Unidad 5

DIPLOMATURA EN PROGRAMACION .NET

Capítulo 9

Cadenas y Expresiones Regulares

En este capítulo

- Manejo de cadenas
- La Clase String
- La clase System.Text.StringBuilder
- Formatos numéricos
- Introducción a la globalización
- Acceso a la Información de la Cultura
- Formatear y Ordenar datos sensibles a la cultura
- Realización de ordenamientos y comparaciones de texto teniendo en cuenta la cultura
- Expresiones regulares
- La enumeración RegexOptions
- La clase Regex
- Comprendiendo las expresiones regulares
- Gestión de patrones de búsqueda
- Encontrando patrones en cadenas con expresiones regulares
- Métodos estáticos y de instancia de las expresiones regulares
- Extracción de datos usando expresiones regulares

Manejo de cadenas

Una cadena es un tipo de datos incorporado y primitivo. Los datos primitivos mapean al tipo cadena en la clase System.String. Los objetos de la clase String (o string) son inmutables por naturaleza. Por inmutable significa que el estado del objeto no se puede cambiar por cualquier operación. Esta es la razón por la que cuando se llama a ToUpper() en una cadena, no se cambia la original, sino que crea y retorna un nuevo objeto del tipo cadena que es la representación en mayúsculas del objeto original. La versión mutable (cuyo contenido se puede alterar) de la cadena en la plataforma .Net es de clase System.Text.StringBuilder. Los objetos de esta clase son mutables, es decir, su estado se puede cambiar mediante sus operaciones. Por lo tanto, si se llama al método append() en un objeto de la clase StringBuilder, la nueva cadena se añadirá a (agregará al final de) el objeto original.

La Clase String

Se ha estado utilizando la clase String desde el principio. También se han visto algunas de sus propiedades (como Length) y métodos (como Equals ()).

Una cadena de texto o String, es un conjunto de caracteres Unicode que están dispuestos en un orden específico de manera que tengan sentido para el lector.

El Framework. NET proporciona el objeto System.String para representar cadenas en el código. El objeto String es una colección secuencial inmutable de objetos System.Char que representa una cadena de texto. El valor del objeto String es el contenido de la colección representa como un valor único objeto.

La clase String es un tipo por referencia, una instancia de la clase String no contiene la cadena como un vector que sea una propiedad de la misma (aunque se usa muchas veces este concepto por motivos didácticos), sino que contiene una dirección a un espacio en la memoria que contiene realmente la cadena.

El tipo String permite almacenar texto en la memoria. También proporciona funcionalidad para realizar operaciones ordinales y sensibles a la cultura. Las operaciones ordinales se ejecutan según el valor numérico de cada objeto del tipo Char que compone la cadena, como por ejemplo, la igualdad. Las operaciones sensibles a la cultura tienen que ver cómo se interpreta una determinada cadena en un entorno de ejecución configurado según elementos culturales (como el alfabeto, caracteres diacríticos, etc…) de una determinada región del mundo. Este concepto se explicará posteriormente.

Los objetos del tipo System.String son inmutables. Esto significa que, después de crear un objeto, su contenido no se puede cambiar. Aunque pueda parecer que el tipo de cadena dispone de varios métodos que cambian la cadena, esto es un error. Cualquier método que parece modificar una cadena retornará siempre una referencia a un nuevo objeto del tipo String. El objeto cadena que

se pasa en el método permanece en la memoria hasta que no hay referencias que apuntan al mismo, en cuyo punto el recolector de basura lo puede quitar. Por lo tanto, si la cadena retornada por un método se asigna nuevamente a la referencia que se utilizó para invocarlo, esto provoca que se descarte el primer objeto haciéndolo seleccionable para el recolector de basura y se asigna la nueva referencia retornada por el método a la que se tenía previa a la invocación.

Nota: La clase String posee, por ejemplo, constructores que permiten manejar punteros a memoria. Para poder manejar un atributo o método con punteros se debe declara un contexto "unsafe". Esta explicación está fuera del alcance de la actual y sólo se la menciona para que le lector conozca su existencia, pero no se explicarán elementos de clase declarados de esta manera. La razón es que no es aconsejable usar código "unsafe".

A continuación, se van a describir algunos de los constructores de la clase String.

Constructor	Descripción
String(Char[])	Inicializa una nueva instancia de la clase String con el valor indicado por un vector de caracteres Unicode.
String(Char, Int32)	Inicializa una nueva instancia de la clase String con el valor indicado por un carácter Unicode especificado que se repite un número determinado de veces.
String(Char[], Int32, Int32)	Inicializa una nueva instancia de la clase String con el valor indicado por un vector de caracteres Unicode, una posición de carácter inicial dentro de dicho vector, y una longitud.

Esta clase posee un atributo estático de solo lectura y público llamado "Empty" que representa una cadena de 0 caracteres de longitud y es utilizado por muchos de sus métodos miembros.

A continuación, se van a describir las propiedades de la clase String.

Propiedad	Descripción
Length	Obtiene el número de caracteres que contiene el objeto String.
Char	Obtiene el objeto Char en una posición especificada en el objeto String actual.

A continuación, se van a describir algunos de los métodos más comunes de la clase String.

Método	Retorna	Significa	Parámetros	Argumento Recibido	Descripción
Compare	**int**	Si es menor a cero el primer argumento es menor que el segundo. Si es igual a cero el primer argumento es igual que el segundo. Si es mayor a cero el primer argumento es mayor que el segundo.	**Object**	Primer objeto a comparar	Compara dos objetos y retorna el resultado. Este método es estático.
			Object	Segundo objeto a comparar	
CompareTo	**int**	Si es menor a cero la instancia actual es menor que el argumento. Si es igual a cero la instancia actual es igual que el argumento. Si es mayor a cero la instancia actual es mayor que el argumento.	**Object**	El objeto a comparar con la instancia actual	Compara la instancia actual del objeto con otra instancia del mismo tipo y retorna el resultado
Concat	**string**	La cadena que resulta de concatenar la cadena actual con la que resulta del argumento. Si este es null usa String.Empty.	**string**	La primera cadena que se concatenará	Realiza una concatenación de la cadena actual con la que recibe de argumento o el valor retornado por el ToString de éste.
			string	La segunda cadena que se concatenará	
Contains	**bool**	Retorna true o True si encuentra la cadena recibida en la actual o si es String.Empty	**string**	La cadena que se verificará si existe en la actual	Retorna un valor indicando si se encuentra la cadena recibida como argumento dentro de la actual
Copy	**string**	La misma cadena que se copió sobre la referencia del argumento	**string**	Referencia donde se copia la cadena	Crea una nueva instancia de String con el mismo valor de la actual. Este método es estático.
EndsWith	**bool**	Si se encuentra la cadena recibida como argumento retorna true o True , sino false o False	**string**	La cadena que se verificará si existe al final de la actual	Determina si la cadena actual termina con la subcadena que recibe como argumento
Equals	**bool**	Retorna true o True si ambos objetos son iguales	**Object**	Cadena a comparar con la actual	Determina si dos cadenas son iguales

Método	Retorna	Significa	Parámetros	Argumento Recibido	Descripción
IndexOf	**int**	Posición a partir de 0 donde se encuentra el carácter o -1 si no lo encuentra	**char**	Carácter a buscar	Determina la primera ocurrencia de un carácter dentro de una cadena
IndexOf	**int**	Posición a partir de 0 donde se encuentra la subcadena o -1 si no lo encuentra. Si la cadena es String.Empty, retorna 0	**string**	La cadena a buscar	Determina la primera ocurrencia de una cadena dentro de la actual
Insert	**string**	La nueva cadena con la que recibió insertada	**int**	Índice a partir del cual insertar	Retorna una nueva cadena con la que recibió insertada a partir del índice indicado
			string	Cadena a insertar	
LastIndexOf	**int**	Posición a partir de 0 donde se encuentra la subcadena o -1 si no lo encuentra. Si la cadena es String.Empty, retorna 0	**char**	Carácter a buscar	Determina la última ocurrencia de un carácter dentro de una cadena
LastIndexOf	**int**	Posición a partir de 0 donde se encuentra la subcadena o -1 si no lo encuentra. Si la cadena es String.Empty, retorna la posición del último caracter	**string**	La cadena a buscar	Determina la última ocurrencia de una cadena dentro de la actual
Replace	**string**	Una cadena equivalente a la actual excepto por los caracteres reemplazados	**char**	Carácter a ser reemplazado	Retorna una nueva cadena reemplazando todos los caracteres del primer argumento con el del segundo. Si no encuentra ocurrencias del segundo retorna la cadena actual.
			char	Carácter que reemplaza todas las ocurrencias del reemplazado	
Replace	**string**	Una cadena equivalente a la actual excepto por las cadenas reemplazadas	**string**	La cadena a ser reemplazada	Retorna una nueva cadena reemplazando todas las cadenas del primer argumento con la del segundo. Si no encuentra ocurrencias del segundo retorna la cadena actual.
			string	La cadena que reemplaza todas las ocurrencias de la reemplazada	

Método	Retorna	Significa	Parámetros	Argumento Recibido	Descripción
Split	**string[]**	Un vector que resulta de separar la cadena actual según los separadores indicados	**params char[]**	Vector de caracteres separadores. El parámetro puede ser nulo	Retorna un vector de subcadenas que se generan separando en base a los caracteres recibidos como argumentos. Si dos caracteres son adyacentes, ese elemento en el resultado es String.Empty. Si es null el argumento, separa por espacios en blanco
StartsWith	**bool**	Si se encuentra la cadena recibida como argumento retorna true o True , sino false o False	**string**	La cadena que se verificará si existe al comienzo de la actual	Determina si la cadena actual empieza con la subcadena que recibe como argumento
Substring	**string**	Retorna una nueva cadena generada a partir del índice recibido	**int**	Índice a partir del cual se generará la subcadena	Retorna una nueva cadena a partir del índice en base a 0 indicado en el argumento
Substring	**string**	Retorna una nueva cadena generada desde el primer índice recibido hasta el segundo	**int**	Índice a partir del cual se generará la subcadena	Retorna una nueva cadena que se genera como una subcadena de la actual desde y hasta los índices recibidos
			int	Índice que determina el final de la subcadena a generar	
ToCharArray	**char[]**	Un vector con los caracteres que componen la cadena actual	**N / A**	N / A	Genera un nuevo vector de caracteres en el cual cada elemento es cada uno de los que componía la cadena actual.
ToLower	**string**	Una cadena con todos los caracteres en minúsculas	**N / A**	N / A	Crea una nueva cadena a partir de la actual convirtiendo todos sus caracteres en minúsculas
ToUpper	**string**	Una cadena con todos los caracteres en mayúsculas	**N / A**	N / A	Crea una nueva cadena a partir de la actual convirtiendo todos sus caracteres en mayúsculas

Método	Retorna	Significa	Parámetros	Argumento Recibido	Descripción
Trim	**string**	Una cadena sin caracteres en blanco ni al principio ni al final	**N / A**	N / A	Genera una nueva cadena removiendo los espacios en blanco al principio y al final de la misma
TrimEnd	**string**	Una cadena sin caracteres en blanco al final	**N / A**	N / A	Genera una nueva cadena removiendo los espacios en blanco al final de la misma
TrimStart	**string**	Una cadena sin caracteres en blanco al principio	**N / A**	N / A	Genera una nueva cadena removiendo los espacios en blanco al principio de la misma

El siguiente programa se mostrará el uso de muchos de los métodos enumerados en la tabla para la clase String

Ejemplo

C#

```
namespace cadenas
{
    class Program
    {
        static void Main(string[] args)
        {
            string s1 = "palabras";
            string s2 = "palabra";
            string s3 = "Palabras";
            string s4 = "Este es un ejemplo de manejo de cadenas";
            string s5 = " Este es el texto a manipular ";
            Console.WriteLine("El largo de {0} es {1}", s1, s1.Length);
            Console.WriteLine("El resultado de comparar {0} con {1} es {2}", s1, s2,
              s1.CompareTo(s2));
            Console.WriteLine("Verificando igualdad entre {0} y {1} retorna {2}",
              s1, s3, s1.Equals(s3));
            Console.WriteLine(
      "Verificando igualdad entre {0} en minúscula con {1} ({2}) retorna {3}",
              s1, s3, s3.ToLower(), s1.Equals(s3.ToLower()));
            Console.WriteLine("El índice de \"a\" en {0} es {1}", s3,
              s3.IndexOf('a'));
            Console.WriteLine("El último índice de \"a\" en {0} es {1}", s3,
              s3.LastIndexOf('a'));
            Console.WriteLine("Las palabras individuales de \"{0}\" son", s4);
            string[] palabras = s4.Split(' ');
            foreach (string palabra in palabras)
            {
                Console.WriteLine("\t {0}", palabra);
            }
```

```
            Console.WriteLine("\nla subcadena de \n\t\"{0}\"\n"
              + "desde el índice 3 hasta el índice 10 es \n\t\"{1}\"",
              s4, s4.Substring(3, 10));
            Console.WriteLine("\nLa cadena \n\t\"{0}\"\nsin espacios "
              + "en blanco es\n\t\"{1}\"", s5, s5.Trim());
            Console.WriteLine("\nLas cadenas originales contenadas separadas "
              + "por punto: \n {0}", String.Concat(s4,String.Concat(". ",s5)));
            Console.ReadKey();
        }
    }
}
```

VB

```
Module Module1

    Sub Main()
        Dim s1 As String = "palabras"
        Dim s2 As String = "palabra"
        Dim s3 As String = "Palabras"
        Dim s4 As String = "Este es un ejemplo de manejo de cadenas"
        Dim s5 As String = " Este es el texto a manipular "
        Console.WriteLine("El largo de {0} es {1}", s1, s1.Length)
        Console.WriteLine("El resultado de comparar {0} con {1} es {2}",
              s1, s2, s1.CompareTo(s2))
        Console.WriteLine("Verificando igualdad entre {0} y {1} retorna {2}",
              s1, s3, s1.Equals(s3))
        Console.WriteLine(
       "Verificando igualdad entre {0} en minúscula con {1} ({2}) retorna {3}",
              s1, s3, s3.ToLower(), s1.Equals(s3.ToLower()))
        Console.WriteLine("El índice de 'a' en {0} es {1}", s3, s3.IndexOf("a"))
        Console.WriteLine("El último índice de 'a' en {0} es {1}", s3,
              s3.LastIndexOf("a"))
        Console.WriteLine("Las palabras individuales de '{0}' son", s4)
        Dim palabras() As String = s4.Split(" ")
        For Each palabra As String In palabras
            Console.WriteLine(vbTab + " {0}", palabra)
        Next
        Console.WriteLine(vbNewLine + "la subcadena de " + vbNewLine + vbTab + _
                "'{0}'" + vbNewLine + "desde el índice 3 hasta el índice 10 es " + _
              vbNewLine + vbTab + "'{1}'", s4, s4.Substring(3, 10))
        Console.WriteLine(vbNewLine + "La cadena " + vbNewLine + vbTab + "'{0}'" + _
              vbNewLine + "sin espacios en blanco es" + _
              vbNewLine + vbTab + "'{1}'", s5, s5.Trim())
        Console.WriteLine(vbNewLine + _
              "Las cadenas originales contenadas separadas por punto:  " + _
              vbNewLine + vbTab + "{0}", String.Concat(s4, String.Concat(". ", s5)))
        Console.ReadKey()
    End Sub
End Module
```


La clase System.Text.StringBuilder

Esta clase tiene un comportamiento muy similar a la clase anterior salvo que se debe utilizar el constructor para crear una nueva instancia. El estado interno de la cadena que se almacenar en el

interior del objeto puede ser modificado, esto implica que se puede mutar. El valor se dice que es mutable, ya que puede ser modificado después de la creación por adición, eliminación, sustitución o inserción de caracteres. Como las cadenas de caracteres no se pueden modificar (son inmutables y las modificaciones siempre generan una nueva cadena) cada vez que se desee cambiar el contenido de las mismas se debe utilizar esta clase.

La mayoría de los métodos que modifican una instancia de esta clase retornan una referencia a la misma instancia y se puede llamar a un método o propiedad mediante dicha referencia. Esto puede ser conveniente si se quiere escribir en una sola declaración operaciones con cadenas que se sucedan unas a otras por notación de puntos. Sin embargo es preferible evitar este tipo de técnicas porque enrarecen el código y se hace más difícil de depurar.

La capacidad de una instancia de StringBuilder es el número máximo de caracteres que puede almacenar la misma en un momento dado. La capacidad es mayor que, o igual a, la longitud de la cadena que representa el valor de la instancia. La capacidad puede ser aumentada o disminuida con la propiedad Capacity o con el método EnsureCapacity, pero no puede ser menor que el valor de la propiedad de longitud. Para realizar una operación de ese tipo, primero se deben borrar caracteres para disminuir la longitud.

El tamaño actual de un objeto StringBuilder se define por su propiedad Length. Se puede acceder a los caracteres en la cadena que es representada por un objeto del tipo StringBuilder mediante la propiedad Chars. Las posiciones de índices comienzan *siempre* desde cero.

La clase StringBuilder incluye métodos que pueden reducir el tamaño de la instancia actual. El método Clear quita todos los caracteres y se establece la propiedad de longitud cero. El método Remove elimina una serie de caracteres.

La clase StringBuilder también incluye métodos que pueden expandir la instancia actual. Los métodos Append y AppendLine agregan datos al final del objeto del tipo StringBuilder, y el método Insert inserta datos en una posición de carácter especificada en el objeto del tipo StringBuilder actual. El método AppendFormat permite dar formato a una cadena que se quiera agregar como texto al final de un objeto del tipo StringBuilder.

El método Replace reemplaza todas las apariciones de un carácter o una cadena en todo el objeto del tipo StringBuilder o en un intervalo en particular.

Se debe convertir el objeto del tipo StringBuilder en un objeto String para poder pasar la cadena representada por el objeto del tipo StringBuilder a un método que tiene un parámetro String o mostrarlo en la interfaz de usuario. Se debe realizar esta conversión llamando al método ToString.

En un objeto del tipo String, la operación de concatenación siempre crea un nuevo objeto a partir de la cadena existente y aquello que se añadirá. Un objeto del tipo StringBuilder mantiene un

buffer para dar cabida a la concatenación de datos nuevos. Estos se anexan a la memoria intermedia si el espacio está disponible, de lo contrario, es asignado un nuevo buffer más grande, los datos del búfer original se copian en el búfer de nuevo y los nuevos datos se adjuntan a continuación en el nuevo buffer.

El rendimiento de una operación de concatenación para un objeto del tipo String o StringBuilder depende de la frecuencia de las asignaciones en memoria. Una operación de concatenación de cadenas siempre asigna memoria, mientras que una operación de concatenación con StringBuilder asigna memoria sólo si el buffer del objeto del tipo StringBuilder es demasiado pequeño para almacenar a los nuevos datos. Utilizar la clase String si se está concatenando un número fijo de objetos del tipo String. En ese caso, el compilador puede, incluso, combinar las operaciones individuales de concatenación en una sola. Utilizar un objeto del tipo StringBuilder si está concatenando un número arbitrario de cadenas, por ejemplo, si se está utilizando un ciclo para concatenar un número aleatorio de cadenas ingresadas por el usuario.

Los constructores de la clase son:

Constructor	Descripción
StringBuilder()	Inicializa una nueva instancia de la clase StringBuilder.
StringBuilder(Int32)	Inicializa una nueva instancia de la clase StringBuilder con la capacidad especificada.
StringBuilder(String)	Inicializa una nueva instancia de la clase StringBuilder con la cadena especificada.
StringBuilder(Int32, Int32)	Inicializa una nueva instancia de la clase StringBuilder que empieza con una capacidad concreta y puede crecer hasta un máximo especificado.
StringBuilder(String, Int32)	Inicializa una nueva instancia de la clase StringBuilder con la cadena y la capacidad especificada.
StringBuilder(String, Int32, Int32, Int32)	Inicializa una nueva instancia de la clase StringBuilder con la cadena y la capacidad especificada.

