UT 4: String, StringBuilder y StringBuffer

1. Gestión de memoria de String

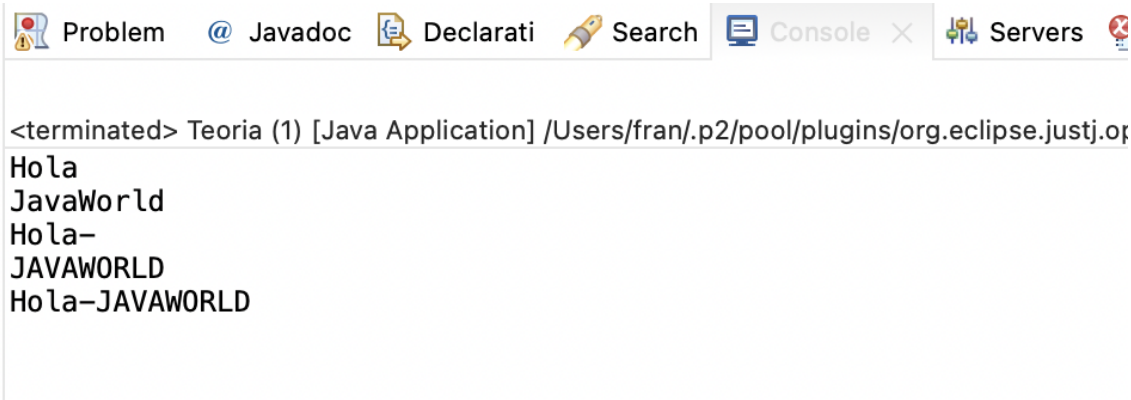
Quizás uno de los aspectos más desconocidos de esta clase desde el punto de vista de Java sea la gestión interna de memoria que hace para la misma con lo que se conoce como pool de constantes *String*.

Este pool funciona de tal forma que cada vez que el compilador encuentra un literal *String* en nuestro código procede a ver si existe en este pool, actuando:

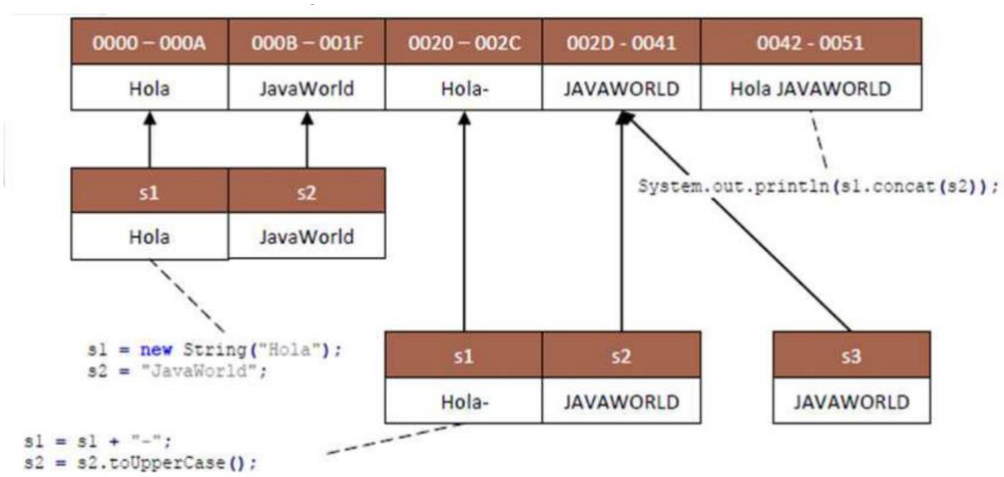
- Si existe, devuelve un referencia al objeto existente.
- Si no existe, se crea un objeto y se devuelve una referencia al mismo, depositándose al mismo tiempo en el pool por si se necesita mas tarde.

Otro de los aspectos a tener en cuenta para esta clase es que se trata de un objeto inmutable. Esto quiere decir, que un objeto de este tipo no puede cambiar su valor (ojo no confundir con que apunte a sitios diferentes).

```
private static void ejemploString() {
    String s1 = new String("Hola");
    String s2 = "JavaWorld";

    System.out.println(s1);
    System.out.println(s2);

    s1 = s1 + "-";
    s2 = s2.toUpperCase();

    System.out.println(s1);
    System.out.println(s2);

    System.out.println(s1.concat(s2));

    String s3 = s2;

}
```



Internamente en memoria lo que esta ocurriendo es:



Se puede ver que en ningún momento se modificó ningún objeto que estuviera en memoria, pero sí cambiaron las referencias de s1 y s2.

Por último, cuando se crea un nuevo objeto (s3), y se le asigna el valor de s2, no se crea un nuevo objeto, se asigna la misma posición de memoria.

2. StringBuilder y StringBuffer

Frente a la inmutabilidad de *String*, nos encontramos con las clases *StringBuffer* y *StringBuilder*, que presentan diferencias básicas sobre *String*:

- Representan cadenas de texto modificables.
- Para crear un objeto de alguno de estos dos tipos hay que hacerlo explícitamente con el operador new.
- Nunca se almacena en el pool de constantes como hace *String*.
- No admiten el operador + para la concatenación de cadenas.

La diferencia entre *StringBuffer* y *StringBuilder*, es que la primera se corresponde con un *Thread-Safe*, es decir, que todos sus métodos están sincronizados. Java nos recomienda el uso de *StringBuilder*, salvo en casos de lectura de ficheros, o en el uso de hilos, ya que es más rápida en procesamiento que *StringBuffer*.

MUY IMPORTANTE, *StringBuffer/StringBuilder* NO SOBRESCRIBEN el método *equals*, por lo tanto dos objetos creados de este tipo solo retornaran TRUE si ambas referencias apuntan al mismo objeto. Dicho de otra forma, devolverán FALSE aunque dispongan del mismo contenido.

Los métodos más importantes son:

RETORNO	MÉTODO	SOBRECARGA	DESCRIPCIÓN
StringBuffer/ StringBuilder	append(String s)	Sí	CONCATENA UNA CADENA SOBRE OTRA. NO HACE FALTA OBTENER LA REFERENCIA PARA QUE EL CAMBIO SE REALICE.
StringBuffer/ StringBuilder	delete(int inicio, int fin)	No	ELIMINA LA SUBCADENA DENTRO DE LA CADENA DEL OBJETO DESDE LA POSICION INICIO A LA FINAL.
StringBuffer/ StringBuilder	insert(int indice, String s)	Sí	INSERTA UNA NUEVA CADENA A PARTIR DE LA POSICION INDICADA POR INDICE DENTRO DE LA CADENA DEL OBJETO.
StringBuffer/ StringBuilder	reverse()	No	INVIERTE LA CADENA.