

Programación

# Cadenas de texto y Funciones

Unidad 6

Jesús Alberto Martínez  
versión 0.2





Reconocimiento – NoComercial – CompartirIgual (by-nc-sa): No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original.  
Basado en los apuntes del CEEDCV




# Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

 Importante

 Atención

 Interesante

## Unidad 6. Cadenas de texto y Funciones

- 1 La clase String..... 3
  - 1.1 Valor null..... 4
  - 1.2 Comparación..... 4
  - 1.3 Métodos más utilizados..... 5
  - 1.4 Conversiones entre tipos..... 9
  - 1.5 Clase Character..... 10
  - 1.6 StringBuilder y StringBuffer..... 10
    - Métodos de StringBuilder vs métodos de String..... 12

# 1 La clase String

Las cadenas de texto son objetos especiales, son una secuencia de caracteres, ya sean números, letras u otros símbolos. Los textos deben manejarse creando objetos de tipo **String**, no son datos primitivos, nos van a servir para introducir el uso de objetos.

Vamos a simplificar un poco los conceptos, ya que se introducirán de una forma más teórica más adelante. String es una clase, y una clase es un tipo de objetos, simplificando es como si fuera un tipo de datos. Cuando definimos una variable de tipo String, en realidad estamos creando un objeto de la clase String.

```
// Definimos una variable de tipo int o entero
int numero=10;
// Definimos un objeto o instancia de la clase String, de nombre mensaje
String mensaje="Hola Mundo";
```

Realmente, la sintaxis pura sería

```
String mensaje=new String("Hola Mundo");
```

Creamos objetos con el operador **new**, pero en la creación de cadenas de texto nos lo podemos ahorrar.

La diferencia entre usar datos primitivos o clases, es que éstas tienen definidas una serie de operaciones, a las que llamamos **métodos** que vamos a poder usar. La forma de usar estos métodos es a través de un punto sobre el objeto sobre el que queremos operar.

Ya hemos usado algún método de String, hemos comparado dos cadenas de caracteres.

```
String color="rojo";
if (color.equals("negro")){ ...
```

Estamos usando el método equals sobre el objeto color. Vemos que al método hay que pasarle otra cadena de texto para compararla, a esto se llama **parámetro** del método. Un método puede no tener parámetros, o tener varios, depende de si necesita más información para la operación que realiza.

Las cadenas pueden ocupar varias líneas utilizando el operador de concatenación "+".

```
String texto2 ="Este es un texto que ocupa " +
               "varias líneas, no obstante se puede " +
               "perfectamente encadenar";
```

También se pueden crear objetos String sin utilizar constantes entrecomilladas, usando otros constructores:

```
char[] palabra = {'P', 'a', 'l', 'a', 'b', 'r', 'a'}; // Array de char
String cadena = new String(palabra);
```

## 1.1 Valor null

Como hemos expuesto, un String es un objeto, y otra propiedad que tienen es que pueden no contener ningún valor, esto no pasa con los datos primitivos.

**null** es una palabra reservada que representa un valor especial, la ausencia de valor, cuando no se almacena nada.

```
String mensaje=null; // La variable no está almacenando nada
```

El uso de primario de null es comprobar si un objeto está vacío, si almacenamos información o no.

```
if (mensaje!=null) System.out.println(mensaje);
```

## 1.2 Comparación

⚡ Los objetos *String* NO pueden compararse directamente con los operadores de comparación `==` como las variables simples

En su lugar se deben utilizar estos métodos:

- **cadena1.equals(cadena2)**. El resultado es true si la cadena1 es igual a la cadena2. Ambas cadenas son variables de tipo String.
- **cadena1.equalsIgnoreCase(cadena2)**. Como la anterior, pero en este caso no se tienen en cuenta mayúsculas y minúsculas.
- **s1.compareTo(s2)**. Compara ambas cadenas, considerando el orden alfabético. Si la primera cadena es mayor en orden alfabético que la segunda, devuelve la diferencia positiva entre una cadena y otra, si son iguales devuelve 0 y si es la segunda la mayor, devuelve la diferencia negativa entre una cadena y otra. Hay que tener en cuenta que el orden no es el del alfabeto español, sino que usa la tabla ASCII, en esa tabla la letra ñ es mucho mayor que la o.
- **s1.compareToIgnoreCase(s2)**. Igual que la anterior, sólo que además ignora las mayúsculas.