Las propiedades son:

Propiedad	Descripción
Capacity	Obtiene o establece el número máximo de caracteres que pueden estar contenidos en la memoria asignada por la instancia actual.
Chars	Obtiene o establece el carácter en la posición en la que se encuentra la cadena dentro de la instancia.
Length	Obtiene o establece la longitud del objeto StringBuilder actual.
MaxCapacity	Obtiene la capacidad máxima de esta instancia.

Algunos de los métodos más importantes de la clase son los siguientes:

Método	Retorna	Significa	Parámetros	Argumento Recibido	Descripción
Append	StringBuilder	Una referencia a la instancia actual luego de agregar al parámetro a la cadena interna.	Boolean, Byte, Char, Char[], Decimal, Double, Int16, Int32, Int64, Object, SByte, Single, String, UInt16, UInt32 o UInt64	Uno de los tipos base definidos que se convertirá en cadena, un objeto o un vector de caracteres	Método sobrecargado que recibe un argumento que convierte a un tipo String y lo agrega al final de la cadena interna de la instancia actual
Append	StringBuilder	Una referencia a la instancia actual luego de agregar al parámetro a la cadena interna.	Char	El carácter a agregar	Agrega un número especificado de copias de la representación en formato de cadena de un carácter Unicode a la instancia actual.
			Int32	Número de veces que se agrega	
Append	StringBuilder	Una referencia a la instancia actual luego de agregar al parámetro a la cadena interna.	Char[]	Vector de caracteres a agregar	Agrega desde un vector la cantidad de caracteres indicadas según el índice de comienzo hasta el índice de finalización inclusive
			Int32	Índice del comienzo del vector	
			Int32	Índice del final del vector	
Append	StringBuilder	Una referencia a la instancia actual luego de agregar al parámetro a la cadena interna.	String	Cadena o subcadena de ésta a insertar	Agrega desde una cadena la cantidad de caracteres indicados según el índice de comienzo hasta el índice de finalización inclusive de esta.
			Int32	Índice de comienzo de la cadena recibida	
			Int32	Índice de finalización de la cadena recibida	
AppendLine	StringBuilder	Una referencia a la instancia actual	N / A	N / A	Agrega el terminador de línea predeterminado al final del objeto StringBuilder actual.
AppendLine	StringBuilder	Una referencia a la instancia actual luego de agregar al parámetro a la cadena interna.	String	La cadena que se va a agregar	Agrega la cadena de argumento y luego el terminador de línea predeterminado al final del objeto StringBuilder actual.

Método	Retorna	Significa	Parámetros	Argumento Recibido	Descripción
Clear	StringBuilder	Una referencia a la instancia actual luego de agregar al parámetro a la cadena interna.	N / A	N / A	Clear es un método útil que es equivalente a establecer la propiedad Length = 0 en la instancia actual. Al llamar al método Clear no se modifica las propiedades Capacity ni MaxCapacity.
CopyTo	N / A	N / A	Int32	El índice a partir de donde se copiarán los datos de origen	Copia los caracteres de un segmento específico de esta instancia en una posición determinada de un vector destino del tipo Char.
			Char[]	El vector destino de los datos	
			Int32	El índice a partir de donde se copiarán los datos en el destino	
			Int32	Cantidad de caracteres a copiar	
Equals	Boolean	Retorna verdadero si ambas instancias son iguales	StringBuilder	El objeto del tipo StringBuilder con el que comparará esta instancia	Devuelve un valor que indica si esta instancia equivale a un objeto del tipo StringBuilder
Insert	StringBuilder	Una referencia a la instancia actual luego de agregar al parámetro a la cadena interna.	Int32	Posición en donde se insertará el elemento	Inserta en la cadena interna a partir del índice del primer argumento el tipo recibido como segundo argumento. Luego de la inserción se copian los caracteres que existían anteriormente a partir del punto de inserción
			Boolean, Byte, Char, Char[], Decimal, Double, Int16, Int32, Int64, Object, SByte, Single, String, UInt16, UInt32 o UInt64	Uno de los tipos base definidos que se convertirá en cadena, un objeto o un vector de caracteres	
Insert	StringBuilder	Una referencia a la instancia actual luego de agregar al parámetro a la cadena interna.	Int32	Posición en donde se insertará el elemento	Inserta una o más copias de una cadena especificada en la instancia actual, en la posición del carácter especificado.
			String	La cadena a insertar	
			Int32	Cantidad de veces a realizar la inserción	
Insert	StringBuilder	Una referencia a la instancia	Int32	Posición en donde se insertará el	Inserta en la cadena interna a partir del índice del

Método	Retorna	Significa	Parámetros	Argumento Recibido	Descripción
		actual luego de agregar al parámetro a la cadena interna.		elemento	primer argumento la cantidad de caracteres especificado en el último argumento desde la posición inicial indicada en el tercer argumentos desde el vector pasado como segundo argumento
			Char[]	El vector origen de datos	
			Int32	Posición inicial en el vector de origen de datos	
			Int32	Cantidad de caracteres a copiar	
Remove	StringBuilder	Una referencia a la instancia actual luego de agregar al parámetro a la cadena interna.	Int32	Índice a partir del cual borrar	Borra una cantidad de caracteres igual al rango especificado en los argumentos
			Int32	Cantidad de caracteres a borrar	
Replace	StringBuilder	Una referencia a la instancia actual luego de agregar al parámetro a la cadena interna.	Char	Carácter a reemplazar	Reemplaza todas las apariciones de un carácter especificado en la instancia por el carácter del segundo argumento.
			Char	Nuevo carácter	
Replace	StringBuilder	Una referencia a la instancia actual luego de agregar al parámetro a la cadena interna.	String	Cadena a reemplazar	Reemplaza todas las apariciones de una cadena especificado en la instancia por la cadena del segundo argumento.
			String	Nueva cadena	
Replace	StringBuilder	Una referencia a la instancia actual luego de agregar al parámetro a la cadena interna.	Char	Carácter a reemplazar	Reemplaza todas las apariciones de un carácter especificado en la instancia por el carácter del segundo argumento, dentro del rango que indican el tercer y cuarto argumento
			Char	Nuevo carácter	
			Int32	Índice a partir del cual se empieza a reemplazar	
			Int32	Índice en donde se termina de reemplazar	
Replace	StringBuilder	Una referencia a la instancia actual luego de	String	Cadena a reemplazar	Reemplaza todas las apariciones de una cadena especificado en la instancia
			String	Nueva cadena	

Método	Retorna	Significa	Parámetros	Argumento Recibido	Descripción
		agregar al parámetro a la cadena interna.	Int32	Índice a partir del cual se empieza a reemplazar	por la cadena del segundo argumento, dentro del rango que indican el tercer y cuarto argumento
			Int32	Índice en donde se termina de reemplazar	
ToString	String	Una cadena con el contenido de la cadena interna	N / A	N / A	Retorna la cadena interna en formato String
ToString	String	Una cadena con el contenido de la cadena interna	Int32	Índice a partir del cual se crea la subcadena a retornar	Retorna una subcadena a a partir de la cadena interna en formato String en el rango especificado en los argumentos
			Int32	Índice en donde se termina de crear la subcadena	

Ejemplo

C#

```
namespace mutables
{
    class Program
    {
        static void Main(string[] args)
        {
            bool bandera = false;
            System.Text.StringBuilder sb = new System.Text.StringBuilder();
            sb.Append("El valor booleano es ").Append(bandera).Append(".");
            Console.WriteLine(sb.ToString());
            Console.WriteLine("-------------------------------------");
            decimal valor = 1346.19m;
            sb.Append('*', 5).Append(valor).Append('*', 5);
            Console.WriteLine(sb);
            Console.WriteLine("-------------------------------------");
            sb.AppendLine();
            int indiceAux = sb.Length - 2;
            string str = "El tercer planeta;Tierra;Venus;Marte";
            int indice = 0;
            int largo = str.IndexOf(';', indice);
            sb.Append(str, indice, largo).Append(", la ");
            indice += largo + 1;
            largo = str.IndexOf(';', indice) - indice;
            sb.Append(str, indice, largo).Append(", está entre ");
            indice += largo + 1;
            largo = str.IndexOf(';', indice) - indice;
            sb.Append(str, indice, largo).Append(" y ");
            indice += largo + 1;
            sb.Append(str, indice, str.Length - indice);
```

```
            Console.WriteLine(sb);
            Console.WriteLine("-------------------------------------");
            sb.Insert(indiceAux, "\nEn el sistema solar: ");
            Console.WriteLine(sb);
            Console.WriteLine("-------------------------------------");
            sb.Replace('*', '#');
            Console.WriteLine(sb);

            Console.ReadKey();
        }
    }
}
```

VB

```
Module Module1

    Sub Main()
        Dim bandera As Boolean = False
        Dim sb As New System.Text.StringBuilder
        sb.Append("El valor booleano es ").Append(bandera).Append(".")
        Console.WriteLine(sb.ToString())
        Console.WriteLine("-----------------------------------")
        Dim valor As Decimal = 1346.19D
        sb.Append("*"c, 5).Append(valor).Append("*"c, 5)
        Console.WriteLine(sb)
        Console.WriteLine("-----------------------------------")
        sb.AppendLine()
        Dim indiceAux As Integer = sb.Length - 2
        Dim s1 As String = "El tercer planeta;Tierra;Venus;Marte"
        Dim indice As Integer = 0
        Dim largo As Integer = s1.IndexOf(";"c, indice)
        sb.Append(s1, indice, largo).Append(", la ")
        indice += largo + 1
        largo = s1.IndexOf(";"c, indice) - indice
        sb.Append(s1, indice, largo).Append(", está entre ")
        indice += largo + 1
        largo = s1.IndexOf(";"c, indice) - indice
        sb.Append(s1, indice, largo).Append(" y ")
        indice += largo + 1
        sb.Append(s1, indice, s1.Length - indice)
        Console.WriteLine(sb)
        Console.WriteLine("-------------------------------------")
        sb.Insert(indiceAux, vbNewLine + "En el sistema solar: ")
        Console.WriteLine(sb)
        Console.WriteLine("-------------------------------------")
        sb.Replace("*", "#")
        Console.WriteLine(sb)
        Console.ReadKey()
    End Sub
End Module
```


Formatos numéricos

Hasta el momento se han utilizado salidas en cadenas con formato sin especificar como hacerlo. System.Console.WriteLine y System.Console.Write, por ejemplo, pueden definir parámetros en la

cadena de salida que recibirán como argumento. Los parámetros de definen dentro de la cadena entre llaves y numerándolos a partir del 0. El método asigna el resto de los argumentos que recibe en la posición indicada empezando desde el segundo argumento recibido hasta el último. Si se encuentra la definición de un parámetro pero no hay un argumento que se pueda asignar, el método WriteLine ignora la salida especificada sacando la cadena sin adjudicar un argumento en dicho lugar.

Ejemplo

C#

```
Console.WriteLine("El resultado de comparar {0} con {1} es {2}", s1, s2,
      s1.CompareTo(s2));
```

VB

```
Console.WriteLine("El resultado de comparar {0} con {1} es {2}", _
      s1, s2, s1.CompareTo(s2))
```

Además, las cadenas pueden tener un formato que se defina para los parámetros. Se puede dar formato, por ejemplo, a los resultados numéricos mediante el uso del método String.Format, o a través de los métodos System.Console.WriteLine y System.Console.Write, que luego llaman a String.Format. El formato se especifica mediante el uso de cadenas de formato. La siguiente tabla contiene las cadenas de formato estándar compatibles. La cadena de formato tiene la siguiente forma:

- A:xx:
  - A es el especificador de formato
  - xx es el especificador de precisión.

El especificador de formato controla el tipo de formato aplicado al valor numérico, y el especificador de precisión controla el número de dígitos significativos o posiciones decimales de la salida formateada.

La siguiente tabla muestra formatos y los ejemplifica:

Caracter	Descripción	Ejemplos	Salida
C or c	Monetario	Console.Write("{0:C}", 2.5)	$2.50
		Console.Write("{0:C}", -2.5)	($2.50)
D or d	Decimal	Console.Write("{0:D5}", 25)	25
E or e	Científico	Console.Write("{0:E}", 250000)	2,50E+11
F or f	Punto fijado	Console.Write("{0:F2}", 25)	25.00
		Console.Write("{0:F0}", 25)	25
G or g	General	Console.Write("{0:G}", 2.5)	2.5
N or n	Numérico	Console.Write("{0:N}", 2500000)	2,500,000.00
X or x	Hexadecimal	Console.Write("{0:X}", 250)	FA

Caracter	Descripción	Ejemplos	Salida
		Console.Write("{0:X}", 0xffff)	FFFF

Introducción a la globalización

Cuando se desarrolla una aplicación para que se distribuya globalmente, se debe proporcionar los medios para que se adapten a los símbolos y necesidades de diversas lenguas y culturas. Existen varios términos que se deben entender y diferenciar cuando se trabaja con una aplicación multicultural.

Globalización

Cuando se diseña una aplicación que funciona para múltiples culturas, este proceso se llama globalización. Una aplicación verdaderamente global está disponible para los usuarios de todo el mundo. Es posible que aparezca el término globalización abreviada de la escritura técnica como "g11n", donde 11 es el número de caracteres entre la "g" y la "n" de globalización. La globalización a menudo abarca los dos procesos: internacionalización y localización. Sin embargo, una aplicación globalizada no necesariamente contiene recursos localizados.

Internacionalización

Cuando se diseña una aplicación a escala mundial, un objetivo clave es crear software que puede adaptarse fácilmente a diferentes idiomas y culturas sin cambios en la estructura del código de la misma. Este proceso de creación de una aplicación que maneja los bloques de código de los de manera que estos sean independientes de la cultura, sobre todo en los bloques de la interfaz de usuario, se llama internacionalización (i18n).

Localización

El adaptar una aplicación a una lengua o cultura, se llama proceso de localización (l10n). El proceso de localización requiere traducir los recursos que utiliza la aplicación en versiones localizadas de cada cultura que admite la aplicación.

Esto incluye traducciones para los elementos de interfaz de usuario y otros datos que utiliza la aplicación. La capacidad de localización de una aplicación determina si la misma está lista para prestar servicios con las necesidades que implican ese lugar en particular. Cuando se prepara una aplicación para localización, se asegura que el idioma de los contenidos relacionados con la cultura y es independiente del código fuente.

Cultura

Una cultura es una combinación de un código de lenguaje y país, que identifica la combinación del lenguaje y país específicos. Las culturas pueden tener diferentes formatos para elementos como cadenas, calendarios, fechas, horas y tamaños de letras. Como culturas de ejemplo se incluyen en-GB para Inglés-Reino Unido y en-US para Inglés-Estados Unidos. Los dos primeros caracteres en minúsculas identificar el idioma, y los dos últimos caracteres en mayúsculas representar al país.

Algunas culturas contienen un tercer segmento que identifica una secuencia de comandos específicos de dicha cultura, por ejemplo, uz-UZ-Cyrl representa entre Uzbek-Uzbekistán-Cirílico. Las culturas disponibles se definen en un estándar de codificación geográfica a la que se denomina ISO 3166. La Organización Internacional de Normalización (ISO-International Organization for Standardization) publica el mencionado estándar.

El espacio de nombres System.Globalization

Se deben utilizar las clases del espacio de nombres System.Globalization para desarrollar una aplicación globalizada. En la siguiente tabla se muestran varias de las clases clave del espacio de nombres System.Globalization.

Clases	Descripción
Calendar	Representa divisiones de tiempo, como semanas, meses y años.
CompareInfo	Implementa un conjunto de métodos para las comparaciones de cadenas sensibles a la cultura.
CultureAndRegionInfoBuilder	Define una referencia cultural personalizada. Para utilizar esta clase, se debe agregar una referencia al ensamblado sysglobl.dll, que se puede encontrar en el directorio C:\WINDOWS\Microsoft.NET\Framework\vX.Y.ZZZZZ (las letras x, y, z representan la versión del Framework de .Net instalada).
CultureInfo	Proporciona información sobre una cultura específica.
DateTimeFormatInfo	Define el formato y la presentación de los valores DateTime.
NumberFormatInfo	Define el formato y la visualización de valores numéricos.
RegionInfo	Proporciona información sobre el país o la región.
StringInfo	Proporciona funciones para dividir una cadena en elementos de texto y recorrerlos en iteración.
TextInfo	Define las propiedades y comportamientos tales como los casos que son específicos de un sistema de escritura.

Acceso a la Información de la Cultura

Cuando se tiene que acceder a la información de referencia cultural, hay dos clases que son las principales del espacio de nombres System.Globalization que se pueden utilizar: CultureInfo y RegionInfo.

La clase CultureInfo

Se usa la clase CultureInfo para convertir valores a formatos específicos de la cultura. La clase CultureInfo proporciona información acerca de las culturas, incluyendo el formato del calendario, fecha y tiempo, reglas de ordenamiento, formatos numéricos, formatos de moneda y sistema de escritura.

Los constructores de la clase son:

Constructor	Descripción

Constructor	Descripción
CultureInfo(Int32)	Inicializa una nueva instancia de la clase CultureInfo de acuerdo con la referencia cultural especificada por el id de la cultura.
CultureInfo(String)	Inicializa una nueva instancia de la clase CultureInfo de acuerdo con la referencia cultural especificada por el nombre.
CultureInfo(Int32, Boolean)	Inicializa una nueva instancia de la clase CultureInfo de acuerdo con la referencia cultural especificada por el identificador de la cultura y el valor booleano que especifica si se utiliza la configuración de referencia cultural seleccionada por el usuario actual del sistema.
CultureInfo(String, Boolean)	Inicializa una nueva instancia de la clase CultureInfo de acuerdo con la referencia cultural especificada por el nombre y el valor booleano que indica si se utiliza la configuración de referencia cultural seleccionada por el usuario actual del sistema.

El siguiente ejemplo de código inicializa una nueva instancia de la clase CultureInfo de acuerdo con la referencia cultural especificada. El valor booleano especifica si se utiliza la configuración de referencia cultural seleccionada por el usuario del sistema si la cultura que se presenta como el primer parámetro coincide con la referencia cultural del sistema actual.

Ejemplo

C#

```
CultureInfo cultureArgentina = new CultureInfo("es-AR", false);
```

VB

```
Dim cultureArgentina As New CultureInfo("es-AR", False)
```

La clase CultureInfo proporciona propiedades y métodos que se pueden utilizar para dar formato a la información de presentación de la cultura específica.

Las propiedades de la clase son:

Propiedad	Descripción
Calendar	Obtiene el calendario predeterminado utilizado por la cultura.
CompareInfo	Obtiene el CompareInfo que define cómo comparar cadenas para la cultura.
CultureTypes	Obtiene los tipos de cultura que pertenecen al objeto CultureInfo actual.
CurrentCulture	Obtiene el objeto CultureInfo que representa la referencia cultural utilizada por el subproceso (thread) actual.

