Curso de Programación .NET nivel I

Unidad 5

“Tipos de Datos complejos”

Índice

Contenido

Fecha y hora....3
Explicación....3
Declaración....3
Conversiones....4
Otras Conversiones útiles....5
El Método Now() y UtcNow()....5
Métodos Arítmeticos de fecha y hora....6
Funciones con Strings....8
Structs....10
Explicación....10
Declaración....11
Utilización en la práctica....11
Arrays, ArrayLists y Lists....11
Arrays....11
Implementación....11
ArrayLists....12
Implementación....12
List....13
Implementación....13
Comparación....13

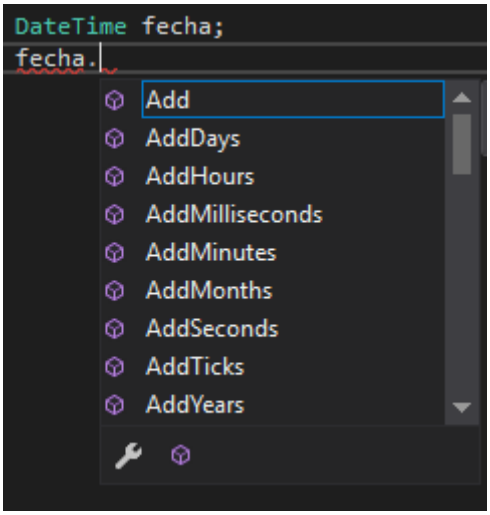
Fecha y hora

Explicación

Muchos sistemas que utilizamos hoy en día se valen de la fecha y hora para funcionar. Están presentes desde que nos levantamos de la cama como el caso de la aplicación despertador, que compara la fecha actual contra la de la alarma para despertarnos, o una aplicación de reporte que tiene un filtro de fecha para ir y comparar cada registro y traer solamente los que correspondan a la selección.

Una fecha se puede representar con un número entero como por ejemplo: 20181102180000. Luego para se pueden realizar ciertas funciones matemáticas para descomponerlo y operar sobre él.

Por suerte para evitar reinventar la rueda .NET ya viene con un tipo de dato entero con estas funciones ya disponibles:

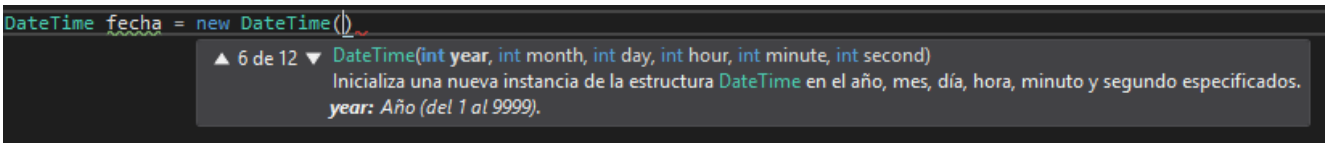


Declaración

Es muy simple se declara como cualquier variable:

```
DateTime fecha;
```

La fecha se puede inicializar con un valor específico:



por ejemplo las 18 horas del primero de febrero del 2019:

```
DateTime fecha = new DateTime(2019, 2, 1, 18, 0, 0);
```

DateTime también permite la definición de fecha sin hora en el caso que lo necesite:

```
DateTime fecha = new DateTime(2019, 02, 01);
```

Conversiones

Veamos ahora una las ventajas de tener la fecha en este tipo especial: las conversiones a **String** con formato personalizable por medio de la función `ToString()`.

`ToString()` me permite recibir un formato de fecha y hacer su conversión a texto con los valores de mi `DateTime`. Ilustremos esto con un ejemplo:

```
   fecha.ToString("yyyy/MM/dd hh:mm:ss");
//esto me devuelve: 2019/02/01 18:00:00
```

- **yyyy** representa al año en 4 dígitos
- **MM** representa al mes en 2 dígitos
- **dd** representa al día en 2 dígitos
- **HH** a las horas en formato 00-24
- **mm** a los minutos en formato de 2 dígitos
- **ss** a los segundos en formato de 2 dígitos

Básicamente al repetir el caracter de formato establezco el mínimo de dígitos para representarlo.

por ejemplo: al determinar **mm** me aseguro que los minutos del 0 al 9 se les complete con otro 0 adicional por delante.

