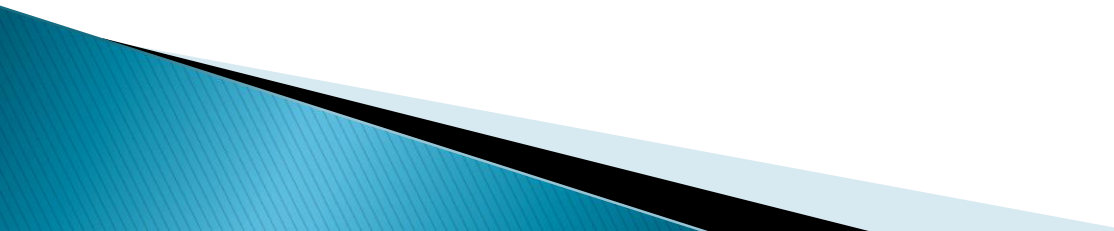


Visual Basic .Net

Programación Orientada a Objetos

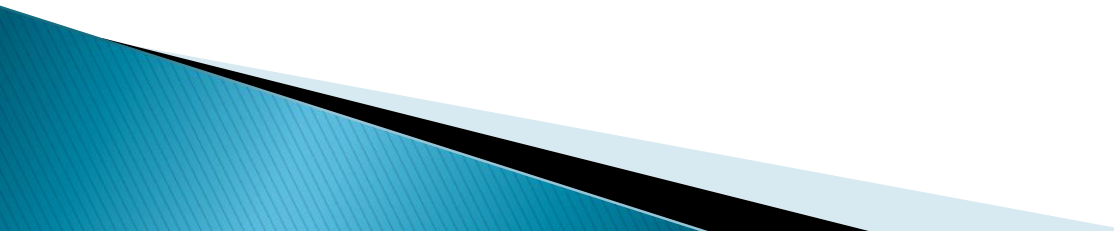
Objetos

- ▶ Un objeto puede ser visto como una entidad que posee atributos y efectúa acciones
 - ▶ Los **objetos** son modelados a partir de los objetos reales, basándose en sus características y comportamientos.
 - Las **propiedades** de un objeto se almacenan en **variables**.
 - Los **comportamientos** se implementan utilizando los **métodos**, que son funciones o procedimientos asociados a un objeto.
- 

Clase

- ▶ Una clase es un molde o bien prototipo en donde se definen los atributos (variables) y las acciones (métodos) comunes de una entidad.

Características básicas de un sistema orientado a objetos

- ▶ Abstracción
 - ▶ Modularidad
 - ▶ Herencia
 - ▶ Encapsulamiento
 - ▶ Polimorfismo
- 

Clases

- ▶ Una clase tiene dos partes claramente diferenciadas
 - los atributos
 - los métodos.
- ▶ **Los atributos serían las variables del objeto mientras que los métodos son las funciones que manipulan dichos variables**

Clases

► Definición

```
Ámbito Class nombre_clase  
    Ámbito componente  
    Ámbito componente  
End Class
```

Classes

- ▶ Mas formalmente

```
[ <attributelist> ] [ accessmodifier ] [ Shadows ]  
[ MustInherit | NotInheritable ] [ Partial ] _  
Class name [ ( Of typelist ) ]  
    [ Inherits classname ]  
    [ Implements interfacenames ]  
    [ statements ]  
End Class
```

Clases

► Instanciación

- Un **objeto** es una particularización o **instancia** de una clase, por tanto un objeto tiene existencia real y determinada
- para instanciarlo tenemos que llamar al método **new**, y hay dos modos de realizarlo:
- Modo1
 - Dim objeto as NombreClase
 - objeto = **New** NombreClase()
- Modo2
 - Dim objeto as Nombreclase = **New** NombreClase

Classes

Class persona

```
Private _Nombre As String
Private _Domicilio As String
Private _Telefono As String
Private _edad As Byte
Public Sub nombre()
```

```
Console.Write("Introduzca el  
nombre : ")
```

```
_Nombre = Console.ReadLine
```

End Sub

Public Sub Domicilio()

```
Console.WriteLine("Introduzca la  
direccion :")
```

```
_Domicilio = Console.ReadLine
```

End Sub

Public Sub ListarDatos()

Console.WriteLine()

```
Console.WriteLine("Datos  
personales")
```

```
Console.WriteLine("_____")
```

Console.WriteLine()

```
Console.WriteLine("Nombre:  
{0}", _Nombre)
```

```
Console.WriteLine("Domicilio:  
{0}", _Domicilio)
```