Propiedad	Descripción
CurrentUICulture	Obtiene el objeto CultureInfo que representa la cultura actual de la interfaz de usuario que utiliza el Administrador de recursos para buscar recursos específicos de la referencia cultural en tiempo de ejecución.
DateTimeFormat	Obtiene o establece un DateTimeFormatInfo que define el formato culturalmente apropiado para mostrar fechas y horas.
DefaultThreadCurrentCulture	Obtiene o establece la referencia cultural predeterminada para los threads en el dominio de la aplicación actual.
DefaultThreadCurrentUICulture	Obtiene o establece la referencia cultural de interfaz de usuario por defecto para los threads en el dominio de la aplicación actual.
DisplayName	Obtiene el nombre completo de cultura localizada.
EnglishName	Obtiene el nombre de referencia cultural en el formato languagefull [country/regionfull] en Inglés.
InstalledUICulture	Obtiene el CultureInfo que representa la referencia cultural instalada con el sistema operativo.
InvariantCulture	Obtiene el objeto CultureInfo que es independiente de la cultura (invariable).
IsNeutralCulture	Obtiene un valor que indica si el CultureInfo actual representa una referencia cultural neutra.
IsReadOnly	Obtiene un valor que indica si el CultureInfo actual es de sólo lectura.
KeyboardLayoutId	Obtiene el identificador de configuración regional de entrada activa.
LCID	Obtiene el identificador de referencia cultural para el CultureInfo actual.
Name	Obtiene el nombre de referencia cultural en el formato languagecode2-country/regioncode2.
NativeName	Obtiene el nombre de la cultura, que consiste en el idioma, el país / región y la escritura opcional, que la cultura está configurado para mostrar.
NumberFormat	Obtiene o establece un NumberFormatInfo que define el formato culturalmente apropiado de mostrar números, moneda y porcentaje.
OptionalCalendars	Obtiene la lista de calendarios que pueden ser utilizados por la cultura.
Parent	Obtiene el CultureInfo que representa la referencia cultural principal del CultureInfo actual.

Propiedad	Descripción
TextInfo	Obtiene el TextInfo que define el sistema de escritura asociado con la cultura.
ThreeLetterISOLanguageName	Obtiene el código de tres letras ISO 639-2 para el idioma del CultureInfo actual.
ThreeLetterWindowsLanguageName	Obtiene el código de tres letras para el idioma definido en la API de Windows.
TwoLetterISOLanguageName	Obtiene el código de dos letras ISO 639-1 para el idioma del CultureInfo actual..
UseUserOverride	Obtiene un valor que indica si el CultureInfo actual utiliza la configuración regional seleccionados por el usuario.

Esta clase proporciona además acceso a instancias específicas de DateTimeFormatInfo, NumberFormatInfo, CompareInfo y TextInfo respecto de la referencia cultural. Estos objetos contienen la información necesaria para las operaciones específicas de la referencia cultural, como la distinción entre mayúsculas y minúsculas, la aplicación de formato a fechas y números y la comparación de cadenas.

La clase String utiliza de forma indirecta esta clase para obtener información acerca de la referencia cultural predeterminada.

En la siguiente tabla se describen varios de estos miembros.

Método	Retorna	Significa	Parámetros	Argumento Recibido	Descripción
ClearCachedData	N / A	N / A	N / A	N / A	Actualiza la información en caché relacionados con la cultura.
Clone	Object	Una copia del CultureInfo actual.	N / A	N / A	Crea una copia del CultureInfo actual.
CreateSpecificCulture	CultureInfo	Un objeto CultureInfo que, según el valor, representa: • Una cadena vacía (""): La referencia cultural invariable. • Una referencia cultural neutra: Una cultura específica asociado con el nombre. • Una cultura específica: La cultura se especifica por su nombre.	String	Un nombre de CultureInfo predefinido o el nombre de un objeto CultureInfo existente. No distingue entre mayúsculas y minúsculas.	Método estático. Crea un CultureInfo que representa la cultura específica que está asociada con el nombre especificado.

Método	Retorna	Significa	Parámetros	Argumento Recibido	Descripción
GetConsoleFallbackUICulture	CultureInfo	Una cultura alternativa que se utiliza para leer y mostrar texto en la consola.	N / A	N / A	Obtiene una cultura para interfaz de usuario alternativa adecuada para aplicaciones de consola cuando la cultura por defecto de la interfaz gráfica de usuario es inadecuada.
GetCultureInfo	CultureInfo	Un objeto CultureInfo de sólo lectura.	Int32	Un identificador de configuración regional (LCID)	Recupera una instancia de una cultura en caché de sólo lectura mediante el identificador de referencia de cultura especificado.
GetCultureInfo	CultureInfo	Un objeto CultureInfo de sólo lectura.	String	El nombre de una cultura. El argumento no distingue entre mayúsculas y minúsculas.	Recupera una instancia de una cultura en caché de sólo lectura mediante el nombre de la cultura especificado
GetCultureInfo	CultureInfo	Un objeto CultureInfo de sólo lectura.	String	El nombre de una cultura. El argumento no distingue entre mayúsculas y minúsculas.	Método estático. Recupera una instancia de una cultura en caché de sólo lectura. Los parámetros especifican una cultura que se inicializa con los objetos TextInfo y CompareInfo especificados por otra cultura.
			String	El nombre de una cultura que suministra los objetos TextInfo y CompareInfo utilizados para inicializar el primer argumento. El segundo argumento no distingue entre mayúsculas y minúsculas.	
GetCultures	CultureInfo[]	Un vector que contiene las culturas especificadas por el parámetro de tipo. El conjunto de culturas está sin ordenar.	CultureTypes	Una combinación bit a bit de los valores de la enumeración que filtran las culturas a recuperar.	Método estático. Obtiene la lista de referencias culturales admitidas filtradas por el parámetro CultureTypes especificado.

Método	Retorna	Significa	Parámetros	Argumento Recibido	Descripción
GetFormat	Object	Retorna, si el argumento recibido fue: • Un objeto Type de la clase NumberFormatInfo: El valor de la propiedad NumberFormat, que es un NumberFormatInfo que contiene la información predeterminada del formato numérico para el CultureInfo actual. • Un objeto Type de la clase DateTimeFormatInfo: El valor de la propiedad DateTimeFormat, que es un DateTimeFormatInfo que contiene la fecha y la información por defecto el formato de hora para el CultureInfo actual. • Nulo, si es formatType cualquier otro objeto.	Type	El Type del cual obtener un objeto para formato. Este método sólo es compatible con el tipo NumberFormatInfo y DateTimeFormatInfo.	Obtiene un objeto que define cómo dar formato al tipo especificado.
ReadOnly	CultureInfo	Una envoltura de CultureInfo de sólo lectura en torno al argumento	CultureInfo	El CultureInfo para envolver.	Método estático. Devuelve un contenedor de envoltorio de sólo lectura con el CultureInfo especificado.
ToString	String	Una cadena que contiene el nombre del CultureInfo actual.	N / A	N / A	Devuelve una cadena que contiene el nombre del CultureInfo actual en el formato *languagecode2-country/regioncode2*. (Sobrescribe Object.ToString.)

Recordar que los nombres de las referencias culturales e identificadoras representan sólo un subconjunto de referencias culturales que se pueden buscar en un equipo determinado. Las versiones o los service pack de Windows pueden cambiar las referencias culturales disponibles. Las aplicaciones agregan referencias culturales personalizadas mediante la clase CultureAndRegionInfoBuilder.

La clase CultureInfo especifica un nombre único para cada referencia cultural. Una aplicación puede utilizar el método GetCultures para recuperar una lista completa de todas las referencias culturales. En el ejemplo siguiente se presenta una lista de todas las referencias culturales.

Ejemplo

C#

```
namespace culturas
{
    class Program
    {
        static void Main(string[] args)
        {
            CultureInfo cultureArgentina = new CultureInfo("es-AR", false);
            Console.WriteLine(cultureArgentina);
            Console.WriteLine(cultureArgentina.DisplayName);
            Console.ReadKey();
            Console.WriteLine("----------------------------------------");
            foreach (CultureInfo ci in
                  CultureInfo.GetCultures(CultureTypes.AllCultures))
            {
                Console.Write(ci + "\t");
            }
            Console.ReadKey();
            Console.WriteLine("\n----------------------------------------");
            CultureInfo[] culturas =
                CultureInfo.GetCultures(CultureTypes.SpecificCultures);
            foreach (CultureInfo cultura in culturas)
            {
                Console.Write(cultura.Name + "\t");
            }
            Console.ReadKey();
        }
    }
}
```

VB

```
Imports System.Globalization

Module Module1

    Sub Main()
        Dim cultureArgentina As New CultureInfo("es-AR", False)
        Console.WriteLine(cultureArgentina)
        Console.WriteLine(cultureArgentina.DisplayName)
        Console.ReadKey()
        Console.WriteLine("----------------------------------------")
        Dim ci As CultureInfo
        For Each ci In _
        CultureInfo.GetCultures(CultureTypes.AllCultures)
            Console.Write(ci.ToString + vbTab)
        Next ci
        Console.ReadKey()
        Console.WriteLine(vbNewLine + _
                          "----------------------------------------")
        Dim culturas As CultureInfo() = _
            CultureInfo.GetCultures(CultureTypes.SpecificCultures)
        For Each cultura As CultureInfo In culturas
```

```
            Console.Write(cultura.Name + vbTab)
        Next cultura
        Console.ReadKey()
    End Sub
End Module
```

La clase RegionInfo

La clase RegionInfo proporciona información acerca de las regiones geográficas, aunque no proporciona la misma funcionalidad respecto al formato que la clase CultureInfo. Para crear una nueva instancia de la clase RegionInfo, se pasa el nombre de dos letras que identifica a la región al constructor de la clase.

La clase RegionInfo proporciona varias propiedades que se pueden utilizar para recuperar información sobre una región. Estas propiedades incluyen el nombre, DisplayName, EnglishName y CurrencySymbol.

El siguiente ejemplo de código crea una instancia nueva de una clase RegionInfo utilizando las dos letras del nombre de la cultura, que se suministran en el código.

Ejemplo

C#

```
namespace region
{
    class Program
    {
        static void Main(string[] args)
        {
            RegionInfo region = new RegionInfo("AR");
            Console.WriteLine("Name: {0}", region.Name);
            Console.WriteLine("DisplayName: {0}", region.DisplayName);
            Console.WriteLine("CurrencySymbol: {0}", region.CurrencySymbol);
            Console.ReadKey();
        }
    }
}
```

VB

```
Imports System.Globalization

Module Module1

    Sub Main()
        Dim region As New RegionInfo("AR")
        Console.WriteLine("Name: {0}", region.Name)
        Console.WriteLine("DisplayName: {0}", region.DisplayName)
        Console.WriteLine("CurrencySymbol: {0}", region.CurrencySymbol)
        Console.ReadKey()
    End Sub
End Module
```

El código produce el siguiente resultado:

```
Name: AR
DisplayName: Argentina
CurrencySymbol: $
```

Formatear y Ordenar datos sensibles a la cultura

Cada cultura tiene diferentes convenciones para mostrar las fechas, hora, números, moneda, y otra información. El espacio de nombres System.Globalization contiene clases que se pueden utilizar para modificar la forma en la cual una aplicación muestra los valores específicos de la cultura.

La interfaz IFormatProvider

La interfaz sirve para proporcionar un objeto del tipo IFormatProvider brinda información acerca del formato para darlo y analizarlo en las operaciones. Las operaciones de formato convierten el valor de un tipo a la representación en cadena de dicho valor. Los métodos típicos para dar el formato son: ToString y Format. Las operaciones de análisis convierten la representación de cadena de un valor a un tipo con ese valor. Los métodos típicos de análisis son Parse y TryParse.

Nota: Los métodos del tipo Parse son referidos muchas veces como analizadores, puesto que esa es su función. Los métodos del tipo Format son referidos como formateadores por la misma causa

La interfaz IFormatProvider consiste en un método único, IFormatProvider.GetFormat. GetFormat es un método de devolución de llamada (callback): el analizador o método de formato que lo llama le pasa un objeto del tipo Type que representa el tipo de objeto acerca del cual el método de formato o analizador espera se proporcionará información de formato. El método GetFormat es responsable de retornar un objeto de ese tipo.

Las implementaciones de IFormatProvider se utilizan a menudo implícitamente por métodos que dan formato y analizan. Por ejemplo, DateTime.ToString(String) utiliza implícitamente una implementación IFormatProvider que representa la cultura actual del sistema. Las implementaciones de IFormatProvider también pueden ser invocadas explícitamente por los métodos que tienen un parámetro del tipo IFormatProvider, como Int32.Parse (String, IFormatProvider) y String.Format (IFormatProvider, String, Object []).

El Framework de .Net incluye las siguientes tres implementaciones predefinidas de IFormatProvider para proporcionar información específica de la cultura que se utiliza para dar formato o analizar los valores numéricos y de fecha y hora:

- CultureInfo: representa una cultura en particular. Su método GetFormat retorna un objeto del tipo NumberFormatInfo específico de una cultura o un objeto del tipo

DateTimeFormatInfo, dependiendo de si el objeto CultureInfo se utiliza en una operación de formato o análisis que involucra números o fechas y horas.

- NumberFormatInfo: proporciona información que se utiliza para los formatos numéricos, como el de la moneda, el separador de miles y símbolos separadores decimales para una cultura en particular.
- DateTimeFormatInfo: proporciona información que se utiliza para formatear fechas y horas, como el símbolo de separación de fecha y hora para una determinada cultura o el orden y formato de una fecha en base a sus componentes año, mes y día.

El Framework. NET también soporta la creación de formatos personalizados. Normalmente, esto implica la creación de una clase de formato que implementa IFormatProvider e ICustomFormatter. Una instancia de esta clase se pasa como parámetro a un método que realiza una operación de formato personalizado, como String.Format (IFormatProvider, String, Object []).

El siguiente ejemplo muestra cómo una implementación de IFormatProvider puede cambiar la representación de un valor de fecha y hora. En este caso, una sola fecha se muestra utilizando distintos objetos del tipo CultureInfo que representan culturas diferentes.

Ejemplo

C#

```
using System.Globalization;

namespace formatos
{
    class Program
    {
        static void Main(string[] args)
        {
            DateTime dateValue = new DateTime(2009, 6, 1, 4, 37, 0);
            CultureInfo[] cultures = { new CultureInfo("es-AR"),
                                       new CultureInfo("en-US"),
                                       new CultureInfo("fr-FR"),
                                       new CultureInfo("it-IT"),
                                       new CultureInfo("de-DE") };
            foreach (CultureInfo culture in cultures)
                Console.WriteLine("{0}: {1}",
                    culture.Name, dateValue.ToString(culture));
            Console.ReadKey();
        }
    }
}
```

VB

```
Imports System.Globalization

Module Module1

    Sub Main()
```

```
        Dim dateValue As Date = #6/1/2009 4:37:00 PM#
        Dim cultures() As CultureInfo = {New CultureInfo("es-AR"), _
                                         New CultureInfo("en-US"), _
                                         New CultureInfo("fr-FR"), _
                                         New CultureInfo("it-IT"), _
                                         New CultureInfo("de-DE")}
        For Each culture As CultureInfo In cultures
            Console.WriteLine("{0}: {1}", culture.Name, dateValue.ToString(culture))
        Next
        Console.ReadKey()
    End Sub
End Module
```

La salida del programa es:

```
es-AR: 01/06/2009 04:37:00 p.m.
en-US: 6/1/2009 4:37:00 PM
fr-FR: 01/06/2009 16:37:00
it-IT: 01/06/2009 16:37:00
de-DE: 01.06.2009 16:37:00
```

El siguiente ejemplo muestra el uso de una clase que implementa la interfaz IFormatProvider y el método GetFormat. La clase FormatoDelNumeroDeCuenta convierte un valor Int64 que representa un número de cuenta a un número de cuenta de 12 dígitos con formato de cadena. Su método GetFormat devuelve una referencia a la instancia actual FormatoDelNumeroDeCuenta si el parámetro que es el formato de un tipo refiere a una clase que implementa ICustomFormatter, de lo contrario, GetFormat retorna un valor nulo.

Ejemplo

C#

```
using System.Globalization;

namespace cuentas
{
    public class FormatoDelNumeroDeCuenta : IFormatProvider, ICustomFormatter
    {
        private const int LARGO_CUENTA = 12;

        public object GetFormat(Type formatoDelTipo)
        {
            if (formatoDelTipo == typeof(ICustomFormatter))
                return this;
            else
                return null;
        }

        public string Format(string formato, object arg,
              IFormatProvider formatProvider)
        {
            // Proveer un formato por defecto si el argumento no es un Int64.
```

```
            if (arg.GetType() != typeof(Int64))
                try
                {
                    return ManejarOtrosFormatos(formato, arg);
                }
                catch (FormatException e)
                {
                    throw new FormatException(String.Format(
                           "El formato '{0}' es inválido.", formato), e);
                }

            // Proveer un formato por defecto si es una cadena no soportada.
            string tipoF = formato.ToUpper(CultureInfo.InvariantCulture);
            if (!(tipoF == "G" || tipoF == "E"))
                try
                {
                    return ManejarOtrosFormatos(formato, arg);
                }
                catch (FormatException e)
                {
                    throw new FormatException(String.Format(
                           "El formato '{0}' es inválido.", formato), e);
                }

            // Convertir el argumento a string.
            string resultado = arg.ToString();

            // Si el número de cuenta tiene menos de 12 dígitos, completar con 0
            if (resultado.Length < LARGO_CUENTA)
                resultado = resultado.PadLeft(LARGO_CUENTA, '0');
              // Si el número de cuenta tiene menos de 12 dígitos, truncar a 12
              //caracteres.
            if (resultado.Length > LARGO_CUENTA)
                resultado = resultado.Substring(0, LARGO_CUENTA);

            if (tipoF == "E")                       // Formato sólo para enteros.
                return resultado;
            // Agregar guiones para el formato G.
            else                                                // Formato con guiones.
                return resultado.Substring(0, 5) + "-" + resultado.Substring(5, 3) +
                     "-" + resultado.Substring(8);
        }

        private string ManejarOtrosFormatos(string formato, object arg)
        {
            if (arg is IFormattable)
                return ((IFormattable)arg).ToString(
                           formato, CultureInfo.CurrentCulture);
            else if (arg != null)
                return arg.ToString();
            else
                return String.Empty;
        }
    }
```

```
namespace cuentas
{
    class Program
    {
        public enum DiasDeLaSemana { Lunes = 1,
                                     Martes = 2,
                                     Miercoles = 3,
                                     Jueves = 4,
                                     Viernes = 5
                                   };
        static void Main(string[] args)
        {
            long nroCuenta;
            double balance;
            DiasDeLaSemana diaSemana;
            string salida;

            nroCuenta = 104254567890;
            balance = 16.34;
            diaSemana = DiasDeLaSemana.Lunes;

            salida = String.Format(new FormatoDelNumeroDeCuenta(),
                                   "El {2}, el balance de la cuenta {0:G} fue {1:C2}.",
                                    nroCuenta, balance, diaSemana);
            Console.WriteLine(salida);

            diaSemana = DiasDeLaSemana.Martes;
            salida = String.Format(new FormatoDelNumeroDeCuenta(),
                                   "El {2}, el balance de la cuenta {0:E} fue {1:C2}.",
                                   nroCuenta, balance, diaSemana);
            Console.WriteLine(salida);
            Console.ReadKey();
        }
    }
}
```

