La Güeb de Joaquín
Mi sitio de Internet

Interfaz IEquatable

[Descripción general]

En muchas ocasiones tenemos que saber si dos objetos tienen el mismo valor. Para estos casos está diseñada la interfaz IEquatable.
Microsoft recomienda utilizar esta interfaz (preferiblemente) en los tipos por valor, porque el objeto de esta interfaz es determinar si ***dos objetos tienen el mismo valor*** (no la misma referencia).
.NET Framework define y utiliza docenas de interfaces y los programadores expertos en Visual Basic .NET deben aprender a sacar partido de todas ellas

Contenido

[Descripción general] 1
Introducción 2
Las famosas referencias a objetos 2
- Clase para ver el ejemplo 3
Implementando la Interfaz 4
La Interfaz 4
Firma de la interfaz 4
Firma de las funciones de la interfaz 5
Implementando la Interfaz 5
Paso Uno – Definir la interfaz 5
Paso Dos - Implementar la función de la interfaz 5
Paso Tres – Sombrear Object.Equals 5
Paso Cuatro – Implementar Operadores [=], y [<>] 6
Paso Cinco – Sombrear la función Object. GetHashCode 7
A modo de resumen 8
Más información: 10
¿Qué es un número Racional? 10
Código de ejemplo 11
Bibliografía 11
Información de este Documento 11

Introducción

Las famosas referencias a objetos

En .NET todo son objetos, pero existen dos tipos de objetos: los tipos por valor y los tipos por referencia.

- Los tipos por valor son -en general-
  - los números definidos en el sistema (Por ejemplo: Boolean, SByte, Byte, Int16, UInt16, Integer, UInt32, Int64, UInt64, Single, Double, Decimal, DateTime, Char y String,
  - Los tipos numéricos definidos por el usuario, como las estructuras.
- Los tipos por referencia son cualquier otro objeto con el que se trabaje en .NET, incluidos los que definamos nosotros

Cuando se asigna una variable a otra,

```
Objeto2 = Objeto1
```

Lo que ocurre es lo siguiente:

- En los tipos por valor –en general los números - se obtienen dos variables (diferentes) y cada una de ellas apunta a un sitio diferente de la memoria. Es decir, obtengo dos juegos de datos diferentes. (Aunque los valores de cada juego son los mismos)
- En los tipos por referencia, - el resto de los objetos- se obtienen dos variables (diferentes) que apuntan a la misma zona de la memoria, es decir, estamos viendo el mismo juego de datos (el mismo objeto) desde dos variables diferentes

Para determinar la igualdad de instancias La interfaz IEquatable, define el método [Equals]. Dicho de otro modo, tenemos que escribir una función llamada [Equals] que nos diga lo siguiente:

- Si trabajamos con tipos de valor (números) el método Equals nos dirá ***si las cantidades son iguales o no***, ya que por definición cuando hacemos una asignación de variables se obtienen dos variables diferentes. Para obtener el resultado solo tenemos que utilizar el operador de Igualdad (=) para comparar los valores numéricos
- Si trabajamos con tipos de referencia, el método Equals básico de la clase System.Object, heredada por todos los objetos, nos devuelve directamente un valor que indica ***si los objetos son la mima instancia o no***, es decir, en principio no hay que implementar esta interfaz. El problema surge cuando hay tipos de referencia que contienen implícitamente tipo de valor, por ejemplo una clase que implementa un numero complejo, Aunque es un tipo de referencia, se debería implementar la interfaz, porque estamos tratando en realidad con datos numéricos.

NOTA
Para aquellos casos en los que queramos asegurarnos que estamos trabajando con la misma instancia, independientemente del tipo de objeto que sea (valor o referencia) la clase System.Object, dispone de un método, cuya firma se expone a continuación, que devuelve un valor que indica si **dos instancias** son iguales o no Es decir, si las variables apuntan a la misma dirección de memoria o no.

```
Public Shared Function ReferenceEquals (Object, Object) As Boolean
```

