How to use String, StringBuilder and StringBuffer

The most used data types for variables in any programming language are boolean values, numeric primitive values and “strings” (or arrays) of characters. In contrast with C or C++, in Java handling strings is different because:

- in Java every char is a 16 bit Unicode value, and not 1 byte;
- in Java, strings values are managed by String objects;
- in Java, the syntax allows you to use Strings as primitive data types (you can use = operator to initialize them)
- in Java, Strings are immutable objects, meaning that once are created, they can’t change their value.

How to define and initialize a String

Because string values are managed by String objects you can create it using the default syntax:

```
String s = new String();
String sWithValue = new String("Hello String !");
```

As I said, because strings are one of the most used data type, the Java syntax allows you to use them as they are primitive data types, using = operator:

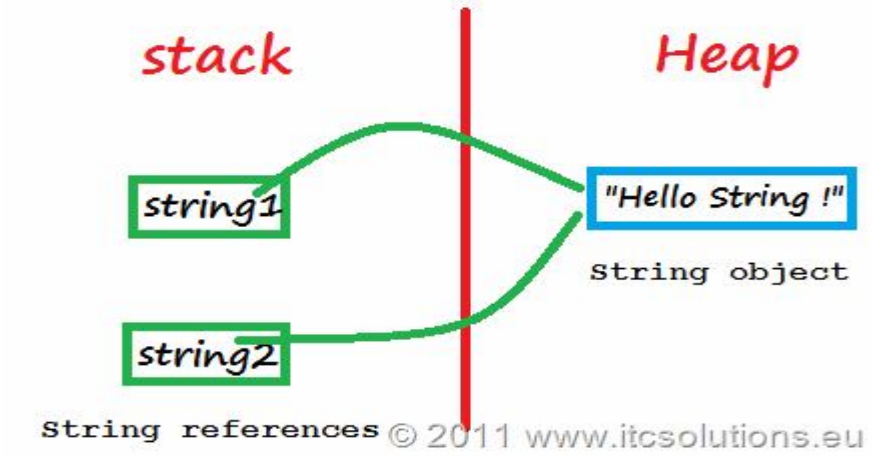
```
String s = "Hello String !";
s = "Hello Java !";
```

Do not forget that in Java strings are objects when using = between 2 String references. You will copy the reference value and not the object value. The next example:

```
String string1 = "Hello String !";
String string2 = string1;

if(string1 == string2)
    System.out.println("EQUAL");
else
    System.out.println("NOT equal");
```

prints “EQUAL” because the references have the same value/address and generates in memory this situation:



Equal String References

What means that String is immutable

An immutable object is an object that once is created it can't change its value. String is immutable and based on previous example let's see what happens when we do something like this:

```
String string1 = "Hello String !";;
System.out.println(string1);
string1 = "Hello !";
System.out.println(string1);
```

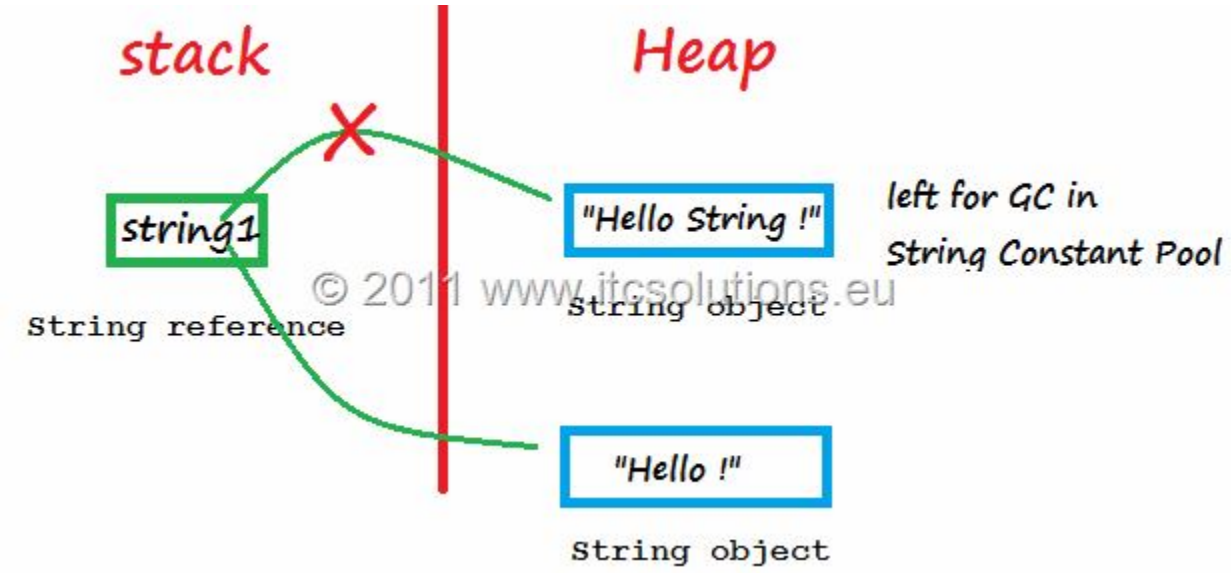
If you run this example, you get

```
Hello String !
Hello !
```

