

# Cadenas en Java

Programación 2021/22

## 1.1. Tipo primitivo (char) vs Clase (Character)

- En Java contamos con el tipo primitivo **char** y la clase **Character** que nos permiten trabajar con caracteres.
- Una variable de tipo primitivo **carácter** se declara con **comillas simples** (').
- Una variable de tipo **cadena** se declara entre **comillas dobles** (").
- Existen métodos básicos en la **clase Character** para trabajar con caracteres (los tipos primitivos no tienen métodos):

<b>isUpper, isLower</b>	<b>isLetter</b>
<b>toUpperCase, toLowerCase</b>	<b>isDigit</b>

● § isLowerCase(char) : boolean  
● § isLowerCase(int) : boolean  
● § isUpperCase(char) : boolean  
● § isUpperCase(int) : boolean  
● § isTitleCase(char) : boolean  
● § isTitleCase(int) : boolean  
● § isDigit(char) : boolean  
● § isDigit(int) : boolean

## 1.2. Character.métodos()

```
char car_c = 'c';
```

```
Character.isUpperCase(car_c); // False  
Character.isLowerCase(car_c); // True  
Character.isDigit(car_c);     // False  
Character.isLetter(car_c);    // True
```

```
char car_D = 'D';
```

```
Character.isUpperCase(car_D); // True  
Character.isLowerCase(car_D); // False  
Character.isDigit(car_D);     // False  
Character.isLetter(car_D);    // True
```

```
char car_9 = '9';
```

```
Character.isUpperCase(car_9); // False  
Character.isLowerCase(car_9); // False  
Character.isDigit(car_9);     // True  
Character.isLetter(car_9);    // False
```

```
char car_$ = '$';
```

```
Character.isUpperCase(car_$); // False  
Character.isLowerCase(car_$); // False  
Character.isDigit(car_$);     // False  
Character.isLetter(car_$);    // False
```

## 2. String

⇒ Las variables de este tipo son **INMUTABLES**

Para crear una variable que contenga una cadena de texto podemos hacerlo de diferentes modos:

```
// Declaración e inicialización de cadena vacía  
String cadenaVacía_1 = "";  
String cadenaVacía_2 = new String("");  
String cadenaVacía_3 = new String();
```

## 2.1. String. Métodos

- **length()**: devuelve el tamaño de la cadena de texto
- **charAt(posicion)**: devuelve el carácter que se encuentra en la posición especificada. Comienza a numerar en la posición 0 y finaliza en length()-1

```
61
62     String cadena = "Cadena de texto uno";
63     int longitud = cadena.length();
64     char quintoCaracter = cadena.charAt(5);
65
66
67
68     System.out.println(
69         String.format("La longitud de la cadena es %d", longitud)
70     );
71     System.out.println(
72         String.format("La letra en la posición quinta es %c", quintoCaracter)
73     );
74
```

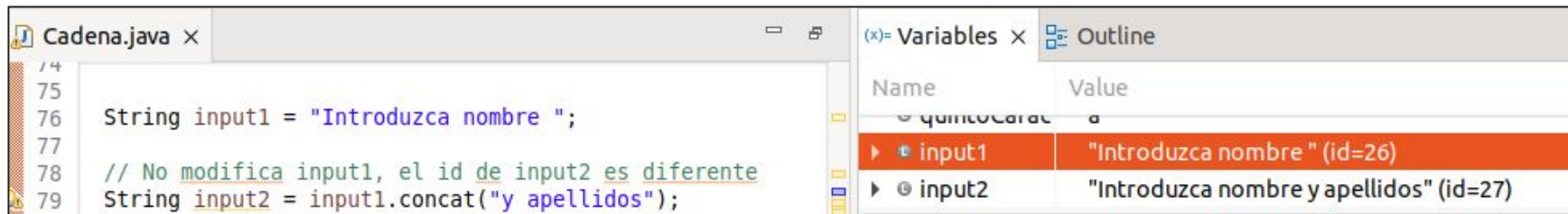
Problems Javadoc Declaration Console x

<terminated> Cadena [Java Application] /home/jm/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.lin

La longitud de la cadena es 19  
La letra en la posición quinta es a

## 2.2. String Métodos. Concatenación

- Al concatenar cadenas creamos nuevos elementos porque las cadenas en Java son Inmutables.
- **concat**(contenido)
- operador **+**



The screenshot shows an IDE window titled 'Cadena.java'. The code is as follows:

```
74  
75  
76 String input1 = "Introduzca nombre ";  
77  
78 // No modifica input1, el id de input2 es diferente  
79 String input2 = input1.concat("y apellidos");
```

To the right of the code editor is a 'Variables' panel. It displays the current state of variables:

Name	Value
input1	"Introduzca nombre " (id=26)
input2	"Introduzca nombre y apellidos" (id=27)

## 2.3. String Métodos ⇒ Búsqueda

- Si no se encuentra la cadena o valor, los métodos devuelven -1
- Si encuentra el valor devuelve la posición, comenzando en 0 y finalizando en n-1

indexOf(valor, posicion)  
lastIndexOf(valor, posicion)

indexOf(valor)  
lastIndexOf(valor)

Caracteres	Cadenas
------------	---------

```
String cadena = "Cadena de texto uno";  
  
int posicionDeLaUMinuscula = cadena.indexOf('u');  
  
int posicionDeLaUltimaE = cadena.lastIndexOf('e');  
posicionDeLaUltimaE = cadena.indexOf('e', 10);  
  
int caracterNoEncontrado = cadena.indexOf('@');
```

```
String cadena = "Cadena de texto uno";  
  
int posicionDeLaPalabraUno = cadena.indexOf("uno");  
  
int posicionDeUltimaDE = cadena.lastIndexOf("de");  
posicionDeUltimaDE = cadena.indexOf("de", 6);
```

## 2.4. String Métodos ⇒ Contiene/Empieza/Termina

- Devuelve true si la palabra/letra a buscar se encuentra en la cadena
- Puede buscarse al principio/fin o en cualquier posición

```
boolean contiene = cadena.contains("de"); //True
```

```
// Similar a  
contiene = cadena.indexOf("de")!=-1;
```

```
boolean empieza = cadena.startsWith("Cad"); //True
```

```
// Similar a  
contiene = cadena.substring(0, "Cad".length()).equals("Cad");
```

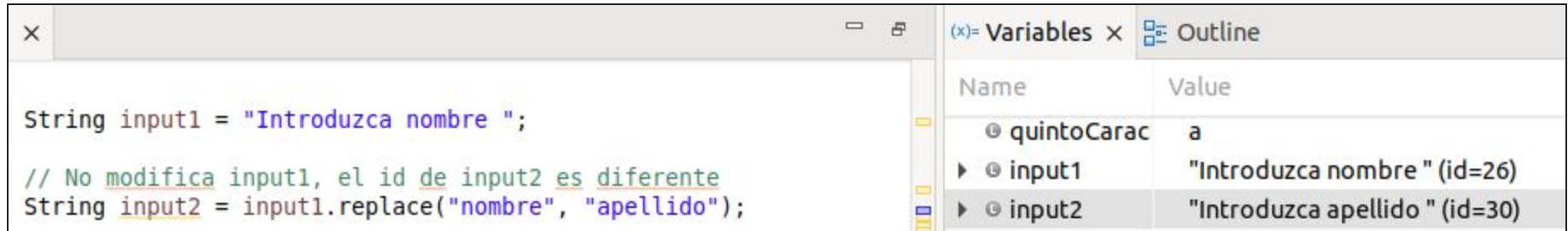
```
boolean termina = cadena.endsWith("uno"); // True
```

```
// Similar a  
contiene = cadena.indexOf("uno", (cadena.length()-"uno".length()-1))!=-1;
```



## 2.4. String Métodos. Reemplazar

- Reemplazar es similar al anterior en el sentido de que cambia la expresión indicada por otra, pero no modifica la cadena de texto original, sino que crea una nueva.
- `replace(buscado, sustituto)`
- Pueden utilizarse caracteres (`char`) o palabras (`CharSequence`, `String`)



The screenshot shows an IDE with a Java code editor on the left and a 'Variables' window on the right. The code defines two String variables: `input1` with the value "Introduzca nombre " and `input2` which is the result of replacing "nombre" with "apellido" in `input1`. The 'Variables' window shows the state of the program, including a variable `quintoCarac` with the value 'a'.

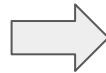
```
String input1 = "Introduzca nombre ";  
  
// No modifica input1, el id de input2 es diferente  
String input2 = input1.replace("nombre", "apellido");
```

Name	Value
quintoCarac	a
input1	"Introduzca nombre " (id=26)
input2	"Introduzca apellido " (id=30)

## 2.5. String Métodos. Formato

- No modifican la cadena original, sino que crean una nueva cadena de texto.
  1. toUpperCase(): convierte a MAYÚSCULAS
  2. toLowerCase(): convierte a minúsculas
  3. trim() ⇒ elimina espacios iniciales y finales (no intermedios)

```
String iesNombre = "    IeS JaCaRanDa    ";  
  
String IES        = iesNombre.toUpperCase();  
String ies        = iesNombre.toLowerCase();  
String iesSinEspacios = iesNombre.trim();
```