- Clase para ver el ejemplo

```
''' <summary>
'''    Estructura que representa a un numero racional
''' </summary>
''' <remarks>
'''   Un número racional es un número representado por el cociente
'''    de dos números enteros (lo que normalmente llamamos quebrado),
'''   como 5/7. El nuero de la izquierda se denomina numerador y el
'''   de la derecha denominador
''' </remarks>
<Serializable()> _
Public Structure NumeroRacional
    Implements IEquatable(Of NumeroRacional)


#Region "Campos"
    '-------------------------------------------------------------
    ' La interfaz de usuario del Numero Racional debe tratar al
    ' número como un objeto indivisible, esto es, el usuario
    ' no tiene que tener la posibilidad de acceder a los campos
    ' numerador y denominador de forma independiente
    '-------------------------------------------------------------
    Private _numerador As Long
    Private _denominador As Long
#End Region


#Region "Constructores"

    '''' <summary>
    ''''  Constructor sin argumentos (valor por defecto 0/1)
    '''' </summary>
    'Shared Sub New()
    '     _numerador = 0L
    '     _denominador = 1L
    'End Sub

   ''' <summary>
    '''   Costructor con argumentos
    ''' </summary>
    ''' <param name="numerador">El numerador del numero racional</param>
    ''' <param name="denominador">El denominador del numero racional</param>
    ''' <remarks>
    ''' Las operaciones que debe realizar el constructor son:
    ''' Si el denominador es cero, forzar a que sea 1.
    ''' Si el denominador es negativo, invertiremos el signo del
    ''' numerador y del denominador.
    ''' Simplificará la fracción siempre que sea posible
    ''' </remarks>
    Public Sub New(ByVal numerador As Long, ByVal denominador As Long)
        Me._numerador = numerador
        Me._denominador = denominador

        If Me._denominador = 0 Then
            Me._denominador = 1
        End If

        If Me._denominador < 0 Then
            Me._numerador = -Me._numerador
            Me._denominador = -Me._denominador
        End If

        Call Simplificar()
    End Sub
```

```
    ''' <summary>
    ''' La función simplificar utiliza el algoritmo de Euclides para
    ''' obtener el máximo común divisor (mcd) del numerador y denominador,
    ''' y simplifica el numero racional dividiendo ambos números por su mcd.
    ''' </summary>
    ''' <returns>El numero racional simplificado</returns>
    Public Function Simplificar() As NumeroRacional

        ' Calculo del máximo común divisor (mcd)
        Dim mcd As Long = 0L
        Dim temp As Long = 0L
        Dim resto As Long = 0L
        mcd = Math.Abs(Me._numerador)
        temp = Math.Abs(Me._denominador)

        Do While (temp > 0)
            resto = mcd Mod temp
            mcd = temp
            temp = resto
        Loop

        'Simplificar
        If mcd > 1 Then
            Me._numerador = CType(Me._numerador / mcd, Long)
            Me._denominador = CType(Me._denominador / mcd, Long)
        End If

        ' devolver el valor obtenido
        Return Me
    End Function

    ''' <summary>
    '''    Constructor de número Entero
    ''' </summary>
    ''' <param name="numerador">Un numero entero</param>
    Public Sub New(ByVal numerador As Long)
        Me._numerador = numerador
        Me._denominador = 1L
    End Sub


    ''' <summary>
    '''   Constructor copia
    ''' </summary>
    ''' <param name="unNumeroRaciona">
    ''' El numero racional que se recibe por parametro cuyos valores se copian
    ''' </param>
    Public Sub New(ByVal unNumeroRaciona As NumeroRacional)
        Me._numerador = unNumeroRaciona._numerador
        Me._denominador = unNumeroRaciona._denominador
    End Sub

#End Region
End Structure
```

Implementando la Interfaz

La Interfaz

Firma de la interfaz

La interfaz es una interfaz genérica cuya firma es:

```
    Public Interface IEquatable (Of T)
```

Firma de las funciones de la interfaz

Y la función que hay que implementar tiene la firma

```
    Public Overloads Function Equals(ByVal d As T) As Boolean _
                Implements System.IEquatable(Of T).Equals
```

Implementando la Interfaz

Paso Uno – Definir la interfaz

Implementaremos la interfaz en nuestra estructura

```
Public Structure NumeroRacional
    Implements IEquatable(Of NumeroRacional)
        ...
        ...
End Structure
```

