La Güeb de Joaquín
Mi sitio de Internet

Interfaz IComparer

[Descripción general]

Hay ocasiones en las que es necesario saber si un objeto es mayor, menor, o igual que otro. Mejor dicho, cuando el VALOR del objeto es mayor, igual o menor que otro.
Cuando se necesita **ordenar** objetos de **una única manera** Microsoft recomienda que se implemente la interfaz IComparable.
Pero en el mundo real, los objetos se pueden **ordenar de varias formas**, por ejemplo, los nombres de las personas pueden ordenarse por nombre o por el apellido, para cubrir estos casos deberemos implementar la interfaz IComparer
.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

Introducción

Casi todas las clases Colección, (por ejemplo, System.Array, List, ArrayList, SortedDictionary (Of T), etc.) Exponen el método compartido Sort que permite ordenar los objetos contenidos en la colección, siempre y cuando sean del tipo de datos sencillos tales como números o cadenas (integer, doublé, char, string, etc.).

La ordenación de los datos se realiza por un proceso interno que compara los datos entre si y averigua si un dato es mayor igual o menor que otro y entonces se ordenan según ese valor.

Sin embargo, el método Sort, no puede ordenar objetos más complejos, por ejemplo, Persona, porque no sabe como comparar esos objetos entre sí.

El método **Sort**, está sobrecargado y una de sus sobrecargas **admite la interfaz** IComparable. Esto quiere decir que Sort sabe como ordenar a cualquier objeto que implemente la interfaz IComparable. Dicho de otra forma. Si queremos que un objeto cualquiera, por ejemplo Persona, pueda ser ordenado dentro de una colección utilizando el método Sort, deberemos implementar la interfaz IComparable.

La interfaz IComparable, expone un único método CompareTo, que recibe un objeto y devuelve -1, 0, +1, dependiendo de que el objeto actual sea menor que, igual a, o mayor que el objeto que se ha pasado como argumento.

La interfaz IComparablees útil y suficiente si solamente se quieren ordenar objetos de una única manera, pero en el mundo real, los objetos se pueden ordenar de varias formas, por ejemplo, los nombres de las personas pueden ordenarse por nombre o por el apellido, para cubrir estos casos deberemos implementar la interfaz IComparer

La interfaz IComparer, expone un único método Compare, que recibe dos objetos y devuelve -1, 0, +1, dependiendo de que el primer objeto sea menor que, igual a, o mayor que el segundo objeto.

El método **Sort**, está sobrecargado y otra de sus sobrecargas **admite la interfaz** IComparer. Esto quiere decir que Sort Sabe como ordenar a cualquier objeto que implemente la interfaz IComparer.

La interfaz IComparer solo expone un método (**Compare**), luego si implemento esta interfaz en el objeto Persona, solo podre ordenar de una manera (la forma que se describa en el código de la función **Compare**), y además, Microsoft recomienda que la forma de ordenación sea la misma que la que realiza el método **CompareTo**, de hecho recomienda que **Compare** llame a **CompareTo** para obtener el valor de la comparación (-1, 0,+1)

Entonces, ¿Como me las arreglo para poder ordenar de formas diferentes un objeto? Pues utilizando clases Internas que implementan la interfaz IComparer, y en cada una de ellas se define el método especifico de comparación. Después solo hay que llamar al método Sort y pasar el valor de la interfaz IComparer de las clases internas para que Sort, ordene de la forma que queremos. Sigue leyendo este documento y veras una descripción detallada y un ejemplo que muestra como se hace.

Observación Muy Importante,

Toda clase que implemente la interfaz IComparer, También debe, NECESARIAMENTE, implementar la interfaz IComparable. La Razón es que la función [Compare] que implementa esta interfaz [IComparer] debe disparar la excepción [ArgumentException] cuando: Ni x ni y implementan la interfaz IComparable. Ver referencia [MSDN Apartado Excepciones], y además, Microsoft recomienda utilizar la función CompareTo para realizar la operación de comparación

Toda clase que implemente la interfaz IComparable, debería implementar la Interfaz IEquatable. La razón es que IComparable.CompareTo realiza una comparación de objetos para ver si el primero es mayor, igual, o menor que el otro. Y la comparación de igualdad (si un objeto es igual que otro) debería realizarla la función Equals, que pertenece a la interfaz IEquatable.

