Apunte sobre el tipo TEXTO, Cadena de Caracteres o STRING

Cátedra de Resolución de Problemas y Algoritmos
Facultad de Informática
Universidad Nacional del Comahue

---

1. EL TIPO TEXTO

En la etapa de diseño, al especificar un pseudocódigo de un algoritmo utilizamos el tipo TEXTO para representar una cadena de caracteres. Una variable de este tipo se declara de la siguiente forma:
TEXTO mensaje
mensaje ← "Ingrese la cantidad de alumnos"
ESCRIBIR(mensaje)

En este ejemplo, la variable *mensaje* es de tipo TEXTO. La variable asume un valor particular en la segunda línea a partir de la asignación. Y ese valor, es utilizado para mostrar por pantalla un mensaje al usuario.

Cuando queremos representar el literal de un TEXTO, o la cadena de texto en sí, lo escribimos entre doble comillas. Hemos utilizado cadenas de texto con distintos propósitos. Por ejemplo:

—Cuando queremos mostrar un mensaje en pantalla para indicárle al usuario que ingrese un valor particular.
ESCRIBIR("Ingrese la cantidad de alumnos")

—Cuando queremos producir mensajes más precisos. Por ejemplo en el siguiente ejemplo el módulo ejemplo recibe por parámetro un entero i, y muestra los números naturales desde el uno hasta i. El mensaje por pantalla produce un mensaje indicando: Número: 1, Número: 2, etc.

```
MÓDULO ejemplo(ENTERO i) RETORNA ∅
    (*....*)
    ENTERO x, j
    x ← 1
    PARA j← 0 HASTA i PASO 1 HACER
        ESCRIBIR("Número: " + x)
        x ← x + 1
    FIN PARA
FIN MÓDULO ejemplo
```

.

Las cadenas de texto se representan en Java por el tipo de dato String.

2. STRING EN JAVA

En Java no existe un tipo de dato primitivo que sirva para la manipulación de caracteres. En su lugar se utiliza la clase String. Esto significa que en Java las cadenas de caracteres son objetos que se manipulan como tales, aunque existen ciertas operaciones que nos permiten acceder a ciertas funcionalidades.
Un valor de tipo String es una secuencia de caracteres tratado como un ítem único.
Cuando se dice en Java que una variable es un String lo que se quiere decir en realidad es que la variable es una referencia a un String.

2.1. DECLARACIÓN EN JAVA

Un string se puede declarar de la siguiente forma:

```
String saludo;
saludo = "Hola!";
```

ó:

```
String saludo = "Hola!";
```

ó un String puede crearse como se crea cualquier otro objeto de cualquier clase, mediante el operador new:

```
String saludo = new String("Hola!");
```

Los literales de cadena de caracteres se indican entre comillas dobles, a diferencia de los caracteres que utilizan comillas simples.

2.2. SALIDA POR PANTALLA

Para mostrar el contenido de un String utilizamos la siguiente instrucción:

```
System.out.println(saludo);
```

2.3. CONCATENACIÓN DE STRINGS EN JAVA

Java define el operador + (suma) con un significado especial cuando los operandos son de tipo String. En este caso el operador suma significa concatenación.

En el siguiente ejemplo el resultado de la concatenación es un nuevo String compuesto por las dos cadenas, una tras otra. Por ejemplo:

```
String x = "Problemas " +"y Algoritmos ";
```

da como resultado el String "Problemas y Algoritmos".

Veamos otro ejemplo:

```
String saludo = "Hola";
String sentencia;
sentencia = saludo + " operador";
System.out.println(sentencia);
```

También es posible concatenar a un String datos primitivos, tanto numéricos como booleanos y char. Por ejemplo, se puede usar:

```
int i = 5;
String x = "El valor de i es "+ i;
```

Cuando se usa el operador + y una de las variables de la expresión es un String, Java transforma la otra variable (si es de tipo primitivo) en un String y las concatena.

Otro ejemplo:

```
String solucion;
solucion = "La temperatura es " + 72;
System.out.println (solucion);
```

2.4. CLASES, OBJETOS, Y MÉTODOS EN JAVA

Como dijimos previamente String es una clase. Una clase es un tipo usado para producir objetos. Cada declaración de un String en realidad implica la creación de un objeto de la clase String.

