LAS CADENAS

Las cadenas son necesarias si queremos guardar cualquier información de tipo alfanumérico. También son utilizadas para guardar nombres, direcciones, etcétera. Para poder utilizar cadenas necesitamos hacer uso de la clase **String**, que nos da todo lo necesario para poder trabajar con ellas. En su interior la cadena se guarda como una colección y cada carácter es un elemento.

EJEMPLO

```
String miCadena = "Hola a todos"; String dato = "";
```

EL MÉTODO TOSTRING()

Un método común para muchas clases que existen en C# y .NET es **ToString()**. Éste es usado para convertir el tipo de dato o su representación a una cadena y es muy útil cuando necesitamos desplegar esa información para que el usuario la pueda leer. El método siempre regresará una cadena, que puede ser formateada para nuestras necesidades.

CÓMO CONVERTIR Y FORMATEAR FECHAS A CADENAS

Uno de los usos más comunes de las cadenas es el de poder formatear información, que puede ser de cualquier tipo. Uno bastante común que no hemos utilizado hasta el momento es el de las fechas. Para utilizar la fecha y la hora, lo haremos mediante una clase conocida como **DateTime**. Esta clase provee todos los elementos necesarios para poder trabajar con la información relacionada con el tiempo, como son la fecha y las horas del día. Si deseamos tener un objeto de este tipo y que tenga la hora actual de la máquina en el momento en que se instanció podemos hacerlo de la siguiente manera:

EJEMPLO

```
DateTime miTiempo = DateTime.Now;
```

La propiedad **Now** de **DateTime** contiene el tiempo actual del sistema en su interior. Ahora lo único que necesitamos es poder pasar esa información a un formato útil y práctico de trabajar para nosotros. Podemos hacer uso del método **Format()** de **String**. Este método nos permite reemplazar los contenidos de la cadena por una cadena con un formato en particular.
Para llevar a cabo un formato debemos usar los especificadores de formato. **DateTime** contiene un conjunto de estos especificadores. A continuación veamos los más importantes de estos, presentados en la siguiente tabla de una forma clara de leer y comprender.

ESPECIFICADOR	DESCRIPCIÓN
d	Día del mes.
dd	Día del mes, pero los primeros días se representan con 01, 02, 03, etcétera.
ddd	Presenta el nombre del día abreviado.
dddd	Presenta el nombre del día completo.
h	Hora en el rango de 1 a 12.
hh	Hora en el rango de 1 a 12. Las primeras horas se presentan como 01, 02, 03, etcétera.
H	Hora en el rango de 0 a 23.
HH	Hora en el rango de 0 a 23. Las primeras horas se presentan como 01, 02, 03, etcétera.
m	Minuto.
mm	Minuto. Los primeros minutos se muestran como 01, 02, 03, etcétera.
M	Mes.
MM	Mes. Los primeros meses se presentan como 01, 02, 03, etcétera.
MMM	Nombre del mes abreviado.
MMMM	Nombre del mes completo.
s	Segundos.
ss	Segundos. Los primeros segundos se presentan como 01, 02, 03, etcétera.
t	Despliega A o P según AM o PM.
tt	Despliega AM o PM.
yyyy	Muestra el año.

OTROS EJEMPLO

```
String  formato;
Formato = String.Format("La fecha es: {0:dddd yyyy M}",DateTime.Now);

string name = "Fred";
String.Format("Name = {0}, hours = {1:hh}", name, DateTime.Now);

string primes;
primes = String.Format("Prime numbers less than 10: {0}, {1}, {2}, {3}",
                       2, 3, 5, 7 );
Console.WriteLine(primes);

string multiple = String.Format("0x{0:X} {0:E} {0:N}",
                                Int64.MaxValue);
Console.WriteLine(multiple);

int value = 6324;
string output = string.Format("{0}{1:D}{2}",
                              "{", value, "}");
```

Para mas información visite:
https://docs.microsoft.com/es-es/dotnet/standard/base-types/formatting-types

PARA DARLES FORMATO A VALORES NUMÉRICOS

Al igual que con la información de la fecha y hora, es posible darles formato a los valores numéricos que deseemos mostrar. Para éstos también tendremos una serie de especificadores que podemos utilizar.

