Fundamentos de la Interacción Persona-Ordenador

6. Elementos del lenguaje VB.NET

Luís Rodríguez Baena (luis.rodriguez@upsam.net)

Universidad Pontificia de Salamanca (campus Madrid)
Facultad de Informática

Elementos del lenguaje

- Caracteres.
  - Utiliza caracteres Unicode de 16 bits.
- Identificadores.
  - De 1 a 16.386 caracteres Unicode.
    - ✓ Caracteres alfabéticos, numéricos o el carácter de subrayado.
  - Debe empezar por un carácter alfabético o el subrayado.
    - ✓ Si comienza por el subrayado debe tener al menos otro carácter alfabético o numérico.
      - No cumple el estándar del CLS.
- Comentarios.
  - Cualquier texto que aparezca después del apóstrofo (‘) es ignorado por el compilador.
  - Sentencia `REM`.
  - Visual Studio 2005 permite añadir comentarios XML en el editor de código utilizando tres apóstrofos (‘‘‘).
    - ✓ Inserta en el editor de código una estructura XML.

Elementos del lenguaje (II)

- Palabras reservadas.

AddHandler	AddressOf	Alias	And	AndAlso	As	Boolean	ByRef
Byte	ByVal	Call	Case	Catch	CBool	CByte	CChar
CDate	CDec	CDbl	Char	CInt	Class	CLng	CObj
Const	Continue	CSByte	CShort	CSng	CStr	CType	CUInt
CULng	CUShort	Date	Decimal	Declare	Default	Delegate	Dim
DirectCast	Do	Double	Each	Else	ElseIf	End	**EndIf**
Enum	Erase	Error	Event	Exit	False	Finally	For
Friend	Function	Get	GetType	Global	**GoSub**	GoTo	Handles
If	Implements	Imports	In	Inherits	Integer	Interface	Is
IsNot	**Let**	Lib	Like	Long	Loop	Me	Mod
Module	MustInherit	MustOverride	MyBase	MyClass	Namespace	Narrowing	New
Next	Not	Nothing	NotInheritable	NotOverridable	Object	Of	On
Operator	Option	Optional	Or	OrElse	Overloads	Overridable	Overrides
ParamArray	Partial	Private	Property	Protected	Public	RaiseEvent	ReadOnly
ReDim	REM	RemoveHandler	Resume	Return	SByte	Select	Set
Shadows	Shared	Short	Single	Static	Step	Stop	String
Structure	Sub	SyncLock	Then	Throw	To	True	Try
TryCast	TypeOf	**Variant**	**Wend**	UInteger	ULong	UShort	Using
When	While	Widening	With	WithEvents	WriteOnly	Xor	#Const
#Else	#ElseIf	#End	#If	-	&	&=	*
*=	/	/=	\	\=	^	^=	+
+=	=	-=					

Estructura de un programa VB.NET

- Una aplicación VB.NET se almacena en uno o más archivos de proyecto.
  - Cada proyecto consta de uno o más archivos de código (módulos) que se compilan para crear aplicaciones.
- Las categorías de instrucciones dentro de cada módulo deben seguir este orden:
  1. Instrucciones `Option`.
  2. Instrucciones `Imports`.
  3. Instrucciones `Namespace`.
  4. Declaraciones de módulos o clases.

Estructura de un programa VB.NET (II)

- Instrucciones `Option`.
  - Establecen las reglas base del código que aparece en el archivo.
  - `Option Explicit`, `Option Compare`, `Option Strict`, `Option Infer`.
- Instrucciones `Imports`.
  - Facilitan el empleo de los espacios de nombres y clases dentro del código del archivo.
  - Permiten evitar la referencia por el nombre cualificado.

```
Imports System.Text
...
Dim sb1 As New System.Text.StringBuilder(20) 'Nombre cualifacado
Dim sb2 As New StringBuilder(30)
```

Estructura de un programa VB.NET (III)

- Declaraciones.
  - Todo el código ejecutable (declaraciones, métodos, procedimientos, funciones) debe estar contenido en una clase o un módulo.
    - ✓ La clase o módulo puede estar dentro de un espacio de nombres.
- La mayoría de las veces el módulo o clase deberá tener un método `main`.
  - Es el punto de entrada de la aplicación.

```
Sub Main()
Sub Main(ByVal CmdArgs() As String)
Function Main() As Integer
Function Main(ByVal CmdArgs() As String) As Integer
```

Tipos de datos

- Tres categorías.
  - Tipos de valores.
    - ✓ Tipos primitivos (tipos de datos predefinidos o tipos valor integrados).
      - Numéricos, reales, lógicos, caracteres.
    - ✓ Enumeraciones.
    - ✓ Estructuras.
  - Tipos de referencia.
    - ✓ Cadenas, arrays, clases, módulos estándar, interfaces y delegados.
  - Tipo `Object`.
    - ✓ Alias de la clase `System.Object`.
    - ✓ De ella descienden todos los tipos.
    - ✓ Puede contener cualquier tipo de dato.

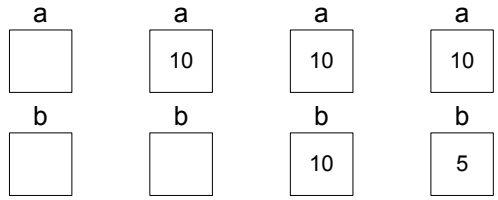
Tipos de datos (II)

- Almacenamiento en memoria.
  - Los tipos de valores se almacenan en la pila.
    - ✓ Se crean y reservan en tiempo de compilación.
    - ✓ Su acceso es directo.
  - Los tipos de referencia se almacenan en el montículo.
    - ✓ Son dinámicos, se guardan en tiempo de ejecución.
    - ✓ El acceso se hace a través de una referencia.
      - Cuando la referencia se pierde no se puede acceder al dato.
    - ✓ La asignación a un dato de referencia copia la referencia, no su contenido.

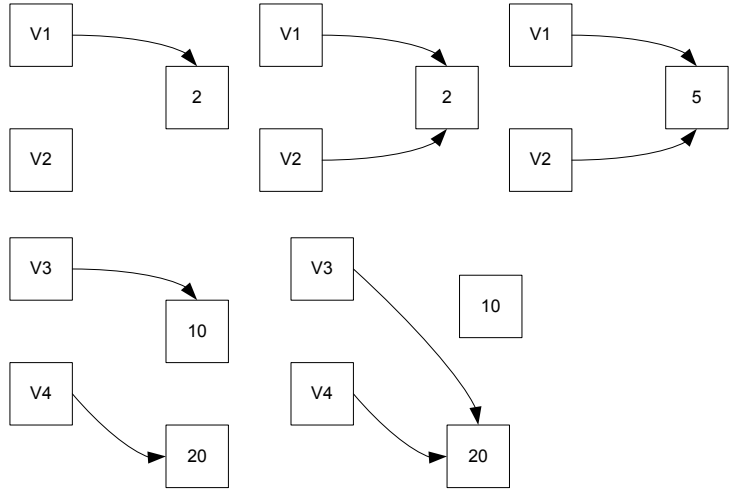
Tipos de datos (III)

```
Module Prueba
 Class MiClase
  Public valor As Integer
 End Class
 Sub Main()
  'a y b son tipos valor
  Dim a as integer
  Dim b as integer
  a = 10
  b = a
  b = 5
  'v1, v2, v3 y v4 son tipos de referencia
  Dim v1 As New MiClase 'Nueva instancia
  Dim v2 As MiClase
  v1.valor = 2
  v2 = v1
  v2.valor = 5
  System.Console.WriteLine(v1.valor _
    & " " & v2.valor) 'La salida es 5 5

  Dim v3 as New MiClase
  Dim v4 as New MiClase
  v3.valor = 10
  v4.valor = 20
  v3 = v4
End Sub
End Module
```

Tipos valor



Tipos de referencia



Tipos de datos
Tipos de datos numéricos

- Tipos de datos enteros.