End Sub

End Class

Clases

- ▶ Module Module1
- ▶ Dim pers As persona = New persona()'reservo una variable en memoria
- ▶ Sub Main()
 - ▶ pers = New persona()'instancia del objeto
 - ▶ pers.nombre()'llamada a un método
 - ▶ pers.Domicilio()'llamada a un método
 - ▶ pers.ListarDatos()'llamada a un método
 - ▶ Console.WriteLine()
- ▶ End Sub
- ▶ End Module

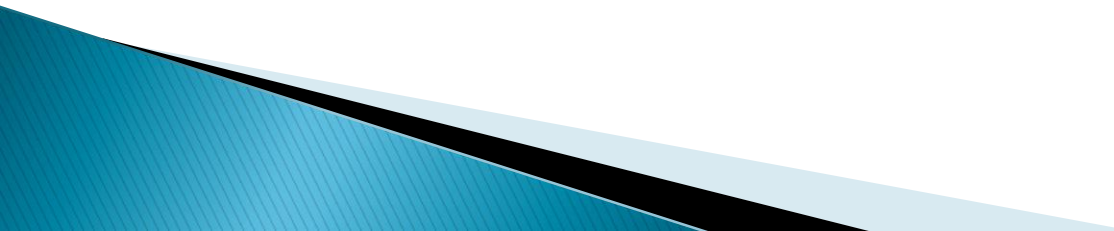
Ámbitos de una clase

- ▶ Los modificadores de ámbito de las clases son:
 - Private
 - Public
 - Protected
 - Friend
 - Protected Friend

Mas informacion:

- <http://msdn.microsoft.com/es-es/library/76453kax.aspx>

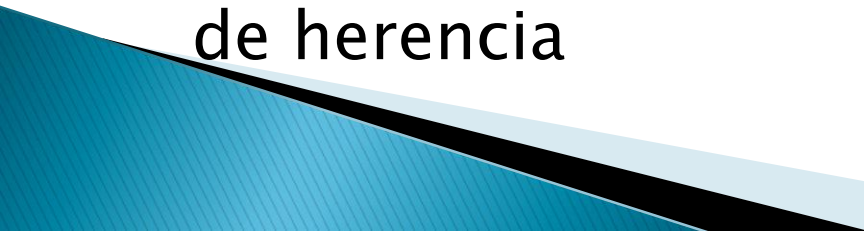
Constructores

- ▶ Un *constructor* es un método especial de una clase que es llamado automáticamente siempre que se crea un objeto de esa clase. Su función es iniciar el objeto.
 - ▶ Un *constructor* tiene el nombre New y no puede retornar un valor (procedimiento de tipo Sub). Cuando en una clase no escribimos explícitamente un *constructor*, VB.NET asume uno por omisión.
- 

Herencia VB .Net

- ▶ La herencia permite crear una clase base o padre, con especificaciones generales, y a partir de ella, crear nuevas clases derivadas o hijas
- ▶ Una clase hija puede servir a su vez como clase base para la creación de otra clase derivada, y así sucesivamente
- ▶ Para crear una clase derivada, debemos declarar una nueva clase, especificando cuál es su clase base mediante la palabra clave **Inherits**

Propiedades de la herencia

- ▶ Una subclase hereda todos los miembros de su superclase, excepto los constructores
 - ▶ Una subclase no tiene acceso directo a los miembros privados de su superclase
 - ▶ Una subclase puede añadir sus propios atributos y métodos. Si el nombre de alguno de estos miembros coincide con el de un miembro heredado, este último queda oculto para la subclase
 - ▶ Los miembros heredados por una subclase pueden, a su vez, ser heredados por más subclases de ella, a esto se le llama propagación de herencia
- 

Miembros sobrecargados

- ▶ La *sobrecarga de métodos*, es una técnica que consiste en crear varios métodos con idéntico nombre dentro de la misma clase, distinguiéndose entre sí por el número de parámetros o con el tipo de estos
- ▶ Para declarar un método como sobrecargado, podemos utilizar opcionalmente la palabra clave ***Overloads*** después del modificador de ámbito siempre que los métodos formen parte de la misma clase. Si el método sobrecargado se hereda de una clase base, por el contrario, si es obligatorio utilizar ***Overloads***

Miembros Redefinidos

- ▶ Para poder redefinir un método tiene que ser definido como *Overridable*, permitiendo así su redefinición en las clases derivadas. Éstas deben usar el modificador *Overrides* para indicar que redefinen el método
- ▶ Para redefinir un método es necesario que su *firma*, compuesta por el nombre, lista de parámetros y tipo de retorno, sea idéntica a la del método redefinido

