Programación con
Visual Basic .NET

http://alarcos.inf-cr.uclm.es/per/fruiz/cur/vbn/vbn.htm

3 – Orientación a Objetos en Visual Basic .NET

Francisco Ruiz

Manuel Serrano

Escuela Superior de Informática

Universidad de Castilla-La Mancha

Manuel Ángel Serrano Martín

Contacto Personal:

Email: Manuel.Serrano@uclm.es

Web: alarcos.inf-cr.uclm.es/per/mserrano/

Curso: http://alarcos.inf-cr.uclm.es/per/fruiz/cur/vbn/vbn.htm

alarcos.inf-cr.uclm.es

Contenidos sesión 3

- Orientación a Objetos.
- Encapsulación.
- Polimorfismo.
- Espacios de Nombres.
- Herencia.
- Interfaces.
- Estructuras.
- Eventos.
- Clases Predefinidas.

Orientación a Objetos.

Programación Estructurada vs. OO

Ej. Gestión de una Empresa

```
Module General
  Public psNombre As String
  Public Sub Main()
    ' ......
  End Sub
  Public Sub CalcularVacaciones(ByVal liIDEmpleado As Integer, _
    ByVal ldtFechaInicio As Date, ByVal liNumDias As Integer)
  ' ......
  End Sub
  ' otros procedimientos del módulo
  ' ......
End Module

Module Varios
  Public psApellidos As String
  Public Sub CrearEmpleado(ByVal liIDEmpleado As Integer, _
    ByVal lsNombre As String, ByVal lsApellidos As String, _
    ByVal lsDNI As String, ByVal ldtFechaAlta As Date)
    ' ......
  End Sub
  ' otros procedimientos del módulo
  ' ......
End Module
```

```
Module Pagos
  Public Sub TransfNomina(ByVal liIDEmpleado As Integer,
      ByVal _ ldbImporte As Double)
    ' ......
  End Sub
  Public Sub MostrarEmpleado(ByVal liIDEmpleado As
      Integer)
    ' ......
  End Sub
  ' otros procedimientos del módulo
  ' ......
End Module
```

- Variables públicas en los módulos
- Demasiados procedimientos
- Código disperso
- Difícil mantenimiento

Estructuras de código

Conjunto de procedimientos e información que ejecutan una serie de procesos destinados a resolver un grupo de tareas con un denominador común.

Habrá tantas estructuras de código como aspectos del programa sea necesario resolver.

Los procedimientos y los datos que contenga la estructura sólo podrán acceder y ser accedidos por otros procedimientos y datos bajo una serie de reglas.

estructuras de código → objetos

Objetos

Un objeto es una agrupación de código, compuesta de propiedades (atributos) y métodos, que pueden ser manipulados como una entidad independiente.

Las propiedades definen los datos o información del objeto, permitiendo consultar o modificar su estado; mientras que los métodos son rutinas que definen su comportamiento.

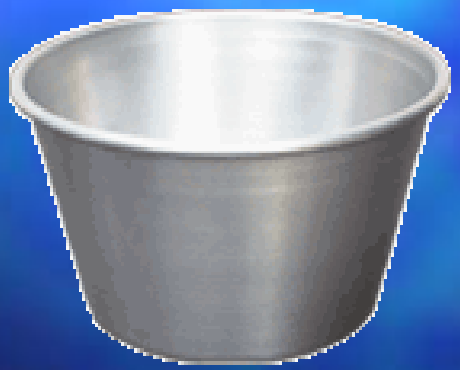
Un objeto desempeña un trabajo concreto dentro de una estructura organizativa de nivel superior, formada por múltiples objetos, cada uno de los cuales ejerce una tarea particular para la cual ha sido diseñado.

Objeto "Empleado"

Clases

Conjunto de especificaciones o normas que definen cómo va a ser creado un objeto de un tipo determinado.

≈ *manual de instrucciones* que contiene las indicaciones para crear y manjar un objeto.

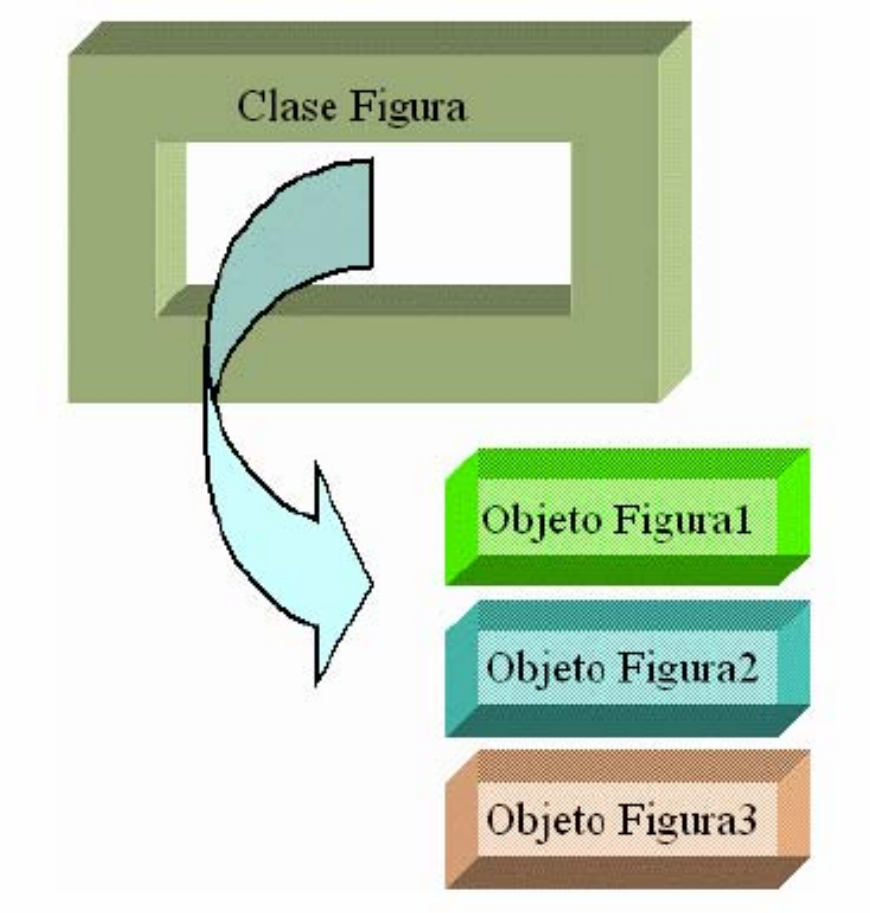


clase

objeto

Orientación a Objetos.

Instancias de una clase



Características básicas de un SOO (i)

- Abstracción
  - Identificación de objetos a través de sus aspectos conceptuales.
  - {Porsche 911, Opel Astra, Seat 600} ϵ **Coche**
- Encapsulación
  - Separación entre el interfaz del objeto y su implementación.
  - Ventajas:
    - Seguridad: evita accesos no deseados.
    - Simplicidad: no es necesario conocer la implementación de un objeto para poder utilizarlo. **Empleado.Alta(id)**

Características básicas de un SOO (ii)

- Polimorfismo
  - Varios métodos con el mismo nombre pueden realizar acciones diferentes.

    Pelota.Tirar() **VasoCristal.Tirar()**
- Herencia
  - Partiendo de una clase (base, padre o superclase) creamos una nueva (derivada, hija o subclase).
  - La clase hija posee TODO el código de la clase padre, más el código propio que se quiera añadir.
  - La clase derivada puede ser, a la vez, base.
  - Simple (.NET) / Múltiple

Características básicas de un SOO (iii)

- Herencia (ejemplos)

Coche (Motor, Ruedas, Volante, Acelerar…)

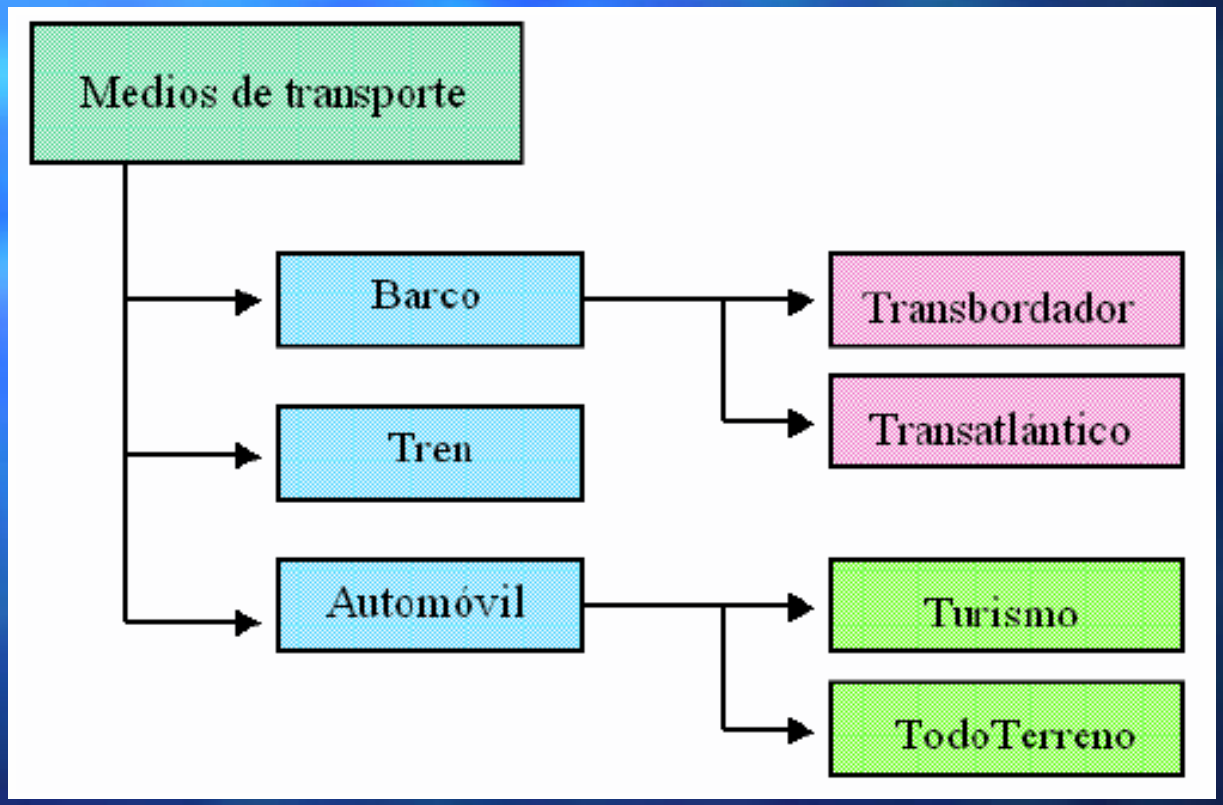
CocheDeportivo (… + ABS, Turbo…)

