Tabla 2 Métodos y atributos de la clase String (1/9)

Constructores

	String() String(String s) Construye una nueva cadena, nula (que no existe) en el primer caso y una copia de s en el segundo. En ambos casos regresa una referencia donde está la cadena en el heap.

Métodos para crear una nueva cadena a partir de otro(s) objetos(s):

String	concat(String s)
	Crea una nueva cadena en la que aparece la cadena a la que se le solicita concat y se le pega s.
	Ejemplo:
	`String s1 = "Estamos␣jugando␣";`
	`String s2 = s1.concat("a␣las␣matatenas");`
	En s2 queda la cadena Estamos␣jugando␣a␣las␣matatenas. La cadena s1, que es la que invoca, no se modifica.

String	replace(**char** a, **char** b) Crea una nueva cadena en la que reemplaza las presencias del primer carácter por el segundo. **Ejemplo:** `s1 = s2.replace('a','x');` El resultado que queda en la variable s1 es una referencia a la cadena Estxmos␣jugxndo␣x␣lxs␣mxtxtenxs, donde podemos observar que todas las letras a fueron sustituidas por x. Debes notar que s2 no se modifica.

String	replace(String sx, String sy)
	Crea una nueva cadena en la que reemplaza las presencias de la primera cadena por la segunda.
	Ejemplo:
	`String trad = "por";`
	`s1 = s2.replace("ta",trad);`
	Ahora, el resultado que queda en s1 es la referencia a la cadena Espormos jugando a las maportenas. Como podemos observar, s2 permanecía igual que antes del replace anterior y la cadena obtenida tiene mayor tamaño que la que se usó para remplazar.

String String	substring(**int**) substring(**int**, **int**) Crean una nueva cadena que es una subcadena de la cadena original. La subcadena empieza en el primer entero y termina, en el primer caso al final de la cadena y en el segundo caso termina en el segundo entero, pero no lo incluye. El primer carácter de la cadena ocupa la posición 0.

Tabla 2 Métodos y atributos de la clase String (2/9)

	Ejemplo:
	`String corta1 = s1.substring(23);`
	`String corta2 = s1.substring(15,26);`
	En el primer caso, corta1 contiene la referencia a la cadena maportenas, mientras que corta2 contiene una referencia a la cadena do␣a␣las␣ma.

String	toLowerCase() Crea una nueva cadena convirtiendo todos los caracteres a minúsculas. **Ejemplo:** `String minusc = "Esta␣es␣una␣CADENA".toLowerCase();` En minusc queda la referencia a la cadena esta␣es␣una␣cadena.
String	toUpperCase() Crea una nueva cadena convirtiendo todos los caracteres de la que invoca a mayúsculas. **Ejemplo:** `String mayusc = "EsTa␣Es␣UnA␣pRuEbItA".toUpperCase();` Deja en mayusc una referencia a ESTA␣ES␣UNA␣PRUEBITA.
String	trim() Crea una nueva cadena quitando los blancos del principio y final de la invocante. **Ejemplo:** `String sinBlancos = ("␣␣␣" + s1 + "␣␣␣" + mayusc.substring(1,10) + "␣␣␣").trim();` La cadena a la que le aplica trim( ) es la siguiente: ␣␣␣Espormos␣jugando␣a␣las␣maportenas␣␣␣STA␣ES␣UN␣␣␣ Y en sinBlancos queda la cadena: Espormos␣jugando␣a␣las␣maportenas␣␣␣STA␣ES␣UN
static String	valueOf(**boolean**)
static String	valueOf(**char**)
static String	valueOf(**int**)
static String	valueOf(**long**)
static String	valueOf(**float**)
static String	valueOf(**double**)

Tabla 2 Métodos y atributos de la clase String (3/9)

Crea una cadena con el valor que corresponde al tipo del dato. Como es estática se puede llamar desde la clase: String.valueOf(valor).

Ejemplos:

```
int j = 1270;
double Pi = 3.141594;
char c = 'a';
String todos = String.valueOf(j) + "*"
     + String.valueOf(Pi) + "*"
     + String.valueOf(c) + "*";
```

Lo que queda guardado en la cadena todos es la cadena (sin las comillas):

```
1270*3.141594*a*
```

Estas funciones son muy útiles cuando, por ejemplo, tenemos un método que regresa una cadena y lo que tenemos es un valor primitivo, aunque conseguimos volverlo cadena simplemente pegándolo:

```
String  todosA = "" + j + "*" + Pi + "*"
  + c + "*";
```

La cadena todos contiene exactamente lo mismo que todosA.

