La Güeb de Joaquín
Mi sitio de Internet

Interfaz IConvertible

[Descripción general]

Cuando necesitamos convertir el VALOR de un objeto (no su referencia) en otro tipo CLR con un valor equivalente, debemos utilizar la interface IConvertible.
Por ejemplo convertir un número entero en un entero largo, o en un número decimal. Observa que lo que hacemos es convertir un objeto Integer en un objeto Long o Decimal, pero con valor equivalente
CLR = Common Language Runtime

Contenido

Interfaz IConvertible

Introducción

La interfaz proporciona métodos que convierten el valor de una instancia de un tipo de implementación en un tipo de Common Language Runtime con un valor equivalente. Los tipos de Common Language Runtime son Boolean, SByte, Byte, Int16, UInt16, Int32, UInt32, Int64, UInt64, Single, Double, Decimal, DateTime, Char y String.

Si no se puede realizar la conversión se produce un error InvalidCastException.

- Por ejemplo, si esta interfaz se implementa en un tipo booleano, la implementación del método ToDateTime produce una excepción porque no existe ningún tipo DateTime significativo que sea equivalente a un tipo booleano.

- 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

#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

    Public ReadOnly Property Valor() As Decimal
        Get
            Return CType(Me._numerador / Me._denominador, Decimal)
        End Get
    End Property

#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>
Private Sub Simplificar()

    ' 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)

    'Algoritmo de Euclides
    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

End Sub


''' <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 para entrar un numero decimal
''' </summary>
Public Sub New(ByVal value As Decimal)


    If value = 0 Then
        ' no tiene decimales
        Me._numerador = 0L
        Me._denominador = 1L
    Else
        ' tiene decimales
        ' Proceso a seguir  '
        ' Primero calcular la unidad seguida de ceros
        ' por ejemplo, 2 decimales = 100, 3 decimales = 1000
        '        5,32 * 100      532      133
        ' 5,32= ------------ = ------ = ------
        '           100          100       25

        Dim numeroDecimales As Integer = 0
        numeroDecimales = DigitosDecimalesCount(value)
        Dim UnidadConCeros As Long = CLng(Math.Pow(10, numeroDecimales))
        Me._numerador = CType(value * UnidadConCeros, Long)
        Me._denominador = UnidadConCeros
    End If

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

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

    Call Simplificar()

End Sub

''' <summary>
'''  cuenta el número de cifras decimales (significativas)
'''   p.e   4,567 devuelve tres(3)
'''   p.e  12,00  devuelve cero (0)
''' </summary>
''' <param name="value">Un numero decimal </param>
''' <returns>
'''  un numero entero que indica las cifras decimales (significativas) del numero.
''' </returns>
''' <remarks>
''' Ejemplo, 12,12   devuelve 2
''' Ejemplo, 12,00   devuelve 0 ' atencion
''' Ejemplo, 12,1234 devuelve 4
''' </remarks>
Private Shared Function DigitosDecimalesCount(ByVal value As Decimal) As Integer

    Try

        Dim infoCultura As System.Globalization.CultureInfo
        infoCultura = System.Globalization.CultureInfo.CurrentCulture

        Dim numeroDecimales As Integer = 0
        Dim separadorDecimal As String
        separadorDecimal = infoCultura.NumberFormat.NumberDecimalSeparator

        ' convertir el numero en cadena
        Dim cadena As String = String.Empty
        cadena = value.ToString(infoCultura)

        Dim posicionSeparadorDecimal As Integer = 0
        posicionSeparadorDecimal = cadena.IndexOf( _
                                        separadorDecimal, _
                                        System.StringComparison.CurrentCulture)
```

```
            If posicionSeparadorDecimal <> -1 Then
                ' hay un separador decimal
                ' 0         1         2
                ' 012345678901234567890 [Posiciones]
                ' 12,1234567890123
                ' La coma decimal esta en la posicion 2
                ' el tamaño de la cadena es de 15 caracteres
                ' Cuidado count develve 16 caracteres
                ' (16-1)-2 = 13 posiciones decimales

                numeroDecimales = (cadena.Length - 1) - posicionSeparadorDecimal
                If numeroDecimales < 0 Then
                    numeroDecimales = 0
                End If
            End If

