Bitwise Right Shift Operators in Java

In C/C++ there is only one right shift operator '>>' which should be used only for positive integers or unsigned integers. Use of the right shift operator for negative numbers is not recommended in C/C++, and when used for negative numbers, the output is compiler dependent. Unlike C++, Java supports following two right shift operators.

Here we will be discussing both of right shift operators as listed:

- Signed right shift ">>"
- Unsigned right shift ">>>"

Type 1: Signed Right Shift

In Java, the operator '>>' is signed right shift operator. All integers are signed in Java, and it is fine to use >> for negative numbers. The operator '>>' uses the sign bit (leftmost bit) to fill the trailing positions after the shift. If the number is negative, then 1 is used as a filler and if the number is positive, then 0 is used as a filler. For example, if the binary representation of a number is 10....100, then right shifting it by 2 using >> will make it 11.....1.

Example:

```
// Java Program to Illustrate Signed Right Shift Operator

// Main class
class GfG {

    // Main driver method
    public static void main(String args[]) {

        int x = -4;
```

```
        System.out.println(x >> 1);

        int y = 4;
        System.out.println(y >> 1);
    }
}
```

Output

```
-2
2
```

Explanation:

- Negative Number (`x = -4`):
  - Binary representation of `-4` (in 8-bit, 2's complement): `11111100`.
  - When shifted right by 1: `11111110` (sign bit `1` is preserved).
  - Decimal value: `-2`.
- Positive Number (`y = 4`):
  - Binary representation of `4` (in 8-bit): `00000100`.
  - When shifted right by 1: `00000010`.
  - Decimal value: `2`.

Type 2: Unsigned Right Shift Operator

In Java, the operator '>>>' denotes unsigned right shift operator and always fill 0 irrespective of the sign of the number.

Example:

```
// Java Program to Illustrate Unsigned Right Shift Operator

// Main class
class GfG {

    // main driver method
    public static void main(String args[]) {

        // x is stored using 32 bit 2's complement form.
        // Binary representation of -1 is all 1s (111..1)
        int x = -1;

        // The value of 'x>>>29' is 00...0111
        System.out.println(x >>> 29);

        // The value of 'x>>>30' is 00...0011
        System.out.println(x >>> 30);

        // The value of 'x>>>31' is 00...0001
        System.out.println(x >>> 31);
    }
}
```

Output

```
7
3
```

- `1`

Explanation:

- Initial Value (`x = -1`):
    - Binary representation of `-1` (32-bit): `11111111111111111111111111111111`.
- Right Shift by 29 Positions (`x >>> 29`):
    - Resulting binary: `00000000000000000000000000000111` (last 3 bits remain).
    - Decimal value: `7`.
- Right Shift by 30 Positions (`x >>> 30`):
    - Resulting binary: `00000000000000000000000000000011` (last 2 bits remain).
    - Decimal value: `3`.
- Right Shift by 31 Positions (`x >>> 31`):
    - Resulting binary: `00000000000000000000000000000001` (last 1 bit remains).
    - Decimal value: `1`.