VB

```
Imports System.Globalization

Public Class FormatoDelNumeroDeCuenta : Implements IFormatProvider, ICustomFormatter

    Private Const LARGO_CUENTA As Integer = 12

    Public Function GetFormat(formatoDelTipo As Type) As Object _
                    Implements IFormatProvider.GetFormat
        If formatoDelTipo Is GetType(ICustomFormatter) Then
            Return Me
        Else
            Return Nothing
        End If
    End Function

    Public Function Format(formato As String, arg As Object, _
             formatProvider As IFormatProvider) As String _
              Implements ICustomFormatter.Format
```

```
        ' Proveer un formato por defecto si el argumento no es un Int64.
        If Not TypeOf arg Is Int64 Then
            Try
                Return ManejarOtrosFormatos(formato, arg)
            Catch e As FormatException
                Throw New FormatException(String.Format( _
                    "El formato '{0}' es inválido.", formato), e)
            End Try
        End If

        ' Proveer un formato por defecto si es una cadena no soportada.
        Dim tipoF As String = formato.ToUpper(CultureInfo.InvariantCulture)
        If Not (tipoF = "G" Or tipoF = "E") Then
            Try
                Return ManejarOtrosFormatos(formato, arg)
            Catch e As FormatException
                Throw New FormatException(String.Format( _
                    "El formato '{0}' es inválido.", formato), e)
            End Try
        End If

        ' Convertir el argumento a string.
        Dim resultado As String = arg.ToString()

        ' Si el número de cuenta tiene menos de 12 dígitos, completar con 0
        If resultado.Length < LARGO_CUENTA Then resultado =  _
             resultado.PadLeft(LARGO_CUENTA, "0"c)
        ' Si el número de cuenta tiene menos de 12 dígitos, truncar a 12 caracteres.
        If resultado.Length > LARGO_CUENTA Then resultado =  _
                    Left(resultado, LARGO_CUENTA)

        If tipoF = "E" Then                         ' Formato sólo para enteros.
            Return resultado
            '  Agregar guiones para el formato G.
        Else                                        ' Formato con guiones.
            Return Left(resultado, 5) & "-" & Mid(resultado, 6, 3) & "-" _
                    & Right(resultado, 4)
        End If
    End Function

    Private Function ManejarOtrosFormatos(formato As String, arg As Object) _
       As String
        If TypeOf arg Is IFormattable Then
            Return DirectCast(arg, IFormattable).ToString( _
              formato, CultureInfo.CurrentCulture)
        ElseIf arg IsNot Nothing Then
            Return arg.ToString()
        Else
            Return String.Empty
        End If
    End Function
End Class

Imports System.Globalization
```

```
Public Enum DiasDeLaSemana As Long
    Lunes = 1
    Martes = 2
    Miercoles = 3
    Jueves = 4
    Viernes = 5
End Enum

Module Module1
    Public Sub Main()
        Dim nroCuenta As Long, balance As Double
        Dim diaSemana As DiasDeLaSemana
        Dim salida As String

        nroCuenta = 104254567890
        balance = 16.34
        diaSemana = DiasDeLaSemana.Lunes

        salida = String.Format(New FormatoDelNumeroDeCuenta(), _
             "El {2}, el balance de la cuenta {0:G} fue {1:C2}.", _
             nroCuenta, balance, diaSemana)
        Console.WriteLine(salida)

        diaSemana = DiasDeLaSemana.Martes
        salida = String.Format(New FormatoDelNumeroDeCuenta(), _
             "El {2}, el balance de la cuenta {0:E} fue {1:C2}.", _
             nroCuenta, balance, diaSemana)
        Console.WriteLine(salida)
        Console.ReadKey()
    End Sub
End Module
```


Formato de fechas para diferentes referencias culturales

Las fechas y horas no tienen convenciones universales. Un conocido ejemplo de esto es el formato de fecha corta. En los Estados Unidos (EE.UU.), la fecha 03/10/2005 se lee como 10 de marzo 2005 porque la convención en EE.UU. es mes-día-año. Sin embargo, en Argentina (AR) o el Reino Unido (UK), la fecha sería el 03 de octubre 2005 por la convención del Reino Unido es día-mes-año. Diferencias de la Convención como estas pueden derivar en problemas cuando se desarrolla una aplicación para múltiples culturas. Otra consideración para una aplicación global pueden ser las distintas zonas horarias que se deben acomodar en el diseño de la misma.

Uso de la clase DateTimeFormatInfo

La clase DateTimeFormatInfo, que está en el espacio de nombres System.Globalization, define el formato de los valores de fecha y hora para una referencia cultural específica. Se puede utilizar esta clase para establecer distinto tipo de informaciones sobre la cultura, tales como patrones de fecha, patrones de tiempo y designadores de AM o P.M. También se puede modificar los patrones que la cultura utiliza.

Para utilizar la clase DateTimeFormatInfo, se puede declarar una variable de ese tipo y asignarle el valor de la propiedad DateTimeFormatInfo que se encuentre en un objeto del tipo de la clase CultureInfo. Cuando se crea un objeto de la clase CultureInfo para representar a una cultura determinada, se puede utilizar la propiedad DateTimeFormat para devolver una instancia de la clase DateTimeFormatInfo.

Un objeto de la clase DateTimeFormatInfo contiene información acerca de cómo formatear y usar los valores de fecha y hora en varias regiones. Se puede inspeccionar esta información directamente llamando a los métodos y propiedades de esta clase. Normalmente, se utiliza la clase DateTimeFormatInfo con otros tipos, como la estructura DateTime para recuperar y manipular los valores de fecha y hora para una referencia cultural específica.

El ejemplo siguiente de código realiza las siguientes acciones:

1. Crea dos instancias de la clase DateTimeFormatInfo. Una recupera la información de la referencia cultural en-Us y la otra instancia recupera información de la referencia cultural actual de la aplicación (usando el thread actual que obtiene la información del S.O.).
2. Genera el formato ShortDatePattern para cada uno de los casos.
3. Crear una cadena que recupera la fecha actual usando el formato 1.
4. Muestra el contenido de la cadena en pantalla.
5. Crear una cadena que recupera la fecha actual usando el formato 2.
6. Muestra el contenido de la cadena en pantalla.

Ejemplo

C#

```
using System.Globalization;
using System.Threading;

namespace fechas
{
    class Program
    {
        static void Main(string[] args)
        {
            DateTimeFormatInfo formato1Fecha = new CultureInfo(
                "en-US",false).DateTimeFormat;
            DateTimeFormatInfo formato2Fecha =
            Thread.CurrentThread.CurrentCulture.DateTimeFormat;
            Console.WriteLine("Formato 1 de la fecha: {0}",
                    formato1Fecha.ShortDatePattern);
            Console.WriteLine("Formato 2 de la fecha: {0}",
                    formato2Fecha.ShortDatePattern);
            string fechaActual = DateTime.Now.ToString(formato1Fecha);
            Console.WriteLine("Formato 1 de la fecha actual: {0}", fechaActual);
            fechaActual = DateTime.Now.ToString(formato2Fecha);
            Console.WriteLine("Formato 2 de la fecha actual: {0}", fechaActual);
            Console.ReadKey();
```

```
        }
    }
}
```

VB

```
Imports System.Globalization
Imports System.Threading

Module Module1

    Sub Main()
        Dim formato1Fecha As DateTimeFormatInfo = New CultureInfo("en-US", _
                                                  False).DateTimeFormat
        Dim formato2Fecha As DateTimeFormatInfo = _
            Thread.CurrentThread.CurrentCulture.DateTimeFormat
        Console.WriteLine("Formato 1 de la fecha: {0}",
                          formato1Fecha.ShortDatePattern)
        Console.WriteLine("Formato 2 de la fecha: {0}",
                          formato2Fecha.ShortDatePattern)
        Dim fechaActual As String = DateTime.Now.ToString(formato1Fecha)
        Console.WriteLine("Formato 1 de la fecha actual: {0}", fechaActual)
        fechaActual = DateTime.Now.ToString(formato2Fecha)
        Console.WriteLine("Formato 2 de la fecha actual: {0}", fechaActual)
        Console.ReadKey()
    End Sub
End Module
```

Nota: las bases de datos definen sus métodos de ordenamiento en base a las diferentes culturas especificadas para determinar los formatos en los cuales se almacena la información. Es importante entender el concepto de los cambios de formatos para que la información recuperada sea igual a la del sistema que ejecuta una aplicación o a la cultura que esta especifique.

Formato de números para diferentes referencias culturales

La clase NumberFormatInfo define el formato y la visualización de valores numéricos basados en la cultura actual seleccionada. Se puede utilizar esta clase para mostrar valores numéricos que siguen las convenciones regionales para el usuario.

El constructor para crear instancias de esta clase es el siguiente:

Constructor	Descripción
NumberFormatInfo	Inicializa una nueva instancia modificable de la clase NumberFormatInfo que es independiente de la cultura (invariable).

La clase NumberFormatInfo proporciona muchas propiedades que se pueden utilizar para los formatos de números. Por ejemplo, la propiedad CurrencySymbol obtiene o establece el símbolo de la moneda, y la propiedad NumberDecimalDigits obtiene o establece el número de decimales a utilizar en valores numéricos.

Las propiedades de la clase son:

Propiedad	Descripción
CurrencyDecimalDigits	Obtiene o establece el número de decimales a utilizar en valores de moneda.
CurrencyDecimalSeparator	Obtiene o establece la cadena que se utiliza como separador decimal en valores de moneda.
CurrencyGroupSeparator	Obtiene o establece la cadena que separa grupos de dígitos a la izquierda de la coma decimal para valores monetarios.
CurrencyGroupSizes	Obtiene o establece el número de dígitos en cada grupo a la izquierda de la coma decimal para valores monetarios.
CurrencyNegativePattern	Obtiene o establece el modelo de formato para valores monetarios negativos.
CurrencyPositivePattern	Obtiene o establece el modelo de formato para valores monetarios positivos.
CurrencySymbol	Obtiene o establece la cadena que se utiliza como símbolo de moneda.
CurrentInfo	Obtiene un NumberFormatInfo de sólo lectura que los valores de los formatos basados en la cultura actual.
DigitSubstitution	Obtiene o establece un valor que especifica cómo la interfaz gráfica de usuario muestra el formato de un dígito.
InvariantInfo	Obtiene un objeto NumberFormatInfo de sólo lectura que es independiente de la cultura (invariable).
IsReadOnly	Obtiene un valor que indica si este objeto NumberFormatInfo es de sólo lectura.
NaNSymbol	Obtiene o establece la cadena que representa el valor NaN del IEEE (no un número).
NativeDigits	Obtiene o establece un vector de cadenas de dígitos nativos equivalentes a los dígitos occidentales del 0 al 9.
NegativeInfinitySymbol	Obtiene o establece la cadena que representa el infinito negativo.
NegativeSign	Obtiene o establece la cadena que denota que el número asociado es negativo.
NumberDecimalDigits	Obtiene o establece el número de decimales a utilizar en valores numéricos.

Propiedad	Descripción
NumberDecimalSeparator	Obtiene o establece la cadena que se utiliza como separador decimal en valores numéricos.
NumberGroupSeparator	Obtiene o establece la cadena que separa grupos de dígitos a la izquierda de la coma decimal en valores numéricos.
NumberGroupSizes	Obtiene o establece el número de dígitos en cada grupo a la izquierda de la coma decimal en valores numéricos.
NumberNegativePattern	Obtiene o establece el patrón de formato para los valores numéricos negativos.
PercentDecimalDigits	Obtiene o establece el número de decimales a utilizar en valores de porcentaje.
PercentDecimalSeparator	Obtiene o establece la cadena que se utiliza como separador decimal en valores de porcentaje.
PercentGroupSeparator	Obtiene o establece la cadena que separa grupos de dígitos a la izquierda de la coma decimal en valores de porcentaje.
PercentGroupSizes	Obtiene o establece el número de dígitos en cada grupo a la izquierda de la coma decimal en valores de porcentaje.
PercentNegativePattern	Obtiene o establece el patrón de formato para los valores porcentuales negativos.
PercentPositivePattern	Obtiene o establece el patrón de formato para los valores de porcentaje positivos.
PercentSymbol	Obtiene o establece la cadena que se utiliza como símbolo de porcentaje.
PerMilleSymbol	Obtiene o establece la cadena que se utiliza como símbolo de por mil.
PositiveInfinitySymbol	Obtiene o establece la cadena que representa el infinito positivo.
PositiveSign	Obtiene o establece la cadena que denota que el número asociado es positivo.

También se pueden especificar valores con formato con el método Parse mediante la enumeración NumberStyles. Se pueden combinar varios valores individuales de la enumeración indicarle al método Parse el tipo de entero que se quiere manejar la cadena numérica. Por ejemplo, el miembro de la enumeración AllowCurrencySymbol informa al método Parse que incluya el símbolo de la moneda.

Algunos de los métodos de la clase son:

Método	Retorna	Significa	Parámetros	Argumento Recibido	Descripción
GetFormat	Object	El NumberFormatInfo actual, si el argumento del tipo Type es del mismo tipo que la instancia actual de NumberFormatInfo, de lo contrario, null.	Type	El Type del servicio de formato requerido.	Obtiene un objeto del tipo especificado que proporciona un servicio de formato de número.
GetInstance	NumberFormatInfo	El NumberFormatInfo asociado al IFormatProvider especificado.	IFormatProvider	El IFormatProvider utilizado para obtener el NumberFormatInfo o null para obtener CurrentInfo.	Obtiene la NumberFormatInfo asociado al IFormatProvider especificado.
ReadOnly	NumberFormatInfo	Un objeto de sólo lectura del tipo NumberFormatInfo que envuelve (contiene) al argumento recibido.	NumberFormatInfo	Un objeto del tipo NumberFormatInfo a envolver	Devuelve un contenedor de sólo lectura de envoltura respecto de un NumberFormatInfo.

Nota: a diferencia del método Parse, el método TryParse no lanza una excepción si la cadena analizada no está en el formato esperado.

El ejemplo siguiente de código crea una nueva instancia de la clase NumberFormatInfo que recupera información de la referencia cultural actual de la aplicación. El código a continuación, muestra las propiedades NumberDecimalSeparator y NumberGroupSeparator de la instancia.

Ejemplo

C#

```
using System.Globalization;
using System.Threading;

namespace numeros
{
    class Program
    {
        static void Main(string[] args)
        {
        NumberFormatInfo numberFormat =
            Thread.CurrentThread.CurrentCulture.NumberFormat;
        Console.WriteLine("Separador decimal: {0}",
            numberFormat.NumberDecimalSeparator);
        Console.WriteLine("Separador de miles: {0}",
            numberFormat.NumberGroupSeparator);
```

```
        Console.ReadKey();
        }
    }
}
```

VB

```
Imports System.Globalization
Imports System.Threading

Module Module1

    Sub Main()
        Dim numberFormat As NumberFormatInfo = _
        Thread.CurrentThread.CurrentCulture.NumberFormat
        Console.WriteLine("Separador decimal: {0}", _
                          numberFormat.NumberDecimalSeparator)
        Console.WriteLine("Separador de miles: {0}", _
                          numberFormat.NumberGroupSeparator)
        Console.ReadKey()
    End Sub
End Module
```

Nota: el espacio de nombres System.Threading es el encargado de manejar subprocesos. Ester tema se abordará más adelante.

Realización de ordenamientos y comparaciones de texto teniendo en cuenta la cultura

Las aplicaciones que tienen en cuenta la cultura pueden generar resultados diferentes para las comparaciones de cadenas cuando la cultura cambia. Por ejemplo, los símbolos, mayúsculas o minúsculas y la fonética pueden producir resultados diferentes de comparación en distintas culturas.

Se utiliza el método String.Compare para comparar las cadenas. Los resultados del método String.Compare son como los siguentes:

- Un número entero negativo si cadena1 es menor que cadena2.
- Cero si cadena1 y cadena2 son iguales.
- Un número entero positivo si cadena1 es mayor que cadena2.

Se puede utilizar tanto el método String.Compare como la clase CompareInfo para realizar comparaciones de cadenas sensibles a la cultura. La propiedad CultureInfo.CompareInfo es una instancia de la clase CompareInfo. Por defecto, el método String.Compare obtiene el objeto CompareInfo del subproceso actual. Se puede recuperar el mismo objeto CompareInfo actual a través de la propiedad Thread.CurrentThread.CurrentCulture.CompareInfo.

Las propiedades de la clase CompareInfo son:

Propiedad	Descripción
LCID	Obtiene el identificador de referencia cultural correctamente formado para el CompareInfo actual.
Name	Obtiene el nombre de la referencia cultural utilizada por este objeto CompareInfo para las operaciones de ordenación.
Version	Obtiene información sobre la versión Unicode utilizado para comparar y ordenar cadenas.

Cuando se llama al método Compare de la clase CompareInfo, se utiliza la enumeración CompareOptions para determinar cómo el método Compare gestiona la comparación de cadenas.

En la siguiente tabla se describen los principales miembros de la enumeración CompareOptions.

Nombre de miembro	Descripción
None	Indica la configuración predeterminada de las opciones para la comparación de cadenas.
IgnoreCase	Indica que en la comparación de cadenas no se debe tener en cuenta la distinción entre mayúsculas y minúsculas.
IgnoreNonSpace	Indica que en las comparaciones de cadenas no deben tenerse en cuenta los caracteres combinables sin espaciado, como los diacríticos.El estándar define los caracteres de combinación como caracteres que se combinan con caracteres base para generar un nuevo carácter.Los caracteres combinables sin espaciado no ocupan por sí mismos un espacio cuando se representan.
IgnoreSymbols	Indica que en la comparación de cadenas no se deben tener en cuenta los símbolos, como los caracteres de espacio en blanco, la puntuación, los símbolos de divisa, el signo de porcentaje, los símbolos matemáticos, la Y comercial (&), etc.
IgnoreKanaType	Indica que en la comparación de cadenas no se debe tener en cuenta el tipo Kana.El tipo Kana hace referencia a los caracteres japoneses hiragana y katakana, que representan sonidos fonéticos del idioma japonés.Los caracteres hiragana se utilizan en expresiones y palabras propias del idioma japonés, mientras que los caracteres katakana se utilizan para préstamos léxicos, como "Internet".Un sonido fonético puede expresarse tanto en caracteres hiragana como katakana.Si se selecciona este valor, el carácter hiragana de un sonido se considera equivalente al carácter katakana del mismo sonido.
IgnoreWidth	Indica que en la comparación de cadenas no se debe tener en cuenta el ancho de los caracteres.Por ejemplo, los caracteres katakana japoneses se pueden escribir como ancho completo o medio ancho.Si se selecciona este valor, los caracteres katakana escritos como ancho completo se consideran iguales que los mismos caracteres escritos como medio ancho.

Nombre de miembro	Descripción
OrdinalIgnoreCase	La comparación de cadenas debe omitir la distinción entre mayúsculas y minúsculas y, a continuación, realizar una comparación de ordinales.Esta técnica es equivalente a poner la cadena en mayúsculas utilizando la referencia cultural de todos los idiomas y realizar después una comparación de ordinales en el resultado.
StringSort	Indica que la comparación de cadenas debe usar el algoritmo de ordenación por cadena.En una ordenación por cadena, el guión y el apóstrofo, así como otros símbolos no alfanuméricos, van delante de los caracteres alfanuméricos.
Ordinal	Indica que la comparación de cadenas debe usar valores sucesivos de la cadena con codificación Unicode UTF-16 (comparación de unidad de código con unidad de código), lo que tiene como resultado una comparación rápida pero que no reconoce la referencia cultural.Una cadena que empiece con una unidad de código XXXX16 va antes que una cadena que empieza por YYYY16, si XXXX16 es menor que YYYY16.Este valor no se puede combinar con otros valores de CompareOptions y se debe usar solo.

Los juegos de caracteres incluyen caracteres a ignorar, que son aquellos que no se consideran al realizar una ordenación lingüística o correspondiente a una cultura. En una búsqueda dependiente de la referencia cultural, si el argumento que indica el valor a buscar contiene un carácter a ignorar, el resultado es equivalente a buscar sin dicho carácter. Si el mencionado argumento solo se compone de uno o más caracteres a ignorar, los método que realizan búsquedas o comparaciones, como IndexOf o Compare siempre devuelve 0 (cero) para indicar que la coincidencia se encuentra al principio de argumento en el cual se busca.