Paso Dos - Implementar la función de la interfaz

Escribimos la función que implementa la interfaz, observa que lo único que hace es llamar a otra función para que haga el trabajo real de comparación

```
    '-----------------------------------------------------------------
    'La interfaz IEquatable
    'http://msdn.microsoft.com/es-es/library/ms131190.aspx
    '-----------------------------------------------------------------
  Public Overloads Function Equals(ByVal other As NumeroRacional) As Boolean _
                Implements System.IEquatable(Of NumeroRacional).Equals
        Return Equals(Me, other)
    End Function
```

Paso Tres – Sombrear Object.Equals

Tengo que "Sombrear" las funciones de [Object.Equals] [] cuyas firmas son las siguientes:

```
Public Shared Function Equals (Object, Object) As Boolean
Public Overridable Function Equals (Object) As Boolean
```

Para ello escribo estas dos funciones

La primera función:

```
    '-----------------------------------------------------------------
    '  Sombreado Object.Equals
    '  http://msdn.microsoft.com/es-es/library/system.object.equals.aspx
    '-----------------------------------------------------------------
   Public Overloads Shared Function Equals( _
                    ByVal nr1 As NumeroRacional, _
                    ByVal nr2 As NumeroRacional) _
                    As Boolean
        '-------------------------------------------------------
        ' Aquí es donde se hace TODO el trabajo de comparación
        '-------------------------------------------------------
        Dim resultado As Boolean = False
        If nr1._numerador = nr2._numerador AndAlso _
           nr1._denominador = nr2._denominador Then
            resultado = True
        End If
        Return resultado
    End Function
```

- Está definida overloads, porque estoy sobrecargando la función [Equals]
- Esta es la función que hace el trabajo real de comparar los dos objetos, para ello compara su VALOR utilizando el operador de igualdad (=). Si las dos cantidades son iguales devuelve [True], en caso contrario devuelve [False]
- El proceso de comparación se hace en un solo punto del código y todo el que necesite realizar una comparación lo llama

La segunda función:

```
'-------------------------------------------------------------------
'  Sombreado Object.Equals
'  http://msdn.microsoft.com/es-es/library/system.object.equals.aspx
'-------------------------------------------------------------------
   Public Overloads Overrides Function Equals(ByVal obj As Object) As Boolean
        'convertirlo a estructura NumeroRacional
        Dim unNumeroRacional As NumeroRacional
        unNumeroRacional = CType(obj, NumeroRacional)

        'realizar la comparacion
        Return Equals(Me, unNumeroRacional)
    End Function
```

- Está definida overloads, porque estoy sobrecargando la función [Equals]
- Está definida overrides porque reemplaza la implementación predeterminada de la clase base, en este caso System.Object.Equals.
- La función lo primero que hace es convertir el objeto genérico [Object] a un Objeto [NumeroRacional] y después llamar a la función [Equals] que hace la comparación.

Paso Cuatro – Implementar Operadores [=], y [<>]

La biblioteca MSDN en su entrada: Capítulo 9: Instrucciones para implementar Equals y el operador de igualdad (==) [http://msdn.microsoft.com/es-es/library/7h9bszxx.aspx] dice lo siguiente (este texto esta resumido del original):

- Implemente el método GetHashCode siempre que implemente el método Equals. De este modo, los métodos Equals y GetHashCode se mantienen sincronizados.
- Reemplace el método Equals siempre que implemente el operador (==), y haga que los dos realicen las mismas acciones.
- Implemente el operador (==) siempre que reemplace el método Equals.
- No inicie excepciones desde los métodos Equals o GetHashCode ni desde el operador de igualdad (==).

La implementación del operador = será la siguiente

```
    '-------------------------------------------------------------------
    'Instrucciones para implementar Equals y el operador de igualdad (==)
    'http://msdn.microsoft.com/es-es/library/7h9bszxx.aspx
    '-------------------------------------------------------------------
Public Overloads Shared Operator =( _
                                    ByVal nr1 As NumeroRacional, _
                                    ByVal nr2 As NumeroRacional) _
                                    As Boolean
        Return Equals(nr1, nr2)
    End Operator
```

La implementación del operador <> será la siguiente

```
'-----------------------------------------------------------------
'Instrucciones para implementar Equals y el operador de igualdad (==)
'http://msdn.microsoft.com/es-es/library/7h9bszxx.aspx
'-----------------------------------------------------------------
Public Overloads Shared Operator <>( _
                        ByVal nr1 As NumeroRacional, _
                        ByVal nr2 As NumeroRacional) _
                        As Boolean
    Return Not (Equals(nr1, nr2))
End Operator
```

