String

1 La clase java.lang.String

1.1 Cadenas vs. referencias a cadenas

Las cadenas de caracteres en Java no son tipos nativos (como `char`, `boolean`, `int` o `float`), sino que son objetos de la clase `String`. Una variable de tipo `String`, por tanto, no almacena una cadena, sino que guarda una referencia (una especie de puntero) a un objeto de la clase `String`, objeto que se ha creado en una zona de la memoria en la que se crean y destruyen objetos dinámicamente (el *heap* o montón).

Ese objeto de la clase `String` almacena la cadena de caracteres propiamente dicha, y contiene una serie de métodos que pueden ser llamados para realizar operaciones sobre la misma.

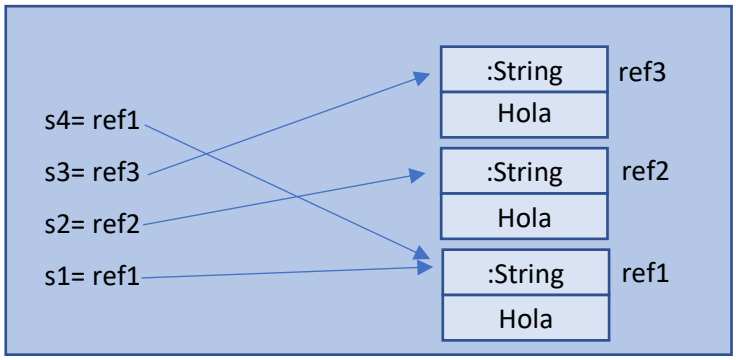
Hay dos formas de declarar una variable de tipo `String`:

- `String s1 = new String("Cadena de ejemplo 1");`
- `String s2 = "Cadena de ejemplo 2";`

La segunda forma es la preferida por ser más eficiente, ya que la primera siempre crea un nuevo objeto `String` para almacenar la cadena, mientras que la segunda no lo hará si ya existe una cadena igual en la memoria.

Por tanto, aunque el valor de la cadena sea el mismo, es posible que en realidad sean dos objetos distintos. Por ejemplo:

- `String s1 = "Hola";` // se crea un nuevo objeto String con referencia ref1
- `String s2 = new String("Hola");` // se crea un nuevo objeto String con referencia ref2
- `String s3 = new String("Hola");` // se crea un nuevo objeto String con referencia ref3
- `String s4 = "Hola";` // no se crea un nuevo objeto String, se asigna la referencia ref1



Por tanto, como `s1` y `s2` almacenan referencias a objetos distintos, la comparación '`s1==s2`' da como resultado `false`. Es por ello que en Java no se utiliza el operador '==', para comparar cadenas, sino que se emplea el método '`equals`' de la clase `String`. Es decir, la llamada '`s1.equals(s2)`' da como resultado `true`, ya que lo que el método '`equals`' comprueba es si los objetos apuntados por ambas referencias tienen el mismo valor.

Lo mismo ocurre con `s1`, `s2` y `s3`, que las tres son referencias a objetos distintos y, por tanto, '`s1==s3`' y '`s2==s3`' también dan ambas un resultado `false`, mientras que '`s1.equals(s3)`' y '`s2.equals(s3)`' dan como resultado `true`.

Por el contrario, tanto '`s1==s4`' como '`s1.equals(s4)`' dan como resultado `true` (una explicación detallada de cómo funciona realmente se proporcionará más adelante, tras presentar algún concepto adicional en teoría).

En todo caso, para evitar errores, es recomendable comparar siempre las cadenas con el método '`equals`'.

1.2 API de la clase String

En la clase `String` se definen un gran número de métodos para realizar las operaciones habituales que se suelen hacer sobre las cadenas (averiguar la longitud de una cadena, convertirla a mayúsculas, obtener una subcadena de la original…). Se puede consultar la sintaxis, parámetros y resultados de todas ellas en la documentación oficial[1] de la clase `String` (hay un enlace en MOOVI).

Los métodos que nos proporciona la clase `String` son métodos que se invocan sobre una instancia de una cadena (una variable `String` o una cadena literal), por lo que la sintaxis de la llamada sería de una de estas formas:

```
variableCadena.metodo(parametros);
"cadena literal".metodo(parametros);
```

A continuación, se presentan algunos de los métodos más útiles, que en ningún caso conforman una lista exhaustiva, para lo cual se recomienda consultar la mencionada documentación oficial.

Métodos que obtienen información sobre la cadena:

- `length()` → devuelve la longitud de la cadena
- `charAt(i)` → devuelve el carácter que está en la posición `i` de la cadena
- `indexOf(c)` → devuelve la posición en la cadena del primer carácter `c`
- `lastIndexOf(c)` → devuelve la posición en la cadena del último carácter `c`
- `split(c)` → divide la cadena en una secuencia de subcadenas (aquellas separadas por el carácter `c`)

Métodos que comparan la cadena:

- `equals(s)` → devuelve `true` o `false` dependiendo de si la cadena es igual o no a la cadena `s`
- `contains(s)` → devuelve `true` o `false` dependiendo de si la cadena contiene la cadena `s`
- `startsWith(s)` → devuelve `true` o `false` dependiendo de si la cadena empieza por la cadena `s`
- `endsWith(s)` → devuelve `true` o `false` dependiendo de si la cadena termina por la cadena `s`
- `compareTo(s)` → devuelve 0, un número positivo, o uno negativo dependiendo de si la cadena es igual, mayor o menor alfabéticamente que la cadena `s`

Métodos que transforman la cadena:

- `toUpperCase()` → devuelve la cadena convertida a mayúsculas
- `toLowerCase()` → devuelve la cadena convertida a minúsculas
- `substring(p1,p2)` → devuelve la subcadena que va desde la posición `p1` (incluida) de la cadena original a la `p2 (excluida)`
- `trim()` → elimina los espacios (tabuladores, fin de línea…) iniciales y finales de la cadena original
- `concat(s)` → añade la cadena `s` a la cadena. Es equivalente al operador '+' (`s1+s2`).
- `replaceAll(s1,s2)` → devuelve la cadena habiendo reemplazado las ocurrencias de `s1` por `s2`

Destacar que estos últimos métodos (los que transforman la cadena), no transforman la cadena original, sino que devuelven una nueva cadena con la transformación aplicada. Por tanto, se usarían de la siguiente forma:

```
s1 = s1.toUpperCase()
```

o asignando el resultado a otra variable si así se prefiere.