Tipo de Visual Basic	Estructura de tipo CLR	Ocupa…	Intervalo de valores
SByte	System.SByte	1 byte	-128 y 127
Byte	System.Byte	1 byte	255
UShort	System.UInt16	2 bytes	0 y 65.535
Short	System.Int16	2 bytes	-32.768 a 32.767
UInteger	System.Int32	4 bytes	0 y 4.294.967.295
Integer	System.Int32	4 bytes	-2.147.483.648 a 2.147.483.647
ULong	System.Int64	8 bytes	18.446.744.073.709.551.615
Long	System.Int64	8 bytes	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807

- Tipos de datos reales

Tipo de Visual Basic	Estructura de tipo CLR	Ocupa…	Intervalo de valores
Single (punto flotante con precisión simple)	System.Single	4 bytes	-3,4028235E+38 a –1,401298E-45 para valores negativos; 1,401298E-45 a 3,4028235E+38 para valores positivos.
Double (punto flotante con precisión doble)	System.Double	8 bytes	-1,79769313486231570E+308 a -4,94065645841246544E-324 para valores negativos; 4,94065645841246544E-324 a 1,79769313486231570E+308 para valores positivos.
Decimal	System.Decimal	16 bytes	0 a +/-79.228.162.514.264.337.593.543.950.335 sin separador decimal; 0 a +/-7,9228162514264337593543950335 con 28 posiciones a la derecha del signo decimal; el número más pequeño distinto de cero es +/-0,0000000000000000000000000001 (+/-1E-28).

Tipos de datos: Boolean y Char

- El tipo de dato `Boolean`.
  - Corresponde al tipo del CLR `System.Boolean`.
  - Ocupa 2 bytes.
  - Puede tomar el valor True o False.
- El tipo de dato `Char`.
  - Corresponde al tipo del CLR `System.Char`.
  - Ocupa 2 bytes (un carácter Unicode de 16 bits).
  - No es compatible con el dato `String`.
  - No se puede utilizar como un dato numérico.
    - ✓ Funciones `Asc()`, `AscW()`, `Chr()` y `ChrW()`.
      - Devuelven respectivamente el código UNICODE de un carácter (`Asc()` y `AscW()`) y el carácter correspondiente a un código Unicode (`Chr()` y `ChrW()`)
  - Métodos estáticos `Char.IsControl()`,`Char.IsDigit()`, `Char.IsLetter()`, `Char.IsDigitOrLetter()`, `Char.IsLower()`, `Char.IsNumber()`, `Char.IsPunctuation()`, `Char.IsSymbol()`, `Char.Upper()`, `Char.IsWhiteSpace()`.

Tipos de datos: el tipo Date

- Características.
  - Corresponde a la clase `System.DateTime` del CLR
  - Opcupa 8 bytes (un entero largo).
  - Puede tomar el valor de fechas y horas desde el 1 de enero del año 1 a las 00:00:00 hasta el 31 de diciembre del año 9.999 a las 11:59:59.
- Constructor: `System.DateTime(año,mes,día).`
- Propiedades:

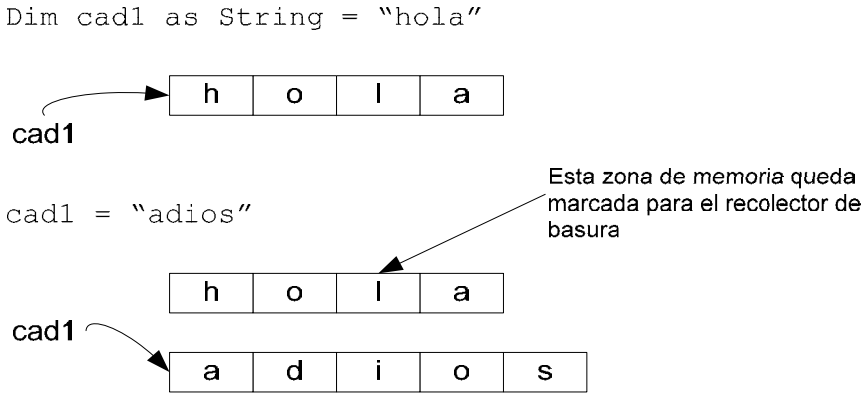
Propiedad	Devuelve…	Propiedad	Devuelve…
Day	El día	Hour	Horas
Month	El mes	Minute	Minutos
Year	El año	Second	Segundo
DayOfWeek	El día de la semana	Millisecon	Milisegundos
DayOfYear	El día del año (0 para el domingo)	Today	La fecha actual
TimeOfDay	La hora del día	Now	La fecha y hora actual
Ticks	Número de ticks de una fecha		

Tipos de datos: el tipo Date (II)

- Aritmética de fechas.
  - `AddDays(n).`
  - `AddMonths(n).`
  - `AddYears(n).`
  - `AddHours(n).`
  - `AddMinutes(n).`
  - `AddSeconds(n).`
- `n` puede ser un número entero o fraccionario, positivo o negativo.

La clase String

- Características.
  - Corresponde a la clase `System.String` del CLR.
  - Permite almacenar de 0 a 2.000 mill. de caracteres Unicode.
- No es un dato valor ni un dato primitivo, sino una clase.
  - Es una referencia que apunta a una zona del montículo.
  - Ocupará aproximadamente 4 bytes de la referencia más el doble del número de caracteres.
- Son inmutables.
  - La clase `StringBuilder`
    - ✓ Cadena modificable.



La clase String (II)

- Constructores.
  - `String(Char,Integer)`; `String(Char())`; etc.
- Algunas propiedades públicas.
  - `Length`. Devuelve el número de caracteres.
  - `Chars(Integer)`. Devuelve él carácter de la posición `Integer`.
    - ✓ El valor que devuelve está en base 0 (el primer carácter es el carácter 0).
- Algunos métodos públicos (contenidos en la clase `String` del CLR).

