GESTIÓN DE CADENAS

En Java las cadenas de caracteres no se corresponden con ningún tipo básico, sino que son objetos pertenecientes a la clase *java.lang.String*.

Esta clase, proporciona una amplia variedad de métodos que permiten realizar las operaciones de manipulación y tratamiento de cadenas de caracteres, habituales en un programa.

Creación de objetos String

Para crear un objeto String podemos seguir el procedimiento general de creación de objetos en Java, utilizando el operador **new.** La siguiente instrucción crea un objeto String a partir de la cadena indicada y lo referencia con la variable *s:*

```
String s = new String("Texto de prueba");
```

Sin embargo, dada la amplia utilización de estos objetos dentro de un programa, Java permite crear y asignar un objeto *String* a una variable como un literal, de la misma forma que se hace con cualquier tipo de dato básico. Así, la sentencia anterior es equivalente a:

```
String s = "Texto de prueba";
```

Una vez creado el objeto y asignada la referencia al mismo a una variable, puede utilizarse ésta para acceder a los métodos definidos en la clase String:

```
s.length(); //llamada al método length()
```

Las variables de tipo String pueden utilizarse también en una expresión que haga uso del operador "**+**" para concatenar cadenas:

```
String  s = "Hola";
String t = s + " que tal";
//La variable t apunta al objeto "Hola que tal"
```

Inmutabilidad de objetos String

En Java, las cadenas de caracteres son objetos inmutables, esto significa que una vez el objeto se ha creado, **no puede ser modificado**. Este hecho nos hace reflexionar sobre qué es lo que sucede cuando escribimos una instrucción como la siguiente:

```
String s = "Hola ";
s = s + "que tal";
```

Como es fácil de intuir, la variable `s` pasa a apuntar al objeto de texto "Hola que tal". Aunque, a raíz de la operación de concatenación pueda parecer que el objeto "Hola " apuntado por la variable s ha sido modificado, no es eso lo que está sucediendo.

Lo que realmente ocurre es que al concatenar "Hola " con "que tal" se **está creando un nuevo objeto de texto,** "Hola que tal", que pasa a ser referenciado por la variable `s` (figura 1). El objeto "Hola", por tanto, deja de estar referenciado por dicha variable, con lo que se pierde toda posibilidad de volver a acceder al mismo. A partir de aquí, el **recolector de basura de Java** puede liberar la memoria ocupada por el objeto, si el sistema así lo requiere.

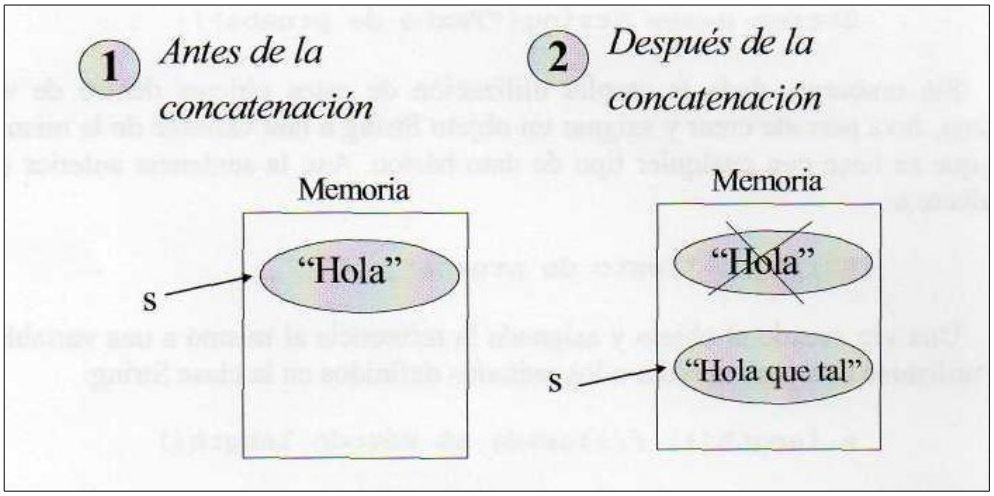

Fig. 1 - Situación de la memoria antes y después de concatenar dos textos

Principales métodos de la clase String

La clase String cuenta con un amplio conjunto de métodos para la manipulación de cadenas de texto, a continuación se indican algunos de los más interesantes:

- `int length()`**:** Devuelve el número de caracteres de la cadena sobre la que se aplica.
- `boolean equals(String otra)`**:** Compara la cadena con otra, haciendo distinción entre mayúsculas y minúsculas. Como ya se comentó anteriormente, se debe utilizar este método, en vez del operador `==`, cuando se van a comparar dos cadenas de texto:

```
String s1, s2;
//suponiendo que las variables adquieren algún valor
if (s1 == s2) //El resultado puede ser falso
//aunque las cadenas sean iguales
if (s1.equals(s2) ) //El resultado siempre será
//verdadero si las cadenas son iguales
```

- **`bolean equalsIgnoreCase(String otra)`:** Actúa igual que el anterior pero sin hacer distinción entre mayúsculas y minúsculas.
- **`char charAt(int pos)`:** Devuelve el carácter que ocupa la posición indicada (base 0) dentro de la cadena. El siguiente programa de ejemplo muestra el número de veces que aparece un carácter dentro de una cadena:

```
public class Principal {
     public static void main(String [] args) {
          String s = "texto de prueba";
          char c = 'e' ;
          int cont = 0;

          for(int i = 0; i<s.length; i++) {
               if (s.charAt(i)==c) {
                    cont + + ;
               }
          }
          System.out.println("La letra " + c +
               "aparece " + cont + " veces");
     }
}
```

- **`String substring(int inicio, int final)`:** Devuelve un trozo de la cadena sobre la que se aplicará. La subcadena devuelta está formada por los caracteres que van desde la posición indicada en *inicio*, hasta la posición *final-1*. Al ejecutar las siguientes instrucciones se mostraría en pantalla el mensaje "de prueba":

```
String s = "texto de prueba";
System.out.println(s.substring(6,15));
```

- **`int indexOf(String cad)`:** Devuelve la posición de la cadena indicada parámetro, dentro de la cadena principal. Si no aparece, devuelve -1. Existe otra versión de este método en la que se puede indicar la posición del carácter en la que se debe comenzar la búsqueda. Se suele utilizar bastante este método para comprobar si un determinado carácter se encuentra presente en una cadena, tal como se indica en el siguiente método de ejemplo:

```
public boolean existe(String texto, String car){
    if (texto.indexOf (car)!=-1){
          return true; //está presente
    } else {
          return false; //no está presente
    }
}
```

- **`String replace(char old, char new)`:** Devuelve la cadena resultante de sustituir todas las apariciones del primer carácter, por el segundo carácter. Este método es muy útil cuando, por ejemplo, se quiere sustituir un tipo de separador decimal por otro:

```
String sPunto, sComa="4,56";
sPunto = sComa.replace(',','.'); //devuelve 4.56;
```

- **`static String valueOf(tipo_basico dato)`:** Método estático que devuelve como cadena el valor pasado como parámetro. Existen tantas versiones de este método como tipos básicos Java.
- **`String toUpperCase()`:** Devuelve la cadena en formato mayúsculas.
- **`String toLowerCase()`:** Devuelve la cadena en formato minúsculas.
- **`String[] split(String regex)`:** Devuelve el array de String resultante de descomponer la cadena de texto en subcadenas, utilizando como separador de elemento el carácter especificado en el parámetro regex. Por ejemplo, dado el siguiente programa:

```
public class ConversionArray {
    public static void main(String [] args){
          String s = "lunes, martes, miércoles";

          String[] cads = s.split(",");

          for ( String aux:cads){
               System.out.println("Día: " + aux);
          }
    }
}
```

Tras la ejecución del método **`main()`** se mostrará en pantalla lo siguiente:

Día: lunes
Día: martes
Día: miércoles

Además de caracteres individuales, se puede utilizar cualquier expresión regular válida como argumento de llamada al método **`split()`** para definir el delimitador de elementos de array en la cadena. En posteriores secciones estudiaremos la sintaxis para la construcción de expresiones regulares.

