Java Tricky Programming questions Part 2

Sometime back I wrote an article with 5 java tricky programming questions and my friends liked it a lot.

Recently I got two java questions that I will be explaining here.

Java Programming Question 1

What is the output of the below program?

```
public class Test {
    public static void main(String[] args) {
        method(null);
    }
    public static void method(Object o) {
        System.out.println("Object impl");
    }
    public static void method(String s) {
        System.out.println("String impl");
    }
}
```

Java Programming Question 2

What will below statements print?

```
long longWithL = 1000*60*60*24*365L;
long longWithoutL = 1000*60*60*24*365;
System.out.println(longWithL);
System.out.println(longWithoutL);
```

****

Java Programming Question 1 Answer with Explanation

As we know that we can assign null to any object, so doesn't compiler complains about this program? According to java specs, in case of overloading, compiler picks the *most specific function*. Obviously String class is more specific that Object class, hence it will print "String impl".
What if we have another method in the class like below:

```
public static void method(StringBuffer i){
        System.out.println("StringBuffer impl");
    }
```

In this case, java compiler will throw error as "The method method(String) is ambiguous for the type Test" because String and StringBuffer, none of them are more specific to others. A method is

more specific than another if any invocation handled by the first method could be passed on to the other one without a compile-time type error. We can pass String as parameter to Object argument and String argument but not to StringBuffer argument method.

Java Programming Question 2 Answer with Explanation

The output of the code snippet will be:

```
31536000000
1471228928
```

In case of first variable, we are explicitly making it a long by placing a “L” at the end, so compiler will treat this at long and assign it to first variable.
In second case, compiler will do the calculation and treat it as a 32-bit integer, since the output is outside the range of integer max value (2147483647), compiler will truncate the most significant bits and then assign it to the variable.

Binary equivalent of 1000*60*60*24*365L = 0111010101111011000100101110000000000 (36 bits)
Removing 4 most significant bits to accommodate in 32-bit int, value = 01010111101100010010110000000000 (32 bits)
Which is equal to 1471228928 and hence the output.