STRING

Un String es una **clase Java** que representa una **cadena de caracteres no modificable** (inmutable). No es necesario hacer **Import** en nuestro programa, dado que la clase **String** está incluida en el paquete **java.lang,** que puede ser usado por defecto y sin importar en nuestros programas.

Todos los literales de la forma "*cualquier texto*", es decir, literales entre comillas dobles, que aparecen en un programa java se implementan como objetos de la clase **String**.

La clase String es **ESPECIAL**, pues nos la proporciona JAVA con la posibilidad de crear Strings como cualquier otro **objeto** en el área de memoria **HEAP**, pero con la opción de manejar a la vez un "pool" de Strings llamado **String Constant Pool** que simplifica la gestión de Strings cuyo contenido se repite.

Así pues, a la hora de trabajar con cadenas de caracteres en nuestros programas, es posible declarar Strings de **2 formas:**

1. **String frase = new String("Hola a tod@s"); (se crea en el HEAP cada "nuevo" String)**

Formas de declarar, instanciar y asignar un String **como objeto**	
String frase = new String("Hola a tod@s");	Declaración + instanciación + asignación
String frase; frase = new String("Hola a tod@s"); String frase = new String(); frase= "Hola a tod@s";	Declaración Instanciación + asignación Declaración + instanciación Asignación
String frase; frase = new String(); frase= "Hola a tod@s";	Declaración Instanciación Asignación

2. **String frase = "Hola a todos"; (Se crea en el HEAP, pero la String Constant Pool gestiona las posibles repeticiones de contenido)**

Formas de declarar y asignar un String **en la String Constant Pool**	
String frase = "Hola a tod@s";	Declaración + asignación
String frase; frase = "Hola a tod@s";	Declaración Asignación

Parece más sencillo hacerlo de esta **2ª forma**, y además es **MAS RECOMENDABLE** si vamos a utilizar el **tipo String original,** por la aportación que supone la String Constant Pool para una gestión más eficiente de los Strings. Sin embargo, en el futuro seguramente nos veremos en la necesidad de usar tipos **String más evolucionados**, como **StringBuilder** y **StringBuffer**, y en este caso **tendremos que declarar y manejar los Strings siempre y solamente como objetos, tal y como hemos hecho en la 1ª tabla.**

MÉTODOS DE LA CLASE STRING

La clase String proporciona métodos para el tratamiento de las cadenas de caracteres: conocer la longitud de una cadena de caracteres, comparar cadenas, acceder a caracteres concretos es una cadena, buscar y extraer una subcadena, copiar cadenas, convertir cadenas a mayúsculas o minúsculas, etc.

(operador concatenación – NO RECOMENDABLE). Además de los métodos de la clase String, existe una sobrecarga del operador **+** que sirve para "unir" 2 o más cadenas de caracteres. **Ejemplo**:

String frase = "hola" + "a" + "tod@s"; // El resultado de la concatenación "hola a tod@s"

Su uso es muy sencillo y puede que al principio hagamos uso de él puntualmente por su simplicidad, pero a medio plazo **mejor olvidarse de que existe**, pues si lo necesitamos usar de forma repetida en algún ejemplo **su ineficiencia es muy grande,** y crea multitud de objetos auxiliares (2 cada vez que se usa) que pueden llegar a saturar la memoria HEAP.

Es recomendable descartar su uso y emplear el método **concat**: String frase= hola.**concat**("a todos");

Concat también tiene limitaciones por el carácter inmutable de los strings, pero **SIEMPRE** será mejor que el operador **+**, a la espera de que en un futuro usemos clases de tipo String más evolucionadas **(StringBuilder)**

MÉTODO	DESCRIPCIÓN
concat(OtroString)	Concatena (une) el String original con "otroString"
length()	Devuelve la longitud de la cadena
indexOf('caracter')	Devuelve la posición de la primera aparición de *carácter* dentro del String. Devuelve -1 si no lo encuentra.
lastIndexOf('caracter')	Devuelve la posición de la última aparición de *carácter* dentro del String. Devuelve -1 si no lo encuentra.
charAt(n)	Devuelve el carácter que está en la posición n
substring(n1,n2)	Devuelve subcadena desde la posición n1 hasta n2 - 1
toUpperCase()	Devuelve la cadena convertida a mayúsculas
toLowerCase()	Devuelve la cadena convertida a minúsculas
equals(otroString)	Compara dos cadenas y devuelve true si son iguales
equalsIgnoreCase(otroString)	Igual que equals pero sin considerar mayúsculas y minúsculas
compareTo(OtroString)	Devuelve 0 si las dos cadenas son iguales. <0 si la primera es alfabéticamente menor que la segunda ó >0 si la primera es alfabéticamente mayor que la segunda.
compareToIgnoreCase(OtroString)	Igual que compareTo pero sin considerar mayúsculas y minúsculas.
valueOf(N)	Convierte el valor N a String. N puede ser de cualquier tipo.