Vamos con un ejemplo práctico para que esto quede mas claro:

```
fecha.ToString("yyyy/MM/dd hh:mm:ss");
// esto me devuelve: 2019/02/01 18:00:ss

fecha.ToString("yyyy/M/dd");
// esto me devuelve: 2019/2/01

fecha.ToString("yyyy/M/d");
// esto me devuelve: 2019/2/1

fecha.ToString("yy/M/d");
// esto me devuelve: 19/2/1
```

Otro caso de conversión especial muy útil es entre horario **0-24** y horario **PM** y **AM.**

- **H**: me devuelve el numero de la hora en formato **0-24**
- **h**: me devuelve el numero de la hora en formato **1-12**
- **tt**: para complementar con **h**, representa **PM** y **AM**.

```
fecha.ToString("HH:mm");
       // esto me devuelve: 18:00

fecha.ToString("HH:mm tt");
       // esto me devuelve: 06:00 PM
```

Otras Conversiones útiles

Una fecha tiene disponibles los siguientes métodos de conversión rápidos bastante útiles:

ToLongDateString()	Domingo, 11 de noviembre de 2018
ToLongTimeString()	12:16:59 PM
ToShortDateString()	11/11/2018
ToShortTimeString()	12:16 PM
ToString()	11/11/2018 12:16:59 PM

El Método Now() y UtcNow()

Estos métodos devuelve la fecha y hora actual. El método now() devuelve la hora local o la que veo en el reloj de Windows, mientras que el método UtcNow() devuelve la hora local expresado en formato de tiempo universal coordinado.

En Argentina nuestra hora local se expresa en UTC-3, por lo tanto si son las 2 de la tarde, el primero me devuelve 14:00:00 y el segundo me devuelve 17:00:00 la cual es la hora universal.

```
DateTime now = DateTime.Now();

DateTime utc = DateTime.UtcNow();
```

Esto es crucial cuando el sistema se va a utilizar en distintos países, en esos casos recomiendo siempre almacenar la información en UTC y luego hacer la conversión para el huso horario correspondiente, esto lo ven más adelante en “Timezones”.

Métodos Arítmeticos de fecha y hora

`.AddYears(double value)`	Suma a una fecha una determinada cantidad de años
`.AddMonths(double value)`	Suma a una fecha una determinada cantidad de meses
`.AddDays(double value)`	Suma a una fecha una determinada cantidad de días
`.AddHours(double value)`	Suma a una fecha una determinada cantidad de horas
`.AddMinutes(double value)`	Suma a una fecha una determinada cantidad de minutos
`.AddSeconds(double value)Suma a una fecha una determinada cantidad de`	Suma a una fecha una determinada cantidad de segundos
.Add(Datetime value)	Suma a una fechahora otra fechahora y devuelve el total.
.Substract(Datetime value)	Suma a una fechahora otra fechahora y devuelve la diferencia.

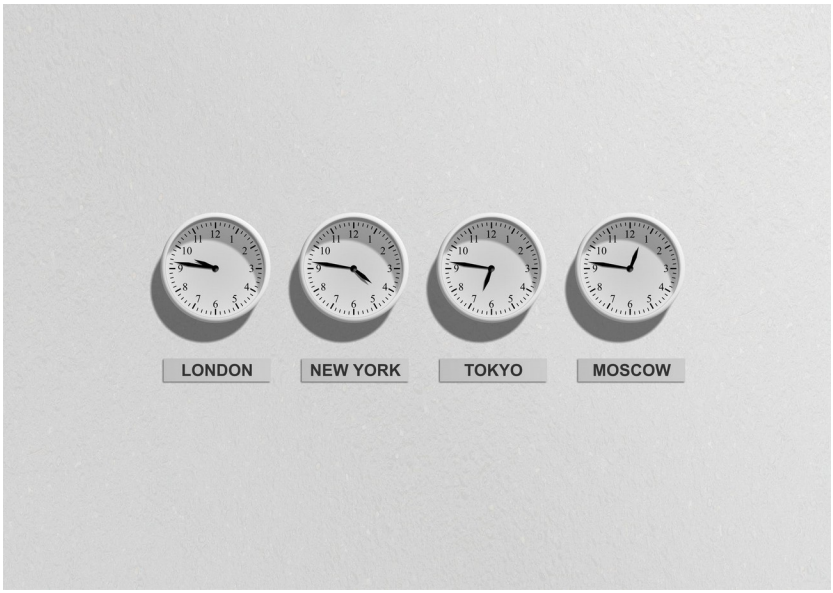
Ejemplo:

```
    //Agregar un dia a hoy
            Console.WriteLine();

            DateTime mañana = now.AddDays(1);

            Console.WriteLine("Hoy es: " + now.ToLongDateString() + ", Mañana es: " +
mañana.ToLongDateString());
```

TimeZones

Como sabemos alrededor del mundo no todos tenemos la misma hora, cada país tiene su hora local dependiendo en que huso horario se encuentre. Este valor es el que me devuelve Now, la hora local.