Empleado (DNI, FechaNacimiento, Fichar, CogerVacaciones…)

CyberEmpleado(… + email, EncriptarDatos…)

Orientación a Objetos.

Jerarquías de clases



Análisis y Diseño OO (i)

www.uml.org

Análisis y Diseño OO (ii)

Crear una aplicación en la que podamos realizar sobre una base de datos, las siguientes operaciones: añadir, borrar y modificar clientes. Por otro lado, será necesario crear facturas, grabando sus datos generales y calcular su importe total.

Programación estructurada:

AñadirCliente( ), BorrarCliente( ), ModificarCliente( ), GrabarFac( ), CalcularTotalFac( )…

Programación OO:

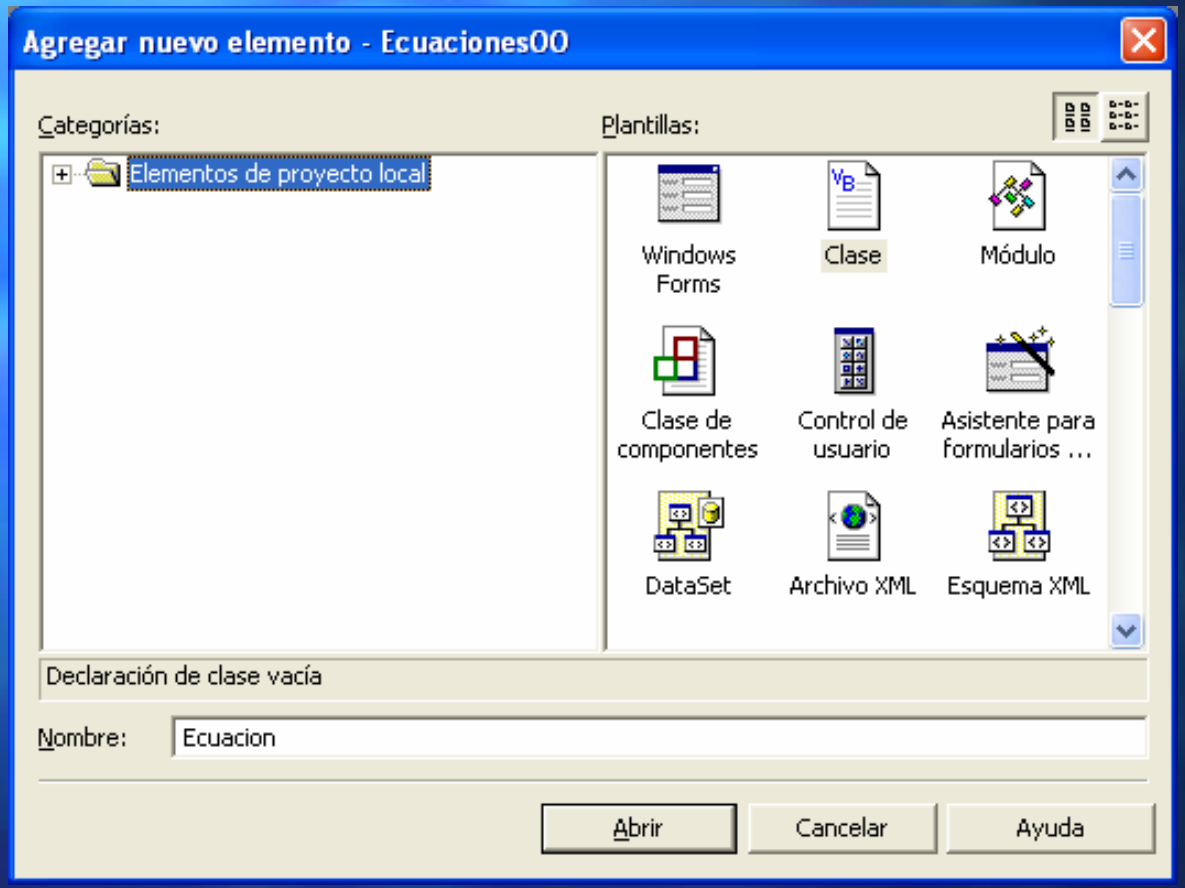
Objetos: Cliente, Factura.

Objeto Cliente: propiedades Nombre, Apellidos, Dirección, DNI, …;
métodos Añadir( ), Borrar( ), Modificar( ), ...

Objeto Factura: propiedades Número, Fecha, Importe, …;
métodos Grabar( ), CalcularTotal ( ), ...

Orientación a Objetos.

Creación de clases



Creación de atributos de una clase

Ecuacion

```
Public Class Ecuacion

    Private ma As Double
    Private mb As Double
    Private mc As Double
    Private mx1 As Double
    Private mx2 As Double
```

Creación de propiedades de una clase

```
Public Property a() As Double
    Get
        Return ma
    End Get
    Set(ByVal Value As Double)
        ma = Value
    End Set
End Property
```

Encapsulación

- El código de una clase debe permanecer protegido de modificaciones no controladas del exterior.
- *caja negra* que expone una interfaz para su uso.

Encapsulación a través de propiedades (i)

```
Module General
    Sub Main()
        Dim loEmpleado As Empleado
        loEmpleado = New Empleado()
        loEmpleado.psNombre = "Juan"
        loEmpleado.piCategoria = 1
        ' atención, el sueldo para este empleado
        ' debería estar entre 1 a 200, debido a su categoría
        loEmpleado.pdbSueldo = 250
    End Sub
End Module

Public Class Empleado
    Public msNombre As String
    Public miCategoria As Integer
    Public mdbSueldo As Double
End Class
```

Encapsulación a través de propiedades (ii)

```
Public Property Sueldo() As Double
    Get
      Return mdbSueldo
    End Get
    ' cuando asignamos el valor a esta propiedad,
    ' ejecutamos código de validación en el bloque Set
    Set(ByVal Value As Double)
      ' si la categoría del empleado es 1...
      If miCategoria = 1 Then
            ' ...pero el sueldo supera 200
            If Value > 200 Then
              ' mostrar un mensaje y asignar un cero
              Console.WriteLine("La categoría no corresponde con el sueldo")
              mdbSueldo = 0
            Else
              ' si todo va bien, asignar el sueldo
              mdbSueldo = Value
            End If
      End If
    End Set
End Property
```

Propiedades de sólo lectura / escritura

```
' esta propiedad sólo permite asignar valores,
' por lo que no dispone de bloque Get
Public WriteOnly Property CuentaBancaria() _
    As String
  Set(ByVal Value As String)
    Select Case Left(Value, 4)
      Case "1111"
        msEntidad = "Banco Universal"
      Case "2222"
        msEntidad = "Banco General"
      Case "3333"
        msEntidad = "Caja Metropolitana"
      Case Else
        msEntidad = "entidad sin catalogar"
    End Select
  End Set
End Property
```

```
' esta propiedad sólo permite obtener
' valores, por lo que no dispone de
'  bloque Set
Public ReadOnly Property EntidadBancaria() _
    As String
  Get
    Return msEntidad
  End Get
End Property
```

Métodos (i)

- With...End With

```
Dim loEmp As Empleado = New Empleado()
With loEmp
        .Nombre = "Ana"
        .Apellidos = "Naranjo"
        .MostrarDatos()
End With
```

Métodos (ii)

- Me / MyClass

```
Public Class Empleado
        Public piID As Integer
        Private msNombre As String
        Public Sub VerDatos()
           ' utilizamos Me y MyClass en este método para tomar el valor de la variable
           ' piID que está en esta misma clase, y para llamar al método NombreMay()
           ' que también está en la clase
           Console.WriteLine("Código del empleado: {0}", Me.piID)
           Console.WriteLine("Nombre del empleado: {0}", MyClass.NombreMay())
        End Sub

        Public Function NombreMay() As String
           Return UCase(msNombre)
        End Function
End Class
```

Polimorfismo.