Paso Cinco – Sombrear la función Object. GetHashCode

Dos normas básicas que supongo que la mayoría ya conoceréis:

1. Si se redefine el método Equals() de una clase debería redefinirse también el método GetHashCode(), para que pueda cumplirse la segunda norma que es…
2. Si la llamada a Equals para dos objetos devuelve *true*, entonces GetHashCode() debe devolver el mismo valor para ambos objetos

El código que se muestra a continuación cumple estas condiciones

```
'-----------------------------------------------------------------
'Sombreado Object.GetHashCode
'http://msdn.microsoft.com/es-es/library/system.object.gethashcode.aspx
'http://msdn.microsoft.com/es-es/library/7h9bszxx.aspx
'-----------------------------------------------------------------
Public Overrides Function GetHashCode() As Integer
    Dim cociente As Double
    cociente = Me._numerador / Me._denominador
    Return cociente.GetHashCode
End Function
```

La biblioteca MSDN, -Biblioteca de clases de .NET Framework-, en su entrada Object. GetHashCode (Método): dice lo siguiente (este texto esta resumido del original):

- Una función hash se utiliza para generar con rapidez un número (código hash) que se corresponda con el valor de un objeto. Las funciones hash suelen ser específicas para cada Type y deben utilizar al menos uno de los campos de instancia como entrada.
- Una función hash debe tener las siguientes características:
    - Si dos objetos resultan iguales al compararlos, el método GetHashCode para cada objeto debe devolver el mismo valor.
    - El método GetHashCode para un objeto debe devolver de forma consecuente el mismo código hash mientras no haya ninguna modificación en el estado del objeto.
    - Las implementaciones del método GetHashCode no deben dar lugar a referencias circulares.
    - Las implementaciones del método GetHashCode no deben producir excepciones.
    - Las clases derivadas que reemplazan el método GetHashCode también deben reemplazar el método Equals para garantizar que dos objetos considerados iguales tengan el mismo código hash.

Una forma fácil y rápida de implementar GetHashCode() y que cumpla las dos normas básicas es algo así:

```
_
Public Class Prueba

    ' en lugar de una propiedad uso variables
    ' asi ahorro codigo aunque no es lo apropiado
    Public Uno As Integer
    Public Dos As Integer


    Public Overrides Function Equals(ByVal obj As Object) As Boolean
        Return TypeOf obj Is Prueba And _
               CType(obj, Prueba).Uno = Me.Uno And _
               CType(obj, Prueba).Dos = Me.Dos
    End Function 'Equals

    Public Overrides Function GetHashCode() As Integer
        Return String.Format("{0},{1}", Me.Uno, Me.Dos).GetHashCode()
    End Function 'GetHashCode

End Class 'Prueba
```

Simplemente creamos una representación en cadena del objeto y llamamos a GetHashCode de dicha cadena.

