**BITWISE COMPLIMENT:**

It is a unary operator that operates on a single operand. It inverts all the bits of the operand. For example, the bitwise complement of 5 (which is 0101 in binary) is -6 (which is 1010 in binary).

**LOGICAL COMPLIMENT:**

It is also a unary operator that operates on a single operand. It inverts the value of the operand. For example, the logical complement of true is false, and the logical complement of false is true.

**DIFFERENCE B/W BITWISE AND LOGICAL COMPLIMENTS:**

* The main difference between bitwise complement and logical complement is the type of operand they operate on and the result they produce.
* Bitwise complement (~) is a unary operator that operates on a single operand and inverts all the bits of the operand. It is used to perform bitwise operations such as bit shifting and masking. The result of the bitwise complement is an integer value.
* Logical complement (!) is also a unary operator that operates on a single operand and inverts the value of the operand. It is used to perform logical operations such as negating a condition or checking for inequality. The result of the logical complement is a boolean value.
* In summary, bitwise complement is used to manipulate the bits of an integer value, while logical complement is used to manipulate the truth value of a boolean expression.

public class Main {

public static void main(String[] args) {

int a = 42;

int b = ~a; // bitwise complement

boolean c = true;

boolean d = !c; // logical complement

System.out.println("Bitwise complement of " + a + " is " + b);

System.out.println("Logical complement of " + c + " is " + d);

}

}

**RANGE OF FLOAT AND DOUBLE**:

* In Java, the range of float is approximately 3.4E-38 to 3.4E+38, and the range of double is approximately 1.7E-308 to 1.7E+308.
* The range of float is smaller than the range of double, but float requires less memory than double. Float is a 32-bit floating-point type, while double is a 64-bit floating-point type.
* It is important to note that the actual range of float and double may vary depending on the implementation of the Java Virtual Machine (JVM).

public class Main {

public static void main(String[] args) {

float a = 3.14f; // float variable

double b = 3.14159; // double variable

System.out.println("The value of a is " + a);

System.out.println("The value of b is " + b);

}

}

***In LONG why we have to initiate with 123L ....if the outer range of long increases, which data type we have to take?***

* In Java, the "L" at the end of a number is used to indicate that it should be treated as a long data type. The long data type is a 64-bit signed two's complement integer, which means it can hold values ranging from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.
* When you declare a long variable, you can assign it a value using the "L" suffix to indicate that the value should be treated as a long. For example, you can declare and initialize a long variable like this:
* long myLong = 123L;
* In this case, the "L" at the end of the number 123 indicates that it should be treated as a long.
* If the range of the long data type is not sufficient for your needs, you can use the BigInteger class, which is part of the Java standard library. The BigInteger class provides arbitrary-precision integers, which means they can hold integers of any size. Here is an example of how you can use the BigInteger class:

import java.math.BigInteger;

public class Main {

public static void main(String[] args) {

BigInteger myBigInt = new BigInteger("12345678901234567890");

System.out.println(myBigInt);

}

}

In this example, we create a BigInteger object and assign it the value "12345678901234567890". The BigInteger class provides a wide range of methods for performing arithmetic operations on large integers.

***How to store the data if more than the size of long?***

* If you need to store data that is larger than the maximum value of a long data type, you can use other data types or data structures to store the data. Some examples of data types that can be used to store larger numbers include BigInteger and BigDecimal.
* BigInteger represents an arbitrarily large integer and can be used to store integers of any size. BigDecimal represents an arbitrarily large decimal number and can be used to store decimal numbers of any size.
* Alternatively, you can use data structures such as arrays, lists, or maps to store large amounts of data. The choice of data structure will depend on the specific requirements of your program and the type of data you need to store.