Método	Observaciones
Compare(String,String) Compare(String,String,bolean)	Sobrecargado. Compara dos objetos **String** especificados. Permite ignorar o no las mayúsculas con el tercer atributo lógico.
CompareTo(String)	Sobrecargado. Compara la instancia en cuestión con un objeto especificado.
IndexOf({Char|String}[,Integer])	Sobrecargado. Devuelve el índice de la primera aparición de un objeto **String**, o de uno o más caracteres de la instancia en cuestión. (-1 si no se encuentra)
Insert(String,Integer)	Inserta una instancia especificada de **String** en una posición de índice especificada de la instancia.
LastIndexOf({Char|String}[,Integer])	Sobrecargado. Devuelve la posición de índice de la última aparición de un carácter Unicode especificado o de un objeto **String** en la instancia.

La clase String (III)

- Algunos métodos públicos *(continuación)*.

Método	Observaciones
Remove(Inicio,Longitud)	Elimina un número de caracteres especificado de la instancia a partir de una posición especificada.
Replace({Char|String},{Char|String})	Sobrecargado. Reemplaza todas las apariciones de un carácter Unicode o un objeto **String** en la instancia por otro carácter Unicode u otro objeto **String**.
Split(Char[,Integer])	Sobrecargado. Identifica las subcadenas de la instancia que están delimitadas por uno o varios caracteres especificados en una matriz, y las coloca después en una matriz de elementos **String**.
ToLower()	Sobrecargado. Devuelve una copia de **String** en minúsculas.
ToString()	Sobrecargado. Reemplazado. Convierte el valor de la instancia en un objeto **String**.
ToUpper()	Sobrecargado. Devuelve una copia de **String** en mayúsculas.
Trim()	Sobrecargado. Quita todas las apariciones de un conjunto de caracteres especificados desde el principio y el final de la instancia.
TrimEnd()	Quita todas las apariciones de un conjunto de caracteres especificados en una matriz de caracteres Unicode desde el final de la instancia.
TrimStart()	Quita todas las apariciones de un conjunto de caracteres especificados en una matriz de caracteres Unicode desde el principio de la instancia.

La clase String (IV)

- Algunas funciones de cadena (exclusivas de VB.NET, no de .NET Framework.).
  - Al no estar incluidas en la biblioteca de .NET Framework es preferible utilizar los métodos equivalentes anteriores.

Función	Acción
StrComp(String1,String2[, {Binary|Text]})	Comparar dos cadenas.
LCase(String), UCase(String)	Convertir en mayúsculas o minúsculas.
Space(Integer), StrDup(Integer,Char)	Crear una cadena de caracteres repetidos.
Len(String)	Buscar la longitud de una cadena.
InStr(inicio,String1,String2[,Binary|Text])	Busca String2 dentro de String1
Left(String, Integer)	Extrae caracteres por la izquierda
Right(String,Integer)	Extrae caracteres por la derecha
Mid(String,inicio,longitud)	Devuelve una subcadena de String
LTrim(String)	Elimina espacios en blanco por la
RTrim(String)	Elimina espacios en blanco por la derecha
Trim(String)	Elimina espacios en blanco por la izquierda y la derecha
Replace(StrPrincipal,StrBuscada,StrSust)	Sustituir una subcadena especificada.
Split(String[,delimitador])	Separa la cadena a partir del delimitador y devuelve un array de cadena

El tipo de dato Object

- Todos los tipos derivan de la clase `System.Object`, por lo que con un dato de tipo `Object` se puede referenciar cualquier tipo de dato.
- Ocupa 4 bytes.
  - Lo que ocupa un puntero.
- Todos los datos heredan los métodos de `Object`.
  - Los métodos se pueden sobrecargar para dotarlos de características especiales.
    - ✓ Método `Equals()`.
    - ✓ Metodo `ToString()`.

El tipo de dato Object
Métodos Equals y toString

```
Public Class MiClase
    Private valor As Integer
    Public Sub New(ByVal n As Integer)
        valor = n
    End Sub
    Public Function getValor() As Integer
        Return valor
    End Function
    Public Overrides Function ToString() As String
        Return "Valor: " & Me.valor
    End Function
    Public Overloads Function Equals(ByVal obj As MiClase) As Boolean
        Return valor = obj.valor
    End Function
End Class

...
Dim x As MiClase = New MiClase(5)      'Crea un objeto con valor=5
System.Console.WriteLine(x)            'No escribe 5, sino "Valor: 5"

Dim y As MiClase = New MiClase(5)      'Crea un objeto con valor=5

If x.Equals(y) Then                    'Ejecuta las acciones, puesto que el atributo
   ...                                 'valor de x y de y es 5
End if
```

Estructuras

- Tipos de datos definidos por el usuario.
- Pueden contener cualquier tipo de dato (valores o referencias) y métodos para manipularlos.
- Declaración:

```
[Modificador de acceso]Structure NombreTipo
        declaración de variables del tipo
        [declaración de métodos]
Emd Structure

Structure Empleado
   Public DNI As String
   Public Nombre As String
   Public SueldoBruto As Decimal
   Public Retenciones As Decimal
   Public Function SueldoNeto() As Decimal
       Return SueldoBruto - Retenciones
   End Function
End Structure
```

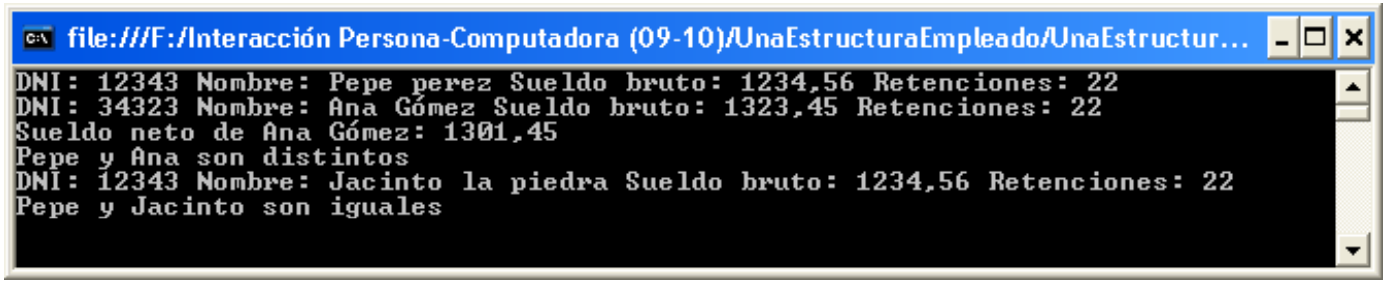
- Se accede a los miembros mediante el punto.