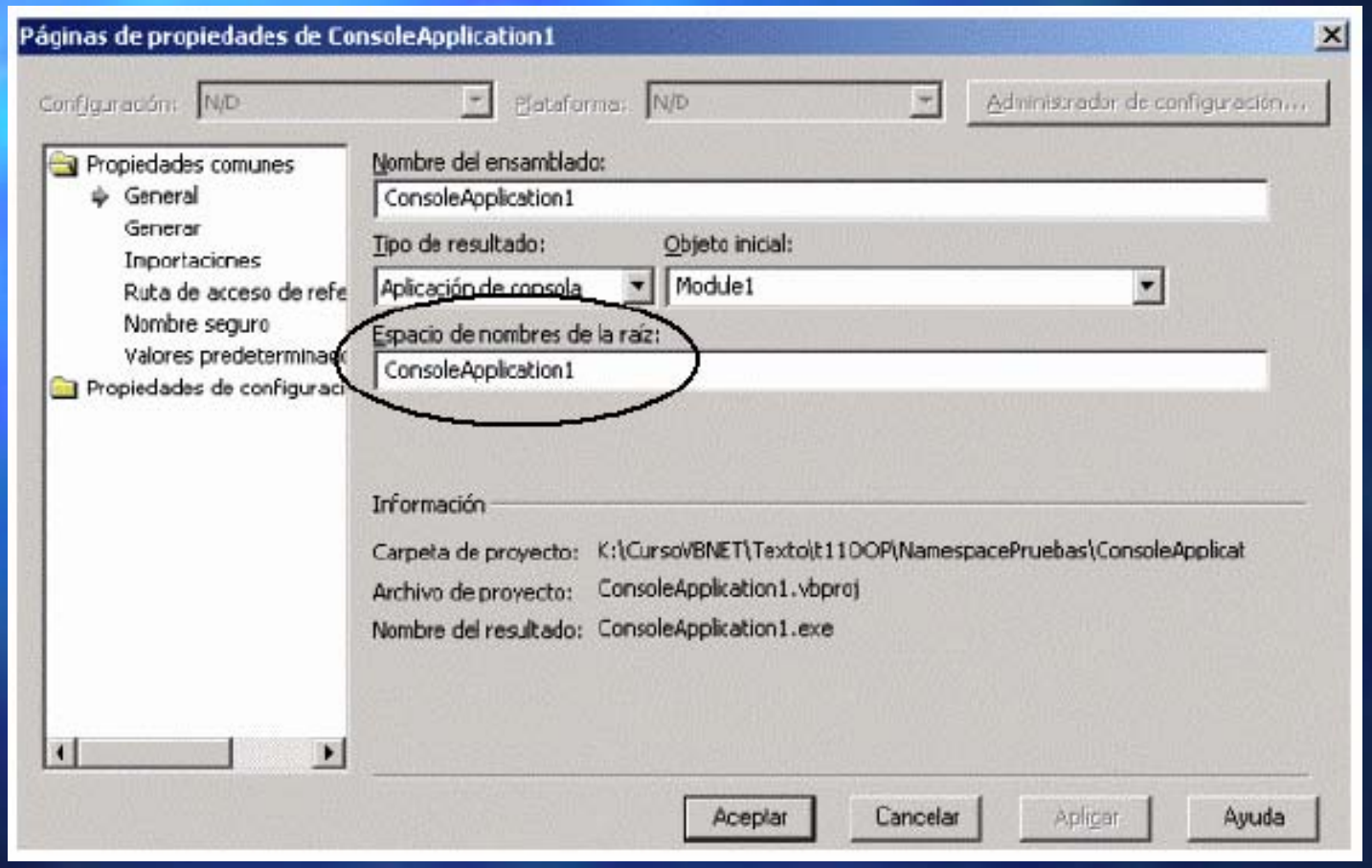
- Varios métodos con idéntico nombre dentro de la misma clase, que se distinguen por su lista de parámetros.
- *Overloads*

Polimorfismo.

Ejemplo.

```
' métodos sobrecargados
Public Overloads Sub Sueldo()
    ' aquí mostramos en consola el importe del sueldo formateado
    Console.WriteLine("El sueldo es {0}", Format(Me.Salario, "#,#.##"))
    Console.ReadLine()
End Sub

Public Overloads Function Sueldo(ByVal liDia As Integer) As String
    ' aquí mostramos la fecha del mes actual en la que se realizará la transferencia
    ' del sueldo al banco del empleado
    Dim ldtFechaActual As Date
    Dim lsFechaCobro As String
    ldtFechaActual = Now()
    lsFechaCobro = CStr(liDia) & "/" & CStr(Month(ldtFechaActual)) & "/" & _
            CStr(Year(ldtFechaActual))
    Return lsFechaCobro
End Function
```

Espacios de Nombres (i).



Espacios de Nombres (ii).

```
Module Module1
    Sub Main()
        ' como la clase Factura se encuentra en el espacio de nombres raíz,
        ' instanciamos normalmente
        Dim loFac As New Factura()
        loFac.piID = 5
        loFac.piImporte = 200
        loFac.Datos()
        Console.ReadLine()
    End Sub
End Module

' clase Factura
' esta clase se encuentra dentro del espacio de nombres raíz del ensamblado
Public Class Factura
    Public piID As Integer
    Public piImporte As Integer
    Public Sub Datos()
        Console.WriteLine("La factura {0}, tiene un importe de {1}", Me.piID, Me.piImporte)
    End Sub
End Class
```

Espacios de Nombres (iii).

```
' clase Empleado
' esta clase se encuentra dentro del espacio de nombres raíz del ensamblado,
' y a su vez, dentro del espacio de nombres Personal
Namespace Personal
    Public Class Empleado
        Public psID As Integer
        Public Sub MostrarDatos()
            Console.WriteLine("Identificador del empleado: {0}", Me.psID)
            Console.ReadLine()
        End Sub
    End Class
End Namespace
```

Espacios de Nombres (iv).

```
' debemos importar el espacio de nombres o no podremos instanciar objetos de las
' clases que contiene
Imports ConsoleApplication1.Personal

Module Module1
    Sub Main()
        ' como hemos importado el espacio de nombres Personal
        ' podemos instanciar un objeto de su clase Empleado
        Dim loEmp As Empleado
        loEmp = New Empleado()
        loEmp.piID = 5
        loEmp.MostrarDatos()
        Console.ReadLine()
    End Sub
End Module
```

Espacios de Nombres (v).

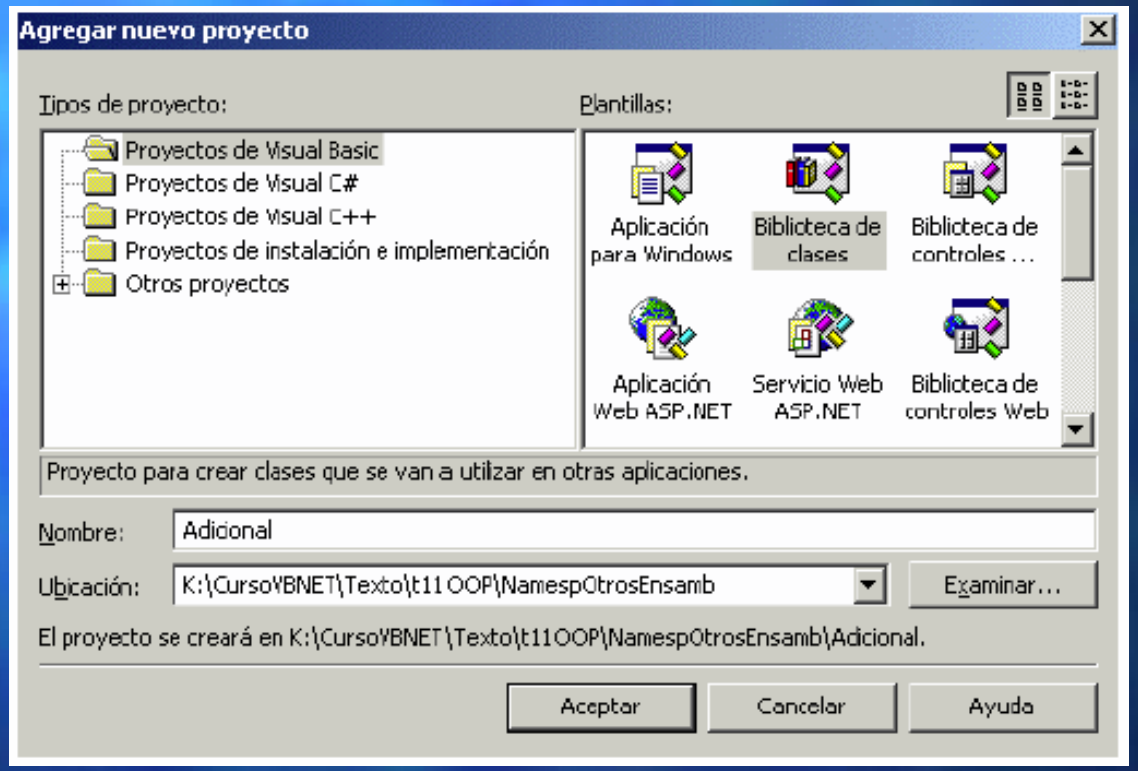
```
Namespace Contabilidad
    Public Class Cuenta
        Public piCodigo As Integer
        Public Function Obtener() As Integer
            Return Me.piCodigo
        End Function
    End Class

    Public Class Balance
        Public psDescripcion As String
        Public Sub MostrarDescrip()
          Console.WriteLine("La descripción del balance es: {0}",Me.psDescripcion)
          Console.ReadLine()
        End Sub
    End Class
End Namespace
```

