



# Tema 03. Programación modular

## Parte 2. Cadenas en Java





## Índice

1. Cadenas de caracteres
2. Clase String
  - Operaciones básicas
  - Búsquedas
  - Comparación
3. Clase StringBuilder
  - Constructores
  - Métodos
4. Operador Suma
5. Clase Character



## 1.- Cadenas de caracteres.

Una cadena es una **secuencia de caracteres**. En Java existen varias clases, pertenecientes al paquete `java.lang`, para manejar cadenas de caracteres:

- `String`.
- `StringBuilder`.
- `StringBuffer`.

Cuándo usar uno y cuándo usar otro

- Si el texto no va a cambiar, utilizar **String**.
- Si va a cambiar, y la aplicación va a tener un solo hilo de ejecución, utilizar **StringBuilder**.
- Si el texto cambia durante la ejecución, y la aplicación que accede a él es multihilo, utilizaremos **StringBuffer**.

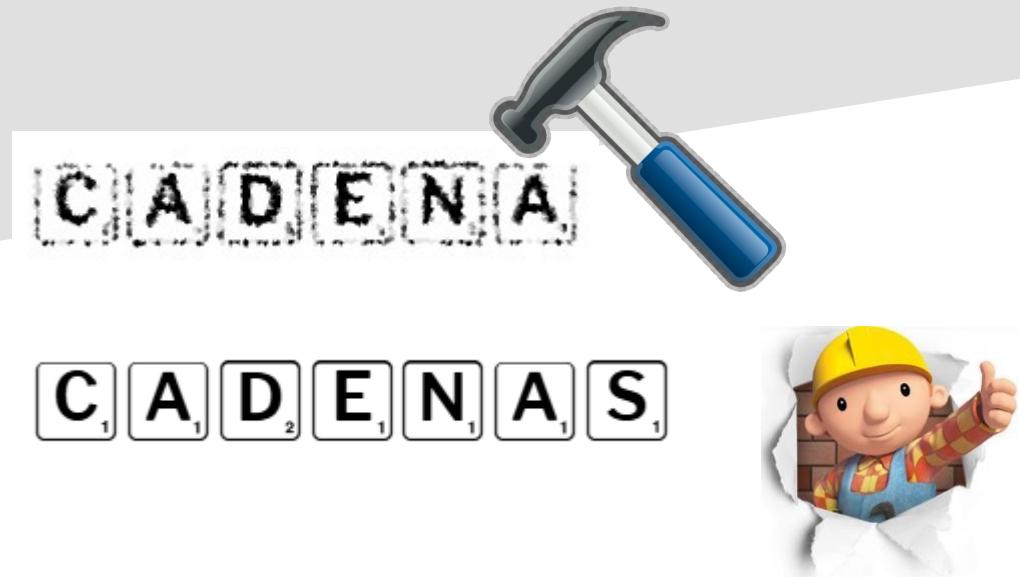


## 2.- Clase String.

La clase String, permite el manejo de cadenas de caracteres no modificables.

Si se realiza una concatenación a un String ya existente, se creará un nuevo objeto String, con el contenido resultante de la operación.

Además esta clase, proporciona métodos para: examinar caracteres individuales de una cadena de caracteres, comparar cadenas, buscar y extraer subcadenas, copiar cadenas y convertir a mayúsculas o a minúsculas,...etc





## 2.- Clase String.

### Constructores de String

Los constructores para crear objetos de la clase String son:

- **String():** crea una cadena vacía.
- **String(String s):** crea una cadena cuyo contenido es copia de otra.

```
String s = new String ("Hola mundo");
```

Se pueden crear Strings realizando una asignación directa del valor, a una variable declarada de tipo String:

```
String s = "Hola mundo";
```



## 2.- Clase String.

### Operaciones básicas

- **int length()**: devuelve la longitud de la cadena.
- **char charAt(int pos)**: devuelve el carácter que ocupa la posición 'pos'.

Las posiciones se numeran de 0 a  $n - 1$ . El acceso a una posición fuera de los límites de la cadena, genera una excepción.

Ejemplos:

```
String str = "abcdabc";
str.length(); // → 7
str.charAt(4); // → 'a'
```



## 2.- Clase String.

Los siguientes métodos devuelven nuevos objetos con la cadena modificada:

- **String concat(String str)**: devuelve una cadena resultante de concatenar 'str' a la cadena (similar a +).

Ejemplo:

```
String cadena1 = "Hola";
String cadena2 = cadena1.concat("¿Qué tal?");
```

El objeto referenciado por *cadena1* permanece inmutable.