Un objeto es una entidad que almacena datos. Podemos realizar acciones con los objetos invocando las acciones que definen su comportamiento. El comportamiento de un objeto se define a partir de métodos.

Un objeto de la clase String almacena datos que consisten en una secuencia de caracteres. Las operaciones sobre objetos de la clase String se ejecutan invocando los metodos correspondientes.

Por ejemplo el método length() retorna el número de caracteres de un objeto String particular, es decir retorna un entero.

```
String cartel;
int longitudCartel;
cartel = "Bienvenido a Java";
longitudCartel = cartel.length();
System.out.println (longitudCartel);
```

La salida por pantalla del fragmento de código anterior es: 17.

3. OPERACIONES CON CADENA DE CARACTERES

A continuación describiremos en forma de tabla diferentes operaciones que pueden realizarse con el tipo TEXTO o cadenas de caracteres. Mostraremos los métodos para la clase String en JAVA. La tabla presenta cuatro columnas: la primera nos muestra como escribir la primitiva en pseudocódigo, la segunda nos permite ver algunos componentes de la signatura del método (nombre y parámetros) en JAVA, la tercera columna describe brevemente la primitiva (ó método en JAVA) y la cuarta columna muestra un ejemplo.

4. POSICIONES DE LOS COMPONENTES DE UNA CADENA DE CARACTERES

Las variables de tipo TEXTO contienen una cadena de caracteres. Podríamos decir que una variable de tipo TEXTO tiene componentes dentro de sí. Estos componentes se encuentran en distintas posiciones comenzando desde 0. Ejemplos:

En "Java es divertido"la 'J' esta en la posición 0 Otros ejemplos":

```
String cartel = "Bienvenido a Java";
System.out.println(cartel.charAt(4));
System.out.println(cartel.substring(3,9));
```

Cuya salida por pantalla es:

v
nvenid

5. COMPARACIÓN DE CADENA DE CARACTERES

El operador == no es apropiado para determinar si dos cadenas de caracteres (dos strings) tienen el mismo valor.

PSEUDOCÓDIGO	JAVA	Descripción	Ejemplo

longitud(cadena)	cadena.length()	Retorna la longitud de la cadena (un String)	`String saludo= Hola! ;`
			`saludo.length()`
			salida por pantalla: 5

igual(cadena1,cadena2)	cadena1.equals(cadena2)	Retorna true si el contenido *cadena1* y de *cadena2* son iguales, sino retorna false.	`String saludo=TecladoIn.readLine();`
			`if (saludo.equals("Hola"))`
			`System.out.println("Saludo␣informal");`

igualIgnoreMayuscula(cadena1, cadena2)	cadena1.equalsIgnoreCase (cadena2)	Retorna true si el contenido de *cadena1* y de *cadena2* son iguales, considerando una letra en mayúscula y minúscula como iguales, sino retorna false.	`String s1="mary";`
			`s1.equalsIgnoreCase("Mary")`
			retorna true

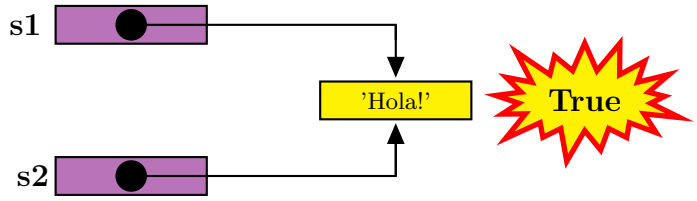
aMinuscula(cadena)	cadena.toLowerCase()	Retorna un String con los mismos caracteres pero convertidos en minúscula	`String s1="Hola␣Mary";`
			`s1.toLowerCase()`
			retorna "hola mary"

aMayuscula(cadena)	cadena.toUpperCase()	Retorna un String con los mismos caracteres pero convertidos en mayúscula	`String s1 =Hola Mary;`
			`s1.toUpperCase()`
			retorna "HOLA MARY"