The first impression is that the string1 variable has changed its value from "Hello String !" to "Hello !", but this is wrong because:

- String is immutable
- string1 is a String reference and NOT an object;

So, in reality, the second instruction generates a new String object and the ***string1*** reference gets the address of the new object, like this:



String is Immutable

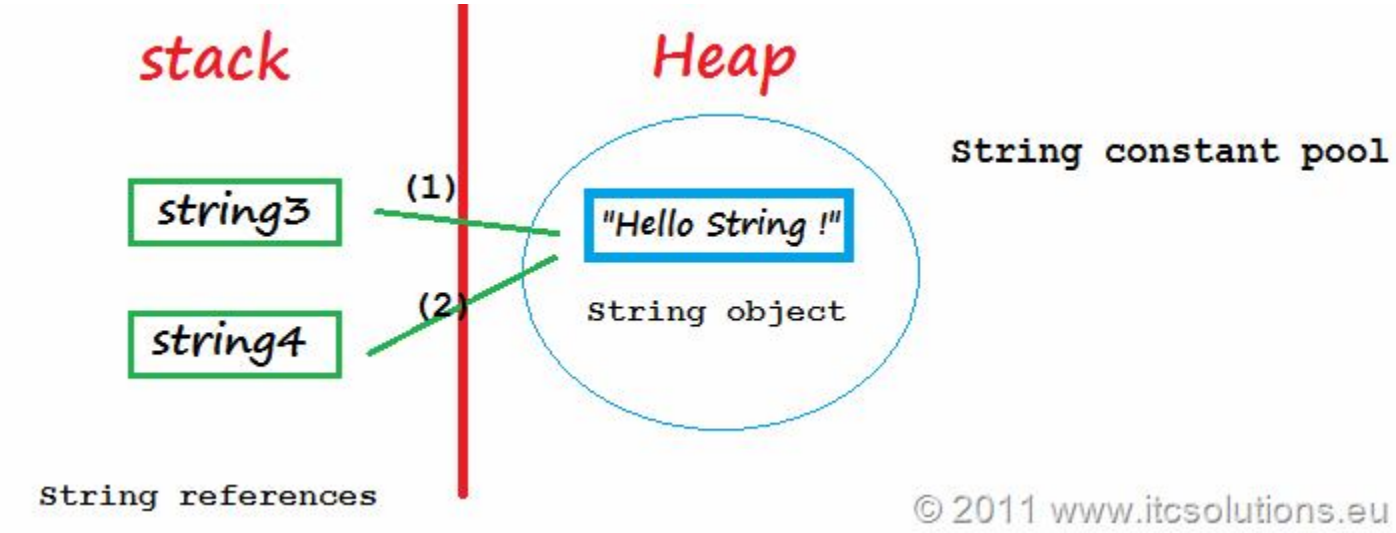
What is String Constant Pool

In Java, String objects are very special (once because they are immutable) because their values are treated in a special way. For an efficient use of memory, JVM manages String values (especially String literals) by putting them in a special area of memory called the "String constant pool". When you use a literal value to initialize a String, the JVM checks the pool to see if the String value is already in memory. If it is, it will not create a new value; it will put its address in the reference.

