 13 then −

Operator	Description	Example
& (bitwise and)	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100

| (bitwise or)	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61 which is 0011 1101
^ (bitwise XOR)	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~ (bitwise compliment)	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.
<< (left shift)	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>> (right shift)	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 1111
>>> (zero fill right shift)	Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.	A >>>2 will give 15 which is 0000 1111

```
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;       /* 15 = 1111 */

     System.out.println("a >> 2  = " + c );


     c = a >>> 2;      /* 15 = 0000 1111 */

     System.out.println("a >>> 2 = " + c );
   }
}
```

Output

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

The Logical Operators

The following table lists the logical operators −

Assume Boolean variables A holds true and variable B holds false, then −

Operator	Description	Example
&& (logical and)	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false
|| (logical or)	Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true.	(A || B) is true
! (logical not)	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true

```
public class Test {


   public static void main(String args[]) {
      boolean a = true;
```

```
    boolean b = false;


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

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

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

  }

}
```

This will produce the following result −

Output

```
a && b = false
a || b = true
!(a && b) = true
```

The Assignment Operators

Following are the assignment operators supported by Java language −

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand.	C = A + B will assign value of A + B into C
+=	Add AND assignment operator. It adds right operand to the left operand and assign the result to left operand.	C += A is equivalent to C = C + A

-=	Subtract AND assignment operator. It subtracts right operand from the left operand and assign the result to left operand.	C -= A is equivalent to C = C – A
*=	Multiply AND assignment operator. It multiplies right operand with the left operand and assign the result to left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides left operand with the right operand and assign the result to left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator. It takes modulus using two operands and assign the result to left operand.	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2

^=	bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
|=	bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2

```
public class Test {

   public static void main(String args[]) {
      int a = 10;
      int b = 20;
      int c = 0;

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

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

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

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

      a = 10;
```

```
c = 15;

c /= a ;

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


a = 10;

c = 15;

c %= a ;

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


c <<= 2 ;

System.out.println("c <<= 2 = " + c );


c >>= 2 ;

System.out.println("c >>= 2 = " + c );


c >>= 2 ;

System.out.println("c >>= 2 = " + c );


c &= a ;

System.out.println("c &= a  = " + c );


c ^= a ;

System.out.println("c ^= a   = " + c );


c |= a ;

System.out.println("c |= a   = " + c );
```

```
   }

}
```

Output

```
c = a + b = 30
c += a  = 40
c -= a = 30
c *= a = 300
c /= a = 1
c %= a  = 5
c <<= 2 = 20
c >>= 2 = 5
c >>= 2 = 1
c &= a  = 0
c ^= a   = 10
c |= a   = 10
```

Miscellaneous Operators

There are few other operators supported by Java Language.

- **Conditional Operator (? :)**

Conditional operator is also known as the **ternary operator**. This operator consists of three operands and is used to evaluate Boolean expressions. The goal of the operator is to decide, which value should be assigned to the variable. The operator is written as −

```
variable x = (expression) ? value if true : value if false
```

Example

```
public class Test {


   public static void main(String args[]) {

      int a, b;


     a = 10;

      b = (a == 1) ? 20: 30;
      System.out.println( "Value of b is : " +  b );


      b = (a == 10) ? 20: 30;

      System.out.println( "Value of b is : " + b );

   }

}
```

Output

```
Value of b is : 30
Value of b is : 20
```

- **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 an example –

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 );
```

```
  } }
```

Output

```
true
```

This operator will still return true, if the object being compared is the assignment compatible with the type on the right. Following is one more example −

Example

```
class Vehicle {}

public class Car extends Vehicle {

  public static void main(String args[]) {

    Vehicle a = new Car();
    boolean result =  a instanceof Car;
    System.out.println( result );
  }
}
```

Output

```
true
```

Precedence of Java Operators

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator −

For example, x = 7 + 3 * 2; here x is assigned 13, not 20 because operator * has higher precedence than +, so it first gets multiplied with 3 * 2 and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Category	Operator	Associativity
Postfix	>() [] . (dot operator)	Left toright
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