En la tabla siguiente se describen algunos de los principales métodos de la clase CompareInfo.

Método	Retorna	Descripción
Compare	Int32	Compara dos cadenas. Son 6 métodos sobrecargados que permiten indicar desde y hasta dónde se compara cada cadena o si se quiere usar la enumeración CompareOptions
GetCompareInfo	CompareInfo	Inicializa un nuevo objeto CompareInfo que está asociado a la referencia cultural con el identificador o la cadena especificada. Son dos métodos sobrecargados

Método	Retorna	Descripción
IndexOf	Int32	Busca el carácter o la cadena especificada y retorna el índice de la primera aparición en la cadena donde se hace la búsqueda. Son 12 métodos sobrecargados que permiten indicar desde y hasta dónde se compara cada cadena o el caracter usado o si se quiere usar la enumeración CompareOptions para buscar tanto una cadena como un caracter
IsPrefix	Boolean	Determina si la cadena en la que se busca comienza con el prefijo especificado como argumento. Son 2 métodos sobrecargados y uno admite la enumeración CompareOptions
IsSortable	Boolean	Indica si se puede ordenar por un carácter Unicode o cadena especificada
IsSuffix	Boolean	Determina si la cadena en la que se busca termina con el sufijo especificado como argumento. Son 2 métodos sobrecargados y uno admite la enumeración CompareOptions
LastIndexOf	Int32	Busca el carácter o la cadena especificada y retorna el índice de la última aparición en la cadena donde se hace la búsqueda. Son 12 métodos sobrecargados que permiten indicar desde y hasta dónde se compara cada cadena o el caracter usado o si se quiere usar la enumeración CompareOptions para buscar tanto una cadena como un caracter

Expresiones regulares

Además de la manipulación de cadenas, haciéndolas más largas, seccionándolas y sustituyendo caracteres, a veces también es necesario procesar la cadena y la información que contiene. Por ejemplo, se puede escribir parte del proceso de autenticación para una aplicación y se desea regular el formato de las contraseñas de los usuarios. En este escenario, se debe decidir si la cadena contiene caracteres especiales o si el usuario debe incluir dígitos de la contraseña.

Otro ejemplo es que se puede tener que manejar la salida de sistemas informáticos heredados en un formato que es incompatible con los sistemas actuales que utiliza. Si se manejan individualmente todos estos requisitos puede ser muy complejo de implementar y requiere analizar en base a cada carácter específico de estar en un lugar específico con un valor particular. Codificar este tipo de funcionalidad puede ser extremadamente laborioso.

Afortunadamente, desde hace décadas, los desarrolladores de Unix y Perl han utilizado una técnica compleja pero muy eficiente para el procesamiento de texto bajo condiciones de este estilo: las expresiones regulares

Una expresión regular (comúnmente llamada una regex o regexp) es una secuencia de caracteres, conocida como cadena de patrón, que el Framework de .NET maneja de una manera específica. Se puede utilizar para varios propósitos, tales como buscar la coincidencia de patrones, la verificación de formato, y las comparaciones para asegurarse de que una cadena cumple varios criterios sin manipularlas directamente. También se puede utilizar expresiones regulares para modificar las cadenas reemplazando o eliminando datos y caracteres individuales, formateándolas o extrayendo datos.

Un punto muy importante acerca de las expresiones regulares es que trabajan sin tomar en cuenta el contexto de los datos con los que interactúan, por ejemplo, se puede escribir una expresión regular para que coincida con el formato de una fecha o una hora, pero la expresión no tiene contexto para los mismos. Por lo tanto, si se desea utilizar una expresión regular para dar formato a un horario, y los datos de entrada se encuentran en formato de reloj de 12 horas, el motor de expresiones regulares que no tienen el concepto que un horario de 12:34 es AM o P.M. (00:34 o 12:34), sino que se debe implementar la funcionalidad para manejar esto si es necesario.

Una expresión regular hace coincidir el texto de entrada con el patrón que se define por la secuencia de caracteres que componen la expresión. La sintaxis de la expresión regular ha existido y evolucionado a lo largo de varias décadas, y sólo aquellos que implementan las expresiones regulares con frecuencia requieren un conocimiento completo del funcionamiento y toda la gama de caracteres que son necesarios para llevarlo a cabo.

El motor de evaluación de expresiones regulares lee la cadena de entrada caracter por caracter e iguala cada carácter a la vez para asegurar que el patrón de la “cadena patrón” coincide con la cadena de entrada. Las igualaciones se realizan cuando el motor alcanza el final de un subpatrón o grupo o al final de la expresión regular (es decir, de esta manera encuentra una coincidencia en un componente de la cadena patrón que se le pasa). Si se solicita todas las ocurrencias del patrón en la cadena de entrada, el motor buscará de forma iterativa todas las ocurrencias en la cadena de entrada hasta que no haya más caracteres coincidentes.

El Framework de .NET proporciona varias clases en el espacio de nombres System.Text.RegularExpressions para que se pueda utilizar expresiones regulares y su funcionalidad relacionada.

La siguiente tabla describe las clases que System.Text.RegularExpressions

Nombre de Clase	Descripción

Nombre de Clase	Descripción
Capture	Retorna una instancia única para cada igualdad de subcadena encontrada que devuelve una expresión regular. La clase Capture se retorna a través de la colección Captures de las clases Match o Group.
CaptureCollection	Representa una colección secuencial de objetos del tipo Capture que devuelve la evaluación de una expresión regular.
Group	Representa los resultados de un grupo de Capture. Se agregan grupos de Capture a expresiones regulares para que se pueda recuperar secciones específicas de los datos de entrada.
GroupCollection	Representa una colección secuencial de objetos de grupo que devuelve una sola coincidencia.
Match	Representa los resultados de una coincidencia única durante una evaluación de la expresión regular. El objeto Match contiene un objeto GroupCollection para exponer todos los resultados de la misma coincidencia. El objeto también contiene un objeto del tipo CaptureCollection para exponer todos los resultados de captura para la coincidencia.
MatchCollection	Representa una colección de coincidencias éxitosas después de iterar a través de la cadena de entrada mediante la expresión regular. Cada vez que se corresponde con la expresión, un objeto del tipo Match se añade a la colección.
Regex	Representa una expresión regular inmutable. Esta clase contiene la funcionalidad principal de una expresión regular. La clase incluye métodos estáticos que permiten utilizar la funcionalidad de expresiones regulares sin definir explícitamente la expresión regular.
RegexCompilationInfo	Proporciona información sobre una expresión regular que se compila en un ensamblado independiente. Las propiedades de la clase definen el nombre de la clase, el nombre completo y el patrón de la expresión con las opciones adicionales para que el ensamblado pueda ejecutar correctamente la expresión.

La enumeración RegexOptions

El espacio de nombres System.Text.RegularExpressions contiene la enumeración RegexOptions, que permite varias opciones que modifican la forma en la cual una expresión regular coincide con su patrón respecto de la cadena de entrada. En la tabla siguiente se muestran los valores de la enumeración y describe la funcionalidad que cada opción activa o desactiva.

Nombre de miembro	Descripción
None	Especifica que no hay opciones establecidas.
IgnoreCase	Especifica la coincidencia sin distinción entre mayúsculas y minúsculas.

Nombre de miembro	Descripción
Multiline	Modo múltiples líneas. Cambia el significado de ^ y $ de manera que coincidan al principio y al final, respectivamente, de cada línea y no justo al principio y al final de toda la cadena.
ExplicitCapture	Especifica que sólo las capturas válidas son explícitamente grupos con nombre o número con la forma (?<nombre>…).Esto permite que los paréntesis sin nombre actúen como grupos sin capturar sin la torpeza sintáctica de la expresión (?:…).
Compiled	Especifica que la expresión regular esté compilada en un ensamblado. Esto proporciona una ejecución más veloz pero incrementa el tiempo de inicio. Este valor no debe asignarse a la propiedad Options al llamar al método CompileToAssembly.
Singleline	Especifica el modo de una sola línea. Cambia el significado del punto (.) de manera que coincida con todos los caracteres (en lugar de hacerlo con todos los caracteres excepto con \n).
IgnorePatternWhitespace	Elimina el espacio en blanco no omitido del patrón y habilita los comentarios marcados con #. Sin embargo, el valor IgnorePatternWhitespace no afecta ni elimina el espacio en blanco en las clases de caracter.
RightToLeft	Especifica que la búsqueda será de derecha a izquierda en lugar de izquierda a derecha.
ECMAScript	Habilita el comportamiento conforme a ECMAScript para esta expresión. Este valor sólo se puede utilizar junto con los valores IgnoreCase, Multiline y Compiled. El uso de este valor con otros valores dará como resultado una excepción.
CultureInvariant	Especifica que las diferencias culturales de idioma se pasan por alto.

Se puede especificar varias opciones proporcionadas por los miembros de la enumeración RegexOptions (en particular, los miembros RegexOptions.ExplicitCapture, RegexOptions.IgnoreCase, RegexOptions.Multiline y RegexOptions.Singleline) usando un caracter de opción insertado en el modelo de expresión regular.

La clase Regex

La clase System.Text.RegularExpressions.Regex es la más importante de las clases del espacio de nombres System.Text.RegularExpressions, esta clase representa la expresión regular, y es desde esta clase donde la evaluación de una expresión regular se inicializa.

Se puede crear una expresión regular creando instancias de esta clase de manera explícita y proporcionando el patrón de coincidencias como un parámetro en el constructor. La clase Regex contiene varios métodos estáticos que permiten utilizar la funcionalidad implícita sin crear un objeto Regex para su uso explícitamente.

En la siguiente tabla se describen los métodos que la clase Regex proporciona, incluyendo sus requisitos de acceso.

Nombre del método	Acceso	Descripción
Escape	Estático	Convierte una cadena de modo que se puede utilizar como una constante en una expresión regular. Los caracteres de espacio en blanco como el asterisco y los paréntesis requieren un carácter de escape para que se puedan usarl cuando la opción IgnorePatternWhitespace está habilitada.
GetGroupNames	Instancia	Retorna un vector que contiene los nombres de los grupos de captura que la expresión regular utiliza.
GetGroupNumbers	Instancia	Retorna un vector que contiene los números que corresponden a los grupos de captura de la expresión regular.
GroupNameFromNumber	Instancia	Retorna el nombre de un grupo de captura que se corresponde con el número de grupo de entrada.
GroupNumberFromName	Instancia	Retorna el número de un grupo de captura que corresponde al nombre de grupo de entrada.
IsMatch	Estático	Retorna un valor booleano que indica si la expresión regular encuentra una coincidencia correcta en la cadena de entrada. La expresión regular debe coincidir una o más veces por el método IsMatch para devolver true (C #) o True (VB). El método Match estático ejecuta la expresión regular en una cadena, y devuelve el resultado exacto de la primera ocurrencia que encuentra. Si no hay ninguna coincidencia, se devuelve un objeto del tipo Match vacío.
Matches	Estático	Ejecuta la función de expresión regular y devuelve todas las coincidencias con éxito en la cadena de entrada como si el método Match se llamara varias veces.
Replace	Estático	Reemplaza cadenas que coinciden con la expresión regular con una cadena de reemplazo específica.
Split	Estático	Divide la cadena de entrada en un vector de cadenas en las posiciones que se derivan de las posiciones de coincidencia con las expresiones regulares.
ToString	Instancia	Devuelve el patrón de expresión regular que contiene el objeto del tipo Regex. Rescribe el método Object.ToString
UnEscape	Estático	Lo contrario del método Escape; elimina los caracteres de escape de caracteres especiales que la expresión regular utiliza como constantes.

Comprendiendo las expresiones regulares

Las expresiones regulares son una serie de caracteres que forman un patrón, normalmente representativo de otro grupo de caracteres mayor, de tal forma que podemos comparar el patrón con otro conjunto de caracteres para ver las coincidencias.

Las expresiones regulares están disponibles en casi cualquier lenguaje de programación, pero aunque su sintaxis es relativamente uniforme, cada lenguaje usa su propio dialecto.

Muchos usuarios, por lo general todos los programadores, usan al menos limitadamente y sin saberlo expresiones regulares. El ejemplo típico es el uso de caracteres comodines en el SO. Por ejemplo, para solicitarle al intérprete de comandos que se listen todos los archivos en un directorio sin importar el nombre o la extensión se suele usar "*" en ambos lugares, el que determina uno u otro.

Las expresiones regulares son cadenas de caracteres ordenadas de acuerdo a un patrón que coincide con otra cadena de caracteres. Su uso está ampliamente difundido al grado de ser considerada una herramienta básica para la gestión de cadenas que está ampliamente utilizada en lenguajes de programación, sistemas operativos y bases de datos, para citar algunos ejemplos. Esto unifica un criterio para buscar, reemplazar o simplemente para "no encontrar" más allá de la plataforma en la cual se utilice. Las expresiones regulares son una forma de abstracción de cualquier cosa que sea texto y sólo cambian detalles en las diferentes implementaciones, por lo tanto su uso "garantiza" la gestión de cadenas más allá de la plataforma en donde se apliquen.

Para comprender su uso, se debe entender primero unas definiciones básicas acerca de las herramientas de gestión que poseen las expresiones regulares:

- **Meta caracter**: es todo carácter que no se debe interpretar literalmente sino como la representación de algo más, por lo general, una funcionalidad (de ahí el prefijo *meta*).
- **Literal**: todo caracter que no sea un meta caracter es un literal (o caracter normal de texto).
- **Unidad atómica**: es aquella sobre la que se aplica un meta carácter. La cantidad menor de componentes es un literal, pero una unidad atómica puede estar compuesta por varios literales agrupados.
- **Patrón**: está formado por un conjunto de caracteres (un grupo de letras, números o signos) o por meta caracteres que representan otros caracteres, o permiten una búsqueda contextual, en otras palabras, la secuencia de literales cuya ocurrencia se busca en la cadena. Los patrones se pueden dividir en diferentes funcionalidades según se definan dentro de él. A dichas funcionalidades se las suele llamar sub patrones.

Para ejemplificar lo enunciado, suponiendo que se crea un patrón que define la coincidencia con la que se busca en una determinada cadena, se puede declarar un método que busque en la cadena dado el patrón recibiendo ambos como argumentos.

Ejemplo

C#

```
        private static void BuscarCoincidencias(string cadena, string patron)
        {
            Regex expRegular = new Regex(patron);
            bool hayCoincidencia = expRegular.IsMatch(cadena);

            Console.WriteLine("Cadena de búsqueda: " + cadena);
            Console.WriteLine("------Valores encontrados---------");
            if (hayCoincidencia)
                foreach (Match coincidencia in Regex.Matches(cadena, patron))
                    Console.WriteLine(coincidencia.Value);
            else
                Console.WriteLine("No se encontraron coincidencias");

            Console.WriteLine("-------------------------------");
        }
```

VB

```
    Sub BuscarCoincidencias(cadena As String, patron As String)
        Dim expRegular As New Regex(patron)
        Dim hayCoincidencia As Boolean = expRegular.IsMatch(cadena)

        Console.WriteLine("Cadena de búsqueda: " + cadena)
        Console.WriteLine("------Valores encontrados---------")
        If hayCoincidencia Then
            For Each coincidencia As Match In Regex.Matches(cadena, patron)
                Console.WriteLine(coincidencia.Value)
            Next
        Else
            Console.WriteLine("No se encontraron coincidencias")
        End If
        Console.WriteLine("-------------------------------")
    End Sub
```

Notar que el objeto del tipo Regex se declara definiendo el patrón y luego se utiliza el método IsMatch sobre la cadena para buscar la ocurrencia o no del patrón definido para la expresión regular.

Meta caracteres de posicionamiento o caracteres de anclaje

Los signos **^** y **$** sirven para indicar dónde debe estar situado el patrón dentro de la cadena para considerar que existe una coincidencia.

Cuando se usa el signo **^** se indica que el patrón debe aparecer al principio de la cadena de caracteres sobre la que se busca la coincidencia. En cambio, con el signo **$** se está indicando que el patrón debe aparecer al final del conjunto de caracteres o, para ser más preciso, antes de un caracter de nueva línea.

Para ejemplificar lo enunciado, suponiendo que se crea un patrón que define la coincidencia con la que se busca en una determinada cadena, se puede declarar un método que busque en la cadena dado el patrón recibiendo ambos como argumentos

Ejemplo

C#

```
using System;
using System.Text.RegularExpressions;

namespace anclaje
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Buscando con ^am (que empiece con 'am')");
            BuscarCoincidencias("amatista", @"^am");
            BuscarCoincidencias("camarones", @"^am");
            BuscarCoincidencias("panam", @"^am");

            Console.WriteLine("Buscando con am$ (que termine con 'am')");
            BuscarCoincidencias("amatista", @"am$");
            BuscarCoincidencias("camarones", @"am$");
            BuscarCoincidencias("panam", @"am$");
            // Debe empezar y terminar exactamente con "am"
            Console.WriteLine(
              "Buscando con ^am$ (empezar y terminar exactamente con 'am')");
            BuscarCoincidencias("amatista", @"^am$");
            BuscarCoincidencias("amatista panam", @"^am$");
            BuscarCoincidencias("am", @"^am$");

            Console.ReadKey();
        }

        private static void BuscarCoincidencias(string cadena, string patron)
        {
            Regex expRegular = new Regex(patron);
            bool hayCoincidencia = expRegular.IsMatch(cadena);

            Console.WriteLine("Cadena de búsqueda: " + cadena);
            Console.WriteLine("------Valores encontrados---------");
            if (hayCoincidencia)
                foreach (Match coincidencia in Regex.Matches(cadena, patron))
                    Console.WriteLine(coincidencia.Value);
            else
                Console.WriteLine("No se encontraron coincidencias");

            Console.WriteLine("---------------------------------");
        }
    }
}
```

VB

```
Imports System.Text.RegularExpressions

Module Module1

    Sub Main()
        Console.WriteLine("Buscando con ^am (que empiece con 'am')")
        BuscarCoincidencias("amatista", "^am")
        BuscarCoincidencias("camarones", "^am")
        BuscarCoincidencias("panam", "^am")

        Console.WriteLine("Buscando con am$ (que termine con 'am')")
        BuscarCoincidencias("amatista", "am$")
        BuscarCoincidencias("camarones", "am$")
        BuscarCoincidencias("panam", "am$")
        ' Debe empezar y terminar exactamente con "am"
        Console.WriteLine( _
             "Buscando con ^am$ (empezar y terminar exactamente con 'am')")
        BuscarCoincidencias("amatista", "^am$")
        BuscarCoincidencias("amatista panam", "^am$")
        BuscarCoincidencias("am", "^am$")

        Console.ReadKey()
    End Sub

    Sub BuscarCoincidencias(cadena As String, patron As String)
        Dim expRegular As New Regex(patron)
        Dim hayCoincidencia As Boolean = expRegular.IsMatch(cadena)

        Console.WriteLine("Cadena de búsqueda: " + cadena)
        Console.WriteLine("------Valores encontrados---------")
        If hayCoincidencia Then
            For Each coincidencia As Match In Regex.Matches(cadena, patron)
                Console.WriteLine(coincidencia.Value)
            Next
        Else
            Console.WriteLine("No se encontraron coincidencias")
        End If
        Console.WriteLine("---------------------------------")
    End Sub
End Module
```

El resultado obtenido es:

```
Buscando con ^am (que empiece con 'am')
amatista: True
camarones: False
panam: False
Buscando con am$ (que termine con 'am')
amatista: False
camarones: False
panam: True
Buscando con ^am$ (empezar y terminar exactamente con 'am')
```

```
amatista: False
amatista panam: False
panam: True
```

Cuantificadores

Los caracteres de anclaje sirven para encontrar elementos al final o al principio, pero ¿qué pasa si se quiere buscar elementos en cualquier lugar de una cadena, incluyendo el principio o el final? En estos casos es necesario saber si un elemento se repite en distintos lugares.