            Return numeroDecimales

        Catch ex As Exception
            Throw
        End Try
    End Function


#End Region


End Structure '/ NumeroRacional
```

La Interfaz

Firma de la interfaz

La interfaz es una interfaz genérica cuya firma es:

```
Public Interface IConvertible
```

Firma de las funciones de la interfaz

Y las funciones que hay que implementar tienen la firma

```
Function ToType(Type,IFormatProvider)As Object   Implements System.IConvertible.ToType
Function GetTypeCode()               As TypeCode Implements System.IConvertible.GetTypeCode
Function ToBoolean(IFormatProvider)  As Boolean  Implements System.IConvertible.ToBoolean
Function ToByte(IFormatProvider)     As Byte     Implements System.IConvertible.ToByte
Function ToChar(IFormatProvider)     As Char     Implements System.IConvertible.ToChar
Function ToDateTime(IFormatProvider) As Date     Implements System.IConvertible.ToDateTime
Function ToDecimal(IFormatProvider)  As Decimal  Implements System.IConvertible.ToDecimal
Function ToDouble(IFormatProvider)   As Double   Implements System.IConvertible.ToDouble
Function ToInt16(IFormatProvider)    As Short    Implements System.IConvertible.ToInt16
Function ToInt32(IFormatProvider)    As Integer  Implements System.IConvertible.ToInt32
Function ToInt64(IFormatProvider)    As Long     Implements System.IConvertible.ToInt64
Function ToSByte(IFormatProvider)    As SByte    Implements System.IConvertible.ToSByte
Function ToSingle(IFormatProvider)   As Single   Implements System.IConvertible.ToSingle
Function ToString(IFormatProvider)   As String   Implements System.IConvertible.ToString
Function ToUInt16(IFormatProvider)   As UShort   Implements System.IConvertible.ToUInt16
Function ToUInt32(IFormatProvider)   As UInteger Implements System.IConvertible.ToUInt32
Function ToUInt63(IFormatProvider)   As ULong    Implements System.IConvertible.ToUInt64
```

Implementando la Interfaz

Paso Uno – Definir la interfaz

Implementaremos la interfaz en nuestra estructura

```
Public Class Complejo
    Implements IConvertible
        ...
        ...
End class
```

Paso Dos - Implementar las funciones de la interfaz

- Microsoft Recomienda implementar estas conversiones usando la clase Convert [Ver Convert (Clase)]

Ejemplo de una conversión personalizada

```
    Public Function ToBoolean(ByVal provider As IFormatProvider) As Boolean _
           Implements IConvertible.ToBoolean

        '-------------------------------------------------------
        ' Forma personalizada de conversion
        If Me._numerador > 0 And Me._denominador > 0 Then
            Return True
        Else
            Return False
        End If
        ''-------------------------------------------------------
        '' Utilizar la clase Convert para realizar la conversion
        'Return Convert.ToBoolean(Me.Valor(), provider)
        '' Alternativa de Convert no recomendada
        'Return Convert.ToBoolean(Me.Valor())
        '' Forma alternativa que realiza el mismo trabajo
        'Return CType(Me.Valor(), IConvertible).ToBoolean(provider)
        ''-------------------------------------------------------
    End Function
```

Ejemplo de la función ToString de la interfaz

```
    ' ------------------------------------------------------------------------
    ' Interfaz System.IConvertible.ToString
    ' ------------------------------------------------------------------------
    ' Esta función lo que hace es llamar a la función que realmente hace el trabajo
    ' Observa que se utiliza un truco que consiste en declararla como privada y
    ' apuntar a la función ToString que hace el trabajo.
    ' De esta forma esta función sólo se ve desde la interfaz (que es lo que queremos)
    ' El nombre no puede ser ToString, porque ya hay una función con la misma firma y
    ' con el mismo nombre, por eso le cambio el nombre y le pongo [ToStringIConvertible]
    ' que considero que es un nombre representativo
    ' ------------------------------------------------------------------------
    Private Overloads Function ToStringIConvertible( _
                    ByVal provider As IFormatProvider) As String _
                    Implements IConvertible.ToString
        '  ------------------------------------------------------------------------
        '  llamar a (4)ToString, La funcion que realmente hace el trabajo de esta interfaz
        Return ToString(provider)
    End Function
```

Ejemplo de una conversión no permitida

