Bitwise Operator in Java

1. **Bitwise AND (&):** The bitwise AND & operator returns 1 if and only if both the operands are 1. Otherwise, it returns 0.

2. **Bitwise inclusive OR (|):** The bitwise OR | operator returns 1 if at least one of the operands is 1. Otherwise, it returns 0.

3. **Bitwise exclusive OR (^):** It returns 0 if both bits are the same, else returns 1.

4. **Bitwise Compliment (~):** The bitwise complement operator is a unary operator (works with only one operand). It is denoted by ~. It is important to note that the bitwise complement of any integer N is equal to - (N + 1).

```
class Main {
  public static void main(String[] args) {
    int y = 9;
    // bitwise compliment
    // ~y = ~9(1001) = 1010 = 2's compliment of -10 (output)
    System.out.println("~y = " + (~y)); // print -10

  }
}
```

5. **Bitwise left shift (<<):** The left shift operator shifts the bits to the left by the number of times specified by the right side of the operand. After the left shift, the empty space in the right is filled with 0. [i.e. **a << b = a * 2^b**]

```
class Main {
  public static void main(String[] args) {
    int y = 9;
    // bitwise left shift
    // y = 9= 1001<<2 = 100100 = 36
    System.out.println("y<<2 = " + (y<<2)); // print 36
  }
}
```

6. **Bitwise signed right shift (>>):** When we shift any number to the right, the least significant bits (rightmost) are discarded, and the most significant position (leftmost) is filled with the sign bit. [i.e. **a >> b = a / 2^b**]

```java
class Main {
  public static void main(String[] args) {
    int y = 9;
    int x = -9;
    // bitwise signed right shift
    // y = 9= 1001 >> 2 = 0010 = 2
    //x= -9= 0111 >> 2= 1101 = 2's complement of -3 or the binary representation of -3
    System.out.println("y>>2 = " + (y>>2)); // print 2
    System.out.println("x>>2 = " + (x>>2)); // print -3
  }
}
```

7. **Bitwise unsigned right shift (>>>):** Java also provides an unsigned right shift. It is denoted by >>>. Here, the vacant leftmost position is filled with 0 instead of the sign bit.

```java
class Main {
  public static void main(String[] args) {
    int y = 9;
    int x = -9;
    // bitwise unsigned right shift (for positive numbers)
    // y = 9= 1001 >>> 2 = 0010 = 2
    System.out.println("y>>>2 = " + (y>>>2)); // print 2
    System.out.println("x>>>2 = " + (x>>>2)); // print 1073741821 (arbitrary number)
  }
}
```

N.B.

i. Extract the rightmost bit using bitwise AND operation

EX:

10100011

& 1

00000001

ii. Flipped the bits using bitwise exclusive OR

EX:

x = 1 0

mask = 1 1

XOR 0 1 (flipped with bitmask)

iii. The expression x & (x - 1) clears the lowest set bit in x

EX:

16 & (16 - 1) = 10000 & 01111 = 00000 (It clears the right most set bit)

11 & (11 - 1) = 10, and

20 & (20 - 1) = 16,

iv. The expression x & ~(x - 1) extracts the lowest set bit of x

EX:

16 & ~(16 - 1) = 16,

11 & ~(11 - 1) = 1, and

20 & ~(20 - 1) = 4.

Problem

Computing the number of bits set to 1 in an integer-valued variable x.

Solution 1

- Iteratively test individual bits using the value 1 as a bitmask.
- Specifically, we iteratively identify bits of x that are set to 1 by examining the bitwise- AND of x with the bitmask, shifting x right one bit at a time.
- The overall complexity is O(n) where n is the length of the integer.

Solution 1

```
public class CountBits {
    public static short countBits(int x) {
        short numBits = 0;
        while (x != 0) {
            numBits += (x & 1) ;
            x >>>= 1;
        }
        return numBits;
    }

    public static void main(String[] args) {
        int x=0b001101;
        System.out.println(countBits(x));
    }
}

Output: 3
```

Solution 2

- Computing y = x & ~ (x−1), where & is the bitwise-AND operator and ~ is the bitwise complement operator.
- The variable y is 1 at exactly the lowest bit of x that is 1; all other bits in y are 0.
- For example, if x = (00101100)2, then x-1 = $(00101011)_2$, ~(x - 1) = $(11010100)_2$, and y = $(00101100)_2$ & $(11010100)_2$ = $(00000100)_2$.
- Consequently, this bit may be removed from x by computing x ⊕ y, where ⊕ is the bitwise-XOR function.
- The time complexity is O(s), where s is the number of bits set to 1 in x.

Solution 2

```
public static int countBits1(int x) {
    int y=0,count=0;
    while (x != 0) {
        y= x & ~(x-1);
        x = x ^ y;
        count=count+1;
    }
    return count;
}
```