Estructuras
Ejemplo

```
Structure Empleado
        Public DNI As String
        Public Nombre As String
        Public SueldoBruto As Decimal
        Public Retenciones As Decimal
        Sub New(ByVal dni As String, ByVal nom As String, ByVal s As Decimal, ByVal r As Decimal)
            Me.dni = dni
            nombre = nom
            SueldoBruto = s
            Retenciones = r
        End Sub

        Public Overrides Function ToString() As String
            Return "DNI: " & DNI & _
                   " Nombre: " & Nombre & _
                   " Sueldo bruto: " & SueldoBruto & _
                   " Retenciones: " & Retenciones
        End Function
        Public Overrides Function Equals(ByVal obj As Object) As Boolean
            'Dos empleados son iguales si tienen el mismo DNI
            Return DNI = obj.dni
        End Function
        Public Function SueldoNeto() As Decimal
            Return SueldoBruto - Retenciones
        End Function
 End Structure
```

Estructuras
Ejemplo (II)

```
Sub Main()
        Dim e As Empleado = New Empleado("12343", "Pepe perez", 1234.56, 22.0)
        Console.WriteLine(e)
        Dim e2 As Empleado = New Empleado("34323", "Ana Gómez", 1323.45, 22.0)
        Console.WriteLine(e2)
        Console.WriteLine("Sueldo neto de " & e2.Nombre & ": " & e2.SueldoNeto())
        If e.Equals(e2) Then
            Console.WriteLine("Pepe y Ana son iguales")
        Else
            Console.WriteLine("Pepe y Ana son distintos")
        End If
        Dim e3 As Empleado = New Empleado("12343", "Jacinto la piedra", 1234.56, 22.0)
        Console.WriteLine(e3)
        If e.Equals(e3) Then
            Console.WriteLine("Pepe y Jacinto son iguales")
        Else
            Console.WriteLine("Pepe y Jacinto son distintos")
        End If
        Console.ReadLine()
End Sub
```



Literales

- Booleanos.
  - `True` y `False`.
- Enteros.
  - Dígitos y, opcionalmente el signo -.
  - El prefijo `&H` indica un valor hexadecimal. El prefijo `&O` indica un valor octal.
  - Pueden incluir un sufijo para indicar el tipo de entero: `S`, `I`,`L`,`US`, ,`UI` o `UL`.
- Reales.
  - Dígitos y opcionalmente el punto decimal y el signo.
  - Formato en coma fija o coma flotante.
  - Pueden incluir un sufijo para indicar el tipo de real: `D`, `F` o `R`.

Literales (II)

- Fecha.
  - Fecha en formato mm/dd/aaaa encerrada entre almohadillas (#).
- Literales de cadena y carácter.
  - Secuencia de 0 o más caracteres Unicode entre comillas.
  - Pueden incluir comillas mediante el carácter de escape ".
  - Los literales de tipo carácter tienen el sufijo `c`.
- El literal `Nothing`.
  - Inicializa cualquier dato con su valor por omisión.
  - Cuando se le asigna a una variable de tipo objeto o a una clase, se elimina la referencia.

Declaraciones de variables

- Declaración implícita y explícita.
  - Instrucción `Option Explicit [{On | Off}]`.
- Declaración de variables a nivel de método: variables locales.
  - `[Static] Dim` *`ident`* `[As [New]` *`TipoDato`*`] [=`*`exprInic`*`]`
    - ✓ Su accesibilidad es dentro del procedimiento.
    - ✓ El modificador `Static` determina el tiempo de vida de la variable.
    - ✓ El tipo de dato no es obligatorio.
      - Si no se pone expresión de inicialización el tipo es `Object`.
      - Si se utiliza una expresión de inicialización y se permite la inferencia de tipo de variable local (`Option Infer On`), el tipo de dato de la variable será del tipo de dato de la expresión.
    - ✓ Para variables de tipos de referencia se puede utilizar la palabra reservada `New` para crear una nueva instancia de la clase.
    - ✓ Admiten una expresión de inicialización del tipo de dato adecuado.
      - Si no se añada una expresión de inicialización, los valores numéricos se inicializan a 0, los lógicos a `false`, los caracteres a carácter nulo, las cadenas a cadena nula y los objetos a `Nothing`.

Declaraciones de variables (II)

- Declaración a nivel de módulo o clase.

```
[{Public | Protected | Friend |
    Protected Friend | Private ] [Static |Shared]
    Dim ident [As [New] tipoDato] [=exprInic]
```

  - `Public`, `Protected`, `Friend`, `Protected Friend` y `Private` son los modificadores de acceso.
    - ✓ `Public` permite acceder a la variable desde otro proyecto o desde cualquier ensamblado del proyecto.
    - ✓ `Friend` permite el acceso desde cualquier parte del mismo ensamblado.
    - ✓ `Private` permite acceder sólo desde dentro del ámbito donde ha sido declarada.
    - ✓ `Protected`, `Protected Friend` y `Shared` se utilizan para la POO.
  - Si se utiliza cualquier de estos modificadores se puede omitir la palabra `Dim`.

Constantes y enumeraciones

- Constantes.

```
[{ Public | Protected | Friend | Protected Friend
  |
Private }] Const identificador [ As tipoDatos ] =
  ExprInic
```

- Enumeraciones.
  - Proporcionan una forma cómoda de trabajar con constantes relacionadas.
  - Declaración:

```
[{ Public | Protected | Friend | Protected Friend
  |
Private }] Enum nombreEnumeración [As tipoDato]
    nombreMiembro [= exprInic]
    …
End Enum
```

Constantes y enumeraciones (II)

- Enumeraciones (*continuación*).
  - El valor de las constantes *nombreMiembro* es de algún tipo entero.
    - ✓ Por omisión son de tipo `Integer` y comienzan a contar desde 0.
    - ✓ Para cambiar el tipo de dato se puede utilizar la cláusula `As` *tipoDato*.
    - ✓ La *exprInic* permite asignar a la constante enumerada un valor distinto.

```
Public Enum DiaSemana
   Domingo      'La constante DiaSemana.Domingo vadrá 0
   Lunes
   Martes
   Miércoles
   Jueves
   Viernes
   Sábado       'La constante DiaSemana.Sábado vadrá 6
End Enum
```

Operadores y expresiones

- Operadores aritméticos.
  - `^`, exponenciación.
    - ✓ Los operandos se convierten a `Double` y el resultado es `Double`.
  - `*`, `+`, `-`.
    - ✓ El resultado depende de los operandos y será del tipo del operando de mayor intervalo (`Byte`, `Short`, `Integer`, `Long`, `Single`, `Double` y `Decimal`).
    - ✓ El signo `+` se puede utilizar para la suma o para la concatenación.
  - División entera (`\`) y módulo (`Mod`).
    - ✓ Los operandos se convierten a entero y el resultado es entero.
- Operadores de concatenación.
  - `+` y `&`.