Todas estas horas locales se calculan en base a la hora de Greenwich en Inglaterra. En Argentina nuestra hora local esta atrasada tres horas de Greenwich. Este horario de Greenwich también es conocido por hora UTC.



Para convertir un horario UTC en el horario de región que queramos .NET viene con una función muy interesante la cual recibe un horario en UTC y lo transforma en la zona horaria que queramos.

```
TimeZoneInfo.ConvertTimeFromUtc(DateTimeUTC, ZonaHoraria);
```

La Zona horaria se obtiene con otra función:

```
TimeZoneInfo ZonaHoraria = TimeZoneInfo.FindSystemTimeZoneById("Argentina Standard Time");
```

Para una lista completa de nombre de timezone para usar en la función, pueden ir a esta web
http://remy.supertext.ch/2012/04/the-net-timezoneinfo-getsystemtimezones-list/

ID Argentina Standard Time
Display Name (UTC-03:00) Buenos Aires
Standard Name Argentina Standard Time
Daylight Name Argentina Daylight Time
Has Daylight Saving Time
Offset from UTC -3 hours, 0 minutes
Adjustment rules 3
From 01.01.2007 00:00:00 to 31.12.2007 00:00:00
Delta: 01:00:00
Begins at 00:00 on Sunday of week 5 of Dezember
Ends at 00:00 on Monday of week 1 of Januar
From 01.01.2008 00:00:00 to 31.12.2008 00:00:00
Delta: 01:00:00
Begins at 23:59 on Saturday of week 3 of Oktober
Ends at 00:00 on Sunday of week 3 of März
From 01.01.2009 00:00:00 to 31.12.2009 00:00:00
Delta: 01:00:00
Begins at 00:00 on Thursday of week 1 of Januar
Ends at 23:59 on Saturday of week 2 of März

Funciones con Strings

Veamos ahora algunas funciones interesantes para usar con cadenas que siempre son muy útiles cuando estamos trabajando con ellas. Estas funciones son en realidad métodos que aceptan todos los objetos de tipo String.

- **Length**
  Esta propiedad devuelve la cantidad de caracteres que contiene un string.

- **contains(String valor)**
  Esta función devuelve true o false dependiendo si una subcadena esta presente en la cadena principal.

- **Replace(string viejoValor, string nuevoValor)**
  Esta función busca en el string un valor y lo reemplaza por uno nuevo, luego devuelve esa cadena sin modificar la original.

- **IndexOf(string valor)**
  Esta función devuelve la posición que se encuentra el valor ingresado.

- **ToUpper()**
  Devuelve una copia de la cadena transformada a mayúscula.

- **ToLower()**
  Devuelve una copia de la cadena transformada a minúscula.

- **Substring(int desde, hasta)**
  Devuelve una porción de un string indicando la posición de corte inicial y la posición final, si esta última no se especifica se usa el Lenght() y se corta hasta el final.

- **Split(string separador)**
  Esta función corta a mi string en partes utilizando una cadena de texto como separadora, Luego devuelve el resultado en un grupo de strings llamado array de strings.

Ejemplos:

```
String texto = "Me cuesta ponerme creativo en estos textos";

            //Tamaño total del texto
            Console.WriteLine(texto.Length);
            //42

            //Contiene la palabra "ponerme"?
            Console.WriteLine(texto.Contains("ponerme"));
            //True

            //reemplazar la palabra "creativo" por "original".
            Console.WriteLine(texto.Replace("creativo","original"));
            //Me cuesta ponerme original en estos textos

            //En que posicion está "cuesta"?
            Console.WriteLine(texto.IndexOf("cuesta"));
            //3

            //Convertir todo a Mayusculas y luego todo a Minusculas
            Console.WriteLine(texto.ToUpper());
            //"ME CUESTA PONERME CREATIVO EN ESTOS TEXTOS
            Console.WriteLine(texto.ToLower());
            //me cuesta ponerme original en estos textos


            //Separar en substrings todo lo que este entre espacios.
            //para mas información revisar arrays en el siguiente tema.
            Console.WriteLine(texto.Substring(1, 10));
            String[] textoEntreEspacios = texto.Split(' ');
            Console.WriteLine(textoEntreEspacios[0]);
            //Me
            Console.WriteLine(textoEntreEspacios[3]);
            //creativo
```

Structs

Explicación

El tipo de dato **struct** es un tipo que se utiliza para encapsular un pequeño grupo de variables que estén relacionadas, como pueden ser los vértices de un cuadrado. Naturalmente se pueden confundir con las clases ya que tiene muchas similitudes como ser: constructores, métodos, propiedades, operadores etc.