No estamos concatenando “¿Qué tal?” al objeto referenciado por *cadena1*, se está creando un nuevo objeto y devolviendo una referencia de él.



## 2.- Clase String.

- **String replace(char c1, char c2)**: devuelve una cadena resultante de sustituir el carácter 'c1' por el 'c2'.

Ejemplo:

```
String cadena1 = "abcdabc";
String cadena2 = cadena1.replace ('b', 'x'); // → "axcdaxc"
```

También puede usarse con cadenas (String) como parámetros.

- **String toUpperCase()**: devuelve una cadena resultante de pasar la cadena a mayúsculas.

Ejemplo:

```
String cadena1 = "abcdabc";
String cadena2 = cadena1.toUpperCase (); // → "ABCDABC"
```



## 2.- Clase String.

- **String toLowerCase()**: devuelve una cadena resultante de pasar la cadena a minúsculas.

Ejemplo:

```
String cadena1 = "ABCDABC";
String cadena2 = cadena1.toLowerCase(); // → " abcdabc "
```

- **String trim()**: devuelve la cadena resultante de eliminar los espacios en blanco que pueda haber al principio y al final.



## 2.- Clase String.

- **String substring(int inicio)**: devuelve una subcadena, desde la posición de 'inicio' (inclusive), hasta el final de la cadena.

```
String cadena1 = "hamburguesa";
String cadena2 = cadena1.substring(3); // cadena 2 vale "burguesa"
```

- **String substring(int inicio, int final)**: devuelve una subcadena desde la posición 'inicio' (inclusive), hasta el 'final' - 1 de la cadena.

```
String cadena1 = "hamburguesa";
String cadena2 = cadena1.substring(3, 7); // cadena 2 vale "burg"
```



## 2.- Clase String.

- **String[] split (separador)**: divide una cadena en subcadenas de acuerdo con un 'separador'. Devuelve un array de String. Podría recibir una expresión regular.

Ejemplo:

```
String cadena = "123-654321";
String[] parts = cadena.split("-");
String part1 = parts[0]; // 123
String part2 = parts[1]; // 654321
```



## 2.- Clase String.

### Búsqueda dentro de cadenas

- **int indexOf(String str)**: devuelve la primera posición que ocupa 'str' dentro de la cadena. Devuelve -1 si el carácter no se encuentra en la cadena.

```
String cadena1 = "abcdabc";
int pos = cadena1.indexOf ("bc"); // → 1
```

- **int indexOf(String str, int inicio)**: devuelve la posición que ocupa 'str' dentro de la cadena, a partir de la posición de inicio (inclusive).

```
String cadena1 = "abcdabc";
int pos = cadena1.indexOf ("bc",4 ); // → 5
```



## 2.- Clase String.

- **int lastIndexOf(String str)**: busca 'str' dentro de la cadena, empezando desde el final de la misma, y devolviendo la primera posición de comienzo de 'str' encontrada.

```
String cadena1 = "abcdabc";
int pos = cadena1.lastIndexOf ("bc"); // → 5
```

- **int lastIndexOf(String str, int final)**: Idem pero indicando la posición 'final' desde la que empezará a buscar.



## 2.- Clase String.

- **boolean contains (String subcadena)**: comprueba si una 'subcadena' está contenida en una cadena.
- **boolean startsWith (String subcadena)**: comprueba si una cadena empieza por una 'subcadena'.
- **boolean endsWith(String subcadena)**: comprueba si una cadena termina con una 'subcadena'.



## 2.- Clase String.

### Comparación de cadenas

Si comparamos dos cadenas: **cad1 == cad2**, estaríamos comparando sus referencias y no su contenido. Para comparar su contenido, se deben utilizar los métodos **equals()** o **compareTo()**:

- **boolean equals(Object objeto)**: si la cadena correspondiente al argumento, es igual que el String de referencia, devuelve *true*, si no, *false*.

Se utiliza de la siguiente forma:

```
cadena1.equals(cadena2);
```

- **boolean equalsIgnoreCase(String s)**: como *equals*, pero sin distinguir mayúsculas y minúsculas.



