**Operators in Java - 2022**

**Right Shift Operator >>**

**a >> b 🡺 a / 2 b**

**3 >> 4 🡺 3 / 2 4 🡺 3 /48 = 0**

**16 >> 3 🡺 16 / 2 3 🡺 16 / 8 = 2**

**Left Shift Operator <<**

**In case of Integer 🡺** a << b 🡺 **a \* 2 b Mod 32**

**In case of long 🡺 a << b 🡺 a \* 2 b Mod 64**

**a << b -> a \* 2 b Mod 32 🡺 a \* 2 (b % 32) 🡺 a \* 2b**

**3 << 4 = 3 \* 2 4 MOD 32 🡺 3 \* 24 🡺3 \* 8 = 48**

**3 << 32 = 3 \* 2 32 Mod 32 🡺 3 \* 20 🡺3 \* 1 = 3**

**XOR ^ in Java**

**More simply, you can also think of xor as "this or that, but not both!".**

Let's take 5^6 as example:

(decimal) (binary)

5 = 101

6 = 110

------------- xor

3 = 011

Table

Description automatically generated A picture containing text, clock

Description automatically generated

[**Real world use cases of bitwise operators**](http://stackoverflow.com/questions/2096916/real-world-use-cases-of-bitwise-operators)

if you have let's say 4 discrete permissions (read, write, execute, change policy), it's better to store this in 1 byte rather than waste 4.

**Communication over ports/sockets:** involves checksums, parity, stop bits

**Compression, Encryption**

|  |  |  |
| --- | --- | --- |
| Table 3.1: Integer Types and Their Representations | | |
| **Type** | **Representation** | **Range** |
| byte | 8-bit, signed, two's complement | -128 to 127 |
| short | 16-bit, signed, two's complement | -32768 to 32767 |
| int | 32-bit, signed, two's complement | -2147483648 to 2147483647 |
| long | 64-bit, signed, two's complement | -9223372036854775808 to 9223372036854775807 |
| char | 16-bit, unsigned, Unicode | '\u0000' to '\uffff' |