ESPECIFICADOR	DESCRIPCIÓN
#	Dígito
.	Punto decimal
,	Separador de miles
%	Porcentaje
E0	Notación científica
;	Separador de secciones

OTROS EJEMPLO

```
Console.WriteLine(String.Format("{0:$#,###0.00;Negativo
$#,###0.00;Cero}", valor));
```

CÓMO CONCATENAR CADENAS

Otra de las manipulaciones que podemos hacer es la concatenación, que en C# a diferencia de otros lenguajes es muy natural e intuitiva. Esto se debe a que podemos utilizar una función especializada para esto o aprovechar la sobrecarga del operador + para la clase **String**. Si deseamos llevar a cabo la concatenación por medio del operador + podemos hacerlo de la siguiente manera:

```
String nombre = "Juan"; String
apellido = "Lopez";
String NombreCompleto = "";
NombreCompleto = nombre + " " + apellido;
```

Método **Concat()**

Otra forma de concatenar es hacer uso del método **Concat()**. Este método pertenece a **String**, es muy sencillo de utilizar y solamente requiere de dos parámetros, que serán las cadenas que deseamos concatenar. El método regresa una cadena, que es la cadena resultante de la concatenación de los parámetros.

EJEMPLO

```
NombreCompleto = String.Concat(nombre, apellido);
```

El método **Concat()** es estático, por eso es posible usarlo sin tener que declarar un objeto de la clase **String**. Para llevar a cabo una concatenación múltiple, podemos hacerlo de la siguiente manera:

EJEMPLO

```
NombreCompleto =String.Concat(nombre, String.Concat(“ “, apellido));
```

USO DE STRINGBUILDER

StringBuilder es una clase que provee .NET y podemos utilizar con C#. Nos permite construir cadenas de forma eficiente y a su vez podemos utilizarla en lugar de las concatenaciones si fuera necesario. Podemos utilizar el constructor de **default** que no necesita de ningún parámetro. Hay una versión en la que podemos pasar como parámetro una cadena con la que podemos inicializar el objeto **StringBuilder**.
La clase tiene varias propiedades que podemos usar, como la propiedad **Capacity**, que nos indica la capacidad actual, que puede o no ser igual a la cantidad de caracteres que tenemos guardados en su interior, aunque generalmente será mayor.
Otra opción es la propiedad **Chars**, mediante la que podemos obtener o modificar un carácter en particular. La modificación se hace indicando el índice donde se encuentra ese carácter. Lo podemos hacer de la siguiente manera:

EJEMPLO

```
StringBuilder sb = new StringBuilder(“Hola a todos”);
...
... sb.Chars[6]=’o’;
```

Si leemos la propiedad de **Length** obtenemos la longitud que tiene el **StringBuilder**. Si colocamos un valor en esta propiedad entonces el **StringBuilder** tendrá el tamaño que indicamos. Si el nuevo tamaño es menor que el actual, entonces se trunca el contenido del **StringBuilder**.
Para llevar a cabo una concatenación, debemos usar el método **Append()**. Este método tiene muchas sobrecargas ya que no solamente puede concatenar cadenas. El método necesita solamente un parámetro y es el valor a concatenar en la cadena. Hay que recordar que **Append()** es eficiente.

EJEMPLO

```
StringBuilder sb = new StringBuilder(“Hola a todos”);
...
...
sb.Append(“Hola mundo”);
```

También podemos agregar una cadena de formato construida de forma similar a la forma como trabajamos con los formatos de **WriteLine()**. Para esto usamos el método **AppendFormat()**. Este método necesitará dos parámetros, pero puede llevar más si fuera necesario. En el primero colocamos la cadena de forma y en el segundo y/o los siguientes la lista de variables a utilizar.