La clase para ver el ejemplo

Para ver el concepto vamos a trabajar con una clase Persona que se muestra a continuación

```
''' <summary>
'''   Clase Persona. EJEMPLO, para mostrar como se implementan las interfaces
''' </summary>
<Serializable()> _
Friend Class Persona
    Implements IComparable
    Implements IComparable(Of Persona)


    Private _nombre As String = String.Empty
    Private _apellidos As String = String.Empty
    Private _dni As String = String.Empty

    Public Property Nombre() As String
        Get
            Return _nombre
        End Get
        Set(ByVal value As String)
            _nombre = value
        End Set
    End Property


    Public Property Apellidos() As String
        Get
            Return _apellidos
        End Get
        Set(ByVal value As String)
            _apellidos = value
        End Set
    End Property


    Public Property DNI() As String
        Get
            Return _dni
        End Get
        Set(ByVal value As String)
            _dni = value
        End Set
    End Property


    Public Sub New(ByVal nombre As String, ByVal apellidos As String, ByVal dni As String)
        Me._nombre = nombre
        Me._apellidos = apellidos
        Me._dni = dni
    End Sub
```

```
    ' Devuelve el nombre y los apellidos
    Public ReadOnly Property ToNombreApellidos() As String
        Get
            Return String.Format(System.Globalization.CultureInfo.CurrentCulture, _
                                 " | {0,10}, {1,15} | ", _
                                 Me.Nombre, _
                                 Me.Apellidos)

        End Get
    End Property

    ' Devuelve los apellidos y el nombre
    Public ReadOnly Property ToApellidosNombre() As String
        Get
            Return String.Format(System.Globalization.CultureInfo.CurrentCulture, _
                                 " | {0,15}, {1,10} | ", _
                                 Me.Apellidos, _
                                 Me.Nombre)

        End Get
    End Property

End Class
```

La Interfaz

Microsoft recomienda utilizar las interfaces [IComparable] y [IComparable (Of T)] en aquellos objetos que necesiten ser ordenados en o por una colección de objetos.

Firma de la interfaz

La interfaz es una interfaz genérica cuya firma es:

- `Public Interface IComparer`
- `Public Interface IComparer (Of T)`

Firma de las funciones de la interfaz

Las funciones que hay que implementar tienen la firma:

```
    Public Overloads Function Compare(ByVal x As Object, ByVal y As Object) As Integer _
           Implements System.Collections.IComparer.Compare
```

y

```
    Public Overloads Function Compare(ByVal x As Persona, ByVal y As Persona) As Integer _
           Implements System.Collections.Generic.IComparer(Of Persona).Compare
```

Implementando la Interfaz

Paso Uno – Definir la interfaz

Ya hemos comentado que toda clase que implemente IComparer, también debe implementar IComparable

Y que toda clase que implemente IComparable debe implementar IEquatable

```
Friend Class Persona
    Implements IComparer
    Implements IComparer(Of Persona)
    Implements IComparable
    Implements IComparable(Of Persona)
    Implements IEquatable(Of Persona)

...
...
End Class
```

Paso Dos - Implementar la función de la interfaz

La función Compare tiene que comparar dos objetos y decidir si el primero es mayor que el segundo, si son iguales o si el primero es menor que el segundo.

Pero observa... hay que decidir si dos objetos son iguales, y esa es la tarea del método Equals. Dicho de otro modo, cuando escribamos el criterio de comparación de dos objetos, la parte del criterio que identifica a dos objetos como iguales debe ser la misma que la empleada en el método Equals.

En este ejemplo, el **criterio de igualdad** empleado en la función Equals, es que el valor del campo DNI. Si dos clases tienen el mismo DNI se supone que corresponden a la misma persona y entonces son iguales.