[1] Para ello, basta con buscar en Internet algo como "javadoc Oracle String" y enseguida nos aparece un enlace a la página de la documentación (mejor si es la correspondiente al JDK que estamos utilizando).

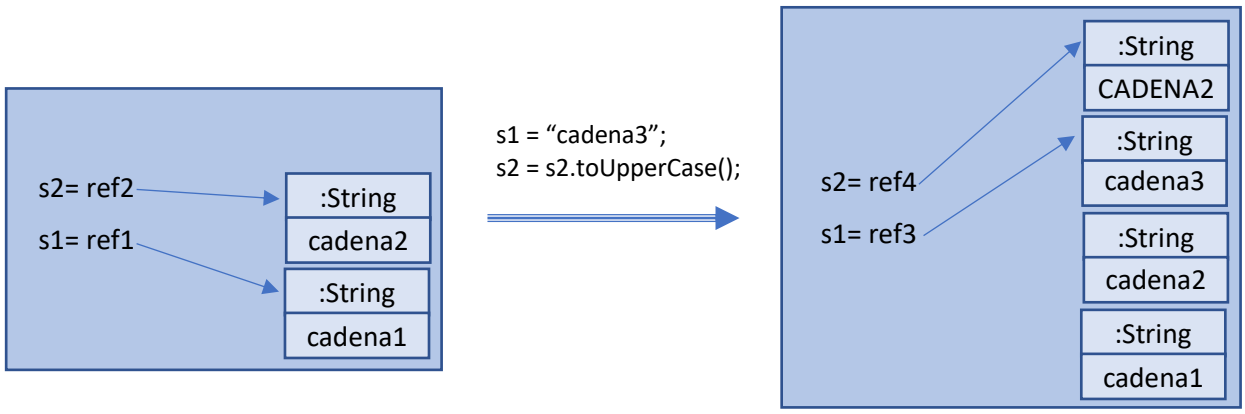
1.3 Inmutabilidad

Es importante destacar que, en Java, las cadenas son inmutables: una vez que se crea un objeto `String`, el contenido de ese objeto no puede cambiarse. Si le asignamos un nuevo valor a la variable, lo que en realidad pasa es que se crea un nuevo objeto `String`. Es decir, no se modifica el objeto original, sino que se crea un nuevo objeto con el nuevo valor de la cadena, y se le asigna su referencia a la variable.

Esto es también así para cualquier método que transforme la cadena original (como los vistos en la sección anterior): como resultado de la ejecución del método no se cambia el objeto original, sino que se crea uno nuevo con el resultado de la transformación, y se devuelve su referencia.

```
String s1 = "cadena1";
String s2 = "cadena2";

s1 = "cadena3";
s2 = s2.toUpperCase()
```



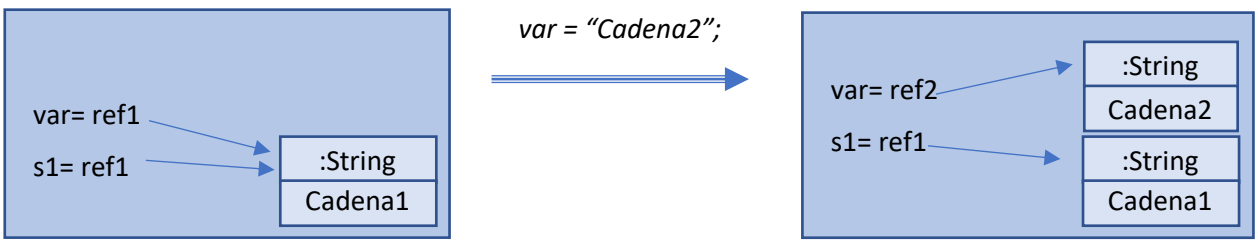
1.4 Paso de cadenas a un método

Cuando se le pasa una cadena (una variable `String` o una cadena literal) como parámetro a un método, lo que realmente se le pasa es la referencia a su objeto `String`. Es decir, no se crea una copia del objeto para el método, y ambos (el método y el programa que lo invoca) trabajan sobre el mismo objeto `String`. El mecanismo de paso de parámetros será explicado con más detalle en breve durante las clases de teoría.

Pero, como las cadenas son inmutables, es claro que el método no puede cambiar el valor del objeto `String` al que apunta la referencia que recibe. Cualquier modificación que haga el método a la variable recibida implicará, como dijimos, la creación de un nuevo objeto `String` con el resultado de esa modificación. En ningún caso el método puede cambiar el valor que ve el programa principal.

```
String s1 = "Cadena1";
objeto.procesar(s1);
```

```
public void procesar (String var) {
      var = "Cadena2";
}
```



Si el método quiere devolver una modificación de la cadena recibida, tendrá que ser a través del resultado devuelto por el método mediante la correspondiente sentencia `return`.