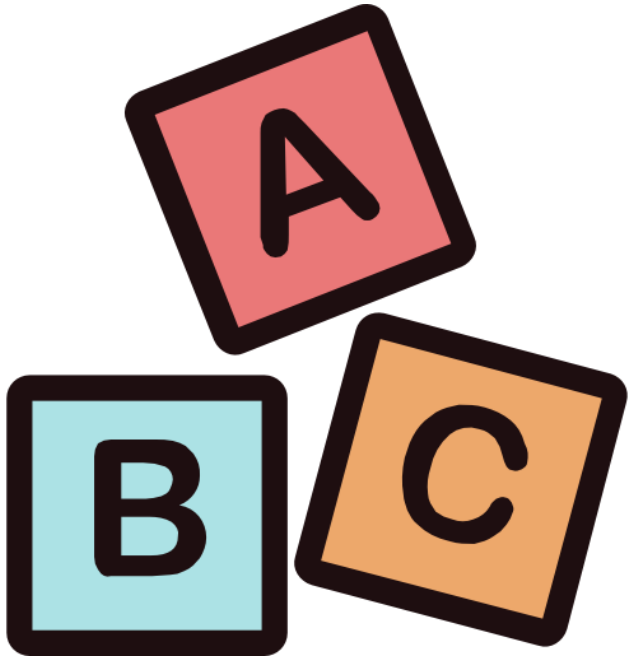

String, StringBuilder, StringBuffer

- Y algo sobre eficiencia -

Chema Durán

Introducción

Java provee distintas clases para el trabajo con cadenas de texto:

- La más básica, la clase **String**.
- Sin embargo existen otras clases como **StringBuffer** y **StringBuilder** que resultan de interés porque facilitan cierto tipo de trabajos y aportan mayor eficiencia en determinados contextos.

StringBuilder

Características 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 constructores 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.

StringBuilder (constructores)

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");

StringBuilder (métodos más comunes)

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

StringBuilder (métodos más comunes)

Retorno	Método	Explicación
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

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***).

En esencia, la multitarea (trabajar con varios ***threads***) 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.

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

Diferencias (String, StringBuilder, StringBuffer)

- **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** tienen diferente tratamiento que los **StringBuffer** y **StringBuilder**.

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

¿Cuál usar?

Normalmente: **String**. No obstante, en algunos casos usaremos **StringBuffer** o **StringBuilder**:

- 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.

¿Cuál usar?

- 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 tiene mejor rendimiento. La propia API de Java recomienda usar **StringBuilder** con preferencia sobre **StringBuffer**, excepto si la situación requiere sincronización.

Comparando rendimientos

```
// StringBuffer
StringBuffer sbuffer = new StringBuffer();
long inicio = System.currentTimeMillis();
for (int i = 0; i < 10000000; i++) {
  sbuffer.append("Elefante");
}
long fin = System.currentTimeMillis();
System.out.println("Tiempo del StringBuffer: " + (fin - inicio));
```

Comparando rendimientos

```
// StringBuilder
StringBuilder sbuilder = new StringBuilder();
inicio = System.currentTimeMillis();
for (int i = 0; i < 10000000; i++) {
  sbuilder.append("Elefante");
}
fin = System.currentTimeMillis();
System.out.println("Tiempo del StringBuilder: " + (fin - inicio));
```

```
Tiempo del StringBuffer:  203
Tiempo del StringBuilder: 285
```

Comparando rendimientos

¿Y si hiciéramos esto mismo con el operador de suma en un **String**?

La misma prueba con el operador de suma consumiría una gran cantidad de tiempo, ya que la creación constante de nuevos objetos hace que la JVM tenga que empezar a limpiar continuamente el Heap. A modo de referencia, concatenar tan sólo 100.000 **String** con el operador de suma se puede demorar por ejemplo 100000 milisegundos frente a por ejemplo 90 milisegundos con **StringBuilder**, por lo que su rendimiento no tiene comparación para este tipo de procesos.

```
Tiempo del StringBuffer:  203
Tiempo del StringBuilder: 285
```

Conclusiones

- Para los usos más habituales usaremos **String**. En situaciones especiales en que se requiera realizar muchas modificaciones de un **String** (por ejemplo ir concatenándole sucesivos fragmentos) usaremos **StringBuilder**.
- Recurriremos a **StringBuffer** sólo cuando trabajemos en entorno multihilos.

Conclusiones

¿Por qué existen distintas clases para realizar tareas tan parecidas?:

- **Motivos históricos. Han ido apareciendo nuevas clases a medida que Java evolucionaba.**

- **Motivos de diseño del lenguaje y compilador Java. Los creadores de Java diseñan el lenguaje de la forma que les parece más adecuada. Casi siempre aciertan (y a veces se equivocan).**

END

jgardur081@g.educaand.es