Espacios de Nombres (vi).

```
Imports ConsoleApplication1.Contabilidad
Module Module1
    Sub Main()
      ' instanciamos con sintaxis calificada
      Dim loCuen As New Contabilidad.Cuenta()
      Dim liDatoCuenta As Integer
      loCuen.piCodigo = 158
      liDatoCuenta = loCuen.Obtener()
      ' al haber importado el espacio de nombres podemos instanciar usando el nombre
      '  de la clase directamente
      Dim loBal As Balance
      loBal = New Balance()
      loBal.psDescripcion = "Resultado trimestral"
      loBal.MostrarDescrip()
      Console.ReadLine()
    End Sub
End Module
```

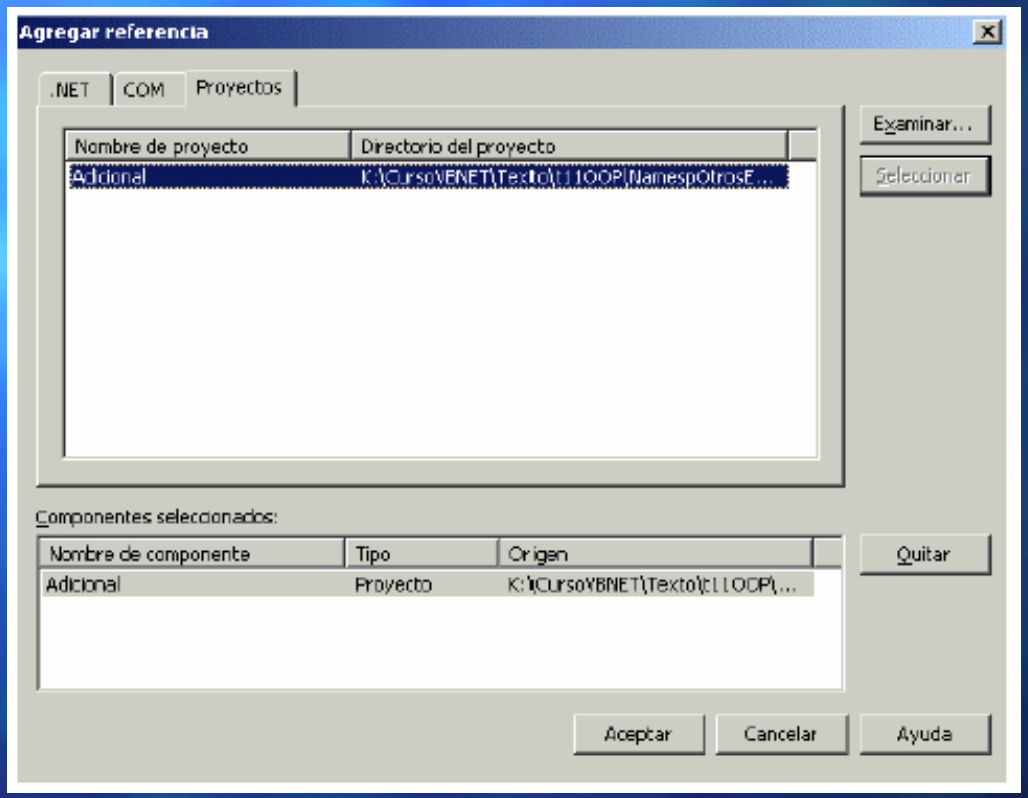
Espacios de Nombres.

Esp. nombres de otros ensamblados (i).



Espacios de Nombres.

Esp. nombres de otros ensamblados (ii).



Métodos constructores.

```
Public Class Empleado

    Private mdtFechaCrea As Date
    Public Property FechaCrea() As Date
            Get
               Return mdtFechaCrea
            End Get
            Set(ByVal Value As Date)
              mdtFechaCrea = Value
            End Set
    End Property

    ' método constructor
    Public Sub New()
            ' asignamos un valor inicial a una variable de propiedad
            Me.FechaCrea = Now
    End Sub

End Class
```

Herencia.

```
' crear clase derivada en dos líneas
Public Class Administrativo
    Inherits Empleado
```

```
' crear clase derivada en la misma línea
Public Class Administrativo : Inherits Empleado
```

```
Public Class Administrativo
    Inherits Empleado
    Public Sub EnviarCorreo(ByVal lsMensaje As String)
        Console.WriteLine("Remitente del mensaje: {0} {1}", _
                Me.Nombre, Me.Apellidos)
        Console.WriteLine("Texto del mensaje: {0}", lsMensaje)
        Console.ReadLine()
    End Sub
End Class
```

Reglas de ámbito (i)

Public vs. Private

```
Module General
    Sub Main()
        Dim loUsu As Usuario
        loUsu = New Usuario()
        ' accedemos al método público del objeto
        loUsu.AsignarNombre("Daniel")
    End Sub
End Module
Public Class Usuario
    ' esta variable sólo es accesible por el código de la propia clase
    Private msNombre As String
    ' este método es accesible desde cualquier punto
    Public Sub AsignarNombre(ByVal lsValor As String)
        msNombre = lsValor
    End Sub
End Class
```

Herencia.

Reglas de ámbito (ii)

Protected

```
Module Module1
  Sub Main()
      ' con una instancia del objeto Empleado
      'o Administrativo no podemos acceder al
      ' método VerFecha() ya que es Protected
      Dim loEmp As Empleado = New Empleado()
      loEmp.psNombre = "Pedro Peral"
      Dim loAdmin As New Administrativo()
      loAdmin.piID = 567
      loAdmin.psNombre = "Juan Iglesias"
      loAdmin.pdtFecha = "5/9/2002"
      loAdmin.AsignarDNI("11223344")
      loAdmin.DatosAdmin()
      Console.Read()
  End Sub
End Module
Public Class Empleado
  Public psNombre As String
  Public pdtFecha As Date
  ' los dos siguientes miembros sólo serán visibles
  ' dentro de esta clase o en sus clases derivadas
  Protected psDNI As String
  Protected Function VerFecha()
      Return pdtFecha
  End Function
  Public Sub AsignarDNI(ByVal lsDNI As String)
      ' desde aquí sí tenemos acceso a la variable
      ' Protected declarada en la clase
      Me.psDNI = lsDNI
  End Sub
End Class
Public Class Administrativo
      Inherits Empleado
  Public piID As Integer
  Public Sub DatosAdmin()
      Console.WriteLine("Datos del administrativo")
      Console.WriteLine("Identificador: {0}", Me.piID)
      Console.WriteLine("Nombre: {0}", Me.psNombre)
      ' desde esta clase derivada sí tenemos acceso
      ' a lo miembtos Protected de la clase padre
      Console.WriteLine("Fecha: {0}", Me.VerFecha())
      Console.WriteLine("DNI: {0}", Me.psDNI)
  End Sub
End Class
```

Herencia.

Reglas de ámbito (iii)

Friend

```
Public Class Empleado
    Public piID As Integer
    Private msNombre As String
    'variable sólo por tipos que estén dentro
    'de este ensamblado
    Friend mdbSueldo As Double
    Public Property Nombre() As String
      Get
            Return msNombre
      End Get
      Set(ByVal Value As String)
            msNombre = Value
      End Set
    End Property

    Public Sub VerDatos()
      Console.WriteLine("Datos del empleado")
      Console.WriteLine("Código: {0}", Me.piID)
      Console.WriteLine("Nombre: {0}", Me.msNombre)
      Console.WriteLine("Sueldo: {0}", Me.mdbSueldo)
    End Sub
End Class
```

Herencia.

Reglas de ámbito (iv)

Friend

```
Public Class Plantilla
  Public Sub Analizar()
    Dim loEmp As Empleado = New Empleado()
    loEmp.piID = 50
    loEmp.Nombre = "Francisco Perea"
    ' desde aquí sí podemos acceder a mdbSueldo, ya
    ' que es el mismo ensamblado
    loEmp.mdbSueldo = 450
    loEmp.VerDatos()
  End Sub
End Class
```

Reglas de ámbito (v)

Friend

```
Imports ClassLibrary1
Module Module1
    Sub Main()
      Dim loEmplea As Empleado = New Empleado()
      ' al acceder a las propiedades del objeto
      ' desde este proyecto, no está disponible
      ' el miembro mdbSueldo ya que está declarado
      ' como Friend en la clase Empleado
      loEmplea.piID = 70
      loEmplea.Nombre = "Alicia Mar"
      loEmplea.VerDatos()
      Console.Read()
    End Sub
End Module
```

Reglas de ámbito (vi)

- Protected Friend
- Herencia + Sobrecarga de métodos
- MyBase

```
Public Class Administrativo
        Inherits Empleado
  Public Overloads Sub CalcularIncentivos(ByVal liHoras As Integer)
  ' llamamos a la clase base con MyBase para hacer en primer lugar los mismos
  ' cálculos de incentivos de la clase Empleado
        MyBase.CalcularIncentivos()
  ' después se hacen los cálculos propios de esta clase
        Me.piIncentivos += liHoras * 15
  End Sub
End Class
```

Herencia.