EJEMPLO

```
StringBuilder sb =  new StringBuilder();
int  valor =  5;
...
...
sb.AppendFormat("El  valor es  {0}", valor);
```

Un método importante en esta clase es **ToString()**, que permitirá convertir **StringBuilder** en una cadena, y de esta manera poder utilizar los contenidos creados por medio de esta clase. Para usarlo simplemente hacemos una asignación:

EJEMPLO

```
String cadena =  "";
cadena = sb.ToString();
```

COMPARACIÓN DE CADENAS

La clase **String** nos da el método **Compare()**. Este método es estático, por lo que podemos hacer uso de éste sin necesidad de declarar un objeto de tipo **String**. El método necesitará dos parámetros, que son las cadenas a comparar. Los llamaremos en nuestro ejemplo **Cadena1** y **Cadena2**. El método regresará luego del análisis, un valor entero y el valor de este entero será el que se encargue de indicarnos la relación que existe entre las cadenas que acabamos de comparar.
Si el valor regresado es igual a **0**, esto significa que **Cadena1** y **Cadena2** son iguales. En el caso de que el valor sea menor que **0**, es decir, un número negativo, **Cadena1** es menor que **Cadena2**. Si se recibe un número mayor que **0** significa que **Cadena1** es mayor que **Cadena2**.

EJEMPLO

```
int  comparación = 0;
String nombre="Juan";
...
...
comparación=String.Compare("Pedro",nombre);

if(comparación==0)
Console.WriteLine("Los  nombres son  iguales");
    else
            Console.WriteLine("Los  nombres son  diferentes");
```

Otra forma que podemos utilizar para saber si dos cadenas son iguales es hacer uso del método **Equals()**, que solamente dirá si dos cadenas son iguales, no realiza ninguna otra comparación. Existen varias sobrecargas del método **Equals()**, por lo que es conveniente aprender el resto con la ayuda de MSDN. La versión que veremos aquí es estática y requiere dos parámetros. Los parámetros son las cadenas a comparar. El método regresa un **bool** con el valor de **true** si las cadenas son iguales y el valor de **false** si son diferentes.

EJEMPLO

```
String cadena1 =  “Hola”;
String cadena2 =  “Todos!”;
...
...
if( String.Equals(cadena1,cadena2) == true)
    Console.WriteLine(“Las cadenas son iguales”);
```

PARA ENCONTRAR UNA SUBCADENA

La cadena puede contener una frase con muchas palabras y para algunas aplicaciones necesitamos saber si determinada palabra se encuentra en esa cadena. Es decir que buscaremos una **subcadena** adentro de otra cadena. Esta **subcadena** es una cadena que queremos saber si se encuentra en la cadena principal.
Esto puede servirnos en búsquedas o cuando necesitamos llevar a cabo determinada lógica si se encuentra una palabra o frase en particular. El método que usaremos se llama **Contains()** y debe ser invocado por la cadena a la que se le realizará la búsqueda. Esto es importante y no debemos olvidarlo ya que la invocación de Contains() en la cadena incorrecta nos puede llevar a errores de lógica.

El método **Contains()** sólo necesita un parámetro. Este parámetro es la **subcadena** a buscar. Ésta puede ser dada explícitamente o por medio de una variable de tipo **String** que la contenga. El método regresa un **bool**. Si la **subcadena** se encontró, el valor regresado es true y si no se encontró el valor regresado es **false**.

EJEMPLO

```
String NombreCompleto =  “Juan Pedro Lopez”;
String NombreBuscar =  “Pedro”;
...
...
if (NombreCompleto.Contains(NombreBuscar)   == true)
    Console.WriteLine(“El  nombre se   encuentra”);
```

PARA OBTENER UNA SUBCADENA

Ahora que ya tenemos una forma de saber si existe una subcadena, también podemos extraer una **subcadena**. Esto nos puede servir para los casos en los que necesitemos tener solamente una parte de la cadena original. La forma cómo lo hacemos es por medio del medio del método **Substring()**.
Para poder usar este método necesitamos contar con dos parámetros, que son valores de tipo entero. El primero indica el índice adentro de la cadena original donde inicia la **subcadena** que nos interesa obtener y el segundo es la cantidad de caracteres que tiene la **subcadena**. El método regresa una cadena que contiene a la **subcadena** que hemos obtenido.