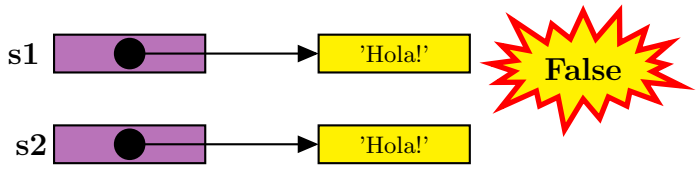
removerEspacios(cadena)	cadena.trim()	Retorna un String con los mismos caracteres que *cadena*, pero remueve los espacios por delante y detrás	`String pausa = "␣Hmmm␣";` `pausa.trim() -- retorna "Hmmm"`

posicion(cadena,pos)	cadena.charAt(*pos*)	Retorna el carácter que en *cadena* se encuentra en la posición *pos*. La posición se cuenta 0, 1, 2, etc.	`String saludo = "Hola!";`
			`saludo.charAt(0) -- retorna "H";`
			`saludo.charAt(3) -- retorna "a";`

(s1 == s2) es *true* si s1 y s2 referencian a la misma locación de memoria



(s1 == s2) es *false* si s1 y s2 no referencian a la misma locación de memoria



Para comparar contenido de Strings hay que utilizar el método **equals**. Ej: s1.equals(s2).

PSEUDOCÓDIGO	JAVA	Descripción	Ejemplo
subcadena(cadena,ini)	substring(*ini*)	Retorna el sub-string de *cadena* desde la posición *ini* hasta el final. Se cuenta 0,1,etc.	`String prueba = "AbcdefG";` `prueba.substring(2) --retorna "cdefG"`
subcadena(cadena, ini,fin)	substring(*ini, fin*)	Retorna el sub-string de *cadena* desde la posición *ini* hasta la posición *fin*, pero sin incluirla. Se cuenta 0,1,etc	`String prueba = "AbcdefG";` `prueba.substring(2,5) -- retorna "cde"`

indiceDe(cadena1, cadena2)	cadena1.indexOf(cadena2)	Retorna la posición de la primer ocurrencia de *cadena2* dentro de *cadena1*. Las posiciones se cuentan 0,1,2,etc. Retorna -1 si *cadena2* no existe dentro de *cadena1*.	`String saludo = "Hola Mary";`
			`saludo.indexOf("Mary") --retorna 5`
			`saludo.indexOf("Sally") --retorna -1`

indiceDe(cadena1, cadena2,ini)	cadena1.indexOf(cadena2, ini)	Retorna la posición de la primer ocurrencia de *cadena2* dentro de *cadena1*, a partir de la posición *ini*. Las posiciones se cuentan 0,1,2, etc. Retorna -1 si *cadena2* no existe dentro de *cadena1*.	`String saludo = "Hola Mary";`
			`saludo.indexOf("Mary",1) --retorna 5`
			`saludo.indexOf("Mary",8) --retorna -1`

ultimoIndiceDe(cadena1, cadena2)	cadena1.lastIndexOf(cadena2)	Retorna la posición de la última ocurrencia de *cadena2* dentro de *cadena1*. Las posiciones se cuentan 0,1,2,etc. Retorna -1 si *cadena2* no existe dentro de *cadena1*.	`String s1 = "Mary, Mary, Mary no";` `s1.lastIndexOf( Mary ) retorna 12`

compararA(cadena1, cadena2)	cadena1.compareTo(cadena2)	Compara *cadena1* con *cadena2* de acuerdo a su orden lexicográfico (que es equivalente al alfabético si las letras están todas en minúsculas o mayúsculas). Si el *cadena1* es menor que *cadena2* devuelve un valor negativo, si son iguales devuelve 0 y si no devuelve un número positivo.	`String entrada = "aventura";`
			`entrada.compareTo("zoo")`
			retorna un valor negativo
			`entrada.compareTo("aventura")`
			retorna 0
			`entrada.compareTo("abajo")`
			retorna un valor positivo

```
1  public class StringEqualityDemo
2  {
3  public static void main(String[] args)
4  {
5  String s1, s2;
6  System.out.println("Ingrese dos lineas de texto:");
7  s1 = TecladoIn.readLine( );
8  s2 = TecladoIn.readLine( );
9  if (s1.equals(s2))
10 System.out.println("Las dos lineas son iguales.");
```

```
11  else
12  System.out.println("Las dos lineas NO son iguales.");
13  if (s2.equals(s1))
14  System.out.println("Las dos lineas son iguales.");
15  else
16  System.out.println("Las dos lineas NO son iguales.");
17  if (s1.equalsIgnoreCase(s2))
18  System.out.println("Pero las lineas son iguales ignorando el tipo de letra.");
19  else
20  System.out.println("Las lineas no son iguales aun ignorando el tipo de letra.");
21  }
22  }
```

La salida por pantalla es la siguiente:

```
Ingrese dos lineas de texto:
Java no es Cafe
Java no es CAFE
s1: Java no es Cafe
s2: Java no es CAFE
Las dos lineas NO son iguales.
Las dos lineas NO son iguales.
Pero las lineas son iguales ignorando el tipo de letra.
```

El orden lexicográfico es similar al orden alfabético pero basado en el orden de las caracteres en ASCII/Unicode.