```
    Public Function ToDateTime(ByVal provider As IFormatProvider) As DateTime _
           Implements IConvertible.ToDateTime
        ''-------------------------------------------------------
        '' Utilizar la clase Convert para realizar la conversion
        'Return Convert.ToDateTime(Me.Valor(), provider)
        '' Alternativa de Convert no recomendada
        'Return Convert.ToDateTime(Me. Valor())
        '' Forma alternativa que realiza el mismo trabajo
        'Return CType(Me. Valor(), IConvertible).ToDateTime(provider)
        ''-------------------------------------------------------
        Throw New InvalidCastException("Conversión no permitida")
    End Function
```

Ejemplo de una función de conversión que no es compatible con CLS

```
    'No es compatible con Common Language Specification (CLS)
    <CLSCompliantAttribute(False)> _
    Public Function ToSByte(ByVal provider As IFormatProvider) As SByte _
           Implements IConvertible.ToSByte
        '-------------------------------------------------------
        ' Utilizar la clase Convert para realizar la conversion
        Return Convert.ToSByte(Me.Valor(), provider)
        '' Alternativa de Convert no recomendada
        'Return Convert.ToSByte(Me.Valor())
        '' Forma alternativa que realiza el mismo trabajo
        'Return CType(Me.Valor(), IConvertible).ToSByte(provider)
    End Function
```

Implementación Avanzada

Otros métodos ToString se implementen en la interfaz IFormattable

```
#Region "    Implements IFormattable"

    ' ------------------------------------------------------------------------
    ' Interfaz System.IFormattable.ToString
    ' ------------------------------------------------------------------------
    ' Esta función lo que hace es llamar a la función que realmente hace el trabajo
    ' Observa que se utiliza un truco que consiste en declararla como privada y
    ' apuntar a la función ToString que hace el trabajo.
    ' De esta forma esta función sólo se ve desde la interfaz (que es lo que queremos)
    ' El nombre no puede ser ToString, porque ya hay una función con la misma firma y
    ' con el mismo nombre, por eso le cambio el nombre y le pongo [ToStringIFormattable]
    ' que considero que es un nombre representativo
    ' ------------------------------------------------------------------------
    Private Overloads Function ToStringIFormattable( _
                               ByVal format As String, _
                               ByVal formatProvider As System.IFormatProvider) As String _
                               Implements System.IFormattable.ToString
        ' ------------------------------------------------------------------------
        ' llamar a (3) La funcion que realmente hace el trabajo de esta interfaz
        Return Me.ToString(format, formatProvider)
    End Function

#End Region
```

Por último queda por mostrar las funciones ToString que se deben implementar en una clase si hay que sombrear la función ToString de Object

```
#Region "   Las Funciones ToString "

    ' ------------------------------------------------------------------------
    ' Funciones que proporcionan un formato de salida
    ' de Cadena para nuestra Clase
    ' Son cuatro funciones las que debes implementar
    ' ------------------------------------------------------------------------
```

```
    ' ------------------------------------------------------------------------
    ' 1) Sombrear System.Object.ToString
    ' ------------------------------------------------------------------------
    ''' <summary>
    '''  Salida en formato de cadena para los valores de nuestra  clase
    ''' </summary>
    ''' <returns>Una cadena con los datos de la clase</returns>
    ''' <remarks>
    '''   Esta función se declara 'overrides' porque sustituye un método que tiene el
    '''   mismo nombre de la clase base, en este caso la clase base es 'Object'
    '''</remarks>
    Public Overloads Overrides Function ToString() As String
        ''--------------------------------------------------------
        '' Forma elemental de obtener la informacion
        'Return "(" & Me._numerador & ", " & Me._denominador & ")"
        ''--------------------------------------------------------
        Dim formatProvider As System.Globalization.CultureInfo
        formatProvider = System.Globalization.CultureInfo.CurrentCulture
        ' ------------------------------------------------------------------------
        ' llamar a (3) La funcion que realmente hace el trabajo
        Return Me.ToString(String.Empty, formatProvider)

    End Function


    ' ------------------------------------------------------------------------
    ' 2) Function ToString(cadenaFormato)As String
    ' ------------------------------------------------------------------------
    Public Overloads Function ToString(ByVal cadenaFormato As String) As String
        ' ------------------------------------------------------------------------
        Dim formatProvider As System.Globalization.CultureInfo
        formatProvider = System.Globalization.CultureInfo.CurrentCulture
        ' ------------------------------------------------------------------------
        ' llamar a (3) La funcion que realmente hace el trabajo
        Return Me.ToString(String.Empty, formatProvider)
    End Function