A modo de resumen

```
#Region "    Implements IEquatable(Of NumeroRacional)"

    '-------------------------------------------------------------------
    'Interfaz IEquatable(Of T)
    'http://msdn.microsoft.com/es-es/library/ms131187.aspx
    'Determinar si dos objetos tienen el mismo VALOR.
    '-------------------------------------------------------------------
    'Para determinar si tienen la misma REFERENCIA utilizar la función
    'Object.ReferenceEquals (Método)
    'http://msdn.microsoft.com/es-es/library/system.object.referenceequals.aspx
    '-------------------------------------------------------------------

    '-------------------------------------------------------------------
    'La interfaz IEquatable
    'http://msdn.microsoft.com/es-es/library/ms131190.aspx
    '-------------------------------------------------------------------
    ''' <summary>
    '''    Determina si dos objetos tienen el mismo valor
    ''' </summary>
    ''' <returns>
    '''    <para>Un valor <see cref="System.Boolean">lógico</see> que indica:</para>
    '''    <para>True:  Si son Iguales</para>
    '''    <para>False: en caso contrario</para>
    ''' </returns>
    Public Overloads Function Equals(ByVal other As NumeroRacional) As Boolean _
                 Implements System.IEquatable(Of NumeroRacional).Equals
        Return Equals(Me, other)
    End Function


    '-------------------------------------------------------------------
    '  Sombreado Object.Equals
    '  http://msdn.microsoft.com/es-es/library/system.object.equals.aspx
    '-------------------------------------------------------------------
    ''' <summary>
    '''    Determina si dos objetos tienen el mismo valor
    ''' </summary>
    ''' <returns>
    '''    <para>Un valor <see cref="System.Boolean">lógico</see> que indica:</para>
    '''    <para>True:  Si son Iguales</para>
    '''    <para>False: en caso contrario</para>
    ''' </returns>
```

```
Public Overloads Overrides Function Equals(ByVal obj As Object) As Boolean

    '-------------------------------------------------------------
    ' El parámetro, obj, debe ser del mismo tipo que la clase o el
    ' tipo de valor que implementa esta interfaz
    '-------------------------------------------------------------
    If Not (obj.GetType Is Me.GetType) Then
        Return False
        '"Los objetos a comparar no son del mismo tipo")
    End If

    'convertirlo a estructura NumeroRacional
    Dim unNumeroRacional As NumeroRacional
    unNumeroRacional = CType(obj, NumeroRacional)

    'realizar la comparacion
    Return Equals(Me, unNumeroRacional)
End Function


'-------------------------------------------------------------------
'  Sombreado Object.Equals
'  http://msdn.microsoft.com/es-es/library/system.object.equals.aspx
'-------------------------------------------------------------------
''' <summary>
'''    Determina si dos objetos tienen el mismo valor
''' </summary>
''' <returns>
'''    <para>Un valor <see cref="System.Boolean">lógico</see> que indica:</para>
'''    <para>True:  Si son Iguales</para>
'''    <para>False: en caso contrario</para>
''' </returns>
Public Overloads Shared Function Equals( _
                 ByVal nr1 As NumeroRacional, _
                 ByVal nr2 As NumeroRacional) _
                 As Boolean
    '-------------------------------------------------------
    ' Aquí es donde se hace TODO el trabajo de comparación
    '-------------------------------------------------------
    Dim resultado As Boolean = False
    If nr1._numerador = nr2._numerador AndAlso _
       nr1._denominador = nr2._denominador Then
        resultado = True
    End If
    Return resultado
End Function

'-------------------------------------------------------------------
'Instrucciones para implementar Equals y el operador de igualdad (==)
'http://msdn.microsoft.com/es-es/library/7h9bszxx.aspx
'-------------------------------------------------------------------
''' <summary>
'''   Devuelve un valor que indica si un objeto Dinero
'''   especificado es igual que otro objeto Dinero especificado.
''' </summary>
''' <returns>
'''    <para>Un valor <see cref="System.Boolean">lógico</see> que indica:</para>
'''    <para>True:  Si son Iguales</para>
'''    <para>False: en caso contrario</para>
''' </returns>
Public Overloads Shared Operator =( _
                              ByVal nr1 As NumeroRacional, _
                              ByVal nr2 As NumeroRacional) _
                              As Boolean
    Return Equals(nr1, nr2)
End Operator


'-------------------------------------------------------------------
```

```
    'Instrucciones para implementar Equals y el operador de igualdad (==)
    'http://msdn.microsoft.com/es-es/library/7h9bszxx.aspx
    '-----------------------------------------------------------------
    ''' <summary>
    '''   Devuelve un valor que indica si un objeto Dinero
    '''   especificado es distinto que otro objeto Dinero especificado.
    ''' </summary>
    ''' <returns>
    '''    <para>Un valor  <see cref="System.Boolean">lógico</see> que indica:</para>
    '''    <para>True:  Si son DIFERENTES</para>
    '''    <para>False: en caso contrario</para>
    ''' </returns>
    Public Overloads Shared Operator <>( _
                            ByVal nr1 As NumeroRacional, _
                            ByVal nr2 As NumeroRacional) _
                            As Boolean
        Return Not (Equals(nr1, nr2))
    End Operator


    '-----------------------------------------------------------------
    'Sombreado Object.GetHashCode
    'http://msdn.microsoft.com/es-es/library/system.object.gethashcode.aspx
    '-----------------------------------------------------------------
    ''' <summary>
    '''  Devuelve el código hash de esta instancia.
    ''' </summary>
    Public Overrides Function GetHashCode() As Integer
        'http://msdn.microsoft.com/es-es/library/system.object.gethashcode.aspx
        'http://msdn.microsoft.com/es-es/library/7h9bszxx.aspx
        Dim cociente As Double
        cociente = Me._numerador / Me._denominador
        Return cociente.GetHashCode
    End Function


#End Region
```

Más información:

¿Qué es un número Racional?