Operadores y expresiones (II)

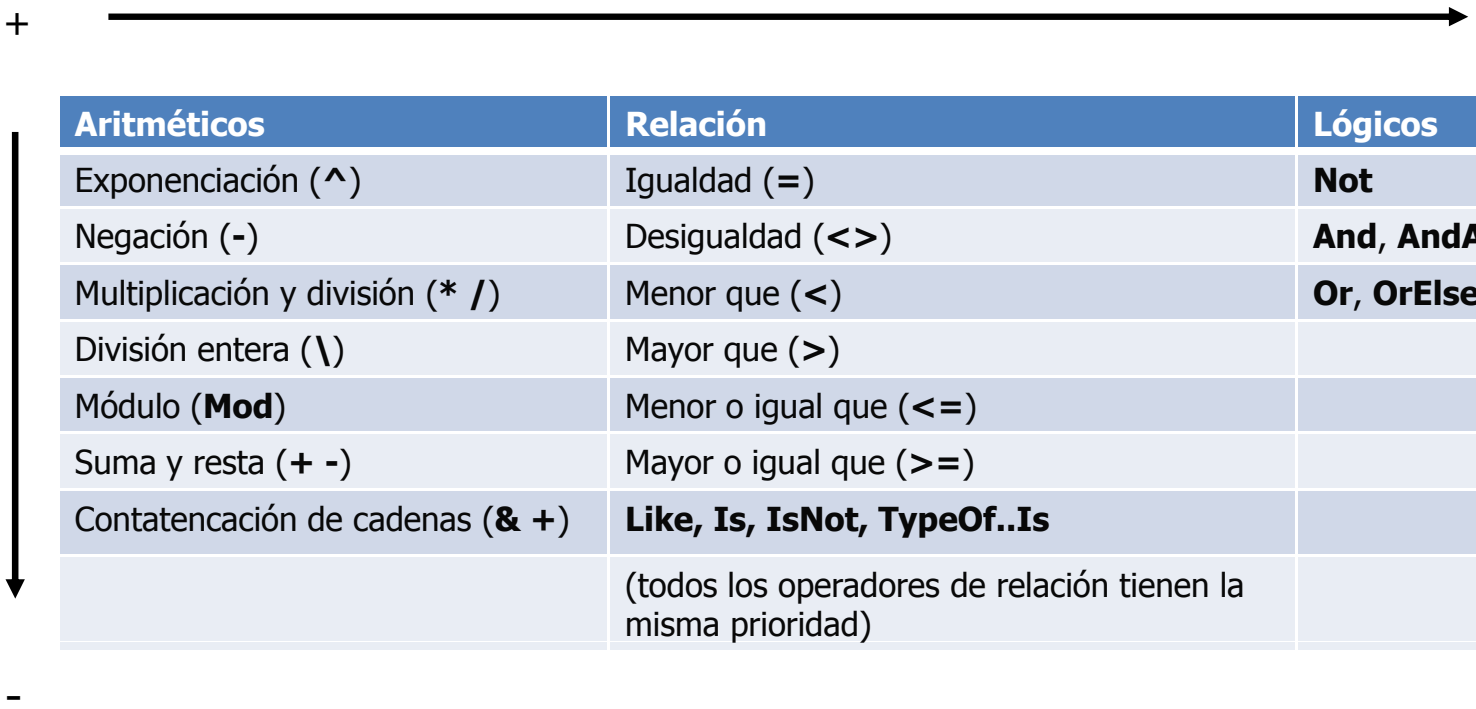
- Operadores de asignación.
  - `=, +=, -=, *=, /=, \=, ^=, &=`.
- Operadores de relación.
  - `=, <>, >=, <=, <, >`.
  - Comparaciones de cadenas.
    - ✓ Comparación binaria y comparación de texto:
      - Instrucción `Option Compare {Binary | Text}`.
    - ✓ La función StrComp.
      - `StrComp(String1,String2[, {Binary|Text}])`.
        - Devuelve 0, 1 o -1 según la cadena 1 sea igual, mayor o menor que la cadena 2.
        - El tercer argumento permite especificar el tipo de comparación.
    - ✓ Método `Compare(String1,String2[,Boolean])`.
    - ✓ Método `CompareTo(String1)`.
      - Ambas devuelven un valor igual, menor o mayor que 0, según la cadena sea igual, menor o mayor que la otra.
      - Si el tercer argumento de `Compare` es `True` ignora mayúsculas y minúsculas.

Operadores y expresiones (III)

- El operador `Like`.
  - Compara una cadena con un patrón.
    expresiónCadena `Like` *patrón*
  - `patrón` es una cadena que puede contener:
    - ✓ *, sustituye a 0 o más caracteres (`"hola" Like "ho*"`).
    - ✓ ?, sustituye a 1 carácter (`"hola" Like "ho?a"`).
    - ✓ #, sustituye a un dígito (`"1234A" Like "####*"`).
    - ✓ [caracteres], sustituye a alguno de lo caracteres (`"hola" Like "hol[aeiou]"`).
    - ✓ [car1-car2], sustituye a alguno de los caracteres del rango (`"A3" Like "A[0-5]"`).
    - ✓ [!caracteres], sustituye a cualquier carácter no incluido en la lista.
- Operador `Is`.
  - Compara si dos referencias de objetos apuntan a la misma instancia.
  - No compara el contenido. Para eso habría que sobrecargar el método `Equals`.
- Operador `IsNot`.
  - Compara si dos referencias de objetos no apuntan a la misma instanacia.
- Operador `Typeof` *expresión* `Is` *clase*.
  - Es verdadero su el tipo de dato de *expresión* coincide con *clase*.

Operadores y expresiones (IV)

- Operadores lógicos.
  - `Not`, `And`, `Or`, `AndAlso`, `OrElse`, `Xor`.
- Prioridad de los operadores.



Aritméticos	Relación	Lógicos
Exponenciación (**^**)	Igualdad (**=**)	**Not**
Negación (**-**)	Desigualdad (**<>**)	**And**, **AndAlso**
Multiplicación y división (*** /**)	Menor que (**<**)	**Or**, **OrElse**, **Xor**
División entera (****)	Mayor que (**>**)	
Módulo (**Mod**)	Menor o igual que (**<=**)	
Suma y resta (**+ -**)	Mayor o igual que (**>=**)	
Contatencación de cadenas (**& +**)	**Like, Is, IsNot, TypeOf..Is**	
	(todos los operadores de relación tienen la misma prioridad)	

Conversión de tipos

- Conversiones de ampliación y restricción.
  - Conversiones de ampliación: el tipo de dato receptor tiene más ocupación en memoria.
  - Conversiones de restricción: el tipo de dato receptor tiene menos ocupación en memoria.
    - ✓ Posible pérdida de precisión.
- Conversión implícita y conversión explícita.
  - Por omisión los tipos de datos se convierten siempre que sea necesario.
  - Instrucción `Option Strict {On | Off}`.
    - ✓ Restringe la conversión a conversiones de ampliación.
    - ✓ En caso de realizar conversiones de restricción produce errores en tiempo de compilación.

Conversiones de tipos (II)

- Funciones de conversión.