    '------------------------------------------------------------------------
    ' Esta es la que realmente hace SIEMPRE el trabajo
    ' ------------------------------------------------------------------------
    ' 3) Funcion que será llamada por la Interfaz IFormattable (Si se Implementa)
    ' ------------------------------------------------------------------------
    Public Overloads Function ToString( _
                    ByVal cadenaFormato As String, _
                    ByVal formatProvider As System.IFormatProvider) As String

        If String.IsNullOrEmpty(cadenaFormato) = True Then
            cadenaFormato = "({0},{1})"
        End If
        Return String.Format(formatProvider, cadenaFormato, Me._numerador, Me._denominador)
    End Function


    ' ------------------------------------------------------------------------
    ' 4) Funcion que será llamada por la Interfaz IConvertible (Si se Implementa)
    '    Esta función tiene que intentar usar la clase [Convert] siempre que sea posible
    ' ------------------------------------------------------------------------
    Public Overloads Function ToString( _
                    ByVal formatProvider As System.IFormatProvider) _
                    As String
        ' ------------------------------------------------------------------------
        ' llamar a (3) La funcion que realmente hace el trabajo
        Return Me.ToString(String.Empty, formatProvider)
    End Function

#End Region
```

A modo de resumen

A continuación se muestra el código completo de la implementación de esa interfaz en la clase del ejemplo.

```
<Serializable()> _
Public Structure NumeroRacional
    Implements IEquatable(Of NumeroRacional)
    Implements ICloneable
    Implements IComparable
    Implements IComparable(Of NumeroRacional)
    Implements IFormattable
    Implements IConvertible


#Region "Campos"
#Region "Constructores"
#Region "Operadores +, -, * , /"
#Region "    Implements IEquatable(Of NumeroRacional)"
#Region "   Las Funciones ToString "
#Region "    Implements IFormattable"
#Region "    Implements ICloneable"
#Region "    Implements IComparable - IComparable(Of NumeroRacional)"
#Region "Operadores >, <, = , <>, >=, <="
#Region " MinusOne, Zero, One, Floor, Ceiling, GetBits, NumeroInverso, Pow, Negate, plus "


#Region "    Implements IConvertible"


    Public Function ToType(ByVal conversionType As Type, _
                           ByVal provider As IFormatProvider) _
                           As Object _
                           Implements IConvertible.ToType

        '-----------------------------------------------------------
        ' Utilizar la clase Convert para realizar la conversion
        Return Convert.ChangeType(Me.Valor(), conversionType, provider)
        '' Alternativa de Convert no recomendada
        'Return Convert.ChangeType(Me.Valor(), conversionType)
        '' Forma alternativa que realiza el mismo trabajo
        'Return CType(Me.Valor(), IConvertible).ToType(conversionType, provider)
    End Function


    Public Function GetTypeCode() As TypeCode _
           Implements IConvertible.GetTypeCode
        '-----------------------------------------------------------
        ' Devuelve siempre un valor Object
        '-----------------------------------------------------------
        'Biblioteca de clases de .NET Framework
        'TypeCode(Enumeración)
        'http://msdn.microsoft.com/es-es/library/system.typecode.aspx
        '-----------------------------------------------------------
        Return TypeCode.Object
    End Function
```

```
Public Function ToBoolean(ByVal provider As IFormatProvider) As Boolean _
       Implements IConvertible.ToBoolean

    '-------------------------------------------------------
    ' Forma personalizada de conversion
    If Me._numerador > 0 And Me._denominador > 0 Then
        Return True
    Else
        Return False
    End If
    ''-------------------------------------------------------
    '' Utilizar la clase Convert para realizar la conversion
    'Return Convert.ToBoolean(Me.Valor(), provider)
    '' Alternativa de Convert no recomendada
    'Return Convert.ToBoolean(Me.Valor())
    '' Forma alternativa que realiza el mismo trabajo
    'Return CType(Me.Valor(), IConvertible).ToBoolean(provider)
    ''-------------------------------------------------------
End Function