Para escribir la función Compare tengo que elegir un **criterio de comparación** que no invalide la forma de trabajar de Equals. Lo que implica que para comparar dos objetos Persona, utilizare también el campo DNI

La función ´Compare´ mirara el valor del DNI de dos clases y en función de sus valores devolverá los valores [-1], [0], [+1]
Por ejemplo si se comparan los objetos D1 y D2 el resultado puede ser

- -1 si D1 < D2
- 0 si D1 = D2
- +1 si D1 > D2

Como son dos funciones que hacen el mismo trabajo, lo que hago es que la función genérica (Of T) es la que haga el trabajo real y la otra la llame para obtener el resultado

Una Observación:
Microsoft Sugiere que la función IComparer.Compare, llame a la función IComparable.CompareTo para hacer el trabajo de comparación
VER - Biblioteca de clases de .NET Framework
[http://msdn.microsoft.com/es-es/library/ms229335.aspx]

- IComparer..::.Compare (Método)
  - http://msdn.microsoft.com/es-es/library/system.collections.icomparer.compare.aspx

Lo que implica que si implemento la interfaz IComparer, Tengo que implementar la Interfaz IComparable. Y como ya sabemos, si implemento la interfaz IComparable, tengo que implementar la Interfaz IEquatable

```
#Region "    Implements IComparer; IComparer(Of Persona)"

    Public Overloads Function Compare(ByVal x As Object, ByVal y As Object) As Integer _
           Implements System.Collections.IComparer.Compare

        '---------------------------
        ' Por definición, dos objetos nulos son iguales
        ' nulo=nulo =0
        If Object.ReferenceEquals(x, Nothing) AndAlso _
         Object.ReferenceEquals(y, Nothing) Then Return 0
        ' Por definición: cualquier objeto es mayor que un objeto nulo
        ' nulo< algo = +1
        If Object.ReferenceEquals(x, Nothing) Then Return +1
        ' algo>nulo = -1
        If Object.ReferenceEquals(y, Nothing) Then Return -1
        ' son la misma referencia (es el mismo objeto)
        If Object.ReferenceEquals(x, y) = True Then Return 0
        '-------------------------------------------------------------
        ' realizar la comparacion
        '-------------------------------------------------------------
        Try
            '-------------------------------------------------------------
            ' Apunte(Táctico)
            ' La función CType dispara las siguientes excepciones
            ' (Documentación MSDN)- Ctype - Ver Comentario
            ' [http://msdn.microsoft.com/es-es/library/4x2877xb.aspx]
            ' Si en una conversión se produce un error en tiempo de ejecución,
            ' se produce la excepción correspondiente. Si se produce un error
            ' en una conversión de restricción, OverflowException es el resultado
            ' más común. Si la conversión es indefinida, se produce una excepción
            ' InvalidCastException..
            '-------------------------------------------------------------
            ' Si alguno de los objetos no es una clase Persona
            ' se dispara la excepcion [ArgumentException]
            '-------------------------------------------------------------
            'Convetirlos a Objetos Persona
            Dim p1 As Persona = CType(x, Persona)
            Dim p2 As Persona = CType(y, Persona)
            ' llamar a la funcion que hace el trabajo
            Return Me.Compare(p1, p2)

        Catch ex As Exception
            Throw New ArgumentException(ex.Message, ex)
        End Try
    End Function



    Public Overloads Function Compare(ByVal x As Persona, ByVal y As Persona) As Integer _
           Implements System.Collections.Generic.IComparer(Of Persona).Compare
        '---------------------------
        ' Por definición, dos objetos nulos son iguales
        If (x Is Nothing) AndAlso (y Is Nothing) Then Return 0
        ' Por definición, cualquier objeto es mayor que un objeto nulo
        If (x Is Nothing) Then Return +1
        If (y Is Nothing) Then Return -1
        ' son la misma referencia (es el mismo objeto)
        If Object.ReferenceEquals(x, y) = True Then Return 0
        '-------------------------------------------------------------
        ' - CRITERIO DE COMPARACION -
        ' - Tienen el mismo valor si tienen el mimso DNI
        '-----------------------------------------------
        ' Menor que cero [-1] -  x  es menor que y.
        ' Cero           [ 0] -  x  es igual a y.
        ' Mayor que cero [+1] -  x  es mayor que y.
        '-------------------------------------------------------------

        ''-------------------------------------------------------------
        '' Codigo que funciona sustituido
        ''-------------------------------------------------------------
        ''Como DNI es una cadena, y voy a comparar dos cadenas,
        ''en lugar de escribir todo el proceso de comparación de cadenas
        ''utilizo la clase StringComparer que realiza ese trabajo
        ''-------------------------------------------------------------
        '' definir una clase [StringComparer] que utilice la
        '' referencia cultural actual y un sistema de comparación IgnoreCase,
        '' es decir que no distinga entre mayúsculas y minúsculas
```

```
        'Dim resultado As Integer = 0
        'Dim SC As StringComparer
        'SC = StringComparer.Create(System.Globalization.CultureInfo.CurrentCulture, True)
        '' realizar la comparacion de objetos
        ''-------------------------------------------------------------
        '' APUNTE TACTICO
        '' Hay un problema con la comparación de cadenas cuando las cadenas
        '' tienen números. Porque se comparan como su fueran letras y se
        '' ordenan como las letras, de forma  que la 'cadena' [2]
        '' está delante la 'cadena' [10] o [159]  (porque una cadena que
        '' tiene menos caracteres que otra siempre es más pequeña y va delante)
        ''
        '' En este caso y como solo es para ejemplo no se trata este problema
        ''-------------------------------------------------------------
        'resultado = SC.Compare(x.DNI, y.DNI)
        ''
        '' devolver el resultado
        'Return resultado
        ''-------------------------------------------------------------
        '' Fin codigo sustituido
        ''-------------------------------------------------------------

        '-------------------------------------------------------------
        ' ¡¡¡¡¡ FIJATE !!!!
        ' Implementacion sugerida por Microsoft
        ' Usar la interfaz IComparable.CompareTo para hacer el trabajo
        ' Biblioteca de clases de .NET Framework
        ' IComparer.Compare (Método) - apartado Comentarios
        ' [http://msdn.microsoft.com/es-es/library/system.collections.IComparer.compare.aspx]
        '-------------------------------------------------------------
        Return x.CompareTo(y)
        '-------------------------------------------------------------
    End Function

#End Region
```