## 1.3 Métodos más utilizados

```
objetoString.método(argumentos);
```

Algunos de los métodos más utilizados son:

**valueOf** : Convierte valores que no son de cadena a forma de cadena.

```
String numero = String.valueOf(1234);
// Convierte el número int 1234 en el String "1234"
```

**length** : Devuelve la longitud de una cadena (el número de caracteres de la cadena):

```
String texto1="Prueba";
System.out.println(texto1.length()); // Escribe un 6
```

**Concatenar (unir) cadenas** : Se puede hacer de dos formas, utilizando el método concat o con el operador +.

```
String s1= "Buenos ", s2= " días", s3, s4;
s3 = s1 + s2;
s4 = s1.concat(s2);
```

En ambos casos el contenido de s3 y s4 sería el mismo: "Buenos días".

**charAt** : Devuelve un carácter concreto de la cadena. El carácter a devolver se indica por su posición (el primer carácter es la posición 0). Si la posición es negativa o sobrepasa el tamaño de la cadena, ocurre un error de ejecución, una excepción tipo *IndexOutOfBoundsException* (recuerda este tipo de error, se repetirá muchas veces).

```
String s1="Prueba";
char c1 = s1.charAt(2); // c1 valdrá 'u'
```

**substring** : Da como resultado una porción del texto de la cadena. La porción se toma desde una posición inicial hasta una posición final (sin incluir esa posición final). Si las posiciones indicadas no son válidas ocurre una excepción de tipo *IndexOutOfBoundsException*. Se empieza a contar desde la posición cero.

```
String s1="Buenos días";
String s2=s1.substring(7,10); // s2 = "día"
```

**indexOf** : Devuelve la primera posición en la que aparece un determinado texto en la cadena. En el caso de que la cadena buscada no se encuentre, devuelve -1.

El texto a buscar puede ser char o String.

```
String s1="Quería decirte que quiero que te vayas";  
System.out.println(s1.indexOf("que")); // Devuelve 15
```

También se puede buscar desde una determinada posición:

```
String s1="Quería decirte que quiero que te vayas";  
System.out.println(s1.indexOf("que",16)); // Ahora devolvería 26
```

**lastIndexOf** : Devuelve la última posición en la que aparece un determinado texto en la cadena. Es casi idéntica a la anterior, sólo que busca desde el final.

```
String s1="Quería decirte que quiero que te vayas";  
System.out.println(s1.lastIndexOf("que")); //Devolvería 26
```

También permite comenzar a buscar desde una determinada posición.

**endsWith** : Devuelve true si la cadena termina con un determinado texto.

```
String s1="Quería decirte que quiero que te vayas";  
System.out.println(s1.endsWith("vayas")); //Devolvería true
```

**startsWith** : Devuelve true si la cadena empieza con un determinado texto.

```
String s1="Quería decirte que quiero que te vayas";  
System.out.println(s1.startsWith("vayas")); // Devolvería false
```

**replace** : Cambia todas las apariciones de un carácter (o caracteres) por otro/s en el texto que se indique y lo almacena como resultado. El texto original no se cambia, por lo que hay que asignar el resultado de replace a un String para almacenar el texto cambiado.

Ejemplo1

```
String s1="Mariposa";  
System.out.println(s1.replace('a', 'e')); //Devuelve "Meripose"  
System.out.println(s1); //Sigue valiendo "Mariposa"
```

Para guardar el valor deberíamos hacer:

```
String s2 = s1.replace('a', 'e');
```

## Ejemplo2

```
String s1="Buscar armadillos";
System.out.println(s1.replace("ar","er")); //Devuelve "Buscer ermadillos"
System.out.println(s1); //Sigue valiendo "Buscar armadillos"
```

**toUpperCase** : Obtiene la versión en mayúsculas de la cadena. Es capaz de transformar todos los caracteres nacionales:

```
String s1 = "Batallón de cigüeñas";
System.out.println(s1.toUpperCase()); //Escribe: BATALLÓN DE CIGÜEÑAS
```

**toLowerCase** : Obtiene la versión en minúsculas de la cadena.

```
String s1 = "Batallón de cigüeñas";
System.out.println(s1.toLowerCase()); //Escribe: batallón de cigüeñas
```

**toCharArray** : Consigue un array de caracteres a partir de una cadena. De esa forma podemos utilizar las características de los arrays para manipular el texto, lo cual puede ser interesante para manipulaciones complicadas.

```
String s="texto de prueba";
char c[]=s.toCharArray();
```

**format** : Modifica el formato de la cadena a mostrar. Muy útil para mostrar sólo los decimales que necesitemos de un número decimal. Indicaremos "%" para indicar la parte entera más el número de decimales a mostrar seguido de una "f" :

```
System.out.println(String.format("%.2f", number));
// Muestra el número con dos decimales.
```

Aplica las mismas reglas que el método System.out.printf

**matches** : Examina la *expresión regular* que recibe como parámetro (en forma de String) y devuelve verdadero si el texto que examina cumple la expresión regular. Una expresión regular es una expresión textual que utiliza símbolos especiales para hacer búsquedas o comprobaciones.