## 2.- Clase String.

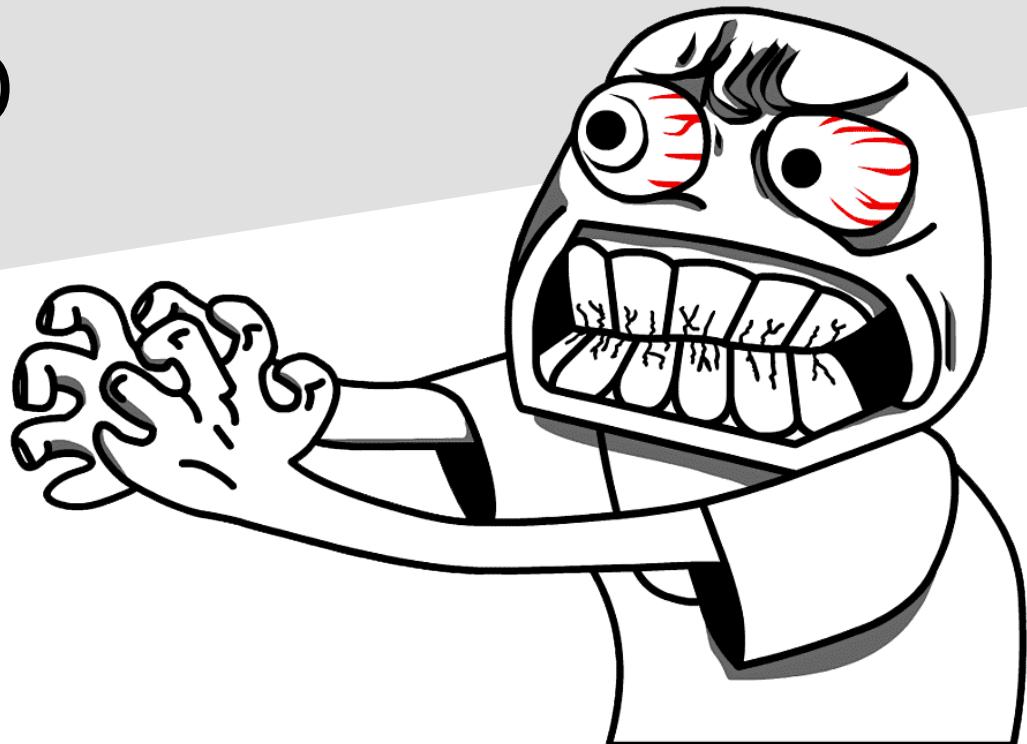
# REPETIMOS

LAS CADENAS SE COMPARAN

CON EL MÉTODO

.EQUALS()

NO CON ==





## 2.- Clase String.

### Comparación de cadenas

- **int compareTo(String s)**: devuelve la diferencia entre los primeros caracteres no iguales entre ambas cadenas

```
cadena1.compareTo(cadena2);
```

Devuelve

- Cero si las cadenas son iguales.
- Un número menor que cero, si cad1 es lexicográficamente menor que cad2.
- Un número mayor que cero, si cad1 es lexicográficamente mayor que cad2.



## 2.- Clase String.

### Comparación de cadenas

Ejemplos:

```
String s1 = "Juan Alvarez";
String s2 = new String ("Juan Alvarez");
String s3 = "Juan Gonzalez";
String s4= "Juan Perez";
boolean valor = (s1 == s2); // Devolvería false
s1.equals (s2);           // Devolvería true
s1.compareTo (s3);         // Devolvería un valor < 0
s4.compareTo(s3);          // Devolvería un valor > 0
```



## 2.- Clase String.

### Método valueOf

Este es un método estático de la clase String. Convierte valores de tipos primitivos, a String. Está sobrecargado con varias definiciones:

#### **String valueOf (<cualquier tipo>)**

- **String valueOf(char a):** de carácter a cadena.
- **String valueOf(int i):** de entero 32 bit a cadena.
- **String valueOf(long l):** de entero 64 bit a cadena.

Ejemplos:

```
String.valueOf (2.34567); // → "2.34567"  
String.valueOf (34); // → "34"
```



### 3.- Clase **StringBuilder**.

La clase **StringBuilder** permite crear y modificar secuencias de caracteres.

- Los caracteres de un StringBuilder sí se pueden modificar.
- Los objetos de tipo StringBuilder gestionan automáticamente su capacidad:
  - Toman una capacidad inicial.
  - La incrementan cuando es necesario.



Posee el método `toString()` que devuelve el resultado de convertir el StringBuilder a una cadena de caracteres.



## 3.- Clase **StringBuilder**.

### Constructores

Para la inicialización de estos objetos, se puede utilizar un constructor con un argumento. Permite reservar espacio para un número de caracteres pasado como parámetro, o bien reservar el espacio según la longitud de un String de entrada al constructor.