Public Function ToByte(ByVal provider As IFormatProvider) As Byte _
       Implements IConvertible.ToByte
    '-------------------------------------------------------
    ' Utilizar la clase Convert para realizar la conversion
    '  Return Convert.ToByte(Me.Valor(), provider)
    '' Alternativa de Convert no recomendada
    'Return Convert.ToByte(Me.Valor())
    '' Forma alternativa que realiza el mismo trabajo
    'Return CType(Me.Valor(), IConvertible).ToByte(provider)
    Throw New InvalidCastException("Conversión no permitida")
End Function



Public Function ToChar(ByVal provider As IFormatProvider) As Char _
       Implements IConvertible.ToChar
    ''-------------------------------------------------------
    '' Utilizar la clase Convert para realizar la conversion
    'Return Convert.ToChar(Me.Valor(), provider)
    '' Alternativa de Convert no recomendada
    'Return Convert.ToChar(Me.Valor())
    '' Forma alternativa que realiza el mismo trabajo
    'Return CType(Me.Valor(), IConvertible).ToChar(provider)
    ''-------------------------------------------------------
    Throw New InvalidCastException("Conversión no permitida")
End Function



Public Function ToDateTime(ByVal provider As IFormatProvider) As DateTime _
       Implements IConvertible.ToDateTime
    Throw New InvalidCastException("Conversión no permitida")
End Function


Public Function ToDecimal(ByVal provider As IFormatProvider) As Decimal _
       Implements IConvertible.ToDecimal
    '-------------------------------------------------------
    ' Utilizar la clase Convert para realizar la conversion
    Return Convert.ToDecimal(Me.Valor(), provider)
End Function


Public Function ToDouble(ByVal provider As IFormatProvider) As Double _
       Implements IConvertible.ToDouble
    ''-------------------------------------------------------
    '' Utilizar la clase Convert para realizar la conversion
    Return Convert.ToDouble(Me.Valor(), provider)
End Function


Public Function ToInt16(ByVal provider As IFormatProvider) As Short _
       Implements IConvertible.ToInt16
    '-------------------------------------------------------
    ' Utilizar la clase Convert para realizar la conversion
    Return Convert.ToInt16(Me.Valor(), provider)
End Function
```

```
Public Function ToInt32(ByVal provider As IFormatProvider) As Integer _
       Implements IConvertible.ToInt32
    '-------------------------------------------------------
    ' Utilizar la clase Convert para realizar la conversion
    Return Convert.ToInt32(Me.Valor(), provider)
End Function


Public Function ToInt64(ByVal provider As IFormatProvider) As Long _
       Implements IConvertible.ToInt64
    '-------------------------------------------------------
    ' Utilizar la clase Convert para realizar la conversion
    Return Convert.ToInt64(Me.Valor(), provider)
End Function


'No es compatible con Common Language Specification (CLS)
<CLSCompliantAttribute(False)> _
Public Function ToSByte(ByVal provider As IFormatProvider) As SByte _
       Implements IConvertible.ToSByte
    '-------------------------------------------------------
    ' Utilizar la clase Convert para realizar la conversion
    Return Convert.ToSByte(Me.Valor(), provider)
End Function


Public Function ToSingle(ByVal provider As IFormatProvider) As Single _
       Implements IConvertible.ToSingle
    '-------------------------------------------------------
    ' Utilizar la clase Convert para realizar la conversion
    Return Convert.ToSingle(Me.Valor(), provider)
End Function


' ------------------------------------------------------------------------
' Interfaz System.IConvertible.ToString
' ------------------------------------------------------------------------
' Esta función lo que hace es llamar a la función que realmente hace el trabajo
' Observa que se utiliza un truco que consiste en declararla como privada y
' apuntar a la función ToString que hace el trabajo.
' De esta forma esta función sólo se ve desde la interfaz (que es lo que queremos)
' El nombre no puede ser ToString, porque ya hay una función con la misma firma y
' con el mismo nombre, por eso le cambio el nombre y le pongo [ToStringIConvertible]
' que considero que es un nombre representativo
' ------------------------------------------------------------------------
Private Overloads Function ToStringIConvertible( _
                ByVal provider As IFormatProvider) As String _
                Implements IConvertible.ToString

    '  ---------------------------------------------------------------------
    '  llamar a (4)ToString, La funcion que realmente hace el trabajo de esta interfaz
    Return ToString(provider)
End Function