Paso Tres – Implementar la interfaz IComparable

OBSERVACION IMPORTANTE:
Solo se muestran la firma de las funciones a implementar

```
#Region "    Implements IComparable; IComparable(Of Persona)"

  '-------------------------------------------------------------------
    'La interfaz IComparable
    'http://msdn.microsoft.com/es-es/library/system.IComparable.compareto.aspx
    '-------------------------------------------------------------------
    Public Function CompareTo(ByVal other As Persona) As Integer _
           Implements System.IComparable(Of Persona).CompareTo


  '-------------------------------------------------------------------
    'La interfaz IComparable(Of Persona)
    'http://msdn.microsoft.com/es-es/library/43hc6wht.aspx
    '-------------------------------------------------------------------
    Public Function CompareTo(ByVal obj As Object) As Integer _
           Implements System.IComparable.CompareTo


    '-------------------------------------------------------------------
    'Implementar el operador de MenorQue (<)
    'http://msdn.microsoft.com/es-es/library/7h9bszxx.aspx
    '-------------------------------------------------------------------
    Public Overloads Shared Operator <( Persona, Persona) As Boolean

    '-------------------------------------------------------------------
    'Implementar el operador de MayorQue (>)
    'http://msdn.microsoft.com/es-es/library/7h9bszxx.aspx
    '-------------------------------------------------------------------
    Public Overloads Shared Operator >( Persona, Persona) As Boolean

#End Region
```