Miembros Ocultos

- ▶ Para redefinir un método de la clase base con distintos parámetros, debemos ocultar el método de la clase base, y definimos el nuevo método en la clase derivada.
- ▶ Para ocultar un miembro heredado tan sólo hay que cambiar la palabra *Overrides* por *Shadows*.
- ▶ Ese nuevo método debe tener el mismo nombre, pero por lo demás puede diferir en parámetros y tipos

Acceso a los miembros de la clase base

- ▶ Cuando en una clase derivada se redefine un método, en ocasiones puede ser necesario invocar, desde el nuevo método, al método de la clase base, para ello usaremos el identificador ***MyBase***.

Inicialización de los constructores

```
Public Class padre
    Public a As Integer = 5
    Public Overridable Function
        Prueba() As String
        Return ("Clase Padre")
    End Function
End Class
```

```
Class Hija
    Inherits padre
    Shadows a As Integer = 10
    Public Overrides Function
        Prueba() As String
        Return ("Clase Hija")
    End Function
End Class
```

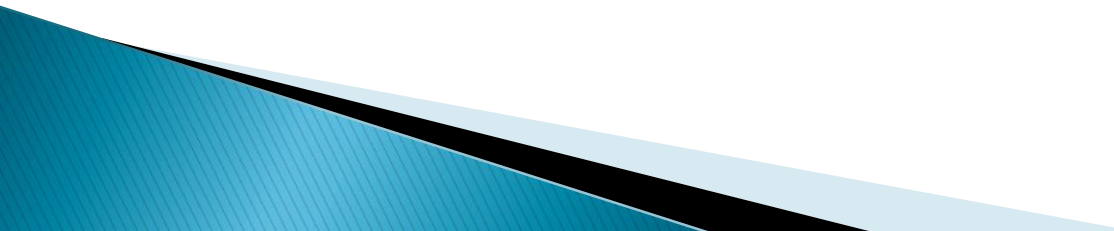
```
Module Module1
    Sub Main()

        Dim Padre As padre = New
            padre()
        Dim hija As Hija = New Hija()
        Padre = hija

        Console.WriteLine(Padre.Prueba)
        Console.WriteLine(Padre.a)

    End Sub
End Module
```

Inicialización de los constructores

- Se crean los objetos en el orden que son llamados, si se crea un objeto de una clase hija, llama al constructor de la clase hija, luego al constructor de la clase padre y así sucesivamente hasta alcanzar la raíz
 - Todas las variables de tipos primitivos son inicializadas a sus valores por defecto y las referencias a objetos son colocadas a nothing
- 

Inicialización de los constructores

- Se inicializan las variables del objeto raíz y se completa el cuerpo del constructor de la clase padre
- Después se inicializan las variables de su hijos y se completa el cuerpo del constructor de la clase hija hasta llegar y así sucesivamente hasta llegar a la base

Propiedades

```
Def
Ambito Property
  Nombre_Propiedad ()
  As Tipo
  Get
    ....
  End Get
  Set
    ....
  End Set
End Property
```

```
Public Class persona
  Private _Nombre As String 'Necesito
                             una variable privada que contenga la
                             variable publica le pongo el _ para
                             diferenciarlo
  Public Property Nombre() As String
    'property para declarar las propiedades
    Get 'Es lo que se ejecuta cuando
        recuperamos el valor de la propiedad
        Return _Nombre
    End Get
    Set(ByVal Value As String) 'es
                                cuando asignamos un valor
    Console.WriteLine("Bautizado")
    _Nombre = Value
    End Set
  End Property
End Class
```

Modulos

- ▶ Un módulo es una clase con algunas características implícitas:
 - no es posible derivarlo de otra clase
 - ni usarlo como base para la definición de una nueva
 - todos sus métodos son compartidos, a pesar de no usar explícitamente el modificador *Shared* por eso no hay que instanciarlos
 - su visibilidad está abierta a todo el módulo de código en el que está definido

Interfaces

- ▶ Un *interfaz* proporciona, a modo de declaración, una lista de propiedades y métodos, que posteriormente serán codificados en una o varias clases.
- ▶ Debido a su naturaleza declarativa, un *interfaz* no contiene el código de los miembros que expresa; dicho código será escrito en las *clases* que implementen el interfaz.
- ▶ si implementamos un interfaz estamos obligados a implementar todos los miembros de dicha interfaz.