International Journal of Engineering and Information Systems (IJEAIS)
ISSN: 2643-640X
Vol. 4, Issue 7, July – 2020, Pages: 110-113

The Practical Importance of Using Literals in Java Script

[1]Olimjonova Saodat G'ulomjon qizi, [2]Dus'yorova Dildora Nurillo qizi, [3]Amirov Qodir

[1,2,3] Student of TUIT branch of Samarkand,Samarkand, Uzbekistan.
E-mail.address: solimjonova95@mail.ru

Abstract: *This article discusses literals, one of the features of the java script programming language. Literals are numeric values and are of great practical importance to the programmer when working with comma numbers. Examples are given in the programmatic section where the literals and instructions for their use are given.*

Keywords: JavaScript, Literals, data types, values, and variables, Java's floaters

INTRODUCTION

Literal - values written directly in the program text. Let's look at some examples together.

In JavaScript, data types can be divided into two classes: simple types and objects. The simplest types of data include numbers,

```
12              // Twelve (literal)
1.2             // One whole twelve digits (numerical literal)
"Hello World"   // Literary line
'Hi'            // The other line is literal
true            // The logical value is literal
false           // Another logical value is literal
/javascript/gi  //  Regular expression literal (to search by
                template)
null            // Empty object literal
```

rows, and logical values.**null** and **undefined** elementary values. But they are not related to any number, line, or matrix value. Each of them defines a unique value of its own type. In JavaScript, an object is a number, a string, a Boolean, and any value that is null or undefined. An object contains properties, each of which has a name and a value (as long as the object belongs to a class).In JavaScript, variables do not have a type: you can assign a value of any type to a variable, and then assign another value to it.

In JavaScript, there is a difference between simple values (undefined, null, logical values, numbers, and strings) and objects (including arrays and functions). Normal values do not change: normal values cannot be changed. This is true for logical and numerical values - for example, it is unthinkable to change the value of a number. As for the lines, some questions may arise. After all, lines are an array of characters. This means that you can change the character of the line. However, in practice, JavaScript does not allow this. Instead, all string methods in JavaScript return a changed string, or more precisely, a new string value.

For example:

```
var s = "hello"; // A line consisting of several characters
s.toUpperCase()  // "HELLO" returns the value, but the value of
                 // line s does not change
s                 // => "hello": s the value of the string has
                not changed
```

It is also compared by values of simple types: literals of two simple types, or variables that hold a value of a simple type, are considered equal only if they have the same value. For numbers, logical values, null and unefined, this is obvious. This method of comparison for rows applies differently. JavaScript considers two-line values to be equal only if their lengths are equal and the characters in the appropriate places match.

Objects are different from ordinary types. First, they are variable - their values can be changed:

```
var o = {x:1}, p = {x:1}; // Two objects with the same
              properties

o === p                   // => false: different objects are
                          // not equal
var a = [], b = [];       // Two different empty arrays
a === b                   // => false: different arrays are
```

MATERIALS AND METHODS

From 1.7v onwards we can use '_' under score symbols between digits of numeric literals. You can place underscores only between digits; you cannot place underscores in the following places:

At the beginning or end of a number, Adjacent to a decimal point in a floating point literal, Prior to an F or L suffix, In positions where a string of digits is expected, We can use under score symbols only between the digits if we are using else we will get compile time error.

Examples:

Input : int i = 12_34_56;

Output : 123456

Input : double db = 1_2_3.4_5_6

Output : 123.456

Some invalid examples:

int i = _12345; // Invalid; This

is an identifier, not a numeric literal

double db = 123._456; // Invalid; cannot put

underscores adjacent to a decimal point

double db 123_.456_; // Invalid; cannot put

underscores at the end of a number

The main advantage of this approach is readability of the code will be improved. At the time of compilation these under score symbols will be removed automatically. We can use more than one under score symbols also between the digits too.

RESULTS

For example, following is a valid numeric literal

int x4 = 5_____2; // OK (decimal literal)

// Java program to illustrate

// using underscore in Numeric Literals

class UnderScoreSymbols

{

  public static void main(String[] args)

```
  {
    int i = 12_34_5_6;
    double db = 1_23.45_6;
    int x4 = 5_____2;
    System.out.println("i = "   i );
    System.out.println("db = "   db );
    System.out.println("x4 = "   x4 );
  }
}
```

Output:

i = 123456

db = 123.456

x4 = 52

DISCUSSION

For testers, there are two ways to express Java-floating logic literature in standard and scientific ways:

standard method

scientific method

Standard way of expressing the floating-point expression:

Example:

float a = 32.569f;

Scientific way of expressing the floating-point expressions:

Scientific way uses floating-point number plus a suffix. The suffix specifies the power of 10.