Nombre de la función	Tipo de valor devuelto	Intervalo de valores del argumento *expression*
CBool(*expr*)	**Boolean**	Cualquier expresión numérica o de cadena (**String**) válida.
CByte(*expr*)	**Byte**	0 a 255; las fracciones se redondean.
CChar(*expr*)	**Char**	Cualquier expresión **String** válida, valores comprendidos entre 0 y 65535.
CDate(*expr*)	**Date**	Cualquier representación válida de fecha y hora.
CDbl(*expr*)	**Double**	-1,79769313486231E+308 a -4,94065645841247E-324 para valores negativos; 4,94065645841247E-324 a 1,79769313486231E+308 para valores positivos.
CDec(*expr*)	**Decimal**	+/-79.228.162.514.264.337.593.543.950.335 para números a partir de cero, es decir, números sin decimales. Para números con 28 decimales, el rango es +/-7.9228162514264337593543950335. El menor número distinto de cero es 0,0000000000000000000000000001.
CInt(*expr*)	**Integer**	-2.147.483.648 a 2.147.483.647; las fracciones se redondean.
CLng(*expr*)	**Long**	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807; las fracciones se redondean.
CObj (*expr*)	**Object**	Cualquier expresión válida.
CShort(*expr*)	**Short**	-32.768 a 32.767; las fracciones se redondean.
CSng(*expr*)	**Single**	De -3,402823E+38 a –1,401298E-45 para valores negativos; de 1,401298E-45 a 3,402823E+38 para valores positivos.
CStr(*expr*)	**String**	

Conversiones de tipos (III)

- Función `CType(`*`expresión`*`,`*`tipo`*`)`.
  - Permite convertir una expresión a un tipo de datos, objeto, estructura, clase o interfaz.

```
Dim num1 As Long
Dim num2 As Single
num1 = 1000
num2 = CType(num1, Single)  'num2 será 1000.0
```

- Función `DirectCast(`*`expresión`*`,`*`tipo`*`)`.
  - Funciona de forma similar a `CType`.
  - Requiere que el tipo especificado sea el mismo que el de la expresión o de un tipo heredado.

```
Dim k As Object = 2.37
Dim i as Integer = CType(k,Integer) 'Funciona
Dim j as Ingeger = DirectCast(k,Integer) 'No funciona
```

  - Es más rápida de `CType`.
  - Devuelve una excepción si no se puede realizar la conversión.
- Función `TryCast(`*`expresión`*`,`*`tipo`*`)`.
  - Devuelve Nothing si la conversión no se puede realizar.

Entrada/salida

- Entrada/salida por consola.
  - `System.Console.ReadLine()`.
  - `System.Console.Write()`.
  - `System.Console.WriteLine()`.
    - ✓ Escribe una representación en formato cadena del argumento, seguida de un salto de línea.
  - `WriteLine()` permite la salida de varios datos, concatenándolos para crear un objeto de tipo cadena:

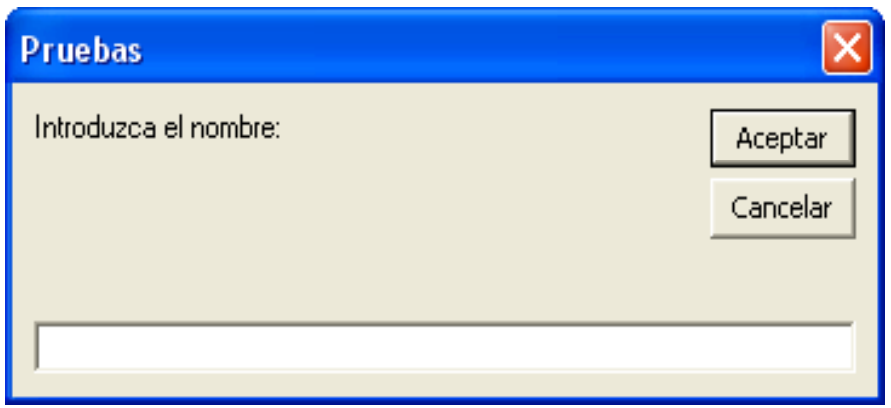
    ```
    Console.WriteLine("Hola" & 23 & date.now())
    ```
  - `WriteLine(cadenaFormato, objetos)`, escribe el objeto u objetos según la cadena de formato.
    - ✓ La cadena de formato puede llevar secuencias `{n° argumento}` para indicar la posición que debe llevar el objeto.

      ```
      Console.WriteLine("Nombre:{1} Edad:{0}", 24, "Ana
        López")
      ```

Función InputBox

- La entrada de datos simples de Windows se puede hacer con la función `InputBox` de Visual Basic.
  - Muestra un cuadro de diálogo con una caja de texto en la que el usuario puede introducir información.

    `InputBox(`*`mensaje`*`[, `*`título`*`][, `*`valorOmisión`*`])`
  - Devuelve un dato de tipo `String`.
  - En general se desaconseja su uso cómo elemento de la interfaz de entrada de datos.



La clase MessageBox

- Se puede utilizar par la salida de mensajes en un entorno de ventanas.
- .NET Framework dispone de la clase `MessageBox` que se utiliza para mostrar cuadros de diálogo por pantalla.
  - Está en el espacio de nombres y en el ensamblado `System.Windows.Forms`.
    - ✓ En aplicaciones de consola es necesario agregar esa referencia (con aplicaciones para Windows no).
- El método estático `Show` muestra el cuadro de diálogo.
  - `MessageBox.Show(`*`texto`*`[,`*`título`*` [,`*`botones`*` [,`*`icono`*` [,`*`botónDefecto`*`]]]])`
  - Devuelve un dato de tipo `DialogResult`:
    - ✓ Puede tomar los valores `DialogResult.Abort`, `DialogResult.Cancel`, `DialogResult.Ignore`, `DialogResult.No`, `DialogResult.OK`, `DialogResult.Retry`, `DialogResult.Yes`.

La clase MessageBox (II)

- *botones* es un miembro de la enumeración `MessageBoxButtons`.

Miembro	Descripción
AbortRetryIgnore	El cuadro de mensaje contiene los botones Abort, Retry y Ignore.
OK	El cuadro de mensaje contiene un botón OK.
OKCancel	El cuadro de mensaje contiene los botones OK y Cancel.
RetryCancel	El cuadro de mensaje contiene los botones Retry y Cancel.
YesNo	El cuadro de mensaje contiene los botones Yes y No.
YesNoCancel	El cuadro de mensaje contiene los botones Yes, No y Cancel.

- *botónDefecto* es un miembro de la enumeración `MessageBoxDefaultButtons`.

Miembro	Descripción
Button1	El primer botón del cuadro de mensaje es el botón predeterminado.
Button2	El segundo botón del cuadro de mensaje es el botón predeterminado.
Button3	El tercer botón del cuadro de mensaje es el botón predeterminado.

La clase MessageBox (III)

- *icono* es un miembro de la enumeración `MessageBoxIcon`.

Miembro	Descripción
Asterisk	El cuadro de mensaje contiene un símbolo que consiste en un letra 'i' minúscula en un círculo.
Error	El cuadro de mensaje contiene un símbolo que consiste en una X blanca en un círculo con fondo rojo.
Exclamation	El cuadro de mensaje contiene un símbolo que consiste en un signo de exclamación en un triángulo con fondo amarillo.
Hand	El cuadro de mensaje contiene un símbolo que consiste en una X blanca en un círculo con fondo rojo.
Information	El cuadro de mensaje contiene un símbolo que consiste en un letra 'i' minúscula en un círculo.
None	El cuadro de mensaje no contiene ningún símbolo.
Question	El cuadro de mensaje contiene un símbolo que consiste en un signo de interrogación en un círculo.
Stop	El cuadro de mensaje contiene un símbolo que consiste en una X blanca en un círculo con fondo rojo.
Warning	El cuadro de mensaje contiene un símbolo que consiste en un signo de exclamación en un triángulo con fondo amarillo.