Se pueden consultar todos estos métodos y constructores en el API de Java
https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html

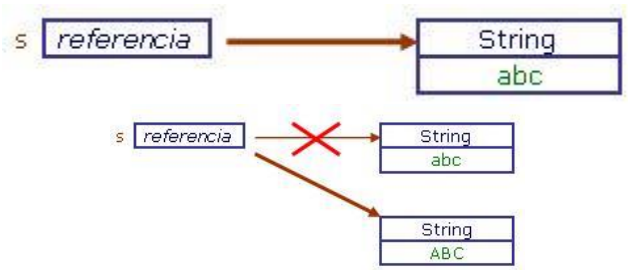
Ejemplos:

“Miguel”.concat(“Rodríguez”)	Miguel Rodríguez
“Miguel”.length()	6
“Miguel”.equals(“Miguel”)	true
“Miguel”.equals(“miguel”)	false
“Miguel”.equalsIgnoreCase(“miguel”)	true
“Miguel”.compareTo(“Saturnino”)	-6 (cualquier entero < 0)
“Miguel”.compareTo(“Miguel”)	0
“Miguel”.compareTo(“Michelin”)	4 (cualquier entero > 0)
“Miguel”.charAt(1)	'i'
“Miguel”.charAt(4)	'e'
“Miguel”.toCharArray()	{ 'M', 'i', 'g', 'u', 'e', 'l' }
“Miguel”.substring(1, 4)	“igu”
“Miguel.substring(1)	“iguel”
“tragaldabas”.indexOf('a')	2
“tragaldabas”.lasIndexOf('a')	9
“tragaldabas”.startsWith(“tragón”)	false
“tragaldabas”.endsWith(“dabas”)	true
“tragaldabas”.split(“a”)	{ “tr”, “g”, “ld”, “b”, “s” }

Los objetos String son inmutables (no es modificable el String original).

Por lo tanto, los métodos que actúan sobre un String con la intención de modificarlo lo que hacen es crear un nuevo String a partir del original y devolverlo modificado.

Por ejemplo: La operación de convertir a mayúsculas o minúsculas un String no lo modificará, sino que creará y devolverá un nuevo String con el resultado de la operación. **String s = "abc"; s = s.toUpperCase();**



El **recolector de basura** de JAVA es el encargado de eliminar de forma automática los objetos a los que ya no hace referencia ninguna variable.

Las secuencias de escape incluidas en el documento *aspectos básicos de JAVA* de la UNIDAD 2 son importantes a la hora de manejar cadenas de caracteres. Podemos insertar diferentes cadenas de escape dentro de Strings para ver sus efectos.

Por ejemplo, no podemos usar "" en medio de un String pues precisamente las comillas delimitran el principio y el final de un String:

String aviso = "aviso a "navegantes": no se puede jugar con los ordenadores"; **Prohibido**

Usando las secuencias de escape convenientes se podría usar comillas "internas" en un String, y algunas cosas más:

String aviso = "aviso a \"navegantes\": no se puede jugar con los ordenadores"; **Permitido.**

Salida por pantalla del String: aviso a "navegantes": no se puede jugar con los ordenadores

Secuencia	**Descripción**
\n	Salto de línea. Sitúa el cursor al principio de la línea siguiente
\b	Retroceso. Mueve el cursor un carácter atrás en la línea actual.
\t	Tabulador horizontal. Mueve el cursor hacia adelante una distancia determinada por el tabulador.
\r	Ir al principio de la línea. Mueve el cursor al principio de la línea actual.
\f	Nueva página. Mueve el cursor al principio de la siguiente página.
\"	Comillas. Permite mostrar por pantalla el caracter *comillas dobles*.
\'	Comilla simple. Permite mostrar por pantalla el carácter *comilla simple*.
\\	Permite mostrar por pantalla el carácter barra inversa.
\udddd	Unicode. Cada **d** representa uno de los 4 dígitos hexadecimales del carácter Unicode a mostrar