And to represent a floating-point value we add f letter at the end of the floating-point literal.

And to represent the exponent value we have to add the letter E before the suffix.

For example:

float a = 3.567E2f;

In the above example,

The letter f is for representing the value as floating-point value.

The letter E is for representing the exponent value

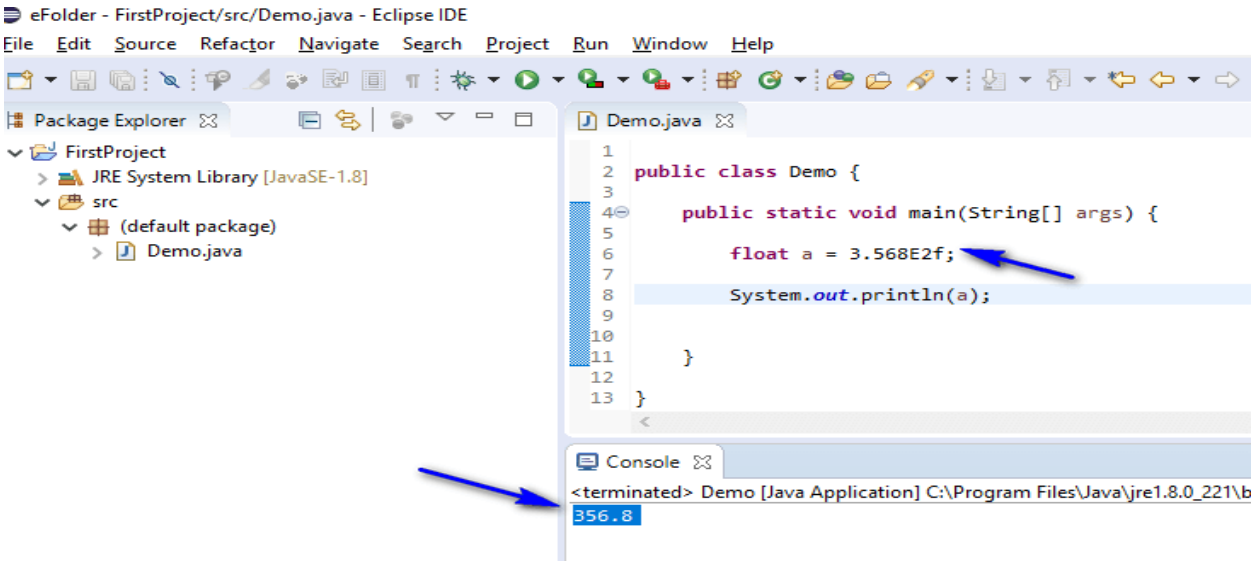
The letter 2 after E is the suffix (2 means 10×10 i.e. 100)

3.567E2 is nothing but 3.567x(10×10)

i.e. nothing but 3.567×100

Which will ultimately become 356.7

The below program practically demonstrate the scientific representation of floating-point value in Java:



Below is the code used in the above demonstration:

```
public class Demo {
        public static void main(String[] args) {
                float a = 3.568E2f;
                System.out.println(a);
        }
}
```

Note: We can also use this scientific representation with the double data type in a similar way. In the case of double data type, we don't have to use the letter f at the end of the literal.

Example:

double a = 3.568E2;

Here conclude this article on expressing the floating and double type literals in a scientific way.

CONCLUSIONS

In JavaScript, I did research on data types, values and variables, fixed simple values, and pointers to variable objects. The deletion operation takes over the pointer: it does not create a new copy of the object in memory. If you want to create a new copy of an object or array in a program, you need to copy the properties of the object or array separately. The below program practically demonstrate the scientific representation of floating-point value in Java. I implemented standard methods of floating point expression as a result.

REFERENCES

1. Introduction to Java programming, short version Daniel Liang (International Economic Edition) Paperback - January 1, 2012

2. Compatibility with Java in version 1 by Brayan Gyotz