Paso Cuatro – Implementar la interfaz IEquatable

OBSERVACION IMPORTANTE:
Solo se muestran la firma de las funciones a implementar

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

    '--------------------------------------------------------------------
    'Interfaz IEquatable(Of T)
    'http://msdn.microsoft.com/es-es/library/ms131187.aspx
    'Determinar si dos objetos tienen el mismo VALOR.
    '--------------------------------------------------------------------
    'La interfaz IEquatable
    '--------------------------------------------------------------------
    Public Overloads Function Equals(ByVal other As Persona) As Boolean _
              Implements System.IEquatable(Of Persona).Equals

    '--------------------------------------------------------------------
    '  Sombreado Object.Equals
    Public Overloads Overrides Function Equals(Object) As Boolean

    '--------------------------------------------------------------------
    '  Sombreado Object.Equals
    '------------------------------------------------------
    ' Aquí es donde se hace TODO el trabajo de comparación
    '------------------------------------------------------
    Public Overloads Shared Function Equals(Persona, Persona) As Boolean

    '--------------------------------------------------------------------
    ' Implementar el operador de igualdad (=)
    '--------------------------------------------------------------------
    Public Overloads Shared Operator =(Persona, Persona) As Boolean

    '--------------------------------------------------------------------
    ' Implementar el operador DiferenteDe (<>)
    Public Overloads Shared Operator <>(Persona, Persona) As Boolean

    '--------------------------------------------------------------------
    'Sombreado Object.GetHashCode
    'http://msdn.microsoft.com/es-es/library/system.object.gethashcode.aspx
    Public Overrides Function GetHashCode() As Integer

#End Region
```

Implementación Avanzada

Cuando necesitamos ordenar una clase por varios campos, o combinaciones de ellos, por ejemplo, los nombres de las personas pueden ordenarse por nombre o por el apellido, existe una 'trampa' que consiste en definir en la clase, dos o más clases anidadas que implementen la interfaz IComparer, usando (y escribiendo) una clase para cada posible método de ordenación.

Este sistema funciona porque el método 'sort' de las colecciones, tiene una sobrecarga que acepta la interfaz IComparer

Ejemplo de una clase auxiliar implementando las interfaces IComparer e IComparer (Of T)

```
'-------------------------------------------------------------------
'Clase auxiliar para ordenar por apellidos  y nombre a la vez
'-------------------------------------------------------------------
Friend Class OrdenarPorApellidosNombre
    Implements IComparer
    Implements IComparer(Of Persona)

    Public Overloads Function Compare(ByVal x As Object, ByVal y As Object) As Integer _
           Implements System.Collections.IComparer.Compare
        ' --------
        ' dos objetos nulos son iguales
        If (x Is Nothing) AndAlso (y Is Nothing) Then Return 0
        ' cualquier objeto es mayor que un objeto nulo
        If (x Is Nothing) Then Return +1
        If (y Is Nothing) Then Return -1
        ' --------
        ' son la misma referencia (es el mismo objeto)
        If Object.ReferenceEquals(x, y) = True Then Return 0
        '-------------------------------------------------------------
        ' realizar la comparacion
        '-------------------------------------------------------------
        Try
            '---------------------------------------------------------
            ' Si alguno de los objetos no es una clase Persona
            ' se dispara la excepcion [ArgumentException]
            Dim p1 As Persona = CType(x, Persona)
            Dim p2 As Persona = CType(y, Persona)
            '---------------------------------------------------------
            ' realizar la comparacion de objetos
            ' llamar a la funcion generica que es la que hace el trabajo
            Return Compare(p1, p2)
        Catch ex As Exception
            Throw New ArgumentException(ex.Message, ex)
        End Try
    End Function

    Public Overloads Function Compare(ByVal x As Persona, ByVal y As Persona) As Integer _
            Implements System.Collections.Generic.IComparer(Of Persona).Compare

        '-------------------------------------------------------------
        ' dos objetos nulos son iguales
        If (x Is Nothing) AndAlso (y Is Nothing) Then Return 0
        ' Valores Nothing -->  nulo< algo = +1
        ' cualquier objeto es mayor que un objeto nulo
        If (x Is Nothing) Then Return +1
        If (y Is Nothing) Then Return -1
        ' --------
        ' son la misma referencia (es el mismo objeto)
        If Object.ReferenceEquals(x, y) = True Then Return 0

