UNARY OPERATOR ASSIGNMENT

Submited by

Harish Kumbhar

1. Unary NOT [!] Operator

Logical NOT [!] is a unary operator that only accepts one operand. It flips the operand values, returning **true** if the value is **false** and **false** if it is **true**.

Example:

```
public class UnaryNot {
   public static void main(String[] args) {
      boolean a = true;
      boolean b = false;

      System.out.println(!a);
      System.out.println(!b);
   }
}

Output:
false
true
```

2. Complementary [~] Operator

It's important to note that the result of the bitwise complement operation depends on the number of bits used to represent the integer type. In this case, int is a 32-bit signed integer type.

Bitwise complement representation of a = 10:

```
 a     = 00000000 00000000 00000000 00001010 (binary)
~a     = 11111111 11111111 11111111 11110101 (bitwise)
```

Mathematical representation **- (n + 1).**
If n = 10 then,
~n = - (n + 1)
~n = - (10+1)
~n = -11

Example:

```
public class UnaryComplementary {
   public static void main(String[] args) {
      int a = 10;

      System.out.println(~a);
   }
}

Output:
-11
```

3. Maximum Integer Range in Java

In Java, if you need to store integer values that are greater than the range of the long data type, you can use the **BigInteger** class from the **java.math** package. BigInteger provides arbitrary precision arithmetic, allowing you to work with integers of any size.

It provides support for operations on large integers that are beyond the range of the built-in numeric data types such as int, long, or double.

Example:

```
import java.math.BigInteger;

public class BigInt {
    public static void main(String[] args) {
                 BigInteger    bigNum    =    new    BigInteger(
"12345678901234567890397669796775987960910580617369797469070
769576976963967475934797367596795769767596795769756956095760
975675769547690750960609095670927509675096709560947597094592
097092709746207672067245694560721234567890");
        System.out.println(bigNum);
    }
}
```

4. Range of float and double

float: The float data type is a 32-bit floating-point number. It has a range of approximately **±3.40282347E+38** and can represent values with a precision of about **7 decimal digits**.

double: The double data type is a 64-bit floating-point number. It has a range of approximately **±1.79769313486231570E+308** and can represent values with a precision of about **15 decimal digits**.

Precision: double has a higher precision than float. It can represent numbers with approximately 15 decimal digits of precision, while float can represent numbers with about 7 decimal digits of precision. This means that double is generally more accurate for storing and performing calculations with decimal values.

Example:

```
public class FloatAndDouble {
    public static void main(String[] args) {
        float pi = 3.142f;
        double PI = 3.14159265359;

        System.out.println(pi);
        System.out.println(PI);
    }
}

Output:
3.142
3.14159265359
```