Las expresiones regulares pueden contener:

- Caracteres. Como a, s, ñ,... y les interpreta tal cual. Si una expresión regular contuviera sólo un carácter, matches devolvería verdadero si el texto contiene sólo ese carácter. Si se ponen varios, obliga a que el texto tenga exactamente esos caracteres.
- Caracteres de control (\n,\\,...)

- Opciones de caracteres. Se ponen entre corchetes. Por ejemplo [abc] significa a, b ó c.
- Negación de caracteres. Funciona al revés impide que aparezcan los caracteres indicados. Se pone con corchetes dentro de los cuales se pone el carácter circunflejo (^). [^abc] significa ni a ni b ni c.
- Rangos. Se ponen con guiones. Por ejemplo [a-z] significa: cualquier carácter de la a a la z.
- Intersección. Usa &&. Por ejemplo [a-x&&r-z] significa de la r a la x (intersección de ambas expresiones).
- Sustracción. Ejemplo [a-x&&[^cde]] significa de la a a la x excepto la c, d ó e.
- Cualquier carácter. Se hace con el símbolo punto (.)
- Opcional. El símbolo ? sirve para indicar que la expresión que le antecede puede aparecer una o ninguna veces. Por ejemplo a? indica que puede aparecer la letra a o no.
- Repetición. Se usa con el asterisco (\*). Indica que la expresión puede repetirse varias veces o incluso no aparecer.
- Repetición obligada. Lo hace el signo +. La expresión se repite una o más veces (pero al menos una).
- Repetición un número exacto de veces. Un número entre llaves indica las veces que se repite la expresión. Por ejemplo \d{7} significa que el texto tiene que llevar siete números (siete cifras del 0 al 9). Con una coma significa al menos, es decir \d{7,} significa al menos siete veces (podría repetirse más veces). Si aparece un segundo número indica un máximo número de veces \d{7,10} significa de siete a diez veces.

Veamos algunos ejemplos:

```
String cadena="Solo se que no se nada";

// ejemplo1. devolverá false, ya que la cadena tiene más caracteres
System.out.println("ejemplo1: "+cadena.matches("Solo"));

// ejemplo2. devolverá true, siempre y cuando no cambiemos la cadena Solo
// .* el punto es cualquier caracter, el asterisco repetición opcional
System.out.println("ejemplo2: "+cadena.matches("Solo.*"));

// ejemplo3. devolverá true
System.out.println("ejemplo3: "+cadena.matches(".*[qnd].*"));
// empieza con cualquier caracter, cualquier número, hay una q o una n o una
// d, y despues cualquier caracter

// ejemplo4. devolverá false, no contiene ninguno de esos caracteres
System.out.println("ejemplo4: "+cadena.matches(".*[xyz].*"));
```



```
// ejemplo5. devolverá true, no contiene ninguno de esos catacteres
System.out.println("ejemplo5: "+cadena.matches(".*[^xyz].*"));

// ejemplo6. devolverá true
System.out.println("ejemplo6: "+cadena.matches("So?lo se qu?e no se n?ada"));
// la interrogación es un caracter opcional, si quitamos los caracteres
anteriores seguirá dando true

// ejemplo7. devolverá true, contiene una letra mayúscula
System.out.println("ejemplo7: "+cadena.matches(".*[A-Z].*"));

String cadena2="abc1234";

// ejemplo8. devolverá true. Con el + indicamos que es obligatorio la
secuencia
System.out.println("ejemplo8: "+cadena2.matches(".*abc+.*"));

// ejemplo9. devolverá true, empieza con abc y a continuación son 4 números
System.out.println("ejemplo9: "+cadena2.matches("abc+\\d{4}"));

// ejemplo10. devolverá true, empieza con abc y a continuación son entre 2 y
10 números
System.out.println("ejemplo10: "+cadena2.matches("abc+\\d{2,10}"));

Scanner entrada=new Scanner(System.in);
String numeros=entrada.nextLine();
if (numeros.matches("[0-9]+[^\.,]")){
    // seguro que es un número entero
    int numero=Integer.parseInt(numeros);
}
```

## 1.4 Conversiones entre tipos

Ya hemos visto a lo largo del curso que podemos realizar distintas conversiones entre tipos, en concreto entre objetos de la clase String y datos primitivos. Éstas conversiones se pueden realizar en ambos sentidos.

```
String textoNumero = String.valueOf(123);
int numeroEntero = Integer.parseInt(textoNumero);
double numeroReal = Double.parseDouble(textoNumero);
long numeroEnteroLargo = Long.parseLong(textoNumero);
```

## 1.5 Clase Character

Una cadena de texto también se puede ver como una sucesión de caracteres individuales, y la posición dentro de la cadena identifica a cada carácter.