—Los digitos van antes que todas las letras.

—Las letras mayúsculas van antes que todas las minúsculas.

—El caracter blanco va antes que los digitos y las letras

El metodo compareTo se debe utilizar combinado con el metodo toUpperCase o toLowerCase.

```
1  String s1 = "HoLa";
2  String s2 = "hola";
3  if (s1.toLowerCase().compareTo(s2.toLowerCase()) == 0)
4  System.out.println("Iguales!");
```

Veamos otro ejemplo:

```
1  public class StringDemo {
2    public static void main(String[] args) {
3      String sentencia = "Procesamiento de texto es duro!";
4      int posicion;
5      posicion = sentencia.indexOf("duro");
6      System.out.println(sentencia);
7      System.out.println("01234567890123456789012");
8      System.out.println("La palabra \"duro\" comienza en el indice "+posicion);
9      sentencia = sentencia.substring(0, posicion) + "facil!";
10     System.out.println("El string cambiado es:");
11     System.out.println(sentencia);
12 }
13 }
```

La salida por pantalla es la siguiente:

```
Procesamiento de texto es duro!
01234567890123456789012 3
La palabra "duro" comienza en el indice 26
El string cambiado es:
Procesamiento de texto es facil!
```

6. CARACTERES DE ESCAPE

¿Cómo podríamos imprimir un cartel como el siguiente?

```
"Java" es un lenguaje
```

Necesitamos indicarle al compilador cuando el símbolo doble comillas (") no es señal de inicio o finalización de un string, sino que debe imprimir las comillas. Por ello, debemos anteponer el símbolo 'barra invertida' al símbolo especial que desea imprimir. De la siguiente forma:

```
System.out.println(" \" Java\" es un lenguaje");
```

La siguiente tabla muestra los caracteres de Escape:

$\backslash''$	Comillas dobles
$\backslash t$	Tabulado: Agrega espacios blancos hasta la próxima marca de tabulación
$\backslash'$	Comilla simple
$\backslash\backslash$	Barra inclinada hacia atrás
$\backslash n$	Bajar de línea. Ir al principio de la línea siguiente
$\backslash r$	Ir al principio de la línea actual

Cada secuencia de escape es **un único carácter** aún cuando está escrito con dos símbolos.

Veamos un ejemplo:

```
System.out.println("abc\\def");
System.out.println();
System.out.println("new\nline");
char singleQuote = '\'';
System.out.println(singleQuote);
System.out.println ("+-------------------------+");
System.out.println ("|\ta\t|\tb\t|\tc\t|");
System.out.println ("|\t12\t|\t\t|\t1\t|");
System.out.println ("+-------------------------+");
System.out.print ("\hola");
System.out.println ("\rcomo\estas\--");
```

Cuya salida es:

7. INMUTABILIDAD

La inmutabilidad de los strings es un concepto muy importante en Java. Al contrario de lo que sucede con la mayoría de objetos, un string no puede ser modificado después de ser creado.

```
String s = "string";
s = s + "\modificado";
System.out.println(s);
```

La salida por pantalla es la siguiente: string modificado.

Y sin embargo acabamos de decir que los strings son inmutables ¿se pueden modificar o no entonces?

No. En realidad lo que está pasando es que en la segunda línea se crea un nuevo objeto, un nuevo String fruto de la concatenación de otros dos. **s** apunta ahora a un objeto de tipo string cuyo valor es *string modificado*.

Por esta razón la sentencia:

```
String s = "string";
s.concat(" modificado");
```

no modifica el string original y **s** sigue conteniendo *string* a secas. Para modificar **s** debemos realizar una asignación nueva:

```
s = s.concat(" modificado");"
```