Para indicar que cierto elemento del patrón va a repetirse un número indeterminado de veces, se usa **+** o *****. Ambos caracteres significan que el patrón de literales que los antecede debe repetirse más de una vez pero ambos difieren entre sí en que el primero exige que aparezca al menos una en el orden que figuran los literales al menos una vez, mientras que el asterisco también es verdadero si no encuentra ninguno (0 ocurrencias).

Ejemplo

C#

```
using System;
using System.Text.RegularExpressions;

namespace cuantificadores1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine(
"Buscando con am* (si aparece el literal 'a' y se repite 0 o más veces 'm')");
            BuscarCoincidencias("amatista", @"am*");
            BuscarCoincidencias("camarones", @"am*");
            BuscarCoincidencias("enmarca", @"am*");
            BuscarCoincidencias("am", @"am*");

            Console.WriteLine(
"Buscando con am+ (si aparece el literal 'a' y se repite 1 o más veces 'm')");
            BuscarCoincidencias("amatista", @"am+");
            BuscarCoincidencias("camarones", @"am+");
            BuscarCoincidencias("enmarca", @"am+");
            BuscarCoincidencias("panam", @"am+");
            Console.ReadKey();
        }

        private static void BuscarCoincidencias(string cadena, string patron)
        {
            Regex myRegex = new Regex(patron);
            bool coincidencia = myRegex.IsMatch(cadena);

            Console.WriteLine("Cadena de búsqueda: " + cadena);
            Console.WriteLine("------Valores encontrados---------");
```

```
            if (coincidencia)
                foreach (Match match in Regex.Matches(cadena, patron))
                    Console.WriteLine(match.Value);
            else
                Console.WriteLine("No se encontraron coincidencias");

            Console.WriteLine("---------------------------------");
        }
    }
}
```

VB

```
Imports System.Text.RegularExpressions

Module Module1

    Sub Main()
        Console.WriteLine( _
"Buscando con am* (si aparece el literal 'a' y se repite 0 o más veces 'm')")
        BuscarCoincidencias("amatista", "am*")
        BuscarCoincidencias("camarones", "am*")
        BuscarCoincidencias("enmarca", "am*")
        BuscarCoincidencias("am", "am*")

        Console.WriteLine( _
       "Buscando con am+ (si aparece el literal 'a' y se repite 1 o más veces 'm')")
        BuscarCoincidencias("amatista", "am+")
        BuscarCoincidencias("camarones", "am+")
        BuscarCoincidencias("enmarca", "am+")
        BuscarCoincidencias("panam", "am+")
        Console.ReadKey()

    End Sub

    Sub BuscarCoincidencias(cadena As String, patron As String)
        Dim expRegular As New Regex(patron)
        Dim hayCoincidencia As Boolean = expRegular.IsMatch(cadena)

        Console.WriteLine("Cadena de búsqueda: " + cadena)
        Console.WriteLine("------Valores encontrados---------")
        If hayCoincidencia Then
            For Each coincidencia As Match In Regex.Matches(cadena, patron)
                Console.WriteLine(coincidencia.Value)
            Next
        Else
            Console.WriteLine("No se encontraron coincidencias")
        End If
        Console.WriteLine("---------------------------------")
    End Sub
End Module
```


La salida que produce el programa es:

```
Buscando con am* (si aparece el literal 'a' y se repite 0 o más veces 'm')
```

```
amatista: True
camarones: True
enmarca: True
panam: True
Buscando con am+ (si aparece el literal 'a' y se repite 1 o más veces 'm')
amatista: True
camarones: True
enmarca: False
panam: True
```

Sin embargo no se deben apresurar conclusiones. Para asegurarse de haber entendido bien el uso del cuantificador, hay que analizar la unidad atómica utilizada, que en este caso es la letra “m”. Cuando se utiliza con el literal +, la letra m debe aparecer al menos una vez a continuación de la “a”. Cuando se utiliza con un asterisco, el caracter “a” debe aparecer obligatoriamente, porque está definido en el patrón y no lo afecta ningún meta carácter, pero la letra m puede aparecer o no. Esto se puede apreciar mejor cuando aparece otro literal a continuación del meta carácter como en el siguiente ejemplo.

Ejemplo

C#

```
using System;
using System.Text.RegularExpressions;

namespace cuantificadores2
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine(
              "Buscando con am*a (si aparece 'a', 0 o más 'm' y una 'a')");
            BuscarCoincidencias("amatista", @"am*a");
            BuscarCoincidencias("camarones", @"am*a");
            BuscarCoincidencias("enmarca", @"am*a");
            BuscarCoincidencias("amrbaaaa", @"am*a");

            Console.WriteLine(
              "Buscando con am+a (si aparece 'a', 1 o más 'm' y una 'a')");
            BuscarCoincidencias("amatista", @"am+a");
            BuscarCoincidencias("camarones", @"am+a");
            BuscarCoincidencias("enmarca", @"am+a");
            BuscarCoincidencias("amraaaaa", @"am+a");
            Console.ReadKey();
        }

        private static void BuscarCoincidencias(string cadena, string patron)
        {
            Regex expRegular = new Regex(patron);
            bool hayCoincidencia = expRegular.IsMatch(cadena);
```

```
            Console.WriteLine("Cadena de búsqueda: " + cadena);
            Console.WriteLine("------Valores encontrados---------");
            if (hayCoincidencia)
                foreach (Match coincidencia in Regex.Matches(cadena, patron))
                    Console.WriteLine(coincidencia.Value);
            else
                Console.WriteLine("No se encontraron coincidencias");

            Console.WriteLine("---------------------------------");
        }
    }
}
```

VB

```
Imports System.Text.RegularExpressions

Module Module1

    Sub Main()
        Console.WriteLine( _
             "Buscando con am*a (si aparece 'a', 0 o más 'm' y una 'a')")
        BuscarCoincidencias("amatista", "am*a")
        BuscarCoincidencias("camarones", "am*a")
        BuscarCoincidencias("enmarca", "am*a")
        BuscarCoincidencias("amrbaaaa", "am*a")

        Console.WriteLine( _
             "Buscando con am+a (si aparece 'a', 1 o más 'm' y una 'a')")
        BuscarCoincidencias("amatista", "am+a")
        BuscarCoincidencias("camarones", "am+a")
        BuscarCoincidencias("enmarca", "am+a")
        BuscarCoincidencias("amraaaaa", "am+a")
        Console.ReadKey()

    End Sub

    Sub BuscarCoincidencias(cadena As String, patron As String)
        Dim expRegular As New Regex(patron)
        Dim hayCoincidencia As Boolean = expRegular.IsMatch(cadena)

        Console.WriteLine("Cadena de búsqueda: " + cadena)
        Console.WriteLine("------Valores encontrados---------")
        If hayCoincidencia Then
            For Each coincidencia As Match In Regex.Matches(cadena, patron)
                Console.WriteLine(coincidencia.Value)
            Next
        Else
            Console.WriteLine("No se encontraron coincidencias")
        End If
        Console.WriteLine("---------------------------------")
    End Sub
End Module
```

La salida obtenida es:

```
Buscando con am*a (si aparece 'a', 0 o más 'm' y una 'a')
amatista: True
camarones: True
enmarca: False
amrbaaaaa: True
Buscando con am+a (si aparece 'a', 1 o más 'm' y una 'a')
amatista: True
camarones: True
enmarca: False
amraaaaa: False
```

Hasta ahora con * y + se analizaron repeticiones (se "cuantificaron" literales), pero el asterisco puede llevar a una confusión ya que determina si se repite 0 o más veces una unidad atómica. Esto no debe interpretarse como "aparece o no el literal". Para esta situación existe el meta carácter **?**. Éste permite analizar en la cadena la ocurrencia o no de un carácter, pero de exactamente uno. Si se repiten caracteres iguales en la misma posición, para esta meta carácter no coincide la cadena con el patrón.

Ejemplo

C#

```
using System;
using System.Text.RegularExpressions;

namespace cuantificadores3
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine(
              "Buscando con am*a (si aparece 'a', 0 o más 'm' y una 'a')");
            BuscarCoincidencias("amatista", @"am*a");
            BuscarCoincidencias("camarones", @"am*a");
            BuscarCoincidencias("enmaa", @"am*a");
            BuscarCoincidencias("ammmmmmma", @"am*a");

            Console.WriteLine(
              "Buscando con am?a (si aparece una 'm' entre dos 'a')");
            BuscarCoincidencias("amatista", @"am?a");
            BuscarCoincidencias("camarones", @"am?a");
            BuscarCoincidencias("enmaa", @"am?a");
            BuscarCoincidencias("ammmmmmma", @"am?a");

            Console.ReadKey();
        }
```

```
        private static void BuscarCoincidencias(string cadena, string patron)
        {
            Regex expRegular = new Regex(patron);
            bool hayCoincidencia = expRegular.IsMatch(cadena);

            Console.WriteLine("Cadena de búsqueda: " + cadena);
            Console.WriteLine("------Valores encontrados---------");
            if (hayCoincidencia)
                foreach (Match coincidencia in Regex.Matches(cadena, patron))
                    Console.WriteLine(coincidencia.Value);
            else
                Console.WriteLine("No se encontraron coincidencias");

            Console.WriteLine("---------------------------------");
        }
    }
}
```

VB

```
Imports System.Text.RegularExpressions

Module Module1

    Sub Main()
        Console.WriteLine( _
             "Buscando con am*a (si aparece 'a', 0 o más 'm' y una 'a')")
        BuscarCoincidencias("amatista", "am*a")
        BuscarCoincidencias("camarones", "am*a")
        BuscarCoincidencias("enmaa", "am*a")
        BuscarCoincidencias("ammmmmmma", "am*a")

        Console.WriteLine("Buscando con am?a (si aparece una 'm' entre dos 'a')")
        BuscarCoincidencias("amatista", "am?a")
        BuscarCoincidencias("camarones", "am?a")
        BuscarCoincidencias("enmaa", "am?a")
        BuscarCoincidencias("ammmmmmma", "am?a")

        Console.ReadKey()

    End Sub

    Sub BuscarCoincidencias(cadena As String, patron As String)
        Dim expRegular As New Regex(patron)
        Dim hayCoincidencia As Boolean = expRegular.IsMatch(cadena)

        Console.WriteLine("Cadena de búsqueda: " + cadena)
        Console.WriteLine("------Valores encontrados---------")
        If hayCoincidencia Then
            For Each coincidencia As Match In Regex.Matches(cadena, patron)
                Console.WriteLine(coincidencia.Value)
            Next
        Else
            Console.WriteLine("No se encontraron coincidencias")
        End If
        Console.WriteLine("---------------------------------")
```

```
    End Sub
End Module
```

La salida obtenida es:

```
Buscando con am*a (si aparece 'a', 0 o más 'm' y una 'a')
amatista: True
camarones: True
enmaa: True
ammmmmmma: True
Buscando con am?a (si aparece una 'm' entre dos 'a')
amatista: True
camarones: True
enmaa: True
ammmmmmma: False
```

Si bien los meta caracteres dan cierto control, no son suficientes para detectar las ocurrencias de una unidad atómica definida en un patrón dentro de una cadena. Un caso típico es encontrar si determinado literal está repito exactamente, una cantidad mayor que o una cantidad menor que o entre determinados valores dentro de una cadena.

Ejemplo

C#

```
using System;
using System.Text.RegularExpressions;

namespace cuantificadores4
{
    class Program
    {
        static void Main(string[] args)
        {
            // {n} Exactamente n ocurrencias
            Console.WriteLine("Buscando con a{2} (si aparece  'a' 2 veces)");
            BuscarCoincidencias("amatista", @"a{2}");
            BuscarCoincidencias("camarones", @"a{2}");
            BuscarCoincidencias("enmaa", @"a{2}");
            BuscarCoincidencias("aaammmmmmma", @"a{2}");

            // {n,} Al menos n ocurrencias
            Console.WriteLine(
              "Buscando con a{2,} (si aparece 'a' al menos 2 veces)");
            BuscarCoincidencias("amatista", @"a{2,}");
            BuscarCoincidencias("camarones", @"a{2,}");
            BuscarCoincidencias("enmaa", @"a{2,}");
            BuscarCoincidencias("aaammmmmmma", @"a{2,}");

            // {,n} Hasta n ocurrencias
            Console.WriteLine(
              "Buscando con a{,2} (si aparece 'a' no más de 2 veces)");
```

```
            BuscarCoincidencias("amatista", @"a{,2}");
            BuscarCoincidencias("camarones", @"a{,2}");
            BuscarCoincidencias("enmaa", @"a{2,}");
            BuscarCoincidencias("aaammmmmma", @"a{,2}");

            // {n,m} Entre n y m ocurrencias (inclusivo)
            Console.WriteLine(
        "Buscando con a{2,3} (si aparece 'a' al menos 2 veces y hasta 3 inclusive)");
            BuscarCoincidencias("amatista", @"a{2,3}");
            BuscarCoincidencias("camarones", @"a{2,3}");
            BuscarCoincidencias("enmaa", @"a{2,3}");
            BuscarCoincidencias("aaammmmmma", @"a{2,3}");
            Console.ReadKey();
        }

        private static void BuscarCoincidencias(string cadena, string patron)
        {
            Regex expRegular = new Regex(patron);
            bool hayCoincidencia = expRegular.IsMatch(cadena);

            Console.WriteLine("Cadena de búsqueda: " + cadena);
            Console.WriteLine("------Valores encontrados---------");
            if (hayCoincidencia)
                foreach (Match coincidencia in Regex.Matches(cadena, patron))
                    Console.WriteLine(coincidencia.Value);
            else
                Console.WriteLine("No se encontraron coincidencias");

            Console.WriteLine("-----------------------------------");
        }
    }
}
```

VB

```
Imports System.Text.RegularExpressions

Module Module1

    Sub Main()
        ' {n} Exactamente n ocurrencias
        Console.WriteLine("Buscando con a{2} (si aparece  'a' 2 veces)")
        BuscarCoincidencias("amatista", "a{2}")
        BuscarCoincidencias("camarones", "a{2}")
        BuscarCoincidencias("enmaa", "a{2}")
        BuscarCoincidencias("aaammmmmma", "a{2}")


        ' {n,} Al menos n ocurrencias
        Console.WriteLine("Buscando con a{2,} (si aparece 'a' al menos 2 veces)")
        BuscarCoincidencias("amatista", "a{2,}")
        BuscarCoincidencias("camarones", "a{2,}")
        BuscarCoincidencias("enmaa", "a{2,}")
        BuscarCoincidencias("aaammmmmma", "a{2,}")

        ' {,n} Hasta n ocurrencias
```

```
        Console.WriteLine("Buscando con a{,2} (si aparece 'a' no más de 2 veces)")
        BuscarCoincidencias("amatista", "a{,2}")
        BuscarCoincidencias("camarones", "a{,2}")
        BuscarCoincidencias("enmaa", "a{2,}")
        BuscarCoincidencias("aaammmmmmma", "a{,2}")

        ' {n,m} Entre n y m ocurrencias (inclusivo)
        Console.WriteLine( _
       "Buscando con a{2,3} (si aparece 'a' al menos 2 veces y hasta 3 inclusive)")
        BuscarCoincidencias("amatista", "a{2,3}")
        BuscarCoincidencias("camarones", "a{2,3}")
        BuscarCoincidencias("enmaa", "a{2,3}")
        BuscarCoincidencias("aaammmmmmma", "a{2,3}")
        Console.ReadKey()
    End Sub

    Sub BuscarCoincidencias(cadena As String, patron As String)
        Dim expRegular As New Regex(patron)
        Dim hayCoincidencia As Boolean = expRegular.IsMatch(cadena)

        Console.WriteLine("Cadena de búsqueda: " + cadena)
        Console.WriteLine("------Valores encontrados--------")
        If hayCoincidencia Then
            For Each coincidencia As Match In Regex.Matches(cadena, patron)
                Console.WriteLine(coincidencia.Value)
            Next
        Else
            Console.WriteLine("No se encontraron coincidencias")
        End If
        Console.WriteLine("-----------------------------------")
    End Sub
End Module
```

La salida obtenida es:

```
Buscando con a{2} (si aparece  'a' 2 veces)
amatista: False
camarones: False
enmaa: True
aaammmmma: True
Buscando con a{2,} (si aparece 'a' al menos 2 veces)
amatista: False
camarones: False
enmaa: True
aaammmmmma: True
Buscando con a{,2} (si aparece 'a' no más de 2 veces)
amatista: False
camarones: False
enmaa: True
aaammmmmma: False
Buscando con a{2,3} (si aparece 'a' al menos 2 veces y hasta 3 inclusive)
```

```
amatista: False
camarones: False
enmaa: True
aaammmmmma: True
```

La siguiente tabla es un resumen de los caracteres especiales usados como cuantificadores

Meta Caracter	Descripción
n*	Cero o más de n
n+	Uno o más de n
n?	Uno o ninguno de n
{n}	Exactamente n ocurrencias
{n,}	Al menos n ocurrencias
{,m}	Hasta m ocurrencias

Agrupadores y rangos

Por cuestiones de simplicidad al comienzo de la explicación siempre se trabajó con unidades atómicas que eran simples literales, pero en la realidad, la mayoría de los casos en los cuales se analiza una cadena, se busca un conjunto de literales con los cuales operar.

Los paréntesis sirven para agrupar un subconjunto de literales de manera que esta puede ser la nueva unidad atómica en lugar de un solo literal como se utilizó hasta el momento.

Por ejemplo, se puede utilizar en combinación con los cuantificadores para realizar operaciones de búsquedas en una cadena, como muestra el siguiente ejemplo.