En muchas ocasiones necesitaremos tratar cada una de las posiciones de la cadena (o bien alguna de ellas), para realizar alguna operación sobre la misma, por ejemplo, saber si el carácter de una posición es un dígito o es una letra minúscula.

Para ello Java nos ofrece estas “funciones”:

```
Character.isLetter(ch1)           //devuelve true o false
Character.isDigit(ch1)           //devuelve true o false
Character.isSpaceChar(ch1)       //devuelve true o false
Character.isUpperCase(ch1)       //devuelve true o false
Character.isLowerCase(ch1)       //devuelve true o false
Character.toUpperCase(ch1)       //devuelve un char
Character.toLowerCase(ch1)       //devuelve un char
Character.toString(ch1)          //devuelve un String
Character.getType(ch1)           //devuelve la categoría del carácter
```

⚡ Integer, Float, Char, Double, etc, son clases que se llaman envoltorio (wrapper class), porque “envuelven” un tipo primitivo, con el mismo valor que tienen, y proporcionándole una serie de propiedades y métodos útiles.

Notamos también que hay métodos que no se llaman desde un objeto, sino que los estamos llamando desde el nombre de la clase (Integer.parseInt) estos tipos de métodos se llaman estáticos, no necesitamos de un objeto para llamarlos.

## 1.6 StringBuilder y StringBuffer

Además de String, existen otras clases como StringBuffer y StringBuilder que resultan de interés porque facilitan cierto tipo de trabajos y aportan mayor eficiencia en determinados contextos.

La clase StringBuilder es similar a la clase String pero presenta algunas diferencias relevantes:

- Su tamaño y contenido pueden modificarse a 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 (en caso de trabajar con hilos se recomienda `StringBuffer`).

Los constructores de `StringBuilder` se resumen en la siguiente tabla:

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

Los principales métodos de `StringBuilder` son:

- `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
- `capacity`. Devuelve la capacidad del `StringBuilder`
- `length`. Devuelve el número de caracteres del `StringBuilder`
- `reverse`. Invierte el orden de los caracteres del `StringBuilder`
- `setCharAt`. Cambia el carácter indicado en el primer argumento por el carácter que se le pasa en el segundo
- `charAt`. Devuelve el carácter asociado a la posición que se le indica en el argumento.
- `setLength`. Modifica la longitud. La nueva longitud no puede ser menor
- `toString`. Convierte un `StringBuilder` en un `String`
- `insert`. Añade la cadena del segundo argumento a partir de la posición indicada en el primero
- `delete`. Borra la cadena de caracteres incluidos entre los dos índices indicados en los argumentos
- `deleteCharAt`. Borra el carácter indicado en el índice
- `replace`. Reemplaza los caracteres comprendidos entre los dos índices por la cadena que se le pasa en el argumento
- `indexOf`. Analiza los caracteres de la cadena y encuentra el primer índice que coincide con el valor deseado

- SubString. Devuelve una cadena comprendida entre la posición inicial incluida y la final (no incluida)

## Métodos de StringBuilder vs métodos de String

El uso de los métodos de StringBuilder y StringBuffer difiere un poco de los String, ya que como estos últimos eran inmutables, devolvían el resultado del método, pero no modificaban el objeto en sí, por ejemplo:

```
cad.toUpperCase();
```

no modificaba cadena así que hay que llamarla así:

```
cad=cad.toUpperCase();
```

Esto no ocurre con los StringBuilder/StringBuffer, los métodos sí modifican el contenido del objeto, así podríamos hacer directamente:

```
StringBuilder sb = new StringBuilder ("abcdefg");  
sb.delete(3,5);
```

Es frecuente convertir String a StringBuider si necesitamos un método de una de las clases que está disponible en la otra. Por ejemplo, si necesitamos eliminar en la tercera posición de una cadena, podríamos hacer:

```
String cadena = "abcdef";  
StringBuilder sb = new StringBuilder(cadena);  
sb.deleteCharAt(3);  
cadena = sb.toString();
```

abreviando:

```
cadena = new StringBuilder(cadena).deleteCharAt(3).toString();
```