Java Operators

Operators is a symbol which is used to perform operations**.** 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
2. Unary Operators
3. Assignment Operator
4. Relational Operators
5. Logical Operators
6. Ternary Operator
7. Bitwise Operators
8. Shift Operators
9. instance of operator

1. **Arithmetic Operators:** They are used to perform simple arithmetic operations on primitive data types.

   - ***:** Multiplication
   - **/:** Division
   - **%:** Modulo
   - **+:** Addition
   - **–:** Subtraction

2. **Unary Operators:** Unary operators need only one operand. They are used to increment, decrement or negate a value.

   a. **–: Unary minus**, used for negating the values.

   b. **++: 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.

   c. **—: 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.

   d. **!: Logical not operator**, used for inverting a boolean value.

3. **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.

Syntax:

```
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.

a. **+=**, for adding left operand with right operand and then assigning it to variable on the left.
b. **-=**, for subtracting left operand with right operand and then assigning it to variable on the left.
c. ***=**, for multiplying left operand with right operand and then assigning it to variable on the left.
d. **/=**, for dividing left operand with right operand and then assigning it to variable on the left.
e. **%=**, 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;
```

4. **<u>Relational Operators</u>:** 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-

a**. ==, Equal to:** returns true of left-hand side is equal to right hand side.
b. **!=, Not Equal to :** returns true of left hand side is not equal to right hand side.
c**. <, less than:** returns true of left-hand side is less than right hand side.
d**.<=, less than or equal to:** returns true of left-hand side is less than or equal to right hand side.
e**. >, Greater than:** returns true of left-hand side is greater than right hand side.
f**. >=, Greater than or equal to:** returns true of left-hand side is greater than or equal to right hand side.

5. **<u>Logical Operators</u>:** 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

Conditional operators are-

a. **&&, Logical AND:** returns true when both conditions are true.

b. **||, Logical OR:** returns true if at least one condition is true.

6. **Ternary operator:** Ternary operator is a shorthand version of if-else statement. It has three operands and hence the name ternary.
   Syntax:
   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 ':'.