La clase MessageBox (IV)


```
If MessageBox.Show("¿Desea actualizar el archivo?", _
                   "Prueba de MessageBox", _
                   MessageBoxButtons.YesNo, _
                   MessageBoxIcon.Question, _
                   MessageBoxDefaultButton.Button2) = DialogResult.Yes Then
   'Acciones del botón "Si"
Else
   'Acciones del botón "No"
End If
```

Instrucciones, líneas y bloques

- VB no tiene formato libre de línea de código.
  - El fin de línea forma parte de la sintaxis del lenguaje.
    - ✓ La siguiente instrucción no sería válida

```
System.Console.WriteLine("Hola" & " "
                         & "qué tal")
```

  - En una línea pueden ir varias instrucciones utilizando el carácter separador de instrucciones (:).

```
a = 3 : b = 4 : c = 6
```

    - ✓ Esto no es recomendable.
  - Es posible extender una instrucción más de una línea de texto utilizando el carácter de continuación de línea (_).
    - ✓ Debe ir precedido de un espacio en blanco.
    - ✓ La siguiente instrucción sería válida

```
System.Console.WriteLine("Hola" & " " _
                         & "qué tal")
```

Instrucciones, líneas y bloques (II)

- Visual Studio 2010 ha incluido la "continuación de línea implícita".
  - En determinadas circunstancias se puede evitar el carácter de continuación.
    - ✓ Después de una coma.
    - ✓ Después de abrir paréntesis o antes de cerrarlo.
    - ✓ Después de abrir llaves o antes de cerrarlas.
    - ✓ Después de los operadores:
      - De concatenación (&)
      - De asignación (=, &=, :=, +=, -=, *=, /=, \=, ^=)
      - De los operadores binarios (+, -, /, *Mod, <>, <, >, <=, >=, ^, And, AndAlso, Or, OrElse, Like, Xor).
      - De los operadores Is e IsNot.
    - ✓ Después de un carácter de calificador de miembro (.) y antes del nombre de miembro.
- Un grupo de líneas de código forman un **bloque de código**
  - Los bloques de código se agrupan en una o más líneas *etiquetadas*, aunque la etiqueta es opcional.
    - ✓ Una etiqueta está formada de un número o un identificador seguido de 2 puntos.
      - Se pueden utilizar para referenciar zonas del código (instrucciones `On Error…Goto`)

Instrucciones de control
Estructuras selectivas

- Declaración `If`.

  `If` *expresiónLógica* `Then` *instrucción* [`Else` *instrucción*]

- Declaración `If` de bloques.

```
If expresiónLógica Then
   bloque de instrucciones
[ElseIf expresiónLógica Then
 bloque de instrucciones]…
[Else
    bloque de instrucciones]
End If
```

Instrucciones de control
Estructuras selectivas (II)

- Declaración `Select Case`.

```
Select Case expresiónPrueba
  [Case listaDePruebas
     [bloque de instrucciones]] ...
  [Case Else
     [bloque de instrucciones]]
End Select
```

- *listaDePruebas* puede ser:
  - ✓ Una expresión.
  - ✓ Dos expresiones separadas por la palabra reservada `To`.
  - ✓ `Is` *operadorDeRelación expresión*.
  - ✓ Una mezcla de todo lo anterior separados por comas.
- La declaración `Select Case` hace cortocircuito.

Instrucciones de control
Estructuras repetitivas

- Declaración `Do..Loop`.

```
Do [{While | Until} exprLógica]
   [bloque de instrucciones]
   [Exit Do]
   [bloque de instrucciones]
Loop

Do
   [bloque de instrucciones]
   [Exit Do]
   [bloque de instrucciones]
Loop [{While | Until} exprLógica]
```

Instrucciones de control
Estructuras repetitivas (II)

- Declaración `While..While End.`

```
While expresiónLógica
    [bloque de instrucciones]
End While
```

- Declaración `For..Next.`

```
For variable [As tipoDato] = exprInicio To exprFin
                                    [Step exprIncremento]
    [bloque de instrucciones]
    [Exit For]
Next [variable]
```

- Instrucción `Continue.`
  - Permite romper la estructura de un bucle, transfiriendo el control a la siguiente iteración del mismo.

```
Continue {Do | While | For}
```

Instrucciones de control
Estructuras repetitivas (III)

- Declaración `For..Each`.

```
For Each elemento [As tipoDato] In grupo
    [bloque de instrucciones]
    [Exit For]
    [bloque de instrucciones]
Next [elemento]
```

  - Repite las instrucciones por cada elemento de una colección o array.
  - *`grupo`* es una colección o array.
    - ✓ Una colección es objeto que contiene una serie de elementos relacionados (objetos, controles, datos…).
  - *`elemento`* es una variable que va tomando el valor de cada uno de los componentes del grupo.
  - Ejemplo:

```
Dim ctl as Object
For Each ctl In Controls
  MsgBox ctl.Name
Next
```

Instrucciones de control
Declaración de variables de bloque

- Una variable declarada dentro de un bloque tiene su ámbito dentro del bloque.
- Los bucles `For` admiten la declaración de la variable del bucle en la propia instrucción.
  - Su ámbito también sería el bloque donde ha sido declarada.

```
For Each x As Integer In a        'a es un array de enteros
    System.Console.WriteLine(x)
Next
For Each ctl As Object In Controls
    Dim i As Integer
    i+=1
    System.Console.WriteLine(i & " " & ctl.name)
Next
'System.Console.WriteLine(i) 'Da un error ya que i no está declarado
```

Procedimientos

- Bloque de código que comienza por una declaración de procedimiento y termina por la palabra `End` correspondiente.
  - Todo el código ejecutable se encuentra dentro de un procedimiento.
- Tipos de declaraciones de procedimiento.
  - Procedimientos `Sub`.
    - ✓ Ejecutan acciones pero no devuelve un valor al código de llamada.
  - Procedimientos `Function`.
    - ✓ Devuelven un valor al código de llamada.
  - Procedimientos de evento.
    - ✓ Procedimientos `Sub` que se ejecutan en respuesta a un evento producido por una acción del usuario en la interfaz o desencadenado por el programa.
  - Procedimientos `Property Get` y `Property Set`
    - ✓ Devuelven y asignan valores de propiedades en clases y módulos.
  - Procedimientos `Operator`.
    - ✓ Se utilizan para sobrecargar los operadores.

