Operators in Java

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

1. [Arithmetic Operators](https://www.geeksforgeeks.org/operators-in-java/#Arithmetic%20Operators)
2. [Unary Operators](https://www.geeksforgeeks.org/operators-in-java/#Unary%20Operators)
3. [Assignment Operator](https://www.geeksforgeeks.org/operators-in-java/#Assignment%20Operator)
4. [Relational Operators](https://www.geeksforgeeks.org/operators-in-java/#Relational%20Operators)
5. [Logical Operators](https://www.geeksforgeeks.org/operators-in-java/#Logical%20Operators)
6. [Ternary Operator](https://www.geeksforgeeks.org/operators-in-java/#Ternary%20Operator)
7. [Bitwise Operators](https://www.geeksforgeeks.org/operators-in-java/#Bitwise%20Operators)
8. [Shift Operators](https://www.geeksforgeeks.org/operators-in-java/#Shift%20Operators)
9. [instance of operator](https://www.geeksforgeeks.org/operators-in-java/#instance%20of%20Operator)
10. [Precedence and Associativity](https://www.geeksforgeeks.org/operators-in-java/#Precedence%20and%20Associativity)

Lets take a look at them in detail.

1. **Arithmetic Operators:** They are used to perform simple arithmetic operations on primitive data types.
   * **\* :**Multiplication
   * **/ :**Division
   * **% :**Modulo
   * **+ :**Addition
   * **– :**Subtraction

|  |
| --- |
| // Java program to illustrate  // arithmetic operators  public class operators  {      public static void main(String[] args)      {          int a = 20, b = 10, c = 0, d = 20, e = 40, f = 30;          String x = "Thank", y = "You";            // + and - operator          System.out.println("a + b = "+(a + b));          System.out.println("a - b = "+(a - b));            // + operator if used with strings          // concatenates the given strings.          System.out.println("x + y = "+x + y);            // \* and / operator          System.out.println("a \* b = "+(a \* b));          System.out.println("a / b = "+(a / b));            // modulo operator gives remainder          // on dividing first operand with second          System.out.println("a % b = "+(a % b));            // if denominator is 0 in division          // then Arithmetic exception is thrown.          // uncommenting below line would throw          // an exception          // System.out.println(a/c);      }  } |

**Output:**

a+b = 30

a-b = 10

x+y = ThankYou

a\*b = 200

a/b = 2

a%b = 0

1. **Unary Operators:** Unary operators needs only one operand. They are used to increment, decrement or negate a value.
   * **– :Unary minus**, used for negating the values.
   * **+ :Unary plus**, used for giving positive values. Only used when deliberately converting a negative value to positive.
   * **++ :Increment operator**, used for incrementing the value by 1. There are two varieties of increment operator.
     + **Post-Increment :**Value is first used for computing the result and then incremented.
     + **Pre-Increment :**Value is incremented first and then result is computed.
   * **— : Decrement operator**, used for decrementing the value by 1. There are two varieties of decrement operator.
     + **Post-decrement :**Value is first used for computing the result and then decremented.
     + **Pre-Decrement :**Value is decremented first and then result is computed.
   * **! : Logical not operator**, used for inverting a boolean value.

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // Java program to illustrate  // unary operators  public class operators  {      public static void main(String[] args)      {          int a = 20, b = 10, c = 0, d = 20, e = 40, f = 30;          boolean condition = true;            // pre-increment operator          // a = a+1 and then c = a;          c = ++a;          System.out.println("Value of c (++a) = " + c);            // post increment operator          // c=b then b=b+1          c = b++;          System.out.println("Value of c (b++) = " + c);            // pre-decrement operator          // d=d-1 then c=d          c = --d;          System.out.println("Value of c (--d) = " + c);            // post-decrement operator          // c=e then e=e-1          c = --e;          System.out.println("Value of c (--e) = " + c);            // Logical not operator          System.out.println("Value of !condition =" + !condition);        }  } |

**Output:**

Value of c (++a) = 21

Value of c (b++) = 10

Value of c (--d) = 19

Value of c (--e) = 39

Value of !condition =false

1. **Assignment Operator : ‘=’** Assignment operator is used to assign a value to any variable. It has a right to left associativity, i.e value given on right hand side of operator is assigned to the variable on the left and therefore right hand side value must be declared before using it or should be a constant.  
   General format of assignment operator is,

variable **=** value;

In many cases assignment operator can be combined with other operators to build a shorter version of statement called **Compound Statement**. For example, instead of a **=** a+5 , we can write a **+=** 5.