Ocultamiento de miembros de una clase

- Shadows

```
Public Class Empleado
    '....
    Public Sub Sueldo()
      ' aquí mostramos en consola el importe del sueldo formateado
      Console.WriteLine("El sueldo es {0}", Format(Me.Salario, "#,#.##"))
      Console.ReadLine()
    End Sub
    '....
End Class
Public Class Administrativo
    '....
    ' si aquí no utilizáramos Shadows, el entorno marcaría este método con un aviso
    Public Shadows Sub Sueldo()
            ' aquí incrementamos el valor actual de la propiedad Salario
            Me.Salario += 250
    End Sub
    '....
End Class
```

Herencia.

Clases Selladas y Clases Abstractas

- Selladas: NotInheritable

  Public NotInheritable Class NombreClase

- Abstractas: MustInherit

  Public MustInherit Class NombreClase

Casting.

```
Module Module1
  Public Sub Main()
     Dim loEmple As New Empleado()
     loEmple.piID = 58
     loEmple.psNombre = "Elena Peral"
     ManipularVarios(loEmple)
     Dim loFac As New Factura()
     loFac.pdtFecha = "25/2/2002"
     loFac.piImporte = 475
     ManipularVarios(loFac)
     Console.Read()
  End Sub
```

```
Public Sub ManipularVarios(ByVal loUnObjeto _
          As Object)
    ' obtenemos información sobre el tipo del objeto
    Dim loTipoObj As Type
    loTipoObj = loUnObjeto.GetType()
    ' comprobamos qué tipo de objeto es, y en
    ' función de eso, ejecutamos el método
    ' adecuado
    Select Case loTipoObj.Name
      Case "Empleado"
        CType(loUnObjeto, Empleado).VerDatos()
      Case "Factura"
        CType(loUnObjeto, Factura).Emitir()
    End Select
  End Sub
End Module
```

Interfaces.

- Proporcionan, a modo de declaración, una lista de propiedades y métodos que se codificarán en una o más clases.
- "Contrato":
  - El interfaz no puede cambiarse.
  - La clase que lo implementa se compromete a crear los miembros que forman el interfaz de la manera en que lo indica.

Ejemplo.

```
' las clases que implementen este interfaz deberán crear la propiedad
' Longitud y el método ObtenerValor(); la codificación de dichos
' miembros será particular a cada clase
Public Interface ICadena
    ReadOnly Property Longitud() As Integer
    Function ObtenerValor() As String
End Interface
Public Class Empleado
    Implements ICadena
    '....
    '....
End Class
```

Estructuras.

```
Public Structure DatosBanco
    Public IDCuenta As Integer
    Public Titular As String
    Public Saldo As Integer
End Structure
```

- Estructuras vs. Clases ¿Qué utilizar?

La estructura *DateTime*

- *DateTime.Today*

  devuelve la fecha actual

- *DateTime.DaysInMonth(año, mes)*

  devuelve el número de días que tiene el mes de un año

- *DateTime.Compare(fecha1, fecha2)*

  compara dos fechas

- *objetoDateTime.AddDays(dias)*

  añade a la fecha tantos días como se indiquen

- *objetoDateTime.AddMonths(meses)*

  añade a la fecha tantos meses como se indiquen

- *objetoDateTime.ToLongDateString()*

  formatea la fecha

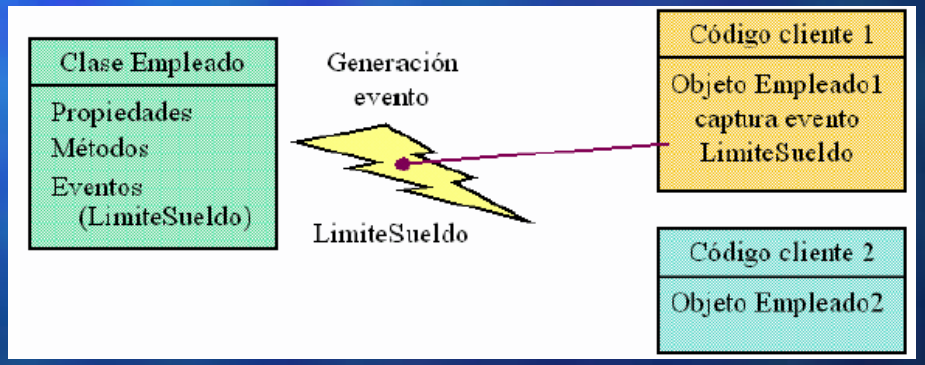
Estructuras.

Enumeraciones

```
Public Enum Musicas As Integer
    Rock ' 0
    Blues ' 1
    NewAge ' 2
    Funky ' 3
End Enum
...
Dim lxMusic as Musicas
lxMusic = Musicas.NewAge
Console.WriteLine(lxMusic)
...
' para obtener las constantes
...System.Enum.GetValues(lxMusic.GetType())
' para obtener los nombres
...System.Enum.GetNames(lxMusic.GetType())
```

Eventos.

- *"Suceso o situación que acontece en una ubicación de espacio y tiempo no predecible."*
- Dentro de una aplicación .NET, es una notificación lanzada por un objeto, que podrá ser respondida por aquellos otros objetos interesados en darle soporte.



Eventos.

Emisión de eventos.

```
Public Class Empleado
' declaramos el evento
  Public Event LimiteSueldo(ByVal _
     ldbImporte As Double)

  Private msNombre As String
  Private mdbSueldo As Double
  Public Property Nombre() As String
     Get
       Return msNombre
     End Get
     Set(ByVal Value As String)
       msNombre = Value
     End Set
  End Property
```

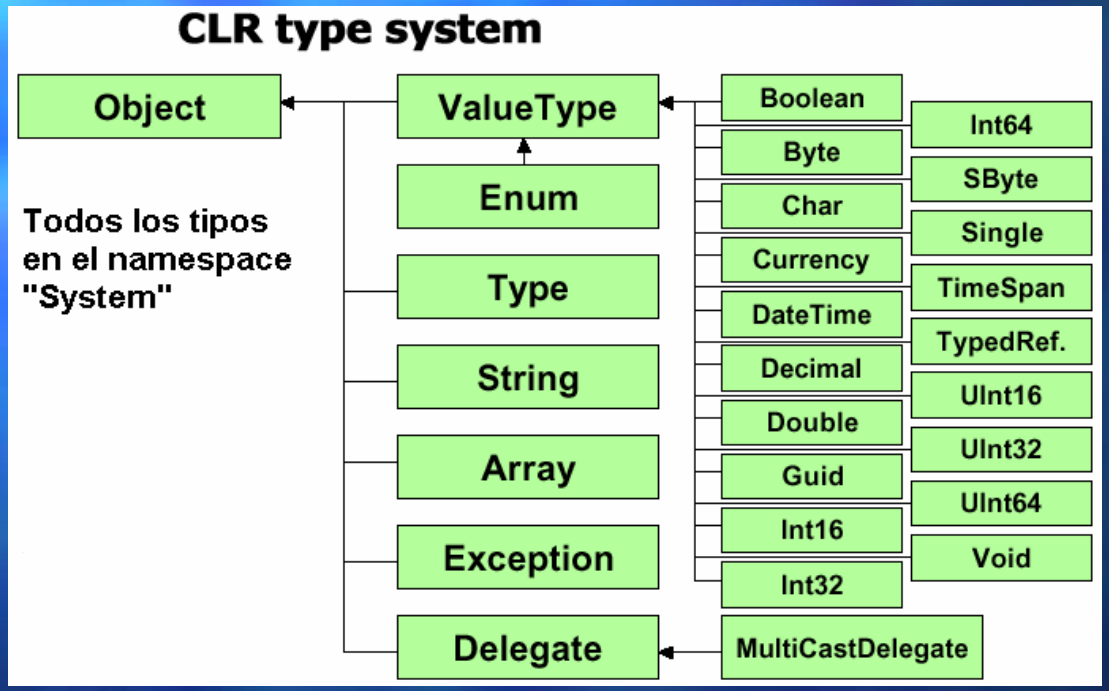
```
Public Property Sueldo() As Double
    Get
      Return mdbSueldo
    End Get
    Set(ByVal Value As Double)
      ' si el valor que intentamos asignar
      ' al sueldo supera el permitido...
      If Value > 1000 Then
      ' ...lanzamos el evento, y le pasamos
      ' como parámetro informativo el valor
      ' incorrecto que intentábamos asignar
            RaiseEvent LimiteSueldo(Value)
      Else
            mdbSueldo = Value
      End If
    End Set
  End Property
End Class
```

Eventos.
Recepción de eventos.

```
Module Module1

    Private WithEvents moEmple As Empleado
    '
     ......
    '
     ......
    Public Sub moEmple_LimiteSueldo(ByVal ldbImporte As Double) _
            Handles moEmple.LimiteSueldo
      Console.WriteLine("Se ha sobrepasado para {0} el límite" & _
            " establecido de sueldo", moEmple.Nombre)
      Console.WriteLine("El importe {0} no es válido", ldbImporte)
      Console.ReadLine()
    End Sub
    '  ...
End Module
```

Clases Predefinidas.