▶ ⓘ iesNombre	" IeS JaCaRanDa " (id=36)
▶ ⓘ IES	" IES JACARANDA " (id=37)
▶ ⓘ ies	" ies jacaranda " (id=38)
▶ ⓘ iesSinEspacios	"IeS JaCaRanDa" (id=39)

## 2.6. String Métodos. Recortar (substring)

1. `substring (posicionInicial)`: recorta la cadena desde la posición establecida (incluida) hasta el final.
2. `substring (posicionInicial, posicionFinal)`: igual que el anterior hasta el valor de la `posicionFinal - 1`

⇒ Generan una nueva cadena, no modifica la original

```
String cadena = "Cadena de texto uno";  
  
String cad2 = cadena.substring(10);  
String cad3 = cadena.substring(10, 12);
```

▶  cadena  
▶  cad2  
▶  cad3

"Cadena de texto uno" (id=19)  
"texto uno" (id=40)  
"te" (id=41)

## 2.7. String Métodos. Dividir $\Rightarrow$ split

- `split("separador")`
- Devuelve una lista de elementos separados por el símbolo usado.

```
String cadena          = "Cadena de texto uno";  
String[] palabras     = cadena.split(" ");  
  
String cadena2        = "rojo,verde,azul,amarillo,marrón";  
@SuppressWarnings("unused")  
String[] colores      = cadena2.split(",");
```

(x)= Variables ×		Problems	@ Javadoc	Declaration	Con
Name		Value			
no method r					
▶	cadena	"Cadena de texto uno" (id=19)			
▼	palabras	String[4] (id=43)			
▶	▶ [0]	"Cadena" (id=47)			
▶	▶ [1]	"de" (id=48)			
▶	▶ [2]	"texto" (id=49)			
▶	▶ [3]	"uno" (id=50)			
▶	cadena2	"rojo,verde,azul,amarillo,marrón" (id=44)			
▼	colores	String[5] (id=45)			
▶	▶ [0]	"rojo" (id=51)			
▶	▶ [1]	"verde" (id=52)			
▶	▶ [2]	"azul" (id=53)			
▶	▶ [3]	"amarillo" (id=54)			
▶	▶ [4]	"marrón" (id=55)			

## 2.8. Comparación de cadenas

1. `==` Útil para tipos básicos o comparar por los ids; con cadenas compara con el contenido en la pila de JVM
2. **`equals`**(cadenaAComparar)
3. **`equalsIgnoreCase`**(cadenaAComparar)
4. **`compareTo`**(otroObjeto): nos da el valor numérico del orden lexicográfico de una cadena respecto a la otra (-0+)

## 2.8. Comparación de cadenas

1. `==` Útil para tipos básicos o comparar por los ids; con cadenas compara con el contenido en la pila de JVM
2. **`equals`**(cadenaAComparar)
3. **`equalsIgnoreCase`**(cadenaAComparar)
4. **`compareTo`**(otroObjeto): nos da el valor numérico del orden lexicográfico de una cadena respecto a la otra (-0+)

## 2.8. Comparación de cadenas

```
String nombre1 = "José Manuel";
String nombre2 = new String("José Manuel");

if (nombre1 == nombre2) {
    System.out.println("Son iguales");
}else {
    System.out.println("Los nombres no coinciden");
}
```



```
String nombre1 = "José Manuel";
String nombre2 = new String("José Manuel");

if (nombre1.equals(nombre2)) {
    System.out.println("Son iguales");
}else {
    System.out.println("Los nombres no coinciden");
}
```



```
String nombre1 = "JOSÉ MANUEL";
String nombre2 = new String("José Manuel");

if (nombre1.equals(nombre2)) {
    System.out.println("Son iguales");
}else {
    System.out.println("Los nombres no coinciden");
}
```



```
String nombre1 = "JOSÉ MANUEL";
String nombre2 = new String("José Manuel");

if (nombre1.equalsIgnoreCase(nombre2)) {
    System.out.println("Son iguales");
}else {
    System.out.println("Los nombres no coinciden");
}
```



## 2.8. Comparación de cadenas

- **compareTo(object)**: compara el orden lexicográfico de dos objetos y proporciona un valor negativo si el primero precede al segundo, mayor que cero si es posterior y cero si son iguales.

```
if (nombre1.compareTo(nombre2)==0) {  
    System.out.println("Son iguales");  
}else {  
    System.out.println("Los nombres no coinciden");  
}
```

```
System.out.println("abc".compareTo("xyz")); // devuelve -23  
System.out.println("xyz".compareTo("abc")); // devuelve 23  
System.out.println("abc".compareTo("abc")); // devuelve 0
```