- **StringBuilder()**: crea un StringBuilder con capacidad inicial para 16 caracteres.
- **StringBuilder(int n)**: crea un StringBuilder con capacidad inicial para  $n$  caracteres.
- **StringBuilder(String s)**: crea un StringBuilder con capacidad inicial para  $s.length() + 16$  caracteres.



## 3.- Clase **StringBuilder**.

### Métodos de **StringBuilder**

- Acceso (igual que para String): *length()*, *charAt(int)*, ...
- Devuelve el espacio total del StringBuilder: *capacity()*
- Conversión a String: *toString()*

Ejemplo:

```
StringBuilder strb1 = new StringBuilder();
StringBuilder strb2 = new StringBuilder(80);
StringBuilder strb3 = new StringBuilder("abcde");
System.out.println(strb1 + " " + strb1.length() + " " + strb1.capacity()); // 0 16
System.out.println(strb2 + " " + strb2.length() + " " + strb2.capacity()); // 0 80
System.out.println(strb3 + " " + strb3.length() + " " + strb3.capacity()); // abcde 5 21
```



### 3.- Clase **StringBuilder**.

Los métodos más interesantes son aquéllos que permiten la modificación de la cadena de caracteres que contiene:

- **StringBuilder append(String str)**: añade la cadena 'str' a la cadena que contiene el receptor. El método devuelve el propio receptor.

Ejemplos:

```
StringBuilder apellidos = new StringBuilder("APELLOS");
String completo = apellidos.append(", ").append ("Nombre").toString( ) ;
//produce la cadena: APELLIDOS, Nombre.
StringBuilder str = new StringBuilder("abcdef");
str.append("ghi"); // str = "abcdefghi"
```



### 3.- Clase **StringBuilder**.

- **StringBuilder insert (int index, String str)**: inserta la cadena 'str' a partir de la posición 'index'.

```
StringBuilder str = new StringBuilder ("abcdef");
str.insert (3, "xyz"); // str = "abcxyzdef"
str.insert (6, 1.23); // str = "abcxyz1.23def"
```

- **StringBuilder delete(int desde, int hasta)**: borra los caracteres que van desde la posición 'desde' hasta la posición 'hasta'-1 (o hasta el último carácter del receptor).

```
StringBuilder str = new StringBuilder("abcdef");
str.delete (2, 3); // str = "abdef"
```



### 3.- Clase **StringBuilder**.

- **void setCharAt(int pos, char c)**: cambia el carácter de la posición 'pos', por el carácter 'c'.

Ejemplo:

```
StringBuilder str = new StringBuilder("abcdef");
str.setCharAt (2, 'q'); // str = "abqdef"
```

- **StringBuilder deleteCharAt(int pos)**: elimina el carácter que ocupa la posición 'pos' del receptor.



### 3.- Clase **StringBuilder**.

- **StringBuilder replace(int desde, int hasta, String str)**: borra del receptor los caracteres que van desde 'desde' hasta 'hasta'-1 (o hasta el último carácter del receptor) e inserta desde la posición 'desde' la cadena 'str'.
- **StringBuilder reverse()**: invierte la secuencia de caracteres de la cadena.

Ejemplo:

```
StringBuilder str = new StringBuilder ("abcdef");
str.reverse (); // str = "fedcba"
```



## 4.- Operador suma

Está sobrecargado para los String, y es la forma más básica de concatenar dos cadenas de caracteres. Y también, es la peor forma en cuanto a rendimiento.

Al concatenar dos String con el operador suma, se crea un nuevo String resultante, con lo que constantemente estamos creando objetos nuevos. Esto resulta muy ineficiente

Ejemplo con String:

```
String strHola = "Hola, ";
String strMundo = "mundo";
String str = strHola + strMundo;
```



## 4.- Operador suma

Con StringBuilder sería:

```
StringBuilder str = new StringBuilder();
str.append("Hola, ");
str.append("mundo");
String strCompleta = str.toString();
```



## 5.- Clase Character

La clase **Character** tiene métodos estáticos para tratar caracteres.

Algunos de estos métodos que suelen ser útiles son:

- **boolean isUpperCase(char c)**: Devuelve *true* si el carácter es una letra mayúscula
- **boolean isLowerCase(char c)**: Devuelve *true* si el carácter es una letra minúsculas
- **boolean isLetter(char c)**: Devuelve *true* si el carácter es una letra
- **boolean isDigit(char c)**: Devuelve *true* si el carácter es un dígito
- **char toUpperCase(char c)**: Devuelve el carácter mayúscula correspondiente
- **char toLowerCase(char c)**: Devuelve el carácter minúscula correspondiente.



# Tema 03. Programación modular



Parte 2.  
Cadenas en Java