Ejemplo

C#

```
using System;
using System.Text.RegularExpressions;

namespace cuantificadores5
{
    class Program
    {
        static void Main(string[] args)
        {
            // {n} Exactamente n ocurrencias
            Console.WriteLine("Buscando con (ama){2} (si aparece  'ama' 2 veces)");
            BuscarCoincidencias("ama", @"(ama){2}");
            BuscarCoincidencias("rpppamaamass", @"(ama){2}");
            BuscarCoincidencias("amaamaamaamaama", @"(ama){2}");
            BuscarCoincidencias("amaama", @"(ama){2}");
            BuscarCoincidencias("amaamaamaama", @"(ama){2}");

            // {n,} Al menos n ocurrencias
            Console.WriteLine(
```

```
            "Buscando con (ama){2,} (si aparece 'ama' al menos 2 veces)");
            BuscarCoincidencias("ama", @"(ama){2}");
            BuscarCoincidencias("rpppamaamass", @"a{2,}");
            BuscarCoincidencias("amaamaamaamaama", @"a{2,}");
            BuscarCoincidencias("amaama", @"a{2,}");
            BuscarCoincidencias("amaamaamaama", @"a{2,}");

            // {,n} Hasta n ocurrencias
            Console.WriteLine(
             "Buscando con (ama){,2} (si aparece 'ama' no más de 2 veces)");
            BuscarCoincidencias("ama", @"(ama){2}");
            BuscarCoincidencias("rpppamaamass", @"(ama){,2}");
            BuscarCoincidencias("amaamaamaamaama", @"(ama){,2}");
            BuscarCoincidencias("amaama", @"(ama){2,}");
            BuscarCoincidencias("amaamaamaama", @"(ama){,2}");

            // {n,m} Entre n y m ocurrencias (inclusivo)
            Console.WriteLine(
"Buscando con (ama){2,3} (si aparece 'ama' al menos 2 veces y hasta 3 inclusive)");
            BuscarCoincidencias("ama", @"(ama){2}");
            BuscarCoincidencias("rpppamaamass", @"(ama){2,3}");
            BuscarCoincidencias("amaamaamaamaama", @"(ama){2,3}");
            BuscarCoincidencias("amaama", @"(ama){2,3}");
            BuscarCoincidencias("amaamaamaama", @"(ama){2,3}");
            Console.ReadKey();
        }

        private static void BuscarCoincidencias(string cadena, string patron)
        {
            Regex myRegex = new Regex(patron);
            bool coincidencia = myRegex.IsMatch(cadena);

            Console.WriteLine("Cadena de búsqueda: " + cadena);
            Console.WriteLine("-----Valores encontrados---------");
            if (coincidencia)
                foreach (Match match in Regex.Matches(cadena, patron))
                    Console.WriteLine(match.Value);
            else
                Console.WriteLine("No se encontraron coincidencias");

            Console.WriteLine("---------------------------------");
        }
    }
}
```

VB

```
Imports System.Text.RegularExpressions

Module Module1

    Sub Main()
        ' {n} Exactamente n ocurrencias
        Console.WriteLine("Buscando con (ama){2} (si aparece  'ama' 2 veces)")
        BuscarCoincidencias("ama", "(ama){2}")
        BuscarCoincidencias("rpppamaamass", "(ama){2}")
```

```
        BuscarCoincidencias("amaamaamaamaama", "(ama){2}")
        BuscarCoincidencias("amaama", "(ama){2}")
        BuscarCoincidencias("amaamaamaama", "(ama){2}")

        ' {n,} Al menos n ocurrencias
        Console.WriteLine( _
             "Buscando con (ama){2,} (si aparece 'ama' al menos 2 veces)")
        BuscarCoincidencias("ama", "(ama){2}")
        BuscarCoincidencias("rpppamaamass", "a{2,}")
        BuscarCoincidencias("amaamaamaamaama", "a{2,}")
        BuscarCoincidencias("amaama", "a{2,}")
        BuscarCoincidencias("amaamaamaama", "a{2,}")

        ' {,n} Hasta n ocurrencias
        Console.WriteLine( _
             "Buscando con (ama){,2} (si aparece 'ama' no más de 2 veces)")
        BuscarCoincidencias("ama", "(ama){2}")
        BuscarCoincidencias("rpppamaamass", "(ama){,2}")
        BuscarCoincidencias("amaamaamaamaama", "(ama){,2}")
        BuscarCoincidencias("amaama", "(ama){2,}")
        BuscarCoincidencias("amaamaamaama", "(ama){,2}")

        ' {n,m} Entre n y m ocurrencias (inclusivo)
        Console.WriteLine( _
"Buscando con (ama){2,3} (si aparece 'ama' al menos 2 veces y hasta 3 inclusive)")
        BuscarCoincidencias("ama", "(ama){2}")
        BuscarCoincidencias("rpppamaamass", "(ama){2,3}")
        BuscarCoincidencias("amaamaamaamaama", "(ama){2,3}")
        BuscarCoincidencias("amaama", "(ama){2,3}")
        BuscarCoincidencias("amaamaamaama", "(ama){2,3}")
        Console.ReadKey()
    End Sub

    Sub BuscarCoincidencias(cadena As String, patron As String)
        Dim expRegular As New Regex(patron)
        Dim hayCoincidencia As Boolean = expRegular.IsMatch(cadena)

        Console.WriteLine("Cadena de búsqueda: " + cadena)
        Console.WriteLine("------Valores encontrados---------")
        If hayCoincidencia Then
            For Each coincidencia As Match In Regex.Matches(cadena, patron)
                Console.WriteLine(coincidencia.Value)
            Next
        Else
            Console.WriteLine("No se encontraron coincidencias")
        End If
        Console.WriteLine("---------------------------------")
    End Sub
End Module
```

La salida obtenida es:

```
Buscando con (ama){2} (si aparece  'ama' 2 veces)
ama: False
```

```
rpppamaamass: True
amaamaamaamaama: True
amaama: True
amaamaamaama: True
Buscando con (ama){2,} (si aparece 'ama' al menos 2 veces)
ama: False
rpppamaamass: True
amaamaamaamaama: True
amaama: True
amaamaamaama: True
Buscando con (ama){,2} (si aparece 'ama' no más de 2 veces)
ama: False
rpppamaamass: False
amaamaamaamaama: False
amaama: True
amaamaamaama: False
Buscando con (ama){2,3} (si aparece 'ama' al menos 2 veces y hasta 3 inclusive)
ama: False
rpppamaamass: True
amaamaamaamaama: True
amaama: True
amaamaamaama: True
```

Los corchetes [] incluidos en un patrón permiten especificar el rango de caracteres válidos a comparar. Basta que exista cualquiera de ellos para que se dé la condición. Dentro de un rango se puede especificar cualquier cantidad de caracteres, uno a continuación del otro, siendo válidos los literales alfanuméricos para definir la unidad atómica del patrón.

Ejemplo

C#

```
using System;
using System.Text.RegularExpressions;

namespace rangos
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Buscando con [0-9999]-[0-9999]");
            BuscarCoincidencias("1253-3122", @"[0-9999]-[0-9999]");
            BuscarCoincidencias("12534-3122", @"[0-9999]-[0-9999]");
            BuscarCoincidencias("1253-31224", @"[0-9999]-[0-9999]");
            BuscarCoincidencias("125-3122", @"[0-9999]-[0-9999]");
            BuscarCoincidencias("1253-312", @"[0-9999]-[0-9999]");
            BuscarCoincidencias("125a-312", @"[0-9999]-[0-9999]");
```

```
            Console.WriteLine("Buscando con [0-9]-[0-9]");
            BuscarCoincidencias("1253-3122", @"[0-9]-[0-9]");
            BuscarCoincidencias("12534-3122", @"[0-9]-[0-9]");
            BuscarCoincidencias("1253-31224", @"[0-9]-[0-9]");
            BuscarCoincidencias("125-3122", @"[0-9]-[0-9]");
            BuscarCoincidencias("1253-312", @"[0-9]-[0-9]");
            BuscarCoincidencias("125a-312", @"[0-9]-[0-9]");

            Console.ReadKey();
        }

        private static void BuscarCoincidencias(string cadena, string patron)
        {
            Regex expRegular = new Regex(patron);
            bool hayCoincidencia = expRegular.IsMatch(cadena);

            Console.WriteLine("Cadena de búsqueda: " + cadena);
            Console.WriteLine("------Valores encontrados----------");
            if (hayCoincidencia)
                foreach (Match coincidencia in Regex.Matches(cadena, patron))
                    Console.WriteLine(coincidencia.Value);
            else
                Console.WriteLine("No se encontraron coincidencias");

            Console.WriteLine("-----------------------------------");
        }
    }
}
```

VB

```
Imports System.Text.RegularExpressions

Module Module1

    Sub Main()
        Console.WriteLine("Buscando con [0-9999]-[0-9999]")
        BuscarCoincidencias("1253-3122", "[0-9999]-[0-9999]")
        BuscarCoincidencias("12534-3122", "[0-9999]-[0-9999]")
        BuscarCoincidencias("1253-31224", "[0-9999]-[0-9999]")
        BuscarCoincidencias("125-3122", "[0-9999]-[0-9999]")
        BuscarCoincidencias("1253-312", "[0-9999]-[0-9999]")
        BuscarCoincidencias("125a-312", "[0-9999]-[0-9999]")

        Console.WriteLine("Buscando con [0-9]-[0-9]")
        BuscarCoincidencias("1253-3122", "[0-9]-[0-9]")
        BuscarCoincidencias("12534-3122", "[0-9]-[0-9]")
        BuscarCoincidencias("1253-31224", "[0-9]-[0-9]")
        BuscarCoincidencias("125-3122", "[0-9]-[0-9]")
        BuscarCoincidencias("1253-312", "[0-9]-[0-9]")
        BuscarCoincidencias("125a-312", "[0-9]-[0-9]")

        Console.ReadKey()

    End Sub
```

```
    Sub BuscarCoincidencias(cadena As String, patron As String)
        Dim expRegular As New Regex(patron)
        Dim hayCoincidencia As Boolean = expRegular.IsMatch(cadena)

        Console.WriteLine("Cadena de búsqueda: " + cadena)
        Console.WriteLine("------Valores encontrados---------")
        If hayCoincidencia Then
            For Each coincidencia As Match In Regex.Matches(cadena, patron)
                Console.WriteLine(coincidencia.Value)
            Next
        Else
            Console.WriteLine("No se encontraron coincidencias")
        End If
        Console.WriteLine("---------------------------------")
    End Sub
End Module
```

La salida obtenida es:

```
Buscando con [0-9999]-[0-9999]
1253-3122: True
12534-3122: True
1253-31224: True
125-3122: True
1253-312: True
125a-312: False
Buscando con [0-9]-[0-9]
1253-3122: True
12534-3122: True
1253-31224: True
125-3122: True
1253-312: True
125a-312: False
```

Recordar que el rango se aplica a cada unidad atómica, lo cual en este caso es cada literal. Esta es la causa por la cual ambas salidas son iguales. En el primer grupo el patrón tiene un rango [0-9999]-[0-9999] lo cual indica que la coincidencia debe ser “número hasta 9999” ”-” “número hasta 9999”, o sea, no está definido bien el rango para un solo literal a pesar que numéricamente sea correcto y ésta sea la causa por la cual lo procesa. Este tipo de “errores” que igual se procesan puede llevar a confusiones o conclusiones falsas. El rango definido para el segundo grupo de análisis es más adecuado. Notar que sólo si se cumple ***número-número*** se considera una coincidencia.

El punto

El punto representa cualquier literal. Escribiendo un punto en un patrón indica que la coincidencia depende que en ese lugar aparezca un caracter cualquiera, sin importar cuál sea.

Si un meta caracter es un literal que puede representar a otros, entonces el punto es el meta caracter por excelencia. Un punto en el patrón representa cualquier caracter excepto el de nueva línea.

Ejemplo

C#

```
using System;
using System.Text.RegularExpressions;

namespace punto
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Buscando con pu..o");
            BuscarCoincidencias("pulgoso", @"pu..o");
            BuscarCoincidencias("apuno", @"pu..o");
            BuscarCoincidencias("apunto", @"pu..o");
            BuscarCoincidencias("puntos", @"pu..o");

            Console.ReadKey();
        }

        private static void BuscarCoincidencias(string cadena, string patron)
        {
            Regex expRegular = new Regex(patron);
            bool hayCoincidencia = expRegular.IsMatch(cadena);

            Console.WriteLine("Cadena de búsqueda: " + cadena);
            Console.WriteLine("-----Valores encontrados---------");
            if (hayCoincidencia)
                foreach (Match coincidencia in Regex.Matches(cadena, patron))
                    Console.WriteLine(coincidencia.Value);
            else
                Console.WriteLine("No se encontraron coincidencias");

            Console.WriteLine("---------------------------------");
        }
    }
}
```

VB

```
Imports System.Text.RegularExpressions

Module Module1

    Sub Main()
        Console.WriteLine("Buscando con pu..o")
        BuscarCoincidencias("pulgoso", "pu..o")
        BuscarCoincidencias("apuno", "pu..o")
        BuscarCoincidencias("apunto", "pu..o")
```

```
        BuscarCoincidencias("puntos", "pu..o")

        Console.ReadKey()

    End Sub

    Sub BuscarCoincidencias(cadena As String, patron As String)
        Dim expRegular As New Regex(patron)
        Dim hayCoincidencia As Boolean = expRegular.IsMatch(cadena)

        Console.WriteLine("Cadena de búsqueda: " + cadena)
        Console.WriteLine("------Valores encontrados---------")
        If hayCoincidencia Then
            For Each coincidencia As Match In Regex.Matches(cadena, patron)
                Console.WriteLine(coincidencia.Value)
            Next
        Else
            Console.WriteLine("No se encontraron coincidencias")
        End If
        Console.WriteLine("-------------------------------")
    End Sub
End Module
```

La salida es:

```
Buscando con pu..o
pulgoso: True
apuno: False
apunto: True
puntos: True
```

Alternación

En la búsqueda de coincidencias se necesitan alternativas que son válidas como coincidencias para una misma solución. Por ejemplo, si se desea tener la capacidad de determinar si una cadena incluye referencias a que una persona va al norte o al sur y que el mismo patrón lo determine, se debe poder considerar válidas las coincidencias con las palabras norte y sur dentro de una cadena. Esto quiere decir que el patrón debe establecer que una alternativa o la otra son coincidencias y como deben estar agrupados todos los literales que componen la palabra, se deben encontrar entre paréntesis. Una solución viable a este problema se encuentra usando el meta caracter **|**, también conocido como literal de alternación.

Ejemplo

C#

```
using System;
using System.Text.RegularExpressions;

namespace alternacion
{
    class Program
```

```
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Buscando con (norte|sur)");
            BuscarCoincidencias("Va al norte", @"(norte|sur)");
            BuscarCoincidencias("Va al oeste", @"(norte|sur)");
            BuscarCoincidencias("Va al este", @"(norte|sur)");
            BuscarCoincidencias("Va al sur", @"(norte|sur)");

            Console.ReadKey();
        }

        private static void BuscarCoincidencias(string cadena, string patron)
        {
            Regex expRegular = new Regex(patron);
            bool hayCoincidencia = expRegular.IsMatch(cadena);

            Console.WriteLine("Cadena de búsqueda: " + cadena);
            Console.WriteLine("------Valores encontrados--------");
            if (hayCoincidencia)
                foreach (Match coincidencia in Regex.Matches(cadena, patron))
                    Console.WriteLine(coincidencia.Value);
            else
                Console.WriteLine("No se encontraron coincidencias");

            Console.WriteLine("-----------------------------------");
        }
    }
}
```

VB

```
Imports System.Text.RegularExpressions

Module Module1

    Sub Main()
        Console.WriteLine("Buscando con (norte|sur)")
        BuscarCoincidencias("Va al norte", "(norte|sur)")
        BuscarCoincidencias("Va al oeste", "(norte|sur)")
        BuscarCoincidencias("Va al este", "(norte|sur)")
        BuscarCoincidencias("Va al sur", "(norte|sur)")

        Console.ReadKey()
    End Sub

    Sub BuscarCoincidencias(cadena As String, patron As String)
        Dim expRegular As New Regex(patron)
        Dim hayCoincidencia As Boolean = expRegular.IsMatch(cadena)

        Console.WriteLine("Cadena de búsqueda: " + cadena)
        Console.WriteLine("------Valores encontrados---------")
        If hayCoincidencia Then
            For Each coincidencia As Match In Regex.Matches(cadena, patron)
                Console.WriteLine(coincidencia.Value)
            Next
```

```
        Else
            Console.WriteLine("No se encontraron coincidencias")
        End If
        Console.WriteLine("---------------------------------")
    End Sub
End Module
```

La salida obtenida es:

```
Buscando con (norte|sur)
Va al norte: True
Va al oeste: False
Va al este: False
Va al sur: True
```

Caracteres de escape

En muchas situaciones es necesario utilizar en el patrón literales considerados meta caracteres para encontrar coincidencias en las cadenas. Este caso, el cual es el más simple, demarca la necesidad de un carácter especial que determine el comportamiento de otro carácter de manera especial. En las expresiones regulares dicho carácter es la contra barra o barra invertida (****). Para los programadores C o C# este tipo de formato o funcionalidad es conocida y se suele utilizar para dar formato de salida a las cadenas. Recordar que estos lenguajes se utiliza dos veces seguidas la contra barra, donde la primera actúa como indicador de carácter de escape y la segunda se interpreta como la que se quiere utilizar. En lenguajes como C# existe el caracter @ que indica al compilador que interprete literalmente la cadena y eso evita la necesidad de usar "****".

La barra invertida se utiliza para "marcar" el siguiente carácter de la expresión de búsqueda de forma que este adquiera un significado especial o deje de tenerlo. O sea, la barra invertida no se utiliza nunca por sí sola, sino en combinación con otros caracteres. Al utilizarlo por ejemplo en combinación con el punto "\." este deja de tener su significado normal y se comporta como un carácter literal.

De la misma forma, cuando se coloca la barra inversa seguida de cualquiera de los caracteres especiales que discutiremos a continuación, estos dejan de tener su significado especial y se convierten en caracteres de búsqueda literal.

Por lo tanto, la barra inversa se utiliza tanto para cambiar el comportamiento por defecto de los meta caracteres y que éstos se comporten como un literal común o determina que una declaración con cierto carácter determine una funcionalidad especial y se los conviertan en declaraciones que afecten a las búsquedas de literales en las cadenas. La siguiente tabla muestra estos caracteres:

Meta Caracter	Descripción

Meta Caracter	Descripción
\t	Representa un tabulador.
\r	Representa el "retorno de carro" o "regreso al inicio" o sea el lugar en que la línea vuelve a iniciar.
\n	Representa la "nueva línea" el carácter por medio del cual una línea da inicio. Es necesario recordar que en Windows es necesaria una combinación de \r\n para comenzar una nueva línea, mientras que en Unix solamente se usa \n y en Mac_OS clásico se usa solamente \r.
\a	Representa una "campana" o "beep" que se produce al imprimir este carácter.
\e	Representa la tecla "Esc" o "Escape"
\f	Representa un salto de página
\v	Representa un tabulador vertical
\x	Se utiliza para representar caracteres ASCII o ANSI si conoce su código hexadecimal.
\u	Se utiliza para representar caracteres Unicode si se conoce su código hexadecimal.

Ejemplo

C#

```
using System;
using System.Text.RegularExpressions;

namespace escape
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Buscando con o\\. E");
            BuscarCoincidencias("Ejemplo. Esta es una cadena", @"o\. E");

            Console.ReadKey();
        }

        private static void BuscarCoincidencias(string cadena, string patron)
        {
            Regex myRegex = new Regex(patron);
            bool coincidencia = myRegex.IsMatch(cadena);

            Console.WriteLine("Cadena de búsqueda: " + cadena);
            Console.WriteLine("------Valores encontrados---------");
            if (coincidencia)
                foreach (Match match in Regex.Matches(cadena, patron))
                    Console.WriteLine(match.Value);
            else
                Console.WriteLine("No se encontraron coincidencias");

            Console.WriteLine("---------------------------------");
```

```
        }
    }
}
```

VB

```
Imports System.Text.RegularExpressions

Module Module1

    Sub Main()
        Console.WriteLine("Buscando con o\\. E")
        BuscarCoincidencias("Ejemplo. Esta es una cadena", "o\. E")

        Console.ReadKey()
    End Sub

    Sub BuscarCoincidencias(cadena As String, patron As String)
        Dim expRegular As New Regex(patron)
        Dim hayCoincidencia As Boolean = expRegular.IsMatch(cadena)

        Console.WriteLine("Cadena de búsqueda: " + cadena)
        Console.WriteLine("------Valores encontrados---------")
        If hayCoincidencia Then
            For Each coincidencia As Match In Regex.Matches(cadena, patron)
                Console.WriteLine(coincidencia.Value)
            Next
        Else
            Console.WriteLine("No se encontraron coincidencias")
        End If
        Console.WriteLine("-------------------------------")
    End Sub
End Module
```

La salida obtenida es:

```
Buscando con o\. E
Cadena de búsqueda: Ejemplo. Esta es una cadena
------Valores encontrados---------
o. E
-------------------------------
```


Clases de caracteres

Una clase de caracteres representa un conjunto de literales que pueden coincidir con una cadena de entrada. Se pueden combinar literales, caracteres de escape, y clases de caracteres para formar el patrón de una expresión regular. Las clases de caracteres definen un conjunto de caracteres.