EJEMPLO

```
String cadena=”Hola  mundo redondo”;
String resultado=””;
...
...
resultado=cadena.Substring(5,5);
```

PARA DETERMINAR SI UNA CADENA FINALIZA EN UNA SUBCADENA

Es posible saber si una cadena finaliza en una **subcadena** en particular. Por ejemplo, si analizamos la cadena, y ésta termina en punto, o si la cadena termina en una palabra en particular. Al igual que en los casos anteriores, tenemos un método que nos permite llevar a cabo esto. El método es **EndsWith()** y necesita un parámetro, que será la **subcadena** que deseamos verificar en la terminación de la cadena principal. La función regresa un valor de tipo **bool**, si es **true** la cadena termina en la **subcadena** en caso contrario, el valor regresado es de **false**.

EJEMPLO

```
String cadena1 =  “Juan Pedro Lopez”;
String cadena2 =  “Lopez”;
...
...
if(cadena1.EndsWith(cadena2) == true)
    Console.WriteLine(“La cadena finaliza en Lopez”);
```

CÓMO COPIAR UNA PARTE DE LA CADENA

En muchas aplicaciones es necesario obtener o copiar una parte en particular de una cadena, y si la cadena contiene una frase, podemos extraer una palabra en particular. Cuando necesitamos procesar cadenas es muy útil hacer esto.
El método que usaremos se conoce como **CopyTo()** y aunque es un poco más complejo que los anteriores es fácil de utilizar ya que necesita el mismo de cuatro parámetros. Antes de ver los parámetros es necesario comentar que la cadena extraída será colocada en un arreglo de tipo **char**. El método no regresa ningún valor.
El primer parámetro es el **índice** adentro de la cadena principal desde donde se empezará a copiar. El segundo parámetro es la **cadena** de **tipo char** donde se colocará la cadena a extraer. El tercer parámetro es el **índice** del **arreglo** a partir de donde se empezarán a colocar los caracteres de la cadena copiada. Esto es útil ya que podemos hacer la copia en cualquier posición del arreglo, no solamente al inicio. El cuarto parámetro indica la **cantidad** de **caracteres** a **copiar**. Con estos parámetros podemos indicar sin problema cualquier sección a copiar.

EJEMPLO

```
char[] destino =  new char[10];
String cadena = “Hola a todos  mis amigos”;
...
...
cadena.CopyTo(7,  destino, 0, 5);
```

CÓMO INSERTAR UNA CADENA

Ya hemos visto una manera fácil de cómo podemos extraer una **subcadena**, sin embargo, también es posible llevar a cabo la inserción de una cadena en otra. Esto nos ayuda cuando necesitamos colocar información nueva en la cadena, pero ya no es posible hacerlo por medio de **concatenación** o **formato**.
Para poder hacer esto, recurriremos a utilizar al método **Insert()**, que nos brinda C#. Este método coloca la cadena dentro de la cadena principal. Este también necesitará dos **parámetros**. El **primero** indicará el índice dentro de la cadena principal donde se va a insertar la nueva cadena. Este valor debe de ser de **tipo entero** y nunca podemos pasarle un **valor negativo**. El **segundo parámetro** es la cadena que deseamos insertar, que puede ser colocada explícitamente o mediante una variable de tipo **String**.
El método **Insert()** regresa un valor de tipo **String**. Este valor regresado sería la instancia de la nueva cadena que ya contiene la inserción. Un ejemplo para llevar a cabo la inserción sería de la siguiente forma:

EJEMPLO

```
String cadena1 = “Hola a todos”;
String cadena2 = “hola ”;
String resultado = “”;
...
...
resultado = cadena1.Insert(5,  cadena2);
```