'No es compatible con Common Language Specification (CLS)
<CLSCompliantAttribute(False)> _
Public Function ToUInt16(ByVal provider As IFormatProvider) As UInt16 _
       Implements IConvertible.ToUInt16
    '-------------------------------------------------------
    ' Utilizar la clase Convert para realizar la conversion
    Return Convert.ToUInt16(Me.Valor(), provider)
End Function


'No es compatible con Common Language Specification (CLS)
<CLSCompliantAttribute(False)> _
Public Function ToUInt32(ByVal provider As IFormatProvider) As UInt32 _
       Implements IConvertible.ToUInt32
    '-------------------------------------------------------
    ' Utilizar la clase Convert para realizar la conversion
    Return Convert.ToUInt32(Me.Valor(), provider)
End Function
```

```
    'No es compatible con Common Language Specification (CLS)
    <CLSCompliantAttribute(False)> _
    Public Function ToUInt64(ByVal provider As IFormatProvider) As UInt64 _
           Implements IConvertible.ToUInt64
        '---------------------------------------------------------
        ' Utilizar la clase Convert para realizar la conversion
        Return Convert.ToUInt64(Me.Valor(), provider)
    End Function

#End Region

End Structure '/ NumeroRacional
```

La función de prueba

```
Module Module1

    Sub Main()

        Dim testComplex As New NumeroComplejo(4, 7)

        WriteObjectInfo(testComplex)
        Console.Read()
    End Sub

    Sub WriteObjectInfo(ByVal testObject As NumeroComplejo)

        Dim provider As System.Globalization.CultureInfo
        provider = System.Globalization.CultureInfo.CurrentCulture

        Console.WriteLine("GetType:     {0}", testObject.GetType)
        Console.WriteLine("GetTypeCode: {0}", testObject.GetTypeCode)

        Console.WriteLine("ToBoolean:   {0}", testObject.ToBoolean(provider))
        Console.WriteLine("ToByte:      {0}", testObject.ToByte(provider))
        Try
            Console.WriteLine("ToChar:      {0}", testObject.ToChar(provider))
        Catch ex As Exception
            Console.WriteLine("ToChar:      Conversion No Permitida")
        End Try
        Try
            Console.WriteLine("ToDateTime:  {0}", testObject.ToDateTime(provider))
        Catch ex As Exception
            Console.WriteLine("ToDateTime:  Conversion No Permitida")
        End Try

        Console.WriteLine("ToDecimal:   {0}", testObject.ToDecimal(provider))
        Console.WriteLine("ToDouble:    {0}", testObject.ToDouble(provider))
        Console.WriteLine("ToInt16:     {0}", testObject.ToInt16(provider))
        Console.WriteLine("ToInt32:     {0}", testObject.ToInt32(provider))
        Console.WriteLine("ToInt64:     {0}", testObject.ToInt64(provider))
        Console.WriteLine("ToSByte:     {0}", testObject.ToSByte(provider))
        Console.WriteLine("ToSingle:    {0}", testObject.ToSingle(provider))
        Console.WriteLine("ToString:    {0}", testObject.ToString(provider))
        Console.WriteLine("ToUInt16:    {0}", testObject.ToUInt16(provider))
        Console.WriteLine("ToUInt32:    {0}", testObject.ToUInt32(provider))
        Console.WriteLine("ToUInt64:    {0}", testObject.ToUInt64(provider))