* + **+=**, for adding left operand with right operand and then assigning it to variable on the left.
  + **-=**, for subtracting left operand with right operand and then assigning it to variable on the left.
  + **\*=**, for multiplying left operand with right operand and then assigning it to variable on the left.
  + **/=**, for dividing left operand with right operand and then assigning it to variable on the left.
  + **%=**, for assigning modulo of left operand with right operand and then assigning it to variable on the left.

int a = 5;

a += 5; //a = a+5;

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // Java program to illustrate  // assignment operators  public class operators  {      public static void main(String[] args)      {          int a = 20, b = 10, c, d, e = 10, f = 4, g = 9;            // simple assignment operator          c = b;          System.out.println("Value of c = " + c);            // This following statement would throw an exception          // as value of right operand must be initialised          // before assignment, and the program would not          // compile.          // c = d;            // instead of below statements,shorthand          // assignment operators can be used to          // provide same functionality.          a = a + 1;          b = b - 1;          e = e \* 2;          f = f / 2;          System.out.println("a,b,e,f = " + a + ","                             + b + "," + e + "," + f);          a = a - 1;          b = b + 1;          e = e / 2;          f = f \* 2;            // shorthand assignment operator          a += 1;          b -= 1;          e \*= 2;          f /= 2;          System.out.println("a,b,e,f (using shorthand operators)= " +                              a + "," + b + "," + e + "," + f);      }  } |

**Output :**

Value of c =10

a,b,e,f = 21,9,20,2

a,b,e,f (using shorthand operators)= 21,9,20,2

1. **Relational Operators :** These operators are used to check for relations like equality, greater than, less than. They return boolean result after the comparison and are extensively used in looping statements as well as conditional if else statements. General format is,

variable **relation\_operator** value

Some of the relational operators are-

* + **== , Equal to :**returns true of left hand side is equal to right hand side.
  + **!= , Not Equal to :**returns true of left hand side is not equal to right hand side.
  + **< , less than :**returns true of left hand side is less than right hand side.
  + **<= , less than or equal to :**returns true of left hand side is less than or equal to right hand side.
  + **> , Greater than :**returns true of left hand side is greater than right hand side.
  + **>= , Greater than or equal to:**returns true of left hand side is greater than or equal to right hand side.

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // Java program to illustrate  // relational operators  public class operators  {      public static void main(String[] args)      {          int a = 20, b = 10;          String x = "Thank", y = "Thank";          int ar[] = { 1, 2, 3 };          int br[] = { 1, 2, 3 };          boolean condition = true;            //various conditional operators          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("a >= b :" + (a >= b));          System.out.println("a != b :" + (a != b));            // Arrays cannot be compared with          // relational operators because objects          // store references not the value          System.out.println("x == y : " + (ar == br));            System.out.println("condition==true :" + (condition == true));      }  } |

**Output :**

a == b :false

a < b :false

a <= b :false

a > b :true

a >= b :true

a != b :true

x == y : false

condition==true :true

1. **Logical Operators :** These operators are used to perform “logical AND” and “logical OR” operation, i.e. the function similar to AND gate and OR gate in digital electronics. One thing to keep in mind is the second condition is not evaluated if the first one is false, i.e. it has short-circuiting effect. Used extensively to test for several conditions for making a decision.  
   Conditional operators are-
   * **&& , Logical AND :**returns true when both conditions are true.
   * **|| , Logical OR :**returns true if at least one condition is true.

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // Java program to illustrate  // logical operators  public class operators  {      public static void main(String[] args)      {          String x = "Sher";          String y = "Locked";            Scanner s = new Scanner(System.in);          System.out.print("Enter username:");          String uuid = s.next();          System.out.print("Enter password:");          String upwd = s.next();            // Check if user-name and password match or not.          if ((uuid.equals(x) && upwd.equals(y)) ||                  (uuid.equals(y) && upwd.equals(x))) {              System.out.println("Welcome user.");          } else {              System.out.println("Wrong uid or password");          }        }  } |

**Output :**

Enter username:Sher

Enter password:Locked

Welcome user.

1. **Ternary operator :** Ternary operator is a shorthand version of if-else statement. It has three operands and hence the name ternary. General format is-

condition **?** if true **:** if false

The above statement means that if the condition evaluates to true, then execute the statements after the ‘?’ else execute the statements after the ‘:’.