PARA ENCONTRAR LA POSICIÓN DE UNA SUBCADENA

En algunas situaciones necesitamos encontrar dónde se encuentra determinada subcadena para poder llevar a cabo alguna operación en ese índice en particular. Para poder hacer esto hacemos uso del método **LastIndexOf()**. Un punto importante a tener en cuenta es que este método nos regresa el índice de la última posición encontrada, es decir que si la cadena principal tiene dos repeticiones de esa cadena, nos da el índice de la segunda repetición.
Este método tiene varias sobrecargas, pero la que veremos sólo necesita un parámetro y es la subcadena a encontrar. Como siempre la cadena puede ser dada de forma explícita o por medio de una variable de tipo **String**. El método regresa un valor de tipo entero que contiene el índice de la última ocurrencia de la subcadena.

EJEMPLO

```
int índice = 0;
String cadena = “Hola a todos. Hola”;
…
…
índice=cadena.LastIndexOf(“Hola”);
```

JUSTIFICACIÓN DEL TEXTO

Hasta el momento hemos visto métodos para manipular las cadenas dentro de C#, pero no hemos hablado aún que la clase **String**, que nos provee de varios más. A continuación, veremos algunos que nos permiten hacer otras modificaciones.
Si necesitamos justificar el texto hacia la derecha es posible hacerlo. La forma cómo funciona esto es la siguiente. Supongamos que tenemos una cadena de 10 caracteres como “0123456789”. Para justificarla necesitamos saber el tamaño de la cadena resultante con la justificación incluida. Supongamos que la cadena resultante será de 25 caracteres. Esto nos daría 15 caracteres del que tenemos que insertar del lado izquierdo para obtener “0123456789” que se encuentra justificada hacia la derecha. Para esto necesitamos un método que nos permita empotrar esos caracteres de espacio. El método **PadLeft()** se encarga de esto. Requiere de un parámetro que es la cantidad de caracteres de la cadena final. Éste es un valor entero y representa los caracteres originales más los espacios en blanco. Regresa la cadena final justificada.

EJEMPLO

```
String cadena=”Hola”;
String resultado=””;
…
…
resultado=cadena.PadLeft(10);
```

De forma similar, podemos hacer una justificación hacia la izquierda. En este caso, los caracteres en blanco serán insertados a la derecha de la cadena. Se insertarán tantos caracteres como sean necesarios hasta llegar al tamaño deseado.
Para esto usaremos un método equivalente a **PadLeft()**, pero hacia la derecha. Este método es conocido como **PadRight()**. El método necesita un parámetro, que es un valor entero que indica la cantidad total de caracteres de la cadena final. Este total debe ser igual a los caracteres de la cadena original más los caracteres vacíos. El método regresa la cadena justificada.

PARA ELIMINAR CARACTERES DE LA CADENA

Otra manipulación que podemos hacer sobre la cadena es la eliminación de caracteres. Es posible borrar determinada parte de la cadena, según sea lo que necesitemos en la lógica del programa.
El método que podemos utilizar se conoce como **Remove()**. Este método está sobrecargado, pero veremos la versión que es más flexible. La eliminación de los caracteres puede hacerse en cualquier parte de la cadena, sólo debemos tener cuidado de no generar ningún error.

El método **Remove()** necesita dos parámetros, ambos de valores enteros. El primer parámetro se usa para indicar el índice a partir del cual se empezarán a eliminar los caracteres de la cadena, y en el segundo parámetro colocamos la cantidad de caracteres a eliminar. El método regresa una cadena, que es la cadena resultante de la eliminación.

EJEMPLO

```
String cadena=”Hola mundo, hola a todos”;
String resultado=””;
...
...
resultado=cadena.Remove(12,4);
```

CÓMO REEMPLAZAR UNA SUBCADENA

Reemplazar una parte de la cadena principal con otra cadena puede ser un proceso que toma tiempo programar. Esto se debe a que necesitamos encontrar la subcadena a eliminar, luego eliminarla y al final introducir los caracteres de la nueva cadena. Como siempre, la clase **String** nos provee de un método que nos facilita la manipulación de la cadena para el reemplazo. Éste es el método **Replace()**. Existen dos versiones del método **Replace()**, una de ellas funciona con caracteres y la que aprenderemos en esta ocasión hace uso de cadenas.
Esto nos puede permitir reemplazar una palabra en particular contenida dentro de la cadena por otra palabra. El método llevará a cabo el reemplazo en todas las ocurrencias de la palabra que tengamos.