- CTS: Sistema Común de Tipos



Clases Predefinidas.

La clase *String* (i).

```
' modo tradicional
Dim lsCad1 As String
lsCad1 = "mesa"

' instanciar un objeto String y asignar valor
Dim loCad2 As New String("silla")

' declarar variable e instanciar un objeto String en la misma línea
Dim loCad3 As String = New String("coche")

' declarar variable e instanciar un objeto String en la misma línea; el constructor
' utilizado en este caso requiere un array de objetos Char
Dim loCad4 As String = New String(New Char() {"t", "r", "e", "n"})
```

Clases Predefinidas.

La clase *String* (ii).

```
Dim lsCadena As String
lsCadena = "          Hola .NET              "
Dim lsQuitar As String
lsQuitar = lsCadena.TrimEnd()          ' "          Hola .NET"
lsQuitar = lsCadena.TrimStart()        ' "Hola .NET            "
lsQuitar = lsCadena.Trim()             ' "Hola .NET"

Dim lsCadena As String
Dim lsRellena As String
lsCadena = "Hola"
lsRellena = lsCadena.PadLeft(10)                  ' "      Hola"
lsRellena = lsCadena.PadRight(10, "W"c)           ' "HolaWWWWWW"

Dim lsCadena As String
Dim lsAgregar As String
lsCadena = "Estamos programando"
lsAgregar = lsCadena.Insert(2, "HOLA") ' "EsHOLAtamos programando"
```

Clases Predefinidas.

La clase *String* (iii).

```
Dim lsCadena As String
Dim lsQuitar As String
lsCadena = "Estamos programando"
lsQuitar = lsCadena.Remove(5, 3)          ' "Estamprogramando"

Dim lsCadCompleta As String
lsCadCompleta = "En el bosque se alza el castillo negro"
Console.WriteLine("Replace --> {0}", lsCadCompleta.Replace("el", "la"))

Dim lsCadena As String
lsCadena = "veinte"
Console.WriteLine(lsCadena.StartsWith("ve"))        ' True
Console.WriteLine(lsCadena.EndsWith("TE"))          ' False
```

Clases Predefinidas.

La clase *String* (iv).

```
Dim lsCadCompleta As String
lsCadCompleta = "En el bosque se alza el castillo negro"
Console.WriteLine("Substring --> {0}", lsCadCompleta.Substring(6, 5))          ' "bosqu"
Console.WriteLine("IndexOf --> {0}", lsCadCompleta.IndexOf("el"))              ' 3
Console.WriteLine("LastIndexOf --> {0}", lsCadCompleta.LastIndexOf("el"))      ' 21

Dim lsCadMayMin As String
lsCadMayMin = "CambIaNDO A mayúsCUlAs Y MINúscULAS"
Console.WriteLine("Pasar a may. --> {0}", lsCadMayMin.ToUpper())
Console.WriteLine("Pasar a min. --> {0}", lsCadMayMin.ToLower())

Dim lsConcatenar As String
lsConcatenar = String.Concat("Hola ", "a todos")
lsConcatenar = "ahora usamos" & " el operador para concatenar"
```

Clases Predefinidas.

La clase *String* (v).

```
Dim lsCadA As String = "uno"
Dim lsCadB As String = String.Copy("OTRO")
Console.WriteLine("CadenaA --> {0}", lsCadA)
Console.WriteLine("CadenaB --> {0}", lsCadB)

Dim lsCompara1, lsCompara2 As String
Dim liResultaComp As Integer
Console.WriteLine("Introducir primera cadena a comparar")
lsCompara1 = Console.ReadLine()
Console.WriteLine("Introducir segunda cadena a comparar")
lsCompara2 = Console.ReadLine()
liResultaComp = String.Compare(lsCompara1, lsCompara2)
Select Case liResultaComp
    Case Is < 0 : Console.WriteLine("Primera cadena es menor")
    Case 0 : Console.WriteLine("Las cadenas son iguales")
    Case Is > 0 : Console.WriteLine("Primera cadena es mayor")
End Select
```

Clases Predefinidas.

La clase *String* (vi).

```
Dim lsCadInicial As String
Dim lsCadComparar As String
lsCadInicial = "Prueba"
Console.WriteLine("Introducir una cadena a comparar con la cadena inicial")
lsCadComparar = Console.ReadLine()
If lsCadInicial.Equals(lsCadComparar) Then
    Console.WriteLine("Las cadenas son iguales")
Else
    Console.WriteLine("Las cadenas son diferentes")
End If
```

- La clase Convert

```
Dim lsCadena As String
lsCadena = Convert.ToString(150)        ' "150"
Dim liNum As Integer
liNum = Convert.ToInt32(lsCadena)       ' 150
```

Clases Predefinidas.

La estructura *Char* (i).

```
Public Sub Main()
    Dim lcCaracter As Char
    Dim lsResultado As String
    Dim lcConvertido As Char

    Do
      Console.WriteLine("Introducir un carácter o cero para salir")
      lcCaracter = Convert.ToChar(Console.ReadLine())
      lsResultado = ""
      lcConvertido = Nothing
      ' IsDigit() indica si el carácter es un dígito decimal
      If Char.IsDigit(lcCaracter) Then
            lsResultado = "dígito"
      End If
      ' IsLetter() indica si el carácter es una letra
      If Char.IsLetter(lcCaracter) Then
            lsResultado = "letra"
      End If
```

Clases Predefinidas.

La estructura *Char* (ii).

```
' IsWhiteSpace() indica si el carácter es un espacio en blanco
If Char.IsWhiteSpace(lcCaracter) Then
        lsResultado = "espacio"
End If
' IsPunctuation() indica si el carácter es un signo de puntuación
If Char.IsPunctuation(lcCaracter) Then
        lsResultado &= "puntuación"
End If
' IsUpper() comprueba si el carácter está en mayúscula
If Char.IsUpper(lcCaracter) Then
        lsResultado &= " mayúscula"
        ' ToLower() convierte el carácter a minúscula
        lcConvertido = Char.ToLower(lcCaracter)
End If
' IsLower() comprueba si el carácter está en minúscula
If Char.IsLower(lcCaracter) Then
        lsResultado &= " minúscula"
        ' ToUpper() convierte el carácter a mayúscula
        lcConvertido = Char.ToUpper(lcCaracter)
End If
```

Clases Predefinidas.

La estructura *Char* (iii).

```
        ' mostramos una cadena con el tipo de carácter
        Console.WriteLine("El carácter es: {0}", lsResultado)
        ' si hemos convertido el caracter a mayúscula/minúscula, lo mostramos
        If Char.IsLetter(lcConvertido) Then
            Console.WriteLine("El carácter se ha convertido: {0}", lcConvertido)
        End If
        Console.WriteLine()
        ' no salimos hasta que no se introduzca un 0
    Loop Until lcCaracter = "0"c
End Sub
```

Clases Predefinidas.

La clase *Math* (i).

```
Sub Main()
    Dim liSigno As Integer
    Dim ldbRedondear As Double
    ' Abs(): devuelve el valor absoluto del número pasado como parámetro
    Console.WriteLine("Abs --> {0}", Math.Abs(-1867.79))
    ' Ceiling(): devuelve el número sin precisión decimal, más grande o igual que el
    ' pasado como parámetro
    Console.WriteLine("Ceiling --> {0}", Math.Ceiling(256.7235))
    ' Floor(): devuelve el número sin precisión decimal,
    ' más pequeño o igual que el pasado como parámetro
    Console.WriteLine("Floor --> {0}", Math.Floor(256.7235))
```

Clases Predefinidas.

La clase *Math* (ii).

```
    ' Sign(): devuelve un valor informando del signo del número pasado como parámetro
    Console.WriteLine("Introducir número para averiguar su signo")
    liSigno = Console.ReadLine()
    Select Case Math.Sign(liSigno)
      Case -1
            Console.WriteLine("El número es negativo")
      Case 0
            Console.WriteLine("El número es cero")
      Case 1
            Console.WriteLine("El número es positivo")
    End Select
    ' Round(): redondea el número pasado como parámetro
    ldbRedondear = Math.Round(28.3215)
    Console.WriteLine("Redondear 28.3215 --> {0}", ldbRedondear)
    ldbRedondear = Math.Round(28.63215)
    Console.WriteLine("Redondear 28.63215 --> {0}", ldbRedondear)
    Console.ReadLine()
End Sub
```

Clases Predefinidas.

Formateo de Fechas.

```
Sub Main()
        Dim ldtFecha As Date
        Dim lsListaFormatos() As String = {"d", "D", "g", "G", "t", "T", "m", "y"}
        Dim lsFormato As String
        ldtFecha = Date.Now()
        For Each lsFormato In lsListaFormatos
            Console.WriteLine("Formato: {0}, resultado: {1}", lsFormato, ldtFecha.ToString(lsFormato))
        Next
End Sub

Sub Main()
        Dim ldtFecha As Date
        ldtFecha = Date.Now()
        ' array para obtener todos los formatos de fecha del sistema
        Dim lsListaFormatos() As String
        lsListaFormatos = ldtFecha.GetDateTimeFormats()
        Dim lsFormato As String
        For Each lsFormato In lsListaFormatos
            Console.WriteLine(lsFormato)
        Next
        Console.ReadLine()
End Sub
```

Clases Predefinidas.

Arrays.