Un número racional es un número representado por el cociente de dos números enteros (lo que normalmente llamamos quebrado), por ejemplo (5/7). El número de la izquierda se denomina numerador y el de la derecha denominador

Hay ocasiones en la que es útil manejar estos números para evitar problemas de redondeos en operaciones
Por ejemplo:

- Utilizando números racionales (Quebrados) 1/3 + 1/3+ 1/3 = 1
- Si utilizamos valores System.Double: 0.33333+ 0.33333 + 0.33333 = 0. 99999
- Se puede apreciar que trabajando con quebrados no existen diferencias por redondeos

Hay que trabajar con los números racionales simplificados al máximo, con la fracción más sencilla posible. Por ejemplo 1/3, 2/6, 4/12 son todos el mismo número racional

Podemos utilizar el algoritmo de Euclides para obtener el máximo común divisor (mcd) del numerador y denominador, y simplificar el número racional dividiendo ambos números por su mcd.

Código de ejemplo

En el siguiente enlace hay un fichero ZIP que contiene ejemplo

- Fichero con el código de ejemplo: [2008_09_27_NumeroRacionalEstructura.zip]
- MD5 checksum: [B6680B49C257AC47F6F11B4423C522D0]
- MD5 checksum: Información
  - [http://www.elguille.info/colabora/MD5_checksum.aspx]

Bibliografía

Redefiniendo GetHashCode

- Publicado 21/5/2010 11:45 por Eduard Tomàs i Avellana
- http://geeks.ms/blogs/etomas/archive/2010/05/21/redefiniendo-gethashcode.aspx

Definición de un numero racional copiada de
Java 2 Curso de Programación Pág. 306
Fco. Javier Ceballos Sierra
ISBN: 84-7897-430-X
RA MA 2000

Información de este Documento

[Historial de este documento]

- *Autor*:
  - *Nombre*: Joaquín Medina Serrano
  - *Email* : joaquin@medina.name
  - *Web*: http://joaquin.medina.name

- *Publicador*:
  - *Nombre*: Joaquín Medina Serrano

- *Fechas* (Formato de fecha ISO 8610:2004. [Año-Mes-DiaTHora:Minutos])
  - *Fecha de Creación*: 2008-09-05T23:06
  - *Fecha de la última modificación*: 2010-05-21T13:13
  - *Fecha de Impresión*: 2010-05-21T13:13

- *Copyright*
  -

[Direcciones Web Interesantes]

Más Información sobre números racionales:

- Wiki - Números Racionales
  - [http://es.wikipedia.org/wiki/N%C3%BAmero_racional]
- Wiki -Categoría: Fracciones
  - [http://es.wikipedia.org/wiki/Categor%C3%ADa:Fracciones]
- Tipos de números Racionales
  - [http://platea.pntic.mec.es/anunezca/ayudas/raiz_de_2_irracional/r_irracional.htm]

Referencia general de .NET Framework

[http://msdn.microsoft.com/es-es/library/cc433286(VS.71).aspx]

- Instrucciones de uso de tipos de valor
  - http://msdn.microsoft.com/es-es/library/cc433409(VS.71).aspx
- Tipos de valor en el sistema de tipos común
  - http://msdn.microsoft.com/es-es/library/34yytbws.aspx
- Tipos de valores y tipos de referencia
  - http://msdn.microsoft.com/es-es/library/t63sy5hs.aspx

Visual Studio Team System

[http://msdn.microsoft.com/es-es/library/fda2bad5.aspx]

- Sobrecargar el operador de igualdad al sobrecargar los operadores de suma y resta
  - http://msdn.microsoft.com/es-es/library/ms182164.aspx

Manual del programador de .NET Framework

[http://msdn.microsoft.com/es-es/library/sxe8hcf2.aspx]

- Instrucciones para implementar Equals y el operador de igualdad (==)
  - http://msdn.microsoft.com/es-es/library/7h9bszxx.aspx
- Implementar el método Equals
  - http://msdn.microsoft.com/es-es/library/336aedhh.aspx
- Instrucciones de uso
  - http://msdn.microsoft.com/es-es/library/ms229035.aspx