Método valueOf()

Convierte valores de tipos primitivos a String.
Está sobrecargado con varias definiciones:
`String valueOf(char datos[])`: de array de caracteres a cadena.
`String valueOf(boolean c)`: de booleano a cadena.
`String valueOf(char c)`: de caracter a cadena.
`String valueOf(int i)`: de entero 32 bit a cadena.
`String valueOf(long l)`: de entero 64 bit a cadena.
`String valueOf(float f)`: de flotante 32 bit a cadena.
`String valueOf(double d)`: de flotante 64 bit a cadena.
`String valueOf(Object objeto)`: convierte el objeto en cadena, invocando al método **`toString()`** del objeto.

Uso de las clases envoltorio para conversiones a y desde String

Las clases envoltorio pertenecen al paquete java.lang y emulan a los tipos primitivos. Se llaman *Boolean*, *Character*, *Double*, *Float*, *Integer*, *Long*. Son de tipo *final*, por lo que no admiten clases descendientes.
Tienen un método que permite obtener el valor subyacente y, por tanto, se puede completar la conversión de la cadena a un tipo básico.
El método **`toString()`**, que suele ser genérico para cualquier clase, convierte un objeto a una cadena de caracteres.

Es posible utilizar los métodos estáticos `parseInt()`, `parseLong()`, `parseFloat()` o `parseDouble()` para realizar conversión de String a cada uno de los tipos básicos corresponfientes.
También es posible utilizar el método estático `decode()` para realizar conversiones de String a int o long: `Integer.decode(String).intValue()`;
En resumen, las conversiones posibles son:

- para int
  - A String: `String.valueOf(int)`
  - A String: new Integer(int).toString() // obsoleto
  - De String: `Integer.parseInt(String)`
  - De String: new Integer(String).intValue() // obsoleto
  - De String: Integer.decode(String).intValue()
- para long: es análogo a int
- para float
  - A String: `String.valueOf(float)`
  - A String: new Float(float).toString() // obsoleto
  - De String: `Float.parseFloat(String)`
  - De String: new Float(String).floatValue() // obsoleto
- para double: es análogo a float
- para boolean
  - A String: `String.valueOf(boolean)`
  - A String: new Boolean(boolean).toString() // obsoleto
  - De String: new Boolean(String).booleanValue() // obsoleto
- para char
  - A String: `String.valueOf(char)`
  - A String: new Character(char).toString() // obsoleto
  - De String: new Character(String).charValue() // obsoleto

Problemas de memoria con Strings

Los objetos String son constantes e inmutables Una vez creados, los objetos Strings no pueden ser modificados.

Cuando manipulamos Strings, concatenando, insertando o reemplazando caracteres se crean muchos objetos que son descartados rápidamente. Esta creación de objetos puede producir un incremento en el uso de la memoria.

El recolector de basura de Java, limpia la memoria pero esto tiene un costo de tiempo. Si creamos y destruimos muchos objetos, la performance de nuestro programa puede decaer sustancialmente.

La solución a los problemas de eficiencia de String es proporcionada por Java mediante las clases StringBuilder y StringBuffer

Clase StringBuilder

La clase StringBuilder es similar a la clase String en el sentido de que sirve para almacenar cadenas de caracteres. No obstante, presenta algunas diferencias relevantes. Señalaremos como características de StringBuilder a tener en cuenta:

- Su tamaño y contenido pueden modificarse. Los objetos de éste tipo son mutables. Esto es una diferencia con los String.

- Debe crearse con alguno de sus costructores asociados. No se permite instanciar directamente a una cadena como sí permiten los String.

- Un StringBuilder está indexado. Cada uno de sus caracteres tiene un índice: 0 para el primero,1 para el segundo, etc.

- Los métodos de StringBuilder no están sincronizados. Esto implica que es más eficiente que StringBuffer siempre que no se requiera trabajar con múltiples hilos (threads), que es lo más habitual.

Los constructores de StringBuilder se resumen en la siguiente tabla:

Constructor	Descripción	Ejemplo
StringBuilder()	Construye un StringBuilder vacío y con una capacidad por defecto de 16 carácteres.	StringBuilder s = new StringBuilder();
StringBuilder(int capacidad)	Se le pasa la capacidad (número de caracteres) como argumento.	StringBuilder s = new StringBuilder(55);
StringBuilder(String str)	Construye un StringBuilder en base al String que se le pasa como argumento.	StringBuilder s = new StringBuilder("hola");

Los métodos principales de StringBuilder se resumen en la siguiente tabla:

Retorno	Método	Explicación
StringBuilder	append(...)	Añade al final del StringBuilder a la que se aplica, un String o la representación en forma de String de un dato asociado a una variable primitiva
int	capacity()	Devuelve la capacidad del StringBuilder
int	length()	Devuelve el número de caracteres del StringBuilder
StringBuilder	reverse()	Invierte el orden de los caracteres del StringBuilder
void	setCharAt(int indice,char ch)	Cambia el carácter indicado en el primer argumento por el carácter que se le pasa en el segundo
char	charAt(int indice)	Devuelve el carácter asociado a la posición que se le indica en el argumento
void	setLength(int nuevaLongitud)	Modifica la longitud. La nueva longitud no puede ser menor
String	toString()	Convierte un StringBuilder en un String
StringBuilder	insert(int indiceIni,String cadena)	Añade la cadena del segundo argumento a partir de la posición indicada en el primero
StringBuilder	delete(int indiceIni,int indiceFin)	Borra la cadena de caracteres incluidos entre los dos indices indicados en los argumentos
StringBuilder	deleteChar(int indice)	Borra el carácter indicado en el índice
StringBuilder	replace(int indiceIni, int indiceFin,String str)	Reemplaza los caracteres comprendidos entre los dos indices por la cadena que se le pasa en el argumento
int	indexOf (String str)	Analiza los caracteres de la cadena y encuentra el primer índice que coincide con el valor deseado
String	subString(int indiceIni,int indiceFin)	Devuelve una cadena comprendida entre los dos índices

El método append será probablemente el más utilizado cuando trabajemos con esta clase.

Clase StringBuffer

La clase StringBuffer es similar a la clase StringBuilder, siendo la principal diferencia que sus métodos están sincronizados, lo cual permite trabajar con múltiples hilos de ejecución (threads).

Los constructores y métodos de StringBuffer son los mismos que los de StringBuilder.

Diferencias String, SttringBuilder y StringBuffer

Vamos a enumerar las principales diferencias entre estas tres clases:

- StringBuffer y StringBuilder **son mutables**, mientras que String es inmutable. Cada vez que modificamos un String se crea un objeto nuevo. Esto no ocurre con StringBuffer y StringBuilder.

- Los objetos String se almacenan en el Constant String Pool que es un repositorio o almacén de cadenas, de valores de Strings. Esto se hace con el fin de que si creamos otro String, con el mismo valor, no se cree un nuevo objeto sino que se use el mismo y se asigne una referencia al objeto ya creado. Los objetos StringBuffer y StringBuilder se almacenan en el heap que es otro espacio de memoria usado en tiempo de ejecución para almacenar las instancias de clases, objetos y arrays. Realmente no nos interesa entrar a nivel de detalle en estas diferencias: simplemente, recordar que **los objetos String tienen diferente tratamiento** que los StringBuffer y StringBuilder.

- La implementación de la clase StringBuffer es **synchronized** (sincronizada) mientras StringBuilder no.

- El operador de concatenación "+" es implementado internamente por Java usando **StringBuilder**.

Criterios para usar String, StringBuilder o StringBuffer

¿Cómo decidir qué clase usar? Normalmente usaremos la clase String. No obstante, en algunos casos puede ser de interés usar StringBuffer o StringBuilder. A continuación damos algunas indicaciones útiles para elegir:

- Si el valor del objeto no va a cambiar o va a cambiar poco, entonces es mejor usar String, la clase más convencional para el trabajo con cadenas.

- Si el valor del objeto puede cambiar gran número de veces y puede ser modificado por más de un hilo (thread) la mejor opción es StringBuffer porque es thread safe (sincronizado). Esto asegura que un hilo no puede modificar el StringBuffer que está siendo utilizado por otro hilo.

- Si el valor del objeto puede cambiar gran número de veces y solo será modificado por un mismo hilo o thread (lo más habitual), entonces usamos StringBuilder, ya que al no ser sincronizado es más rápido y tiene mejor rendimiento. El propio api de Java recomienda **usar StringBuilder con preferencia sobre StringBuffer**, excepto si la situación requiere sincronización. En esencia la multitarea (trabajar con varios thread) nos permite ejecutar varios procesos a la vez "en paralelo". Es decir, de forma concurrente y por tanto eso nos permite hacer programas que se ejecuten en menor tiempo y sean más eficientes.