        '-------------------------------------------------------------
        'Como NOMBRE es una cadena,  voy a comparar dos cadenas,
        'en lugar de escribir todo el proceso de comparación de cadenas
        'utilizo la clase StringComparer que realiza ese trabajo
        '-------------------------------------------------------------
        ' definir una clase [StringComparer] que utilice la
        ' referencia cultural actual y un sistema de comparación IgnoreCase,
        ' es decir que no distinga entre mayúsculas y minúsculas
        Dim SC As StringComparer
        SC = StringComparer.Create(System.Globalization.CultureInfo.CurrentCulture, True)
        ' realizar la comparacion de objetos
        '-------------------------------------------------------------
        Return SC.Compare((x.Apellidos & x.Nombre), (y.Apellidos & x.Nombre))
    End Function
End Class
```

Utilizar estas clases es bastante sencillo

```
Dim AR As ArrayList = New ArrayList
     AR.Add(Persona00)
     AR.Add(persona14)
     AR.Add(persona15)
     AR.Add(persona16)

     Console.WriteLine("================================================")
     Console.WriteLine("Pasamos la interfaz con Ctype")
     Console.WriteLine("Lista Ordenada por el apelidos y por el nombre")
     '------------------------------------------------
     ' Ordenar
     ' pasar la interfaz IComparer al metodo Sort
     ' (Pasamos la interfaz que implementa la clase auxiliar)
     AR.Sort(CType(New Persona.OrdenarPorApellidosNombre, Collections.IComparer))
     ' Listar
     Dim enumerador As System.Collections.IEnumerator
     enumerador = AR.GetEnumerator
     Do While (enumerador.MoveNext)
         Console.WriteLine(enumerador.Current.ToString())
     Loop
```

Como refinamiento se pueden definir unas funciones estáticas que devuelve la interfaz ICompararer, tendrán el siguiente aspecto

```
#Region " metodos compartidos que devuelven la interfaz [IComparer] de las clases auxiliares "

    '-------------------------------------------------------------
    ''' <summary>
    '''  Devuelve la interfaz IComparer para OrdenarPorApellidosNombre
    ''' </summary>
    Public Shared Function CompararPorApellidosNombre() As IComparer
        Return New OrdenarPorApellidosNombre
    End Function

    '-------------------------------------------------------------
    ''' <summary>
    '''  Devuelve la interfaz IComparer(Of Persona) para OrdenarPorApellidosNombre
    ''' </summary>
    Public Shared Function CompararPorApellidosNombreOf() As IComparer(Of Persona)
        Return New OrdenarPorApellidosNombre
    End Function

#End Region
```

Y entones la forma de usarlo será la siguiente

```
     Dim AR As ArrayList = New ArrayList
     AR.Add(Persona00)
     AR.Add(persona14)
     AR.Add(persona15)
     AR.Add(persona16)

     Console.WriteLine("Lista Ordenada por el apelidos y por el nombre")
     '------------------------------------------------
     ' Ordenar
     ' pasar la interfaz IComparer al metodo Sort
     ' (Pasamos la interfaz utilizando un metodo especifico que la devuelve)
     AR.Sort(Persona.CompararPorApellidosNombre)
     ' Listar
     Dim enumerador As System.Collections.IEnumerator
     enumerador = AR.GetEnumerator
     Do While (enumerador.MoveNext)
         Console.WriteLine(enumerador.Current.ToString())
     Loop
```

A modo de resumen

La Interfaz IComparable solo realiza un tipo de comparación y por lo tanto solo permite un tipo de ordenación de objetos

La Interfaz IComparer permite tener definidas varios tipos de ordenaciones para un objeto, de forma que podemos llamar al método Sort y pasarle en cada momento el método de ordenación que nos interese. Su implementación es un poco más complicada pero proporciona mucha más flexibilidad de uso

Más información:

Como escribo la función Compare

En la biblioteca MSDN tenemos la siguiente información

A) La firma de la función tiene que ser la siguiente:

```
Function Compare (x As Object, y As Object) As Integer
```

La Función compara dos objetos y devuelve un valor que indica si uno es menor, igual o mayor que el otro.

B) Parámetros

- x - Primer objeto que se va a comparar.
- y - Segundo objeto que se va a comparar.

C) Valor devuelto: es un valor [Integer] un Entero de 32 bits con signo que indica el orden relativo de los objetos que se está comparando. El valor devuelto tiene los siguientes significados:

- Menor que cero [-1] x es menor que y.
- Cero [0] x es igual a y.
- Mayor que cero [+1] x es mayor que y.

D) Excepciones: La función debe disparar la excepción [ArgumentException] cuando: Ni x ni y implementan la interfaz IComparable. O bien cuando x e y son de tipos diferentes y ninguno puede controlar comparaciones con el otro

E) Comentarios

- La implementación preferida consiste en utilizar el método CompareTo de uno de los parámetros.
- Se puede comparar Null (Nothing en Visual Basic) con cualquier tipo sin que se genere una excepción
  - Por definición, cualquier objeto es mayor que una referencia Null (Nothing en Visual Basic) (Null) se considera menor que cualquier otro objeto.
  - Dos referencias Null (Nothing en Visual Basic) son iguales entre sí.

- Los parámetros x e y, deben ser del mismo tipo que la clase o el tipo de valor que implementa esta interfaz; en caso contrario, se produce la excepción ArgumentException.

Código de ejemplo

En el siguiente enlace hay un fichero ZIP que contiene ejemplo

- Fichero con el código de ejemplo: [2008_10_08_SolucionInterfazIComparer.zip]
- MD5 checksum: [139F3E0FA74D626AE82DC4212DEF6671]
- MD5 checksum: Información
  - [http://www.elguille.info/colabora/MD5_checksum.aspx]

Bibliografía

- *Programación* Avanzada con Microsoft Visual Basic .NET
  Francisco Balena
  ISBN 84-481-3715-9
  McGraw-Hill
  Capitulo 6 Interfaces y Delegados

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*: 2008-09-10T16:40
  - *Fecha de la última modificación*: 2008-10-08T11:31
  - *Fecha de Impresión*: 2008-10-08T11:31

[Direcciones Web Interesantes]

Biblioteca de clases de .NET Framework
[http://msdn.microsoft.com/es-es/library/ms229335.aspx]

- IComparable (Interfaz)
  - http://msdn.microsoft.com/es-es/library/system.icomparable.aspx
- IComparable.CompareTo (Método)
  - http://msdn.microsoft.com/es-es/library/system.icomparable.compareto.aspx
- IComparable(Of T) (Interfaz genérica)
  - http://msdn.microsoft.com/es-es/library/4d7sx9hd.aspx
- IComparable(Of T).CompareTo (Método)
  - http://msdn.microsoft.com/es-es/library/43hc6wht.aspx
- IComparer (Interfaz)
  - http://msdn.microsoft.com/es-es/library/system.collections.icomparer.aspx
- IComparer..::.Compare (Método)
  - http://msdn.microsoft.com/es-es/library/system.collections.icomparer.compare.aspx
- StringComparer (Clase)
  - http://msdn.microsoft.com/es-es/library/system.stringcomparer.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
  - http://msdn.microsoft.com/es-es/library/ms229035.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

Referencia del lenguaje Visual Basic
[http://msdn.microsoft.com/es-es/library/sh9ywfdk.aspx]

- CType (Función)
  - http://msdn.microsoft.com/es-es/library/4x2877xb.aspx

[Palabras Clave]

- IComper, 'interfaz IComparer'Sort, IComparable, 'interfaz IComparable', Equals , interfaz, IEquatable, 'interfaz IEquatable', 'tipos por valor', 'tipos por referencia', variables, Equals, instancia, 'misma instancia', comparación, 'comparación de objetos', GetHashCode

[Grupo de documentos]

- [Documento Index]
- [Documento Start]

Joaquín 'jms32' Medina Serrano

Esta página es española

La Güeb de Joaquín
Mi sitio de Internet