- Permiten agrupar un conjunto de valores del mismo tipo y acceder a ellos a través de un único identificador, especificando el índice donde se encuentra el dato a recuperar.

```
Sub Main()
  ' declarar un array de tipo String, el número de elementos es el indicado en la declaración más
  ' uno,  porque la primera posición de un array es cero
  Dim sNombres(3) As String
  ' asignar valores al array
  sNombres(0) = "Ana" : sNombres(1) = "Pedro" : sNombres(2) = "Antonio" : sNombres(3) = "Laura"
  ' pasar un valor del array a una variable
  Dim sValor As String
  sValor = sNombres(2)
  ' mostrar en la consola el valor pasado a una variable y un valor directamente desde el array
  Console.WriteLine("Valor de la variable sValor: {0}", sValor)
  Console.WriteLine("Valor del array, posición 1: {0}", sNombres(1))
  Console.ReadLine()
End Sub
```

Clases Predefinidas.

La clase *Array* (i).

- El primer índice del Array es siempre 0.
- No pueden crearse Arrays con rangos de índices.
- Todos los Arrays son dinámicos.
- Declaración:

```
Sub Main()
        ' 1)  estableciendo el número de elementos
        Dim sNombres(2) As String
        ' 2)  asignando valores al array al mismo tiempo que se declara
        Dim sEstaciones() As String = {"Ana", "Pedro", "Luis"}
        ' 3)  indicando el tipo de dato pero no el número de elementos
        Dim iValores() As Integer
        ' 4)  indicando el tipo de dato y estableciendo una lista vacía de elementos,
        Dim iDatos() As Integer = {}
        ' 5)  instanciando el tipo de dato, estableciendo el número de elementos al instanciar,
        ' e indicando que se trata de un array al situar las llaves
        Dim iCantidades() As Integer = New Integer(20) {}
        ' 6) declarar primero la variable que contendrá el array, asignar valores al array al mismo tiempo que se instancia
        Dim iNumeros() As Integer
        iNumeros = New Integer() {10, 20, 30, 10, 50, 60, 10, 70, 80}
End Sub
```

Clases Predefinidas.

La clase *Array* (ii).

- Asignación y obtención de valores

```
Sub Main()

    Dim sNombres(4) As String
    ' directamente sobre la variable, haciendo referencia al índice
    sNombres(0) = "Juan" : sNombres(1) = "Ana" : sNombres(2) = "Luis"
    ' o con el método SetValue(), asignando el valor en el primer parámetro y especificando
    ' la posición en el segundo
    sNombres.SetValue("Elena", 3) : sNombres.SetValue("Miguel", 4)

    Dim sValorA As String
    Dim sValorB As String
    sValorA = sNombres(2)                    ' directamente de la variable
    sValorB = sNombres.GetValue(3)                    ' usando el meth GetValue
    Console.WriteLine("Contenido de las variables")
    Console.WriteLine("ValorA: {0} -- ValorB: {1}", sValorA, sValorB)
    Console.ReadLine()

End Sub
```

Clases Predefinidas.

La clase *Array* (iii).

- Recorrer el contenido

```
Sub Main()

    Dim sNombres() As String = {"Ana", "Luis", "Pablo"}
    Dim iContador As Integer
    Dim sUnNombre As String

    ' modo tradicional
    Console.WriteLine("Recorrido del array con LBound() y UBound()")
    For iContador = LBound(sNombres) To UBound(sNombres)
            Console.WriteLine("Posicion: {0} - Valor: {1}", iContador, sNombres(iContador))
    Next
    Console.WriteLine()

    ' con bucle For Each
    Console.WriteLine("Recorrido del array con bucle For Each")
    For Each sUnNombre In sNombres
            Console.WriteLine("Nombre actual: {0}", sUnNombre)
    Next
    Console.WriteLine()
```

Clases Predefinidas.

La clase *Array* (iv).

- Recorrer el contenido (cont.)

```
' usando la propiedad Length
Console.WriteLine("Recorrido del array con propiedad Length")
For iContador = 0 To (sNombres.Length - 1)
        Console.WriteLine("Posicion: {0} - Valor: {1}", iContador, sNombres(iContador))
Next
Console.WriteLine()

' usando los métodos GetLowerBound() y GetUpperBound()
Console.WriteLine("Recorrido del array con métodos GetLowerBound() y GetUpperBound()")
For iContador = sNombres.GetLowerBound(0) To sNombres.GetUpperBound(0)
        Console.WriteLine("Posicion: {0} - Valor: {1}", iContador, sNombres(iContador))
Next
Console.WriteLine()
```

Clases Predefinidas.

La clase *Array* (v).

- Recorrer el contenido (cont.)

```
' recorrer con un enumerador
Console.WriteLine("Recorrido del array con un enumerador")
Dim sLetras() As String = {"a", "b", "c", "d"}
Dim oEnumerador As System.Collections.IEnumerator
' obtener el enumerador del array
oEnumerador = sLetras.GetEnumerator()
' con un enumerador no es necesario posicionarse en el primer elemento ni calcular la cantidad
' de elementos del array, sólo hemos de avanzar posiciones con MoveNext() y obtener el valor
' actual con Current
While oEnumerador.MoveNext()
        Console.WriteLine("Valor actual: {0}", oEnumerador.Current)
End While
Console.ReadLine()
End Sub
```

Clases Predefinidas.

La clase *Array* (vi).

- Modificar el tamaño (*ReDim [Preserve]*)

- Array.Clone()
- Array.Sort() Array.Reverse()
- Array.IndexOf() Array.LastIndexOf()

- Arrays Multidimensionales

Dim iDatos(2, 4) As Integer

(0,0)	(0,1)	(0,2)	(0,3)	(0,4)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)

Clases Predefinidas.

La clase *Array* (vii).

- Arrays Multidimensionales

- Array.Rank()

  Devuelve el número de dimensiones.

- Array.GetLength*(dimension)*

  Devuelve el número de elementos de una dimensión.

- Array.GetLowerBound*(dimension)*

  Devuelve el número de índice inferior de una dimensión.

- Array.GetUpperBound*(dimension)*

  Devuelve el número de índice superior de una dimensión.

Clases Predefinidas.

Colecciones.

- Objetos que, internamente, gestionan arrays, pero que están preparados para gestionarlos de forma optimizada.
- Espacio de Nombres System.Collections
  - ArrayList
  - Hashtable
  - SortedList
  - Queue
  - Stack

Clases Predefinidas.

Colecciones. La clase *ArrayList* (i)

- Array cuyo número de elementos puede modificarse dinámicamente.

```
Sub Main()
    Dim alDatos As New ArrayList(10)
    alDatos.Add("a")
    alDatos.AddRange(New String() {"b", "c", "d"})
    Console.WriteLine("ArrayList después de usar Add() y AddRange()")
    RecorrerAList(alDatos)
    alDatos.Insert(2, "hola")
    Console.WriteLine("ArrayList después de usar Insert()")
    RecorrerAList(alDatos)
    alDatos.InsertRange(1, New Integer() {55, 77, 88})
    Console.WriteLine("ArrayList después de usar InsertRange()")
    RecorrerAList(alDatos)
    alDatos.SetRange(3, New String() {"zzz", "yyy"})
    Console.WriteLine("ArrayList después de usar SetRange()")
    RecorrerAList(alDatos)
    Console.ReadLine()
End Sub
```

Clases Predefinidas.

Colecciones. La clase *ArrayList* (ii)

```
Private Sub RecorrerAList(ByVal alValores As ArrayList)
    Dim oEnumerador As IEnumerator = alValores.GetEnumerator()
    While oEnumerador.MoveNext()
        Console.WriteLine("Valor: {0}", oEnumerador.Current)
    End While
    Console.WriteLine()
End Sub
```

- ArrayList.GetRange *(posicion, n)*

  Obtiene un subarray comenzando por *posicion* y tomando *n* elementos.
- ArrayList.FixedSize *(arraylist)*

  Crea un array de tamaño fijo a partir de otro. No se pueden añadir elementos.
- ArrayList.Repeat *(valor, cantidad)*

  Crea un array de *cantidad* valores repetidos y con el *valor* indicado.
- ArrayList.ToArray *()*

  Copia los elementos del ArrayList en un Array.

Clases Predefinidas.

Colecciones. La clase *ArrayList* (iii)

- ArrayList.Contains *(patron)*
  Indica si *patron* existe en el array.

- ArrayList.Remove *(valor)*
  Elimina un elemento.
- ArrayList.RemoveAt *(posicion)*
  Elimina el elemento situado en el índice indicado.
- ArrayList.RemoveRange *(posicion, n)*
  Elimina un conjunto de *n* elementos comenzando por *posicion*.
- ArrayList.Clear *()*
  Elimina todos los elementos del objeto.

Clases Predefinidas.

Colecciones. La clase *Hashtable* (i)

- El acceso a los valores del array se realiza a través de una clave asociada a cada elemento.

```
Sub Main()
    ' declarar colección Hashtable
    Dim htCliente As Hashtable
    htCliente = New Hashtable()
    ' añadir valores
    htCliente.Add("ID", 22)
    htCliente.Add("Nombre", "Pedro")
    htCliente.Add("Apellidos", "Naranjo")
    htCliente.Add("Domicilio", "C/Rio Bravo, 25")
    htCliente.Add("Edad", 35)
    htCliente.Add("Credito", 250)
    ' mostrar el número de elementos que contiene
    Console.WriteLine("El objeto Hashtable tiene {0} elementos", htCliente.Count)
    Console.WriteLine()
```

Clases Predefinidas.