Métodos de comparación:

int compareTo(String)

Comparan dos cadenas en el orden del código Unicode.

Regresa
- 0 si las cadenas son idénticas
- > 0 si $\langle cad1 \rangle$ va después en el orden que $\langle cad2 \rangle$.
- < 0 si $\langle cad1 \rangle$ va antes en el orden que $\langle cad2 \rangle$.

Ejemplo:

```
String s1 = "Arturo Matluk     ";
int menorQue = s1.compareTo("Arturito Magidin");
```

Como s1 es igual a Arturito Magidin hasta la posición 4 (contando desde 0), pero en la posición 5 s1 tiene una 'o' que va después de la 'i', la cadena en s1 es mayor que la cadena Arturito Magidin por lo que queda en menorQue es un entero mayor que cero. Si hiciéramos la comparación:

```
int menorQue = "Arturito Magidin".compareTo(s1);
```

el resultado de menorQue va ser un entero menor que cero. Si imprimimos cada uno de estos valores, el primero es 6, que es la distancia de la 'i' a la 'o', mientras que el segundo es -6, que es la distancia de la 'o' de regreso a la 'i'.

Tabla 2 Métodos y atributos de la clase String (4/9)

	Si una de las subcadenas es prefijo de la otra, la que es prefijo va a ser menor que la otra.

boolean	equals(Object)
	Dice si la cadena en el parámetro es idéntica a la que invoca.
	Ejemplo
	`String s1 = "Hola␣␣␣␣";`
	`String s2 = "Hola␣␣␣␣␣";`
	`boolean iguales = s1.equals(s2);`
	En este caso iguales toma el valor **false** porque una cadena tiene más blancos que la otra.
	Si hacemos
	`iguales = (s1.trim()).equals(s2.trim());`
	El resultado de la comparación va a ser **true** porque a ambas cadenas les quitamos los blancos del final.

boolean	equalsIgnoreCase(String)
	Dice si la cadena en el argumento es igual a la que invoca, ignorando diferencias entre mayúsculas y minúsculas.
	Ejemplo:
	`String sM = "EsTa␣Es␣UnA␣pRuEbItA";`
	`boolean bSM = sM.equalsIgnoreCase(sM.toLowerCase());`
	`boolean bS = sM.equals(sM.toLOwerCase();`
	El resultado guardado en la variable bSM es **true**, porque iguala los casos en que se comparan mayúsculas con minúsculas o viceversa. En cambio, el valor de la variable bS es **false** porque considera distintas mayúsculas que las minúsculas.

boolean	endsWith(String)
	Dice si la cadena con la que se invoca termina con la cadena en el parámetro.
	Ejemplo:
	`String sOff = "todos␣los␣apagadores␣estan␣en␣off";`
	`boolean bOff = sOff.endsWith(sOff.substring(20));`
	El valor de la variable bOff por supuesto que es **true**, porque estamos tomando el final de la cadena con substring. Pero si la asignación fuera
	`bOff = (sOff+"␣␣␣␣").endsWith("off");`
	El valor de bOff sería **false** pues estamos verificando que la cadena entre paréntesis, a la que le agregamos cuatro blancos a la terminación, termina con `"off"` y sus últimas tres posiciones son blancos.

Tabla 2 Métodos y atributos de la clase String (5/9)

boolean	startsWith(String)
boolean	startsWith(String , **int**)

Determina si es que la cadena empieza con la cadena que trae como argumento. En la segunda versión, compara a partir de la posición denotada por el argumento entero.

Ejemplo:

```
String esta = "abecedario␣en␣espanhol";
boolean bEmpzaCon = esta.startsWith("abece");
```

El valor de bEmpzaCon es **true**, ya que la cadena esta empieza con abece. En cambio, en

```
bEmpzaCon = esta.startsWith("abece",2);
```

bEmpzaConda como valor **false**, ya que la subcadena abece no empieza en la posición 3.

Métodos de búsqueda:

int	indexOf(**int**)
int	indexOf(**int** , **int**)
int	indexOf(String)
int	indexOf(String , **int**)

El primer entero corresponde al código de un carácter en Unicode (se puede pasar como argumentos también un carácter). La cadena se refiere a una subcadena. En las cuatro versiones, regresa la primera posición en la cadena que invoca donde se encuentra el primer parámetro. Si se da un segundo parámetro, éste indica que se busque a partir de esa posición. Regresa -1 si no encuentra lo que está buscando.