Biblioteca de clases de .NET Framework

[http://msdn.microsoft.com/es-es/library/ms229335.aspx]

- Object (Clase)
  - http://msdn.microsoft.com/es-es/library/system.object.aspx
- Object.ReferenceEquals (Método)
  - http://msdn.microsoft.com/es-es/library/system.object.referenceequals.aspx
- Object.Equals (Método)
  - http://msdn.microsoft.com/es-es/library/system.object.equals.aspx
- Object. GetHashCode (Método)
  - http://msdn.microsoft.com/es-es/library/system.object.gethashcode.aspx
- IComparable (Interfaz)
  - http://msdn.microsoft.com/es-es/library/system.icomparable.aspx
- IEquatable(Of T) (Interfaz genérica)
  - http://msdn.microsoft.com/es-es/library/ms131187.aspx
- Type (Clase)

  - http://msdn.microsoft.com/es-es/library/system.type.aspx
- IConvertible (Interfaz)
  - http://msdn.microsoft.com/es-es/library/system.iconvertible.aspx
- IFormattable (Interfaz)
  - http://msdn.microsoft.com/library/system.iformattable.aspx
- CultureInfo (Clase)
  - http://msdn.microsoft.com/es-es/library/system.globalization.cultureinfo.aspx
- IFormatProvider (Interfaz)
  - http://msdn.microsoft.com/es-es/library/system.iformatprovider.aspx
- Decimal..::.ToString (Método) (String)
  - http://msdn.microsoft.com/es-es/library/fzeeb5cd.aspx
- String.Format (Método) (String, Object, Object, Object)
  - http://msdn.microsoft.com/es-es/library/d9t40k6d.aspx
- Convert (Clase)
  - http://msdn.microsoft.com/es-es/library/system.convert.aspx
- Convert.ToInt32 (Método) (Int32)
  - [http://msdn.microsoft.com/es-es/library/f4a76a1x.aspx]
- Boolean (Estructura) [http://msdn.microsoft.com/es-es/library/system.boolean.aspx]
- SByte (Estructura)[http://msdn.microsoft.com/es-es/library/system.sbyte.aspx]
- Byte (Estructura) [http://msdn.microsoft.com/es-es/library/system.byte.aspx]
- Int16 (Estructura) [http://msdn.microsoft.com/es-es/library/system.int16.aspx]
- UInt16 (Estructura) [http://msdn.microsoft.com/es-es/library/system.uint16.aspx]
- Int32 (Estructura) [http://msdn.microsoft.com/es-es/library/system.int32.aspx]
- UInt32 (Estructura) [http://msdn.microsoft.com/es-es/library/system.uint32.aspx]
- Int64 (Estructura) [http://msdn.microsoft.com/es-es/library/system.int64.aspx]
- UInt64 (Estructura) [http://msdn.microsoft.com/es-es/library/system.uint64.aspx]
- Single (Estructura) [http://msdn.microsoft.com/es-es/library/system.single.aspx]
- Double (Estructura) [http://msdn.microsoft.com/es-es/library/system.double.aspx]
- Decimal (Estructura) [http://msdn.microsoft.com/es-es/library/system.decimal.aspx]
- DateTime (Estructura) [http://msdn.microsoft.com/es-es/library/system.datetime.aspx]
- Char (Estructura) [http://msdn.microsoft.com/es-es/library/system.char.aspx]
- String (Clase) [http://msdn.microsoft.com/es-es/library/system.string.aspx]
- Structure [http://msdn.microsoft.com/es-es/library/0awthy7k.aspx]).
- OverflowException (Clase)
  - [http://msdn.microsoft.com/es-es/library/system.overflowexception.aspx]
- InvalidCastException (Clase)
  - [http://msdn.microsoft.com/es-es/library/system.invalidcastexception.aspx]

[Palabras Clave]

- interfaz, IEquatable, 'interfaz IEquatable', 'tipos por valor', 'tipos por referencia', variables, Equals, instancia, 'misma instancia', comparación, 'comparación de objetos', GetHashCode, 'número racional'

[Grupo de documentos]

- [Documento Index]
- [Documento Start]

Joaquín 'jms32' Medina Serrano
Esta página es española

La Güeb de Joaquín
Mi sitio de Internet