EJERCICIOS DE STRINGBUILDER

La clase String es una clase no modificable. Esto quiere decir que cuando se modifica un String se crea un nuevo objeto String modificado a partir del original y el recolector de basura es el encargado de eliminar de la memoria el String original.

Java proporciona la clase StringBuffer y a partir de Java 5 la clase StringBuilder para trabajar con cadenas de caracteres sobre las que vamos a realizar modificaciones frecuentes de su contenido.

La diferencia entre StringBuffer y StringBuilder es que los métodos de StringBuffer están sincronizados y los de StringBuilder no lo están. Por este motivo StringBuilder ofrece mejor rendimiento que StringBuffer y la utilizaremos cuando la aplicación tenga un solo hilo de ejecución.

En general decidiremos cuando usar String, StringBuilder o StringBuffer según lo siguiente:

- Usaremos String si la cadena de caracteres no va a cambiar.

- Usaremos StringBuilder si la cadena de caracteres puede cambiar y solamente tenemos un hilo de ejecución.

- Usaremos StringBuffer si la cadena de caracteres puede cambiar y tenemos varios hilos de ejecución.

En esta entrada utilizaremos StringBuilder teniendo en cuenta que todo lo que se explica aquí es aplicable a StringBuffer.

Constructores de la Clase StringBuilder

Un objeto de tipo StringBuilder gestiona automáticamente su capacidad

– Se crea con una capacidad inicial.

– La capacidad se incrementa cuando es necesario.

La clase StringBuilder proporcionan varios constructores, algunos de ellos son:

CONSTRUCTOR	DESCRIPCIÓN
StringBuilder ()	Crea un StringBuilder vacío. StringBuilder sb = new StringBuilder ();
StringBuilder(int n)	Crea un StringBuilder vacío con capacidad para n caracteres.
StringBuilder(String s);	Crea un StringBuilder y le asigna el contenido del String s. String s = "ejemplo"; StringBuilder sb = new StringBuilder (s);

Métodos de la Clase StringBuilder

La clase StringBuilder proporcionan métodos para acceder y modificar la cadena de caracteres. Algunos de ellos son:

MÉTODO	DESCRIPCIÓN
length()	Devuelve la longitud de la cadena
append(X);	Añade X al final de la cadena. X puede ser de cualquier tipo
insert(posicion, X)	Inserta X en la posición indicada. X puede ser de cualquier tipo.
setCharAt(posicion, c)	Cambia el carácter que se encuentra en la posición indicada, por el carácter c.
charAt(posicion)	Devuelve el carácter que se encuentra en la posición indicada.

indexOf('caracter')	Devuelve la posición de la primera aparición de *carácter*. Devuelve -1 si no lo encuentra.
lastIndexOf('caracter')	Devuelve la posición de la última aparición de *carácter*. Devuelve -1 si no lo encuentra.
substring(n1,n2)	Devuelve la subcadena (String) comprendida entre las posiciones n1 y n2 - 1. Si no se especifica n2, devuelve desde n1 hasta el final.
delete(inicio, fin)	Elimina los caracteres desde la posición *inicio* hasta *fin* - 1.
reverse()	Invierte el contenido de la cadena
toString()	Devuelve el String equivalente.

Ejemplo de uso de la clase StringBuilder:

Vamos a escribir un método separarMiles que reciba un String que representa un número entero y devuelva un String con el mismo número al que se le añadirán los puntos separadores de millares.

Por ejemplo, si el método recibe el String "12345678" debe devolver el String "12.345.678"

Este problema lo podemos resolver de varias formas. En este caso la idea es darle la vuelta al número e insertar el primer punto en la cuarta posición del String, el siguiente punto 4 posiciones más adelante el siguiente otras 4 posiciones más adelante .... hasta llegar al final del número. De esta forma obtendremos grupos de 3 cifras separados por punto.

Finalmente le volvemos a dar la vuelta y ya lo tendremos.