Procedimientos Sub

- Declaración.

```
[modificadores de acceso]Sub nombreProc [(listaArgumentos)]
    [ bloque de instrucciones ]
    [ Exit Sub | Return ]
    [ bloque de instrucciones ]
End Sub
```

- Llamada.

```
[Call] nombreProc [(listaArgumentos)]
```

Procedimientos Function

Declaración.

```
[modificadores de acceso]Function nombreFunc [(listaArgumentos)][As tipoDato]
   [ bloque de instrucciones ]
   [ Exit Function ]
   [ bloque de instrucciones ]
End Function
```

Valores de retorno.

- El valor de retorno de la función se puede hacer mediante:

  `Return expresión`

  `nombreFunción = expresión`

- Si el código no pasa por alguna de estas instrucciones, devuelve el valor por omisión correspondiente al tipo de la función.
  - ✓ El tipo de dato que devuelve por omisión es de tipo `Object`.
    - Si `Option Strict` está puesto a `On`, genera un error.

Procedimientos
Modificadores de acceso

- Los distintos tipos de accesibilidad son:
  - `Public` (tipo por omisión).
    - ✓ El método puede utilizarse en cualquier parte del proyecto, desde otro proyecto o desde un ensamblado generado por el proyecto.
  - `Private`.
    - ✓ El método puede utilizarse sólo dentro del ámbito donde ha sido declarado (módulo, función, etc.).
  - `Friend`.
    - ✓ El método puede utilizarse desde el ámbito donde ha sido declarado y en cualquier sección del mismo ensamblado.
  - `Protected`.
    - ✓ El método sólo puede utilizarse dentro de la clase donde ha sido declarado y en las clases derivadas de ésta.
  - `Protected Friend`.
    - ✓ El método sólo puede utilizarse desde la clase dónde ha sido declarada, en las clases derivadas de ésta y en cualquier sección del mismo ensamblado.

Procedimientos
Paso de argumentos

- Declaración de cada argumento.

  `[ Optional ] [{ ByVal | ByRef }] [ ParamArray ]`
  `nombreArgumento``[( )] [ As` *`tipoDato`* `] [ =` *`valorDefecto`* `]`
- Tipos de los argumentos.
  - Por omisión son argumentos de tipo `Object`.
- Paso por valor y paso por referencia.
  - Por omisión el paso se hace por valor.
  - Las constantes, los literales, las expresiones y las enumeraciones se pueden pasar por referencia, aunque su valor no cambiará en el programa llamador.
  - Los tipos de referencia (clases, arrays,…) se pueden pasar por valor.
    - ✓ El procedimiento no puede cambiar la referencia, pero si puede cambiar los miembros de la instancia que señala.

```
'Ejemplo de paso por valor en tipos de referencias la clase declarada en la página 9)
Sub P1(ByVal x as MiClase)
   Dim y as New MiClase
   x.valor = 8      'Se modifica el atributo valor de la instancia x
   x = y            'x apuntará al mismo objeto que y
End Sub
'Al terminar el procedimiento, x sigue apuntando al mismo objeto,
'pero se mantiene el valor del atributo
```

Procedimientos
Paso de argumentos (II)

- Argumentos opcionales.
  - Se señalan mediante la cláusula `Optional`.
  - Se pueden omitir en la llamada.
  - Deben tener un valor por omisión y ser los últimos argumentos de la lista.
  - Para determinar si un argumento está o no presente en la llamada se puede utilizar un valor centinela.
- Argumentos `ParamArray`.
  - Se utilizan para pasar un número indeterminado de argumentos.
  - Sólo puede haber un argumento `ParamArray` y éste debe ser el último de la lista y pasarse por valor.

Paso de argumentos (III)

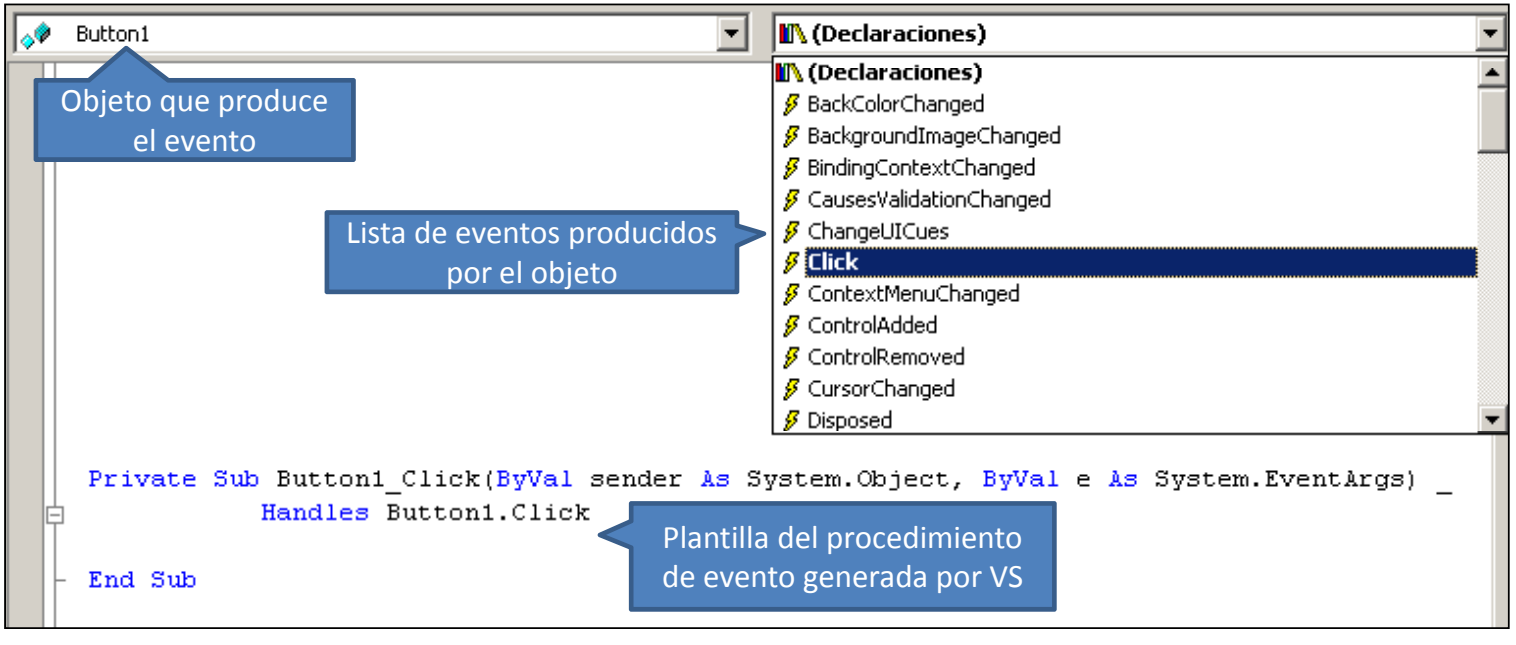
```
Module Module1
    Class clase
        Friend valor As Integer
    End Class
    Sub main()
        'Ejemplo de uso de parámetros opcionales y ParamArray
        System.Console.WriteLine(Sumar(1, 2, 3, 4)) 'Devuelve 10
        System.Console.WriteLine(