Bitwise Operators in Java

Bitwise operators are used to performing the manipulation of individual bits of a number. They can be used with any integral type (char, short, int, etc.). They are used when performing update and query operations of the Binary indexed trees.

Now let's look at each one of the bitwise operators in Java:

1. Bitwise OR (|)

This operator is a binary operator, denoted by '|'. It returns bit by bit OR of input values, i.e., if either of the bits is 1, it gives 1, else it shows 0.

Example:

a = 5 = 0101 (In Binary)
b = 7 = 0111 (In Binary)

Bitwise OR Operation of 5 and 7
0101
| 0111
_____
0111 = 7 (In decimal)

2. Bitwise AND (&)

This operator is a binary operator, denoted by '&.' It returns bit by bit AND of input values, i.e., if both bits are 1, it gives 1, else it shows 0.

Example:

a = 5 = 0101 (In Binary)
b = 7 = 0111 (In Binary)

Bitwise AND Operation of 5 and 7

0101

& 0111

_____

0101 = 5 (In decimal)

3. Bitwise XOR (^)

This operator is a binary operator, denoted by '^.' It returns bit by bit XOR of input values, i.e., if corresponding bits are different, it gives 1, else it shows 0.

Example:

a = 5 = 0101 (In Binary)

b = 7 = 0111 (In Binary)

Bitwise XOR Operation of 5 and 7

0101

^ 0111

_____

0010 = 2 (In decimal)

4. Bitwise Complement (~)

This operator is a unary operator, denoted by '~.' It returns the one's complement representation of the input value, i.e., with all bits inverted, which means it makes every 0 to 1, and every 1 to 0.

Example:

a = 5 = 0101 (In Binary)

Bitwise Complement Operation of 5

~ 0101

1010 = 10 (In decimal)

Note: Compiler will give 2's complement of that number, i.e., 2's complement of 10 will be -6.

```
// Java program to illustrate
// bitwise operators

class GfG {
    public static void main(String[] args)
    {
        // Initial values
        int a = 5;
        int b = 7;

        // bitwise and
        // 0101 & 0111=0101 = 5
        System.out.println("a&b = " + (a & b));

        // bitwise or
        // 0101 | 0111=0111 = 7
        System.out.println("a|b = " + (a | b));

        // bitwise xor
        // 0101 ^ 0111=0010 = 2
        System.out.println("a^b = " + (a ^ b));

        // bitwise not
        // ~00000000 00000000 00000000 00000101=11111111 11111111 11111111
11111010
        // will give 1's complement (32 bit) of 5 = -6
        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
```

```
// Demonstrating the bitwise logical operators


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

        String binary[]={
          "0000","0001","0010","0011","0100","0101",
          "0110","0111","1000","1001","1010",
          "1011","1100","1101","1110","1111"
        };

      // initializing the values of a and b
      int a=3; // 0+2+1 or 0011 in binary
      int b=6; // 4+2+0 or 0110 in binary

      // bitwise or
      int c= a | b;

      // bitwise and
      int d= a & b;

      // bitwise xor
      int e= a ^ b;

      // bitwise not
      int f= (~a & b)|(a &~b);
      int g= ~a & 0x0f;


      System.out.println(" a= "+binary[a]);
      System.out.println(" b= "+binary[b]);
      System.out.println(" a|b= "+binary[c]);
      System.out.println(" a&b= "+binary[d]);
      System.out.println(" a^b= "+binary[e]);
      System.out.println("~a & b|a&~b= "+binary[f]);
      System.out.println("~a= "+binary[g]);
    }
}
```

Output

```
a= 0011
 b= 0110
 a|b= 0111
 a&b= 0010
 a^b= 0101
~a & b|a&~b= 0101
~a= 1100
```

Bit-Shift Operators (Shift Operators)

Shift 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.

Syntax:

number shift_op number_of_places_to_shift;

Types of Shift Operators:

Shift Operators are further divided into 4 types. These are:

1. Left shift operator (<<)
2. Unsigned Left shift operator (<<<)
3. Signed Right shift operator (>>)
4. Unsigned Right shift operator (>>>)

Note: For more detail about the Shift Operators in Java, refer Shift Operator in Java.

Left shift operator (<<)

The left shift operator (<<) is a bitwise operator that shifts the bits of a number to the left by a specified number of positions. It effectively multiplies the number by 2 raised to the

power of the specified shift count. This operator is often used for low-level programming, bit manipulation, and performance optimization.

result = number << shiftCount;

- `number`: The integer value whose bits are to be shifted.
- `shiftCount`: The number of positions to shift the bits to the left.

Here an Example of Positive Number Left Shift:

```
public class GfG {
    public static void main(String[] args) {
        int x = 3;

        // Shift left by 1
        System.out.println(x << 1);

        // Shift left by 2
        System.out.println(x << 2);
    }
}
```

Output
```
6
12
```

Explanation:

- When `3` (binary `0000 0011`) is shifted left by 1, the result is `6` (binary `0000 0110`).
- When shifted left by 2, the result is `12` (binary `0000 1100`).
- Each left shift effectively multiplies the value by `2`.

Example of Comparing Two Numbers:

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

```
        int x = 3;
        int y = 4;

        // Compare using left shift
        int z = (x << y);
        System.out.println(z);
    }
}
```

Output
`48`

Explanation:

- `3` (binary `0000 0011`) is shifted left by `4`, resulting in `48` (binary `0011 0000`).
- This demonstrates how left shift can be used to scale numbers by powers of 2.

Example of Negative Number Left Shift:

```
public class GfG {
    public static void main(String[] args) {
        int x = -1;

        // Shift left by 1
        System.out.println(x << 1);
    }
}
```

Output
`-2`

Explanation:

- The value `-1` is represented in binary as all 1s (`1111 1111` in 8-bit 2's complement representation).
- Shifting left by 1 results in `-2` (binary `1111 1110`).
- The sign bit (leftmost bit) remains `1` for negative numbers, maintaining the number's sign.