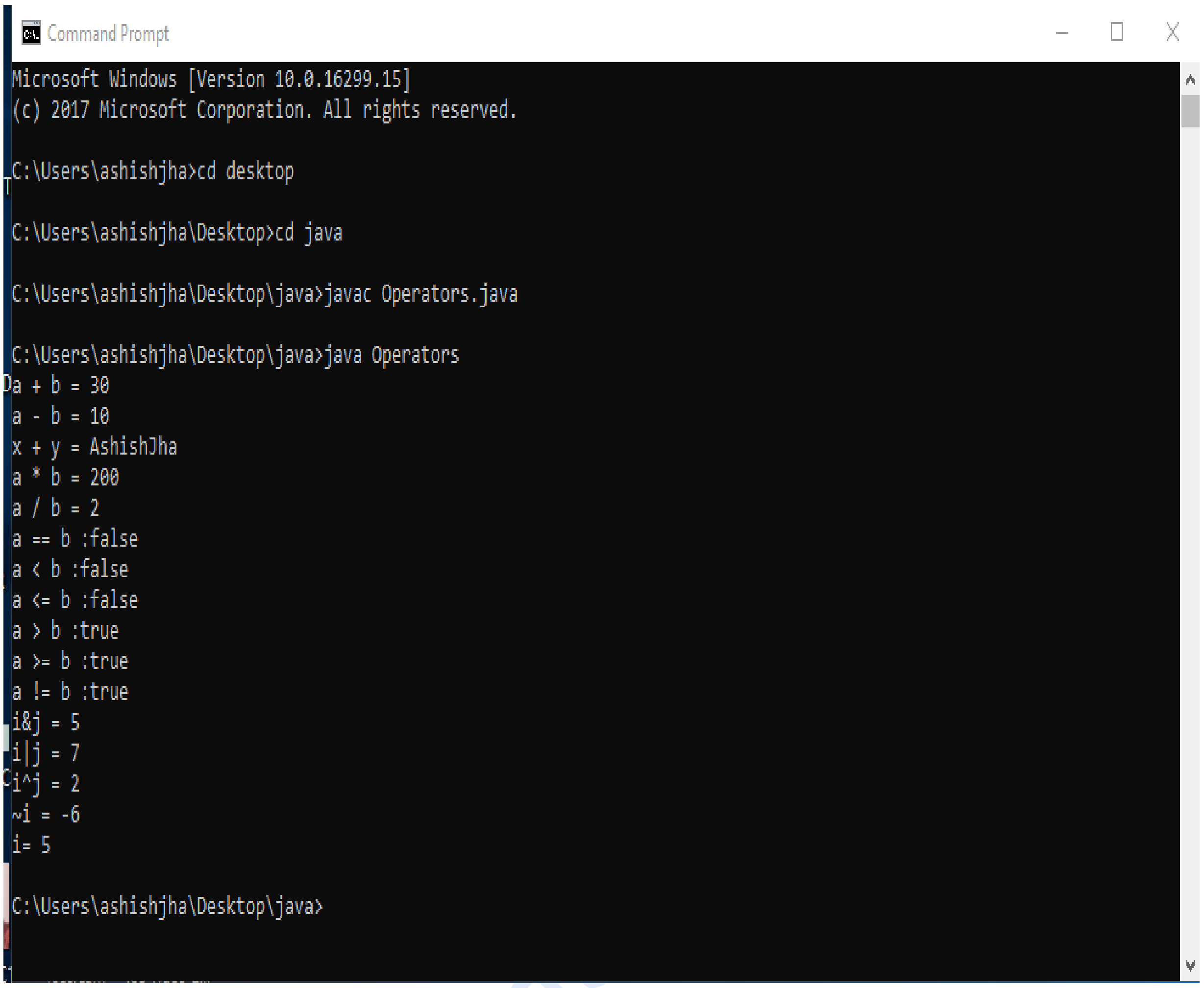
7. **Bitwise Operators:** These operators are used to perform manipulation of individual bits of a number. They can be used with any of the integer types. They are used when performing update and query operations of Binary indexed tree.
   - **&, Bitwise AND operator:** returns bit by bit AND of input 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.

8. **Instance of Operators:** 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.
   Syntax:

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

   Ex:-

```
// arithmetic operators
public class Operators {
    public static void main(String[] args)
    {
        int a = 20, b = 10;
        String x = "Ashish", y = "Jha";

        // + 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));


    // relational operators

        // 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));



      // bitwise operators
          // if int a = 010
          // Java considers it as octal value
          // of 8 as number starts with 0.
          int i = 0x0005;
          int j = 0x0007;

          // bitwise and
          // 0101 & 0111=0101
          System.out.println("i&j = " + (i & j));

          // bitwise and
          // 0101 | 0111=0111
          System.out.println("i|j = " + (i | j));

          // bitwise xor
          // 0101 ^ 0111=0010
          System.out.println("i^j = " + (i ^ j));

          // bitwise and
          // ~0101=1010
          System.out.println("~i = " + ~i);

          // can also be combined with
          // assignment operator to provide shorthand
          // assignment
          // i=i&j
          i &= j;
          System.out.println("i= " + i);
        }
      }
```



```
Microsoft Windows [Version 10.0.16299.15]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\ashishjha>cd desktop

C:\Users\ashishjha\Desktop>cd java

C:\Users\ashishjha\Desktop\java>javac Operators.java

C:\Users\ashishjha\Desktop\java>java Operators
a + b = 30
a - b = 10
x + y = AshishJha
a * b = 200
a / b = 2
a == b :false
a < b :false
a <= b :false
a > b :true
a >= b :true
a != b :true
i&j = 5
i|j = 7
i^j = 2
~i = -6
i= 5

C:\Users\ashishjha\Desktop\java>
```