El método es sencillo de utilizar y necesita dos parámetros. El primero es la cadena que simboliza la palabra que deseamos reemplazar. El segundo es la cadena con la que la reemplazaremos. El método regresa una cadena, que es la resultante con los reemplazos ya llevados a cabo.

EJEMPLO

```
String cadena1=”Hola a todos. Hola mundo”;
String cadena2=”Adios”;
String resultado=””;
...
...
resultado=cadena1.Replace(“Hola”,cadena2);
```

Por el ejemplo, vemos que es posible colocar el parámetro ya sea de forma explícita o por medio de una variable.

CÓMO DIVIDIR LA CADENA

Otro problema clásico con la manipulación de las cadenas es la subdivisión cuando se encuentra algún carácter en particular. Por ejemplo, si la cadena contiene un párrafo de un texto, podemos dividirla en subcadenas, cada una de ellas delimitada por un signo de puntuación. Para lograr esto necesitamos tener un arreglo de caracteres que contenga los caracteres que tomaremos

como delimitadores. Cada vez que el método encuentre uno de estos caracteres, llevará a cabo el corte de la subcadena.
El método que usaremos se conoce como **Split()**, éste sólo requiere de un parámetro, que es el arreglo de caracteres delimitadores. El método regresará un arreglo de tipo **String**. Cada uno de los elementos presentes en el arreglo regresado será una de las subcadenas que se habrán recortado. Después debemos proceder a hacer uso de las cadenas en el arreglo y procesarlas o utilizarlas según sea necesario en la aplicación que estamos desarrollando.

EJEMPLO

```
String cadena1 = (”Hola a todos. Este es un ejemplo, de lo que podemos hacer.”);
String resultado[] = cadena1.Split(new char[] { ‘,’, ‘.’, ‘;’ });
```

Como podemos ver en el ejemplo, usamos los caracteres punto, coma y punto y coma como delimitadores. Adentro de resultado estarán las cadenas que han sido recortadas de **cadena1**.

INTERCAMBIAR MAYÚSCULAS Y MINÚSCULAS

En muchas ocasiones tendremos cadenas que estarán escritas con letras en mayúscula y minúscula mezcladas, pero puede suceder que para facilitar la lógica de aplicación debamos tener la cadena exclusivamente en mayúscula o minúscula.
Esto nos puede ayudar a reducir la cantidad de comparaciones o búsquedas que necesitamos hacer. Si lo que deseamos hacer es convertir la cadena a minúscula, entonces debemos hacer uso del método **ToLower()**. Este método no necesita ningún parámetro. La fuente de información para hacer la conversión es la misma cadena que lo invoca, pero en esta ocasión nos daremos cuenta de que el método sí se encargará de regresar un valor, el valor que se devuelve será la cadena convertida totalmente a letras minúsculas.

EJEMPLO

```
String cadena=”Hola Hola”;
String resultado=””;
…
…
resultado=cadena.ToLower();
```

Al finalizar el código la variable resultado tendrá en su interior a la cadena “hola hola”. De forma similar, podemos pasar la cadena a mayúscula. La forma de hacer esto es con el método **ToUpper()**, que toma la cadena y pasa todas sus letras a mayúscula. El método no necesita ningún parámetro, ya que al igual que **ToLower()**, toma la información directamente de la cadena que lo invoca y regresa una cadena, que es la resultante con todas las letras en mayúscula.