Colecciones. La clase *Hashtable* (ii)

```
        ' obtener algunos valores
        Console.WriteLine("Obtener algunos valores del objeto Hashtable")
        Console.WriteLine("Domicilio: {0} ", htCliente.Item("Domicilio"))
        Console.WriteLine("Nombre: {0} ", htCliente("Nombre"))
        Console.WriteLine("Credito: {0} ", htCliente("Credito"))
        Console.WriteLine()
        ' recorrer el array al completo
        Console.WriteLine("Recorrer objeto Hashtable con un enumerador")
        Dim oEnumerador As IDictionaryEnumerator
        oEnumerador = htCliente.GetEnumerator()
        While oEnumerador.MoveNext()
                Console.WriteLine("Clave: {0} / Valor: {1}", oEnumerador.Key, oEnumerador.Value)
        End While
        Console.ReadLine()
End Sub
```

Clases Predefinidas.

Colecciones. La clase *Hashtable* (iii)

Hashtable.ContainsKey*(clave)*

Comprueba que una clave está en la tabla

Hashtable.ContainsValue*(valor)*

Comprueba que un valor está en la tabla

Hashtable.Remove*(clave)*

Elimina un valor de la tabla

Hashtable.Clear*()*

Elimina todos los valores de la tabla

Hashtable.Keys*()*

Devuelve un array con los nombres de las claves de la tabla

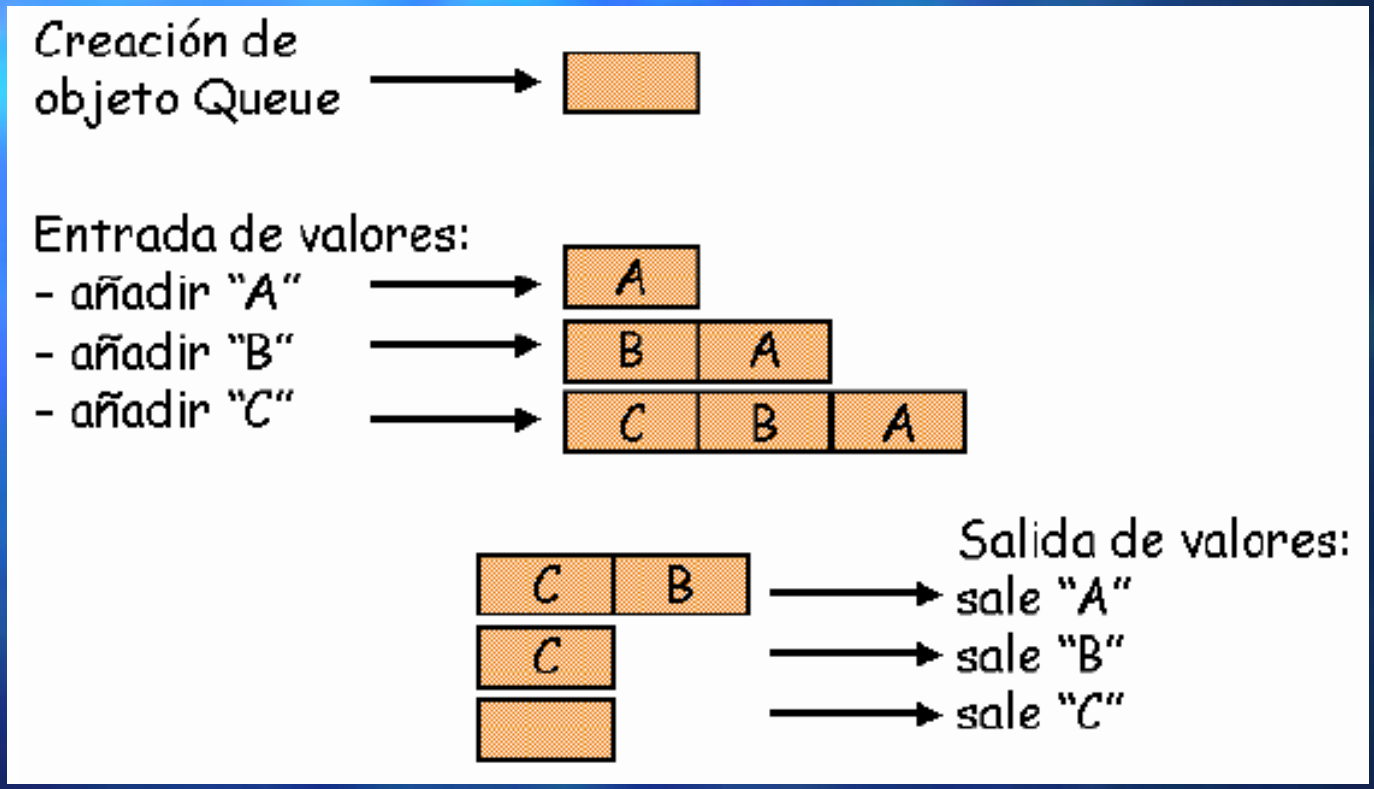
Hashtable.Values*()*

Devuelve un array con los nombres de los valores de la tabla

Colecciones. La clase *SortedList*

- Variación de una Hashtable en la que los elementos se ordenan por la clave según van siendo agregados.
- Funcionamiento similar a una Hashtable.

Colecciones. La clase *Queue* (i)



Clases Predefinidas.

Colecciones. La clase *Queue* (ii)

```
Sub Main()
    ' crear objeto Queue, cola de valores
    Dim aqListaMensa As New Queue()
    Console.WriteLine("Introducir mensajes")
    Dim sMensaje As String
    ' bucle de recepción de mensajes
    Do
        sMensaje = Console.ReadLine()
        ' si hemos escrito algo...
        If sMensaje.Length > 0 Then
          ' añadimos a la cola
          aqListaMensa.Enqueue(sMensaje)
        Else
          ' salimos
          Exit Do
        End If
    Loop
```

Clases Predefinidas.

Colecciones. La clase *Queue* (iii)

```
        ' la propiedad Count nos indica la cantidad de elementos en la lista
        Console.WriteLine("Hay {0} mensajes para procesar", aqListaMensa.Count)
        ' con un enumerador recorremos la lista
        Dim oEnumerador = aqListaMensa.GetEnumerator()
        Console.WriteLine("Contenido del objeto Queue")
        RecorrerEnumerador(oEnumerador)
        Console.ReadLine()
End Sub

Public Sub RecorrerEnumerador(ByVal oEnumerador As IEnumerator)
        While oEnumerador.MoveNext
            Console.WriteLine(oEnumerador.Current)
        End While
        Console.WriteLine()
End Sub
```

Clases Predefinidas.

Colecciones. La clase *Queue* (iv)

```
Sub Main()
    Dim aqListaMensa As New Queue()
    Dim sMensaje As String
    Dim iContador As Integer
    ' bucle de recepción de mensajes
    Do
      iContador += 1
      Console.WriteLine("Mensaje nro. {0}. " & "Pulse [INTRO] para finalizar captura", _
            iContador)
      sMensaje = Console.ReadLine()
      ' si hemos escrito algo...
      If sMensaje.Length > 0 Then
            ' añadimos a la lista
            aqListaMensa.Enqueue(sMensaje)
      Else
            ' salimos
            Exit Do
      End If
    Loop
```

Colecciones. La clase *Queue* (v)

```
    Console.WriteLine()
    ' la propiedad Count nos indica la cantidad de elementos en la lista
    Console.WriteLine("Hay {0} mensajes para procesar", aqListaMensa.Count)
    Console.WriteLine()
    ' procesar los mensajes de la lista
    iContador = 0
    Console.WriteLine("Procesar lista de mensajes")
    While aqListaMensa.Count > 0
      iContador += 1
      Console.WriteLine("Mensaje nro. {0} - texto: {1}", _
            iContador, aqListaMensa.Dequeue())
      Console.WriteLine("Quedan {0} mensajes por procesar", aqListaMensa.Count)
      Console.WriteLine()
    End While
    Console.ReadLine()
End Sub
```

Clases Predefinidas.

Colecciones. La clase *Stack*

```
Public Sub Main()
    ' creamos una colección de tipo pila
    Dim oPila As Stack
    oPila = New Stack()
    ' para añadir valores, usamos el método Push()
    oPila.Push("A") ' este será el último en salir
    oPila.Push("B")
    oPila.Push("C")
    oPila.Push("D")
    oPila.Push("E") ' este será el primero en salir
    ' para extraer un valor de la pila, usamos el método Pop(), dicho valor se eliminará de la lista
    While oPila.Count > 0
            Console.WriteLine("El valor obtenido de la lista es: {0}", oPila.Pop)
    End While
    Console.ReadLine()
End Sub
```

Ejercicio

- Realizar una pequeña aplicación, que cumpla los siguientes requisitos:
  - Debe aceptar una cadena de texto del usuario
  - Debe devolver los siguientes valores:
    - La cadena sin espacios
    - La cadena en la que se substituyen los espacios por #
    - Los 3 primeros caracteres de la cadena
    - El número de caracteres de la cadena
    - La cadena invertida (haced uso de la clase stack)