The Bitwise Operators:

Java defines several bitwise operators, which can be applied to the integer types, long, int, short, char, and byte.

Bitwise operator works on bits and performs bit-by-bit operation. Assume if a = 60; and b = 13; now in binary format they will be as follows:

a = 0011 1100

b = 0000 1101

---

a&b = 0000 1100

a|b = 0011 1101

a^b = 0011 0001

~a = 1100 0011

The following table lists the bitwise operators:

Assume integer variable A holds 60 and variable B holds 13 then:

SR.NO	Operator and Description
1	**& (bitwise and)** Binary AND Operator copies a bit to the result if it exists in both operands. **Example:** (A & B) will give 12 which is 0000 1100
2	**| (bitwise or)** Binary OR Operator copies a bit if it exists in either operand.

	Example: (A | B) will give 61 which is 0011 1101
3	**^ (bitwise XOR)** Binary XOR Operator copies the bit if it is set in one operand but not both. **Example:** (A ^ B) will give 49 which is 0011 0001
4	**~ (bitwise compliment)** Binary Ones Complement Operator is unary and has the effect of 'flipping' bits. **Example:** (~A) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.
5	**<< (left shift)** Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand **Example:** A << 2 will give 240 which is 1111 0000
6	**>> (right shift)** Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand. **Example:** A >> 2 will give 15 which is 1111
7	**>>> (zero fill right shift)** Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros. **Example:** A >>>2 will give 15 which is 0000 1111

```java
public class Test {

  public static void main(String args[]) {
     int a = 60;   /* 60 = 0011 1100 */
     int b = 13;   /* 13 = 0000 1101 */
     int c = 0;

     c = a & b;       /* 12 = 0000 1100 */
     System.out.println("a & b = " + c );

     c = a | b;       /* 61 = 0011 1101 */
     System.out.println("a | b = " + c );

     c = a ^ b;       /* 49 = 0011 0001 */
     System.out.println("a ^ b = " + c );

     c = ~a;          /*-61 = 1100 0011 */
     System.out.println("~a = " + c );

     c = a << 2;     /* 240 = 1111 0000 */
     System.out.println("a << 2 = " + c );

     c = a >> 2;     /* 15 = 1111 */
     System.out.println("a >> 2  = " + c );

     c = a >>> 2;     /* 15 = 0000 1111 */
     System.out.println("a >>> 2 = " + c );
  }
}
```

Hexadecimal numbers:

Hexadecimal (or "hex" for short) is a numbering system which works similarly to our regular decimal system, but where a single digit can take a value of 0-15 rather than 0-9. The extra digits are represented by the letters A-F, as shown in the table below.

They look the same as the decimal numbers up to 9, but then there are the letters ("A","B","C","D","E","F") in place of the decimal numbers 10 to 15.

So a single Hexadecimal digit can show 16 different values instead of the normal 10 like this:

Decimal:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hexadecimal:	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

How to write a hex number in Java code

Sometimes it is convenient to represent a number in hex in your Java code. There are generally two conventions for writing hex numbers. The first is to write *h* after the hex number. For example, **400h** is the equivalent of 1024 decimal.

The second convention is to put **0x** before the number, e.g. **0x400**. **This is the convenion used by Java**:

```
int noBytes = 0x400; // = 1024
```