Based on this rule, the next example

```
String string3 = "Hello String !";
String string4 = "Hello String !";
 if(string3 == string4)    //compare String references, NOT values
    System.out.println("EQUAL");
else
    System.out.println("NOT equal");
```

prints EQUAL, and the process data looks like this



String Constant Pool

- (1) sets the value “Hello String!” in String constant pool
- (2) string4 reference is directed to the existing String in pool

Create a String with new vs. using = operator

As you have seen, there 2 ways to create a String object:

- using new: String s1 = new String(“Hello !”);
- using = operator: String s2 = “Hello !”;

The difference between the two is that **when you use new the String object behaves like a normal object, meaning that its value is placed in Heap and not in the special area, called String constant pool.**

Because, String class is a special one you can’t inherited it and override its functionality. **String class is marked final.**

And the proof:

```
//value in String constant pool
String string1 = "Hello String !";
//value in String constant pool
 String string2 = string1;

 if (string1 == string2) //compare the references, NOT the values
```

```
{
    System.out.println("EQUAL");
} else {
    System.out.println("NOT equal");
}

String string3 = new String("Hello String !");  //value in Heap

if (string1 == string3) //compare the references, NOT the values
{
    System.out.println("EQUAL");
} else {
    System.out.println("NOT equal");
}

if (string1.equals(string3)) //compare the values
{
    System.out.println("EQUAL");
} else {
    System.out.println("NOT equal");
}
```

Running the previous example, you get:

```
EQUAL
NOT equal
EQUAL
```

How can you process a String

The String class has a lot of methods used to process the value. The most used ones are:

Method	Description
charAt()	returns the char at a given index; index takes values from 0 to length()-1;
concat()	appends a String to the end of another; the same as +
equals()	compare at case level 2 String values
length()	return the number of chars; IT IS NOT the length attribute of an array. IT IS A METHOD
replace()	replace occurences of a char with a given one
substring()	returns a substring
toLowerCase()	converts all chars to lowercase
toString()	returns the value of the String object
toUpperCase()	converts all chars to uppercase
trim()	remove whitespace from the end

Because String is immutable, you must pay attention when using some of these functions because their result is a new String object. **These methods will NOT affect the current object.** The next example explains this:

```
String string4 = "TEST";
string4.toLowerCase();      //the result is another String
System.out.println(string4);
string4.concat(" JAVA");     //the result is another String
System.out.println(string4);

String string5 = string4.toLowerCase();      //the result is another String
System.out.println(string5);

string5 = string5.concat(" JAVA");     //the result is another String
System.out.println(string5);
```

The output is:

```
TEST
TEST
test
test JAVA
```

What are StringBuilder and StringBuffer

These two classes are almost the same (they have the same API). The difference between them is that StringBuilder (added from Java 5) is NOT thread safe (more on that in the next posts) and it is faster.

Both classes are defined because they implement in a more efficient manner (memory) processing of string values. One reason is that their values are not stored in String Constant Pool and they behave like normal object – when the object is not references is destroyed by GC and it will not remain in String Constant Pool.

The idea behind StringBuilder and StringBuffer is to provide the means for efficient I/O operations with large streams.

StringBuilder and StringBuffer instances are created using only new operator and NOT with = like Strings:

```
StringBuilder sb1 = new StringBuilder("Hello");
StringBuffer sb2 = new StringBuffer("Hello Java !");

//COMPILER ERROR
sb1 = "Hello World !";          //not allowed
```

If you want to modify the value of a StringBuilder or StringBuffer you can use the methods from this classes. Most used methods are:

Method	Description
append()	adds the argument to the end of the current object
delete()	deletes chars between a start and end index
insert()	inserts a value at a given offset
reverse()	reverse the value of the current object

toString()	returns the value of the StringBuilder or StringBuffer object

These method affects the value of the calling object, like this:

```
StringBuilder sb1 = new StringBuilder("12345");
sb1.append("6789");
System.out.println(sb1);    //prints 123456789

sb1.delete(0,2);
System.out.println(sb1);    //prints 3456789

sb1.reverse();
System.out.println(sb1);    //prints 9876543
```

Another fact about StringBuilder or StringBuffer is that you can call multiple methods – chained methods on the same object. If you prepare for the SCJP exam, you may encounter questions covering this topic, keep in mind that the chained methods affect the calling object and they are executed from left to right.

The next sequence of chaining methods

```
StringBuilder sb2 = new StringBuilder("Test");
sb2.append(" Java ").delete(0, 4).insert(0,"Begin");
System.out.println(sb2);    //prints Begin Java
```

generate this output: *Begin Java*

http://www.itcsolutions.eu/2010/12/22/tutorial-java-6-contents/