La siguiente tabla enumera los más comunes:

Meta Caracter	Descripción
\w	Representa cualquier carácter alfanumérico.
\W	Representa cualquier carácter no alfanumérico.

Meta Caracter	Descripción
\d	Representa un dígito del 0 al 9.
\D	Representa cualquier carácter que no sea un dígito del 0 al 9.
\s	Representa un espacio en blanco.
\S	Representa cualquier carácter que no sea un espacio en blanco.
\A	Representa el inicio de la cadena. No un carácter sino una posición.
\Z	Representa el final de la cadena. No un carácter sino una posición.
\b	Marca el inicio y el final de una palabra.
\B	Marca la posición entre dos caracteres alfanuméricos o dos no-alfanuméricos.

Ejemplo

C#

```
using System;
using System.Text.RegularExpressions;

namespace clases
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Buscando con o\\.\\sE");
            BuscarCoincidencias("Ejemplo. Esta es una cadena", @"o\.\sE");

            Console.ReadKey();
        }

        private static void BuscarCoincidencias(string cadena, string patron)
        {
            Regex expRegular = new Regex(patron);
            bool hayCoincidencia = expRegular.IsMatch(cadena);

            Console.WriteLine("Cadena de búsqueda: " + cadena);
            Console.WriteLine("------Valores encontrados---------");
            if (hayCoincidencia)
                foreach (Match coincidencia in Regex.Matches(cadena, patron))
                    Console.WriteLine(coincidencia.Value);
            else
                Console.WriteLine("No se encontraron coincidencias");

            Console.WriteLine("---------------------------------");
        }
    }
}
```

VB

```
Imports System.Text.RegularExpressions

Module Module1
```

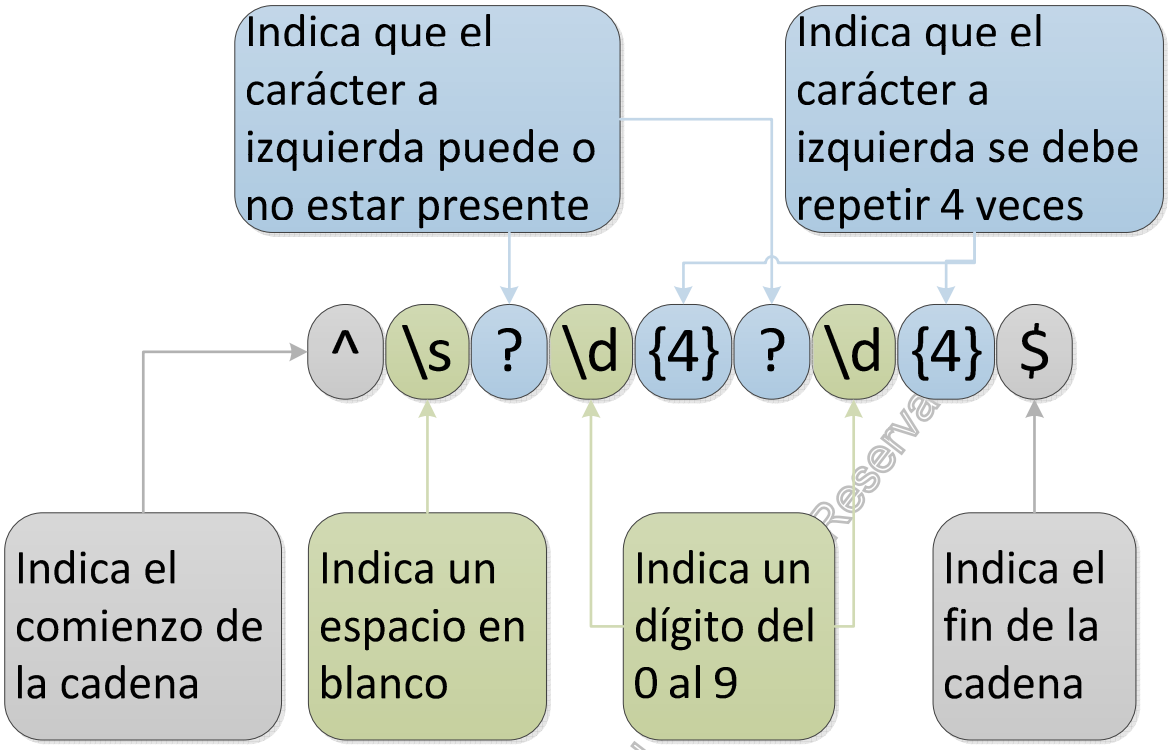
```
    Sub Main()
        Console.WriteLine("Buscando con o\\.\\sE")
        BuscarCoincidencias("Ejemplo. Esta es una cadena", "o\.\sE")

        Console.ReadKey()
    End Sub

    Sub BuscarCoincidencias(cadena As String, patron As String)
        Dim expRegular As New Regex(patron)
        Dim hayCoincidencia As Boolean = expRegular.IsMatch(cadena)

        Console.WriteLine("Cadena de búsqueda: " + cadena)
        Console.WriteLine("------Valores encontrados---------")
        If hayCoincidencia Then
            For Each coincidencia As Match In Regex.Matches(cadena, patron)
                Console.WriteLine(coincidencia.Value)
            Next
        Else
            Console.WriteLine("No se encontraron coincidencias")
        End If
        Console.WriteLine("---------------------------------")
    End Sub
End Module
```

La salida obtenida es:

```
Buscando con o\.\sE
Cadena de búsqueda: Ejemplo. Esta es una cadena
------Valores encontrados---------
o. E
---------------------------------
```

Gestión de patrones de búsqueda

Las expresiones regulares pueden parecer inmensamente complejas, ya que son una forma de codificar que se debe estudiar y practicar para poder dominarlas al escribirlas.



La expresión regular que muestra la figura es relativamente simple. Se ajusta con una secuencia de nueve o diez caracteres: puede comenzar con un espacio en blanco o no, cuatro caracteres a continuación serán dígitos del 0 al 9, un guion opcional (puede estar o no), y luego cuatro dígitos más.

Se debe prestar especial atención cuando se utilizan clases de caracteres como el del espacio en blanco que muestra el ejemplo. Si se confunde y se pone la letra en mayúsculas quiere decir exactamente lo contrario, o sea, cualquier literal a excepción del espacio en blanco y las coincidencias obtenidas van a varia notablemente, como muestra el siguiente ejemplo.

Ejemplo

C#

```
using System;
using System.Text.RegularExpressions;

namespace gestion
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Buscando con ^\\s?\\d{4}-?\\d{4}$");
            BuscarCoincidencias("4567-8910", @"^\s?\d{4}-?\d{4}$");
            BuscarCoincidencias(" 4567-8910", @"^\s?\d{4}-?\d{4}$");
            BuscarCoincidencias("  4567-8910", @"^\s?\d{4}-?\d{4}$");
```

```
            Console.WriteLine("Buscando con ^\\S?\\d{4}-?\\d{4}$");
            BuscarCoincidencias("4567-8910", @"^\S?\d{4}-?\d{4}$");
            BuscarCoincidencias(" 4567-8910", @"^\S?\d{4}-?\d{4}$");
            BuscarCoincidencias("  4567-8910", @"^\S?\d{4}-?\d{4}$");
            Console.ReadKey();
        }

        private static void BuscarCoincidencias(string cadena, string patron)
        {
            Regex expRegular = new Regex(patron);
            bool hayCoincidencia = expRegular.IsMatch(cadena);

            Console.WriteLine("Cadena de búsqueda: " + cadena);
            Console.WriteLine("------Valores encontrados---------");
            if (hayCoincidencia)
                foreach (Match coincidencia in Regex.Matches(cadena, patron))
                    Console.WriteLine(coincidencia.Value);
            else
                Console.WriteLine("No se encontraron coincidencias");

            Console.WriteLine("-----------------------------------");
        }
    }
}
```

VB

```
Imports System.Text.RegularExpressions

Module Module1

    Sub Main()
        Console.WriteLine("Buscando con ^\\s?\\d{4}-?\\d{4}$")
        BuscarCoincidencias("4567-8910", "^\s?\d{4}-?\d{4}$")
        BuscarCoincidencias(" 4567-8910", "^\s?\d{4}-?\d{4}$")
        BuscarCoincidencias("  4567-8910", "^\s?\d{4}-?\d{4}$")

        Console.WriteLine("Buscando con ^\\S?\\d{4}-?\\d{4}$")
        BuscarCoincidencias("4567-8910", "^\S?\d{4}-?\d{4}$")
        BuscarCoincidencias(" 4567-8910", "^\S?\d{4}-?\d{4}$")
        BuscarCoincidencias("  4567-8910", "^\S?\d{4}-?\d{4}$")
        Console.ReadKey()

    End Sub

    Sub BuscarCoincidencias(cadena As String, patron As String)
        Dim expRegular As New Regex(patron)
        Dim hayCoincidencia As Boolean = expRegular.IsMatch(cadena)

        Console.WriteLine("Cadena de búsqueda: " + cadena)
        Console.WriteLine("------Valores encontrados---------")
        If hayCoincidencia Then
            For Each coincidencia As Match In Regex.Matches(cadena, patron)
                Console.WriteLine(coincidencia.Value)
            Next
```

```
        Else
            Console.WriteLine("No se encontraron coincidencias")
        End If
        Console.WriteLine("---------------------------------")
    End Sub
End Module
```

La salida obtenida es:

```
Buscando con ^\s?\d{4}-?\d{4}$
Cadena de búsqueda: 4567-8910
------Valores encontrados---------
4567-8910
---------------------------------
Cadena de búsqueda:  4567-8910
------Valores encontrados---------
 4567-8910
---------------------------------
Cadena de búsqueda:   4567-8910
------Valores encontrados---------
No se encontraron coincidencias
---------------------------------
Buscando con ^\S?\d{4}-?\d{4}$
Cadena de búsqueda: 4567-8910
------Valores encontrados---------
4567-8910
---------------------------------
Cadena de búsqueda:  4567-8910
------Valores encontrados---------
No se encontraron coincidencias
---------------------------------
Cadena de búsqueda:   4567-8910
------Valores encontrados---------
No se encontraron coincidencias
---------------------------------
```


Cuando se utilizan expresiones regulares, puede ser difícil de entender a los caracteres que se utilizan para que coincidan con la cadena. Esto se complica aún más por el hecho que muchos de estos caracteres son contextuales y su interpretación depende de dónde se encuentren en la cadena del patrón y lo que les precede.

Encontrando patrones en cadenas con expresiones regulares

El uso más simple de las expresiones regulares es por coincidencia de patrones. Por ejemplo, puede crear una expresión regular utilizando como patrón la cadena "abc", que se correspondería con cadenas como "abc", "abcde", y "yzabc" porque todas ellas contienen la cadena de literales “abc”.

Para realizar la comparación de patrones, se puede utilizar los métodos IsMatch, Match o Matches de la clase Regex. El método estático Regex.IsMatch acepta la cadena patrón de expresión regular y la cadena en la que se busca la coincidencia, retornando un valor booleano que indica si el motor de expresiones regulares ha encontrado una coincidencia.

El método Match busca una cadena para una secuencia de caracteres que coinciden con una expresión regular. Se devuelve un objeto System.Text.RegularExpressions.Match que indica si la coincidencia ha tenido éxito, y, si es así, un referencia a la secuencia correspondiente de caracteres en la cadena que en la que se está buscando. Si la cadena contiene múltiples secuencias de coincidencias, el método Match devuelve la primera coincidencia. Sin embargo, se puede llamar al método NextMatch del objeto Match para regresar otro objeto del tipo Match que contiene una referencia a la siguiente secuencia de caracteres coincidentes y este proceso se puede repetir para localizar todas las coincidencias encontradas.

El método Matches es similar al método Match, excepto que retorna una referencia a una colección con todas las secuencias coincidentes de la cadena original, en caso que hubiera una o más coincidencias. Es funcionalmente equivalente a llamar al método Match para retornar la primera coincidencia, seguido por repetidas invocaciones del método NextMatch paras recuperar todas las coincidencias halladas.

Métodos estáticos y de instancia de las expresiones regulares

Los métodos IsMatch, Match y Matches están disponibles como métodos estáticos y de instancia.

Las versiones estáticas de estos métodos son ideales si sólo se tiene que realizar una búsqueda rápida de una cadena. Estas implementaciones esperan que se les suministre como parámetros la cadena donde se buscarán las coincidencias mediante la expresión regular y el patrón que define a la misma.

Las versiones de instancia de estos métodos exigen que se cree una instancia de la clase Regex y se proporciona la expresión regular como un parámetro del constructor. A continuación, se puede realizar búsquedas repetidas en varias cadenas respecto del patrón con el cual se creó la expresión regular.

El siguiente ejemplo de código muestra cómo utilizar la versión estática del método IsMatch para realizar la búsqueda de coincidencias según el patrón definido en el segundo argumento.

Ejemplo

C#

```
using System;
using System.Text.RegularExpressions;

namespace estatica
```

```
{
    class Program
    {
        static void Main(string[] args)
        {
            if (Regex.IsMatch("192.168.100.1", @"\d{1,3}"))
            {
                MatchCollection mc = Regex.Matches("192.168.100.1", @"\d{1,3}");
                foreach (Match coincidencia in mc)
                    Console.WriteLine(coincidencia.Value);
            }
            else
            {
                Console.WriteLine("No se encontraron coincidencias");
            }
            Console.ReadKey();
        }
    }
}
```

VB

```
Imports System.Text.RegularExpressions

Module Module1

    Sub Main()
        If Regex.IsMatch("192.168.100.1", "\d{1,3}") Then
            Dim mc As MatchCollection = Regex.Matches("192.168.100.1", "\d{1,3}")
            For Each coincidencia As Match In mc
                Console.WriteLine(coincidencia.Value)
            Next
        Else
            Console.WriteLine("No se encontraron coincidencias")
        End If

        Console.ReadKey()
    End Sub
End Module
```

La salida obtenida es:

```
192
168
100
1
```


Extracción de datos usando expresiones regulares

La gestión de cadenas cuando se analiza el texto que contiene incluye la extracción y manipulación de datos contenidas en ellas. Sin embargo, cuando se trabajan los distintos caracteres que las componen como grupo de literales independientes o cuando se comparan diferentes sub cadenas surgen numerosos problemas algorítmicos. Por ejemplo, la longitud de lo que se busca, las

coincidencias que deben ser consideradas como iguales o diferentes más allá del valor del literal o grupo de literales analizados, el comienzo o la finalización de la cadena a analizar, la reconstitución de una nueva cadena a partir de la anterior, etc.

Cuando se analizan cadenas también es probable que se quieran agrupar coincidencias dependiendo de uno o varios patrones. Más aún, es normal agrupar dichos patrones como uno solo que contenga a los demás como sub patrones. Las expresiones regulares permiten este tipo de gestión en .Net a través de la colección Match.Groups. Para extraer los datos, se pueden acceder los miembros de ésta colección que fueron generados como parte del análisis que la expresión regular realizó en base al patrón que le fuera definido. Estos miembros están indexados para facilitar su acceso, lo cual por las propiedades del indexador, permite el acceso a los elementos en base al valor de un número o una clave. Para poder acceder por clave es interesante tener en cuenta que las expresiones regulares permiten asignar un nombre a cada grupo luego utilizarlo para acceder. Esto se logra incluyendo el valor de la clave (el nombre asignado) con el siguiente formato: **(?<*nombre*>)**. Ese mismo nombre es utilizado posteriormente para el acceso, como se lo hace con cualquier indexador, aplicándolo a la colección Groups.

Anteriormente se explicó como agrupar caracteres mediante el uso se los paréntesis. Cada uno de dichos paréntesis dentro de un patrón se puede considerar un grupo y se lo puede nombrar para luego accederlo. O sea, el cambio que se agrega respecto del concepto anterior de agrupamiento es tan sólo incluir el nombre.

El siguiente ejemplo muestra la utilización de grupos nombrados en el patrón definido para una expresión regular.

Ejemplo

C#

```
using System;
using System.Text.RegularExpressions;

namespace grupos
{
    class Program
    {
        static void Main(string[] args)
        {
            string patron =
@"(?<palabraDuplicada>\w+)\s\k<palabraDuplicada>\W(?<proximaPalabra>\w+)";

            string cadena = "Todo lo que que quiero es una una respuesta correcta.";

            Console.WriteLine("Analizando la cadena: " + cadena);
            Console.WriteLine("--------------------------------------");

            foreach (Match coincidencia in Regex.Matches(
```

```
                    cadena, patron, RegexOptions.IgnoreCase))
            {
                Console.WriteLine("Se encuentra duplicada la palabra: '{0}'.",
                                  coincidencia.Groups["palabraDuplicada"].Value);
                Console.WriteLine("Primera aparición a partir del literal: {0}.",
                                  coincidencia.Groups["palabraDuplicada"].Index + 1);
                Console.WriteLine("Palabra a continuación: '{0}'.",
                      coincidencia.Groups["proximaPalabra"].Value);

                Console.WriteLine();
            }

            Console.ReadKey();
        }
    }
}
```

VB

```
Imports System.Text.RegularExpressions

Module Module1

    Sub Main()
        Dim patron As String =
"(?<palabraDuplicada>\w+)\s\k<palabraDuplicada>\W(?<proximaPalabra>\w+)"

        Dim cadena As String = _
              "Todo lo que que quiero es una una respuesta correcta."

        Console.WriteLine("Analizando la cadena: " + cadena)
        Console.WriteLine("-------------------------------------")

        For Each coincidencia As Match In Regex.Matches( _
              cadena, patron, RegexOptions.IgnoreCase)
            Console.WriteLine("Se encuentra duplicada la palabra: '{0}'.",
                              coincidencia.Groups("palabraDuplicada").Value)
            Console.WriteLine("Primera aparición a partir del literal: {0}.",
                              coincidencia.Groups("palabraDuplicada").Index + 1)
            Console.WriteLine("Palabra a continuación: '{0}'.",
                  coincidencia.Groups("proximaPalabra").Value)

            Console.WriteLine()
        Next

        Console.ReadKey()
    End Sub
End Module
```

La salida obtenida es:

```
Analizando la cadena: Todo lo que que quiero es una una respuesta correcta.
-------------------------------------
Se encuentra duplicada la palabra: 'que'.
Primera aparición a partir del literal: 9.
```

```
Palabra a continuación: 'quiero'.

Se encuentra duplicada la palabra: 'una'.
Primera aparición a partir del literal: 27.

Palabra a continuación: 'respuesta'.
```

La expresión regular utilizada en el ejemplo fue:

```
(?<palabraDuplicada>\w+)\s\k<palabraDuplicada>\W(?<proximaPalabra>\w+)
```

La siguiente tabla muestra como debe ser interpretada:

Modelo	Descripción
(?<palabraDuplicada>\w+)	Busca coincidencias con uno o más caracteres alfabéticos. Asigna a este grupo de capturas el nombre *palabraDuplicada*.
\s	Busca coincidencia con un caracter que sea un espacio en blanco.
\k<palabraDuplicada>	Busque la cadena del grupo capturado denominada *palabraDuplicada*.
\W	Busca coincidencia con un caracter que no sea palabra, espacios en blanco y puntuación incluidos. Esto evita que el modelo de expresión regular coincida con una palabra que comience con la palabra del primer grupo capturado.
(?<proximaPalabra>\w+)	Busca coincidencias con uno o más caracteres alfabéticos. Asigna a este grupo de capturas el nombre *proximaPalabra*.