Por ejemplo si el String es:

"12345678"

Primero le damos la vuelta:

"87654321"

Ahora tenemos que insertar un punto donde está el 5. Nos queda:

"876.54321"

Insertamos otro punto cuatro posiciones más adelante, donde está el 2:

"876.543.21"

Ahora intentaríamos insertar otro punto cuatro posiciones más adelante pero como llegamos al final el proceso termina.

Si le damos la vuelta obtendremos el resultado:

"12.345.678"

```
/*
 * Ejemplo de uso de StringBuilder
 * Separador de millares
 */
package string8;
public class String8 {

    public static void main(String[] args) {
        String s = "1234567890";
        s = separarMiles(s);
        System.out.println(s);
    }

    public static String separarMiles(String s){

        //creamos un StringBuilder a partir del String s
        StringBuilder aux = new StringBuilder(s);

        //le damos la vuelta
        aux.reverse();

        //variable que indica donde insertar el siguiente
punto
        int posicion = 3;

        //mientras no lleguemos al final del número
        while(posicion < aux.length()){
            //insertamos un punto en la posición
            aux.insert(posicion,'.');
            //siguiente posición donde insertar
            posicion+=4;
        }

        //le damos de nuevo la vuelta
        aux.reverse();
```

```
        //el StringBuilder se pasa a String y se devuelve
        return aux.toString();
    }
}
```

Eficiencia de la Clase StringBuilder frente a la Clase String

Podemos comprobar que es más eficiente utilizar StringBuilder frente a String realizando la siguiente prueba:

Vamos a concatenar un número grande de cadenas de caracteres, por ejemplo 100000, y vamos a medir el tiempo que se emplea en hacerlo.

Lo vamos a realizar primero utilizando la clase String. A continuación utilizando la clase StringBuilder y finalmente lo vamos a hacer utilizando StringBuilder pero asignando inicialmente memoria para la longitud final de la cadena resultante.

```
public class String3 {

    public static void main(String[] args) {
        String s = "cadena";
        long t1, t2;
        int n = 100000;

        System.out.print("Concatenar " + n + " cadenas con
String: ");
        t1 = System.currentTimeMillis();
        concatenar(s,n);
        t2 = System.currentTimeMillis();
        System.out.println((t2-t1) + " milisegundos");

        System.out.print("Concatenar " + n + " cadenas con
StringBuilder: ");
        t1 = System.currentTimeMillis();
        concatenar1(s,n);
        t2 = System.currentTimeMillis();
        System.out.println((t2-t1) + " milisegundos");
```

```
        System.out.print("Concatenar " + n + " cadenas con
StringBuilder Optimizado: ");
        t1 = System.currentTimeMillis();
        concatenar2(s,n);
        t2 = System.currentTimeMillis();
        System.out.println((t2-t1) + " milisegundos");


    }

    //método que concatena n cadenas usando la clase String
    public static String concatenar(String s, int n) {
        String resultado = s;
        for (int i = 1; i < n; i++) {
            resultado = resultado + s;
        }
        return resultado;
    }

    //método que concatena n cadenas usando la clase
StringBuilder
    public static String concatenar1(String s, int n) {
        StringBuilder resultado = new StringBuilder(s);
        for (int i = 1; i < n; i++) {
            resultado.append(s);
        }
        return resultado.toString();
    }

    //método optimizado que concatena n cadenas usando la
clase StringBuilder
    //se crea un StringBuilder inicial con el tamaño total
del String resultante
    public static String concatenar2(String s, int n) {
        StringBuilder resultado = new
StringBuilder(s.length() * n);
        for (int i = 0; i < n; i++) {
            resultado.append(s);
        }
        return resultado.toString();
    }
}
```

La salida del programa dependerá del ordenador que utilicemos. En cualquier caso nos debe mostrar que concatenar utilizando String es más lento que si utilizamos StringBuilder.

En mi caso el resultado obtenido ha sido este:

```
Concatenar 100000 cadenas con String: 304435 milisegundos
Concatenar 100000 cadenas con StringBuilder: 15 milisegundos
Concatenar 100000 cadenas con StringBuilder Optimizado: 1 milisegundos
```