    End Sub

End Module
```

Y el resultado obtenido por la función

```
'-------------------------------------------------------
' Salida de esta funcion de prueba
'-------------------------------------------------------
'GetType:     InterfazIConvertible.NumeroComplejo
'GetTypeCode: Object
'ToBoolean:   True
'ToByte:      8
'ToChar:      Conversion No Permitida
'ToDateTime:  Conversion No Permitida
'ToDecimal:   8,06225774829855
'ToDouble:    8,06225774829855
'ToInt16:     8
'ToInt32:     8
'ToInt64:     8
'ToSByte:     8
'ToSingle:    8,062258
'ToString:    (    4,7    )
'ToUInt16:    8
'ToUInt32:    8
'ToUInt64:    8
'-------------------------------------------------------
```

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), como 5/7. El número de la izquierda se denomina numerador y el de la derecha denominador

Una clase que envuelva un número racional es útil porque muchos de estos números NO pueden ser representados exactamente utilizando el tipo Double.
Por ejemplo Realizamos una operación:

- Con Números Racionales 1/3 + 1/3+ 1/3 =1,00
- Con Números Double 0,33333+ 0,33333 + 0,33333 = 0,99999.
- Puedes observar que hay una diferencia en el sexto decimal, que la representación del valor no es exacta con valores Double.

Consideraciones sobre la clase Convert

Microsoft Recomienda implementar estas conversiones usando la clase Convert

La clase Convert devuelve un tipo cuyo valor es equivalente al valor de un tipo especificado. Los tipos base que se admiten son Boolean, Char, SByte, Byte, Int16, Int32, Int64, UInt16, UInt32, UInt64, Single, Double, Decimal, DateTime y String.

Existe un método de conversión para convertir todos y cada uno de los tipos base en los demás tipos base. Sin embargo, la llamada real a un método de conversión determinado puede generar uno de cuatro resultados, dependiendo del valor del tipo base en tiempo de ejecución y el tipo base de destino. Estos cuatro resultados son:

- Ninguna conversión. Esto sucede cuando se intenta convertir un tipo en sí mismo (por ejemplo, llamando a Convert.ToInt32(Int32) con un argumento de tipo (Int32). En este caso, el método devuelve una instancia del tipo original.
- Objeto InvalidCastException. Esto sucede cuando no se admite una conversión determinada. Se produce una InvalidCastException para las conversiones siguientes:
    1. Conversiones de Char a Boolean, Single, Double, Decimal o DateTime.
    2. Conversiones de Boolean, Single, Double, Decimal o DateTime a Char.
    3. Conversiones de DateTime en cualquier otro tipo excepto String.
    4. Conversiones de cualquier otro tipo, excepto String en DateTime.
- Una conversión correcta. Para las conversiones entre dos tipos base diferentes no mostradas en los resultados anteriores, todas las conversiones de ampliación y todas las conversiones de restricción que no producen una pérdida de datos serán correctas y el método devolverá un valor del tipo base concreto.
- Objeto OverflowException. Esto sucede cuando una conversión de restricción produce una pérdida de datos. Por ejemplo, al intentar convertir una instancia Int32 cuyo valor es 10000 en un tipo Byte se produce una OverflowException porque 10000 está fuera del intervalo del tipo de datos Byte.

No se producirá una excepción si la conversión de un tipo numérico produce una pérdida de precisión, es decir, la pérdida de algunos de los dígitos menos significativos. Sin embargo, la excepción se producirá si el resultado es mayor que lo que puede representar el tipo de valor devuelto del método de conversión.

Por ejemplo, cuando un tipo Double se convierte en un tipo Single, se puede producir una pérdida de precisión pero no se produce ninguna excepción. Sin embargo, si la magnitud del tipo Double es demasiado grande para que un tipo Single lo represente, se produce una excepción de desbordamiento.

Existe un conjunto de métodos que admiten la conversión de una matriz de bytes en y desde un tipo String o una matriz de caracteres Unicode formada por dígitos de base 64. Los datos expresados como dígitos de base 64 se pueden transmitir fácilmente en canales de datos que sólo pueden transmitir caracteres de 7 bits.

Algunos de los métodos de esta clase toman un objeto de parámetro que implementa la interfaz IFormatProvider. Este parámetro puede proporcionar información de formato específica de la referencia cultural para ayudar en el proceso de conversión. Los tipos de valor base pasan por alto este parámetro, pero los tipos definidos por el usuario que implementan IConvertible pueden tenerlo en cuenta.

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

- *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. [yyyy-MM-ddThh:mm:ss])
  - *Fecha de Creación*: 2008-09-05T23:07
  - *Fecha de la última modificación*: 2008-10-06T18:53
  - *Fecha de Impresión*: 2008-10-06T18:53

[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]

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

- IConvertible (Interfaz)
  - http://msdn.microsoft.com/es-es/library/system.iconvertible.aspx
- IFormattable (Interfaz)
  - http://msdn.microsoft.com/library/system.iformattable.aspx
- Object (Clase)
  - http://msdn.microsoft.com/es-es/library/system.object.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]
- 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]

-

[Grupo de documentos]

- [Documento Index]
- [Documento Start]