Declaración

```
public struct Book
{
   public decimal price;
   public string title;
   public string author;
}
```

Utilización en la práctica

La recomendación mía y también de Microsoft es utilizarlos simplemente como agrupador de variables, cuando ya se necesiten constructores, métodos y operaciones utilizar clases.

Arrays, ArrayLists y Lists

Arrays

Los arrays son estructuras que permiten almacenar varios datos del mismo tipo de manera conjunta o dicho de otro modo quiero agrupar varias variables del mismo tipo y tratarlas todas juntas.

0	“River”	String
1	“Boca”	String
2	“Racing”	String
3	“Velez”	String
4	“Independiente”	String
5	San Lorenzo	String

En el diagrama de arriba vemos un conjunto de datos tipo **String** que representan a equipos de fútbol, sobre ellos armamos un array y cada elemento del array tiene una posición denotada por un numero de índice.

Gracias a este índice puedo acceder a los elementos del array y leerlos o modificarlos de manera secuencial o no.

Implementación

La implementación es muy similar a otras implementaciones en C#, la gran diferencia son los corchetes. En el momento que agregamos esos corchetes estamos definiendo un conjunto y no una sola variable.

```
String[] listaDeEquipos = new String[6];
```

El numero del final indica cuantos elementos va a tener mi array, los cuales son fijos y no se pueden modificar una vez declarado. **ArrayList** viene a solucionar esa limitación.

Para acceder a los objetos tengo que utilizar ese índice, por ejemplo para obtener el string **"Independiente"**:

```
listaDeEquipos[4];
```

ArrayLists

La principal característica de los arrayLists es que no tengo que conocer el numero de elementos a priori para declararlo, de hecho van a ser dinámicos puedo ir agregando y eliminando elementos.

Otra característica del ArrayList es que permite almacenar distintos tipos:

0	"River"	Object
1	"Boca"	Object
2	22	Object
3	11/11/2018	Object

Como ArrayList no entiende de tipos almacena todo como tipo "Object" que es el padre de todos los tipos.

Implementación

Para declararlo:

```
ArrayList listaDeEquiposEnArrayList = new ArrayList();
```

Para agregar objetos

```
listaDeEquiposEnArrayList.Add("River");
```

Para accederlos requiero un paso adicional:

```
String elementoString = (String) listaDeEquiposEnArrayList[1];
DateTime elementoDate = (DateTime) listaDeEquiposEnArrayList[3];
```

Este paso adicional es el **Casting** y me permite etiquetar un tipo Object en el tipo que le corresponde, en este caso ya se de antemano que tipo de datos contiene, lo complicado es cuando no lo sé. Para evitar esta bolsa de objetos mezclados se utiliza **List.**

List

List funciona de la misma manera que el ArrayList, con la diferencia que le tengo que definir un tipo. Como el ArrayList no requiere que le definan su tamaño antes por lo tanto permite agregar y quitar elementos a demanda.

Implementación

Declaracion:

```
List<String> listaDeEquiposEnList = new List<String>();
```

Para agregar objetos:

```
listaDeEquiposEnArrayList.Add("River");
```

Para accederlos no requiero casting ya **List** tiene definido un tipo:

```
String elemento = listaDeEquiposEnList[2];
```

Comparación

Array	**ArrayList**	**List**
Tengo que definir la cantidad de elementos antes de usarlo.	No hace falta definir la cantidad de elementos antes de usarlo.	No hace falta definir la cantidad de elementos antes de usarlo.
No permite agregar nuevos elementos	Permite agregar nuevos elementos	Permite agregar nuevos elementos
Es Type-Safe	No es Type-Safe, permite almacenar diferentes tipos en un mismo ArrayList.	Es Type-Safe
Es el más eficiente en el uso de recursos.	Consume más recursos que el Array	Consume más recursos que el Array