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // Java program to illustrate  // max of three numbers using  // ternary operator.  public class operators  {      public static void main(String[] args)      {          int a = 20, b = 10, c = 30, result;            //result holds max of three          //numbers          result = ((a > b) ? (a > c) ? a :                     c : (b > c) ? b : c);          System.out.println("Max of three numbers = "+result);      }  } |

**Output :**

Max of three numbers = 30

1. **Bitwise Operators :** These operators are used to perform manipulation of individual bits of a number. They can be used with any of the integer types. They are used when performing update and query operations of Binary indexed tree.
   * **& , Bitwise AND operator:**returns bit by bit AND of input values.
   * **| , Bitwise OR operator:**returns bit by bit OR of input values.
   * **^ , Bitwise XOR operator:**returns bit by bit XOR of input values.
   * **~ , Bitwise Complement Operator:**This is a unary operator which returns the one’s compliment representation of the input value, i.e. with all bits inversed.

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // Java program to illustrate  // bitwise operators  public class operators  {      public static void main(String[] args)      {          //if int a = 010 java considers it as octal value of 8 as number starts with 0.          int a = 0x0005;          int b = 0x0007;            // bitwise and          // 0101 & 0111=0101          System.out.println("a&b = " + (a & b));            // bitwise and          // 0101 | 0111=0111          System.out.println("a|b = " + (a | b));            // bitwise xor          // 0101 ^ 0111=0010          System.out.println("a^b = " + (a ^ b));            // bitwise and          // ~0101=1010          System.out.println("~a = " + ~a);            // can also be combined with          // assignment operator to provide shorthand          // assignment          // a=a&b          a &= b;          System.out.println("a= " + a);      }  } |

**Output :**

a&b = 5

a|b = 7

a^b = 2

~a = -6

a= 5

1. **Shift Operators :**These operators are used to shift the bits of a number left or right thereby multiplying or dividing the number by two respectively. They can be used when we have to multiply or divide a number by two. General format-

number **shift\_op** number\_of\_places\_to\_shift;

* + **<< , Left shift operator:**shifts the bits of the number to the left and fills 0 on voids left as a result. Similar effect as of multiplying the number with some power of two.
  + **>> , Signed Right shift operator:**shifts the bits of the number to the right and fills 0 on voids left as a result. The leftmost bit depends on the sign of initial number. Similar effect as of dividing the number with some power of two.
  + **>>> , Unsigned Right shift operator:**shifts the bits of the number to the right and fills 0 on voids left as a result. The leftmost bit is set to 0.

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // Java program to illustrate  // shift operators  public class operators  {      public static void main(String[] args)      {            int a = 0x0005;          int b = -10;            // left shift operator          // 0000 0101<<2 =0001 0100(20)          // similar to 5\*(2^2)          System.out.println("a<<2 = " + (a << 2));            // right shift operator          // 0000 0101 >> 2 =0000 0001(1)          // similar to 5/(2^2)          System.out.println("a>>2 = " + (a >> 2));            // unsigned right shift operator          System.out.println("b>>>2 = "+ (b >>> 2));        }  } |

**Output :**

a2 = 1

b>>>2 = 1073741821

1. **[instance of operator](https://www.geeksforgeeks.org/java-instanceof-and-its-applications/) :** Instance of operator is used for type checking. It can be used to test if an object is an instance of a class, a subclass or an interface. General format-

object **instance of** class/subclass/interface

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // Java program to illustrate  // instance of operator  class operators  {      public static void main(String[] args)      {            Person obj1 = new Person();          Person obj2 = new Boy();            // As obj is of type person, it is not an          // instance of Boy or interface          System.out.println("obj1 instanceof Person: " +                             (obj1 instanceof Person));          System.out.println("obj1 instanceof Boy: " +                             (obj1 instanceof Boy));          System.out.println("obj1 instanceof MyInterface: " +                             (obj1 instanceof MyInterface));            // Since obj2 is of type boy, whose parent class is          // person and it implements the interface Myinterface          // it is instance of all of these classes          System.out.println("obj2 instanceof Person: " +                             (obj2 instanceof Person));          System.out.println("obj2 instanceof Boy: " +                             (obj2 instanceof Boy));          System.out.println("obj2 instanceof MyInterface: " +                             (obj2 instanceof MyInterface));      }  }    class Person  {    }    class Boy extends Person implements MyInterface  {    }    interface MyInterface  {    } |

**Output :**

obj1 instanceof Person: true

obj1 instanceof Boy: false

obj1 instanceof MyInterface: false

obj2 instanceof Person: true

obj2 instanceof Boy: true

obj2 instanceof MyInterface: true

**Precedence and Associativity of Operators**

Precedence and associative rules are used when dealing with hybrid equations involving more than one type of operator. In such cases, these rules determine which part of equation to consider first as there can be many different valuations for the same equation. The below table depicts the precedence of operators in decreasing order as magnitude with top representing the highest precedence and bottom shows lowest precedence.