Ejemplos:

```
char c = 'r';
int k = "Enero␣␣␣FebreroMarzo".indexOf(c,4);
```

k queda valiendo 11, porque como busca a partir de la posición 4, la 'o' de Enero␣␣␣, encuentra la primera 'r' de Febrero que está en la posición 11 de la cadena que invoca. Si a continuación de este enunciado tenemos:

```
int j = "Enero␣␣␣FebreroMarzo".indexOf(c,k);
```

j quedará valiendo 13, que es la posición que ocupa en la cadena invocante la tercera 'r'.

int	lastIndexOf(**char**)
int	lastIndexOf(**char**, **int**)
int	lastIndexOf(String)
int	lastIndexOf(String , **int**)

Tabla 2 Métodos y atributos de la clase String (6/9)

El carácter corresponde a un carácter (se puede pasar como argumentos también un código entero de un carácter en Unicode). La cadena se refiere a una subcadena. En las cuatro versiones regresa la última posición en la cadena que invoca donde se encuentra el primer parámetro. Si se da un segundo parámetro, éste indica que se busque únicamente hasta esa posición el carácter buscado o el inicio de la cadena solicitada. Regresa -1 si no encuentra lo que está buscando.

Ejemplos:

```
1 String anita = "anita␣lava␣la␣tina";
2 System.out.println("lastIndexOf('t'):\t"
3         + anita.lastIndexOf('t'));
4 System.out.println("lastIndexOf('t',10):\t"
5         + anita.lastIndexOf('t',10));
6 System.out.println("lastIndexOf(\"ni\"):\t"
7         + anita.lastIndexOf("ni"));
8 System.out.println("lastIndexOf(\"ni\",0):\t"
9         + anita.lastIndexOf("ni",0));
```

El resultado de ejecutar este código se encuentra a continuación.

```
lastIndexOf('t'):␣␣␣␣␣␣␣14
lastIndexOf('t',10):␣␣␣␣3
lastIndexOf("ni"):␣␣␣␣␣␣1
lastIndexOf("ni",0):␣␣␣␣−1
```

En el primer caso revisa toda la cadena y la última presencia del carácter `t` está en la posición 14 (recuerda que empiezan en 0); en el segundo caso, como revisa sólo hasta la posición 10 y la última `t` está en la posición 14, la encuentra en la posición 3; el tercer caso es similar al primero y reporta dónde empieza la subcadena; en el cuarto caso, como únicamente le dejas ver la primera posición, no encuentra que ahí empiece la cadena ni, por lo que reporta −1.

boolean regionMatches(**int** esteDesde, String arg, **int** argDesde, **int** tam)

boolean regionMatches(**boolean** ignoraMayusc, **int** esteDesde, String arg, **int** argDesde, **int** tam)

Determina si una región de la cadena que invoca es igual a una región de la cadena en el argumento. esteDesde indica la posición a partir de la cual se va a comparar; arg es la cadena que se va a tratar de aparear; argDesde indica la posición en la cadena arg a partir de la cual se va a comparar; finalmente, tam indica el número de posiciones a comparar en ambas cadenas.

La segunda versión, si ignoraMayusc es verdadero, compara ignorando diferencias entre mayúsculas y minúsculas, con el resto de los argumentos teniendo el mismo significado que en la primera versión.

Tabla 2 Métodos y atributos de la clase String (7/9)

Ejemplo: Usaremos tanto variables de cadena como cadenas armadas "al vuelo". Ambas pueden jugar el papel de invocante (**this**) o de argumento en los métodos no estáticos de cadenas.

```
String sNo = "no tiene otra cosa que hacer";
System.out.println( "Cadena origen: \n*"
                    + (anita + " si " + sNo)
                    + "*\nCadena destino: \n*"
                    + (anita.substring(14)
                       + " porque " + sNo)
                    + "*");
boolean bMatch =
  (anita + " si " + sNo).regionMatches(22,
           (anita.substring(14)
            + " porque " + sNo),
           12,10);
System.out.println("Resultado de la comparacion: "
                   + bMatch);
```

Lo que escribe en la consola este código se encuentra a continuación:

```
Cadena origen:
*anita lava la tina si no tiene otra cosa que hacer*
Cadena destino:
*tina porque no tiene otra cosa que hacer*
Resultado de la comparacion: true
```

Si en lugar de tener 12 en el tercer argumento tuviésemos 11, el resultado en bMatch sería **false**, porque la posición 22 en la cadena invocante tiene una n mientras que la cadena en el argumento en la posición 11 tiene un blanco.

Los casos en los que este método regresa **false**, además de cuando no aparea, son si se presenta al menos uno de:

- Al menos uno de esteDesde o argDesde es negativo.
- esteDesde + tam $\geq$ tamaño del invocante.
- argDesde + tam $\geq$ tamaño del argumento.

Métodos de conversión de un tipo a otro

char charAt(**int**)

Regresa el carácter que se encuentra, en la cadena que invoca, en la posición dada por el argumento ("convierte" de cadena a carácter).

Ejemplo:

```
char car = (anita + sNo).charAt(15);
System.out.println("Cadena origen:\n*"
                   + anita + sNo);
System.out.println
  ("El caracter en la posicion 15 es: "
     + "'" + car + "'");
```

Tabla 2 Métodos y atributos de la clase String (8/9)

El resultado de ejecutar este código es:

```
Cadena origen:
*anita lava la tinano tiene otra cosa que hacer
El caracter en la posicion 15 es: 'i'
```

String

toString()

Genera la representación en cadena del objeto con el que se le invoca. Si se trata de un objeto que no tenga definido este método, lo toma (lo hereda) de la clase Object y simplemente nos da la dirección en el heap donde está guardado el objeto (el valor de la variable de referencia). Si la clase define este método, regresará la descripción, como cadena, que hayas programado en esa clase.
Para los tipos primitivos tenemos "cajas" como:

Tipo primitivo	Objeto "caja"
char	**Character**
boolean	**Boolean**
int	**Integer**
long	**Long**
short	**Short**
byte	**Byte**
double	**Double**
float	**Float**

En el caso de los tipos primitivos, Java se encarga de "encajar" (*boxing*) y "desencajar" (*unboxing*) al valor primitivo en el tipo de referencia correspondiente y convertir el valor en una cadena. Es por eso que cuando aparecen variables primitivas concatenadas con una cadena simplemente toma la cadena correspondiente y la usa (a través del método toString() de la caja correspondiente).
Ejemplos:

```
Scanner sc = new Scanner(System.in);
System.out.println( sc.toString());
System.out.println( "*" + sc + "*" );
Object objeto = new Object();
System.out.println( "El objeto es:*"
   + objeto + "*");
```

Lo que obtenemos de este código se encuentra a continuación (aparece en la consola en una sola línea):

```
ava.util.Scanner[delimiters=\p{javaWhitespace}+]
[position=0][match valid=false][need input=false]
[source closed=false][skipped=false][group separ
ator=\,][decimal separator=\.][positive prefix=]...
.........
```

Tabla 2 Métodos y atributos de la clase String (9/9)

	Sin embargo, el último enunciado imprime en la consola lo siguiente: `El objeto es: *java.lang.Object@43affd0*` que describe, simplemente, de qué clase es el objeto y una referencia a la posición de memoria donde fue creado. Este es el valor por omisión que regresa el método `toString()`.

Otros métodos

int	`length()`
	Regresa el tamaño de la cadena invocante, el número de caracteres.
	Ejemplo:
	`String hola = "␣␣␣␣␣Hola␣que␣tal␣␣␣␣␣";`
	`System.out.println(hola.length());`
	`System.out.println(hola.trim().length());`
	El valor de `hola.length()` es 22, mientras que el de `hola.trim().length()` es 12, pues se le quitan los cinco blancos del principio y los cinco blancos del final.
	Este método es de mucha utilidad porque rara vez, cuando estamos trabajando, conocemos de antemano el tamaño de las cadenas que nos da un usuario de nuestro proyecto o si nos dio una cadena vacía (tecleó inmediatamente el Enter al pedírsele una cadena).

Tabla 3 Algunos métodos de la clase Scanner (10/2)

Modificador y tipo	Firma y descripción
void	close() Cierra el Scanner referido por this.
boolean	nextBoolean() Revisa la entrada para ver si el siguiente elemento es un valor booleano, lo procesa y regresa el valor.
byte	nextByte() Interpreta el siguiente elemento de la entrada como un byte.
double	nextDouble() Interpreta el siguiente elemento de la entrada como un double.
float	nextFloat() Interpreta el siguiente elemento de la entrada como un float.
int	nextInt() Interpreta el siguiente elemento de la entrada como un int.
String	nextLine() Avanza la entrada para pasar el fin de linea y regresa lo que proceso hasta antes del fin de linea.
long	nextLong() Interpreta el siguiente elemento de la entrada como un long.
short	nextShort() Interpreta el siguiente elemento de la entrada como un short.