EJEMPLO

```
String cadena=”Hola  Hola”;
String  resultado=””;
…
…
resultado=cadena.ToUpper();
```

La variable resultado tendrá en su interior **“HOLA HOLA”**. Los dos métodos tienen otra versión que tiene en cuenta las reglas culturales para poder llevar a cabo la conversión. En ese caso tendríamos que pasar como parámetro el identificador de la cultura que se usará como base para convertir los caracteres.
En MSDN podemos encontrar la información sobre los diferentes identificadores de cultura que puede manejar .NET. Para hacer uso del identificador de cultura debemos usar un objeto de tipo **CultureInfo**. En su constructor debemos pasar el ID de la cultura correspondiente.

Por ejemplo, para pasar a mayúscula con las reglas de la cultura en México podemos colocar lo siguiente:

```
String cadena=”Hola  Hola”; String  resultado=””;
…
…
resultado=cadena.ToUpper(new  CultureInfo(“es-MX”));
```

CÓMO PODAR LA CADENA

Cuando trabajamos con las cadenas podemos encontrarnos con situaciones como cuando la cadena tiene exceso de espacios, ya sea al inicio o al final. Algunas veces esto puede ser útil, como cuando justificamos, pero otras veces estos espacios extras pueden ser indeseados. Los espacios extras al inicio o al final pueden deberse a operaciones realizadas sobre la cadena o simplemente a entradas erróneas del usuario, y para librarnos de estos caracteres tenemos diferentes opciones. En primer lugar conoceremos un método que elimina los espacios en blanco extras tanto al inicio como al final de la cadena. Este método se conoce como **Trim()**.
El uso de **Trim()** es muy sencillo ya que no necesita ningún parámetro, simplemente trabajará sobre la cadena que lo invoca. Este método regresará una cadena nueva, que es la cadena original sin los espacios extras.
Es necesario que tengamos en cuenta que la cadena original no se modifica, por esta razón recibiremos una cadena completamente nueva.

EJEMPLO

```
String cadena=”      Hola  a  todos.      “;
String  resultado=””;
…
…
resultado=cadena.Trim();
```

En la cadena resultado tendremos **“Hola a todos”**, que es la cadena sin los espacios extras. El método **Trim()** poda los espacios tanto del inicio como del final de la cadena. Sin embargo, puede haber ocasiones en las que necesitemos podar únicamente el inicio de la cadena. El método para lograr esto es **TrimStart()**. Este método es un poco más complejo que **Trim()** ya que necesita un parámetro. Este parámetro es un arreglo de caracteres y en él tenemos que colocar los caracteres que nos interesa extraer del inicio de la cadena. El método regresará otra cadena, que es la resultante de la cadena original sin los caracteres podados a su inicio.
Es posible que creemos el arreglo de caracteres o también que lo coloquemos explícitamente. Es conveniente pensar bien cuáles serán los caracteres a podar para evitar la eliminación de caracteres que sí pueden ser útiles.

EJEMPLO

```
String cadena=”x x x x x Hola a todos.      “;
String resultado=””;
…
…
resultado=cadena.TrimStart(‘ ‘, ‘x’);
```

En este ejemplo se podan los caracteres espacio y **x**. La cadena final resultante es “Hola a todos. “. Debemos notar que los caracteres al final no han sido podados, ya que solamente trabaja sobre los caracteres al inicio de la cadena.
Si lo que necesitamos es podar el final de la cadena, entonces tenemos que usar el método **TrimEnd()**. Este método es equivalente a **TrimStart()**, pero funciona únicamente al final de la cadena.
Recordemos que el método necesita un parámetro. El parámetro es un arreglo de caracteres. En este arreglo es necesario que coloquemos los caracteres que deseamos eliminar del final de la cadena con la que trabajamos. El método correspondiente se encargará de regresar una cadena, está será el resultado sin los caracteres que fueron eliminados al final de la cadena original.

EJEMPLO

```
String cadena=”x x x x x Hola a todos.      “; String
resultado=””;
…
…
resultado=cadena.TrimEnd(‘ ‘, ‘x’);
```

En la cadena de resultado tendremos **“x x x x x Hola a todos”**, el espacio fue eliminado al final ya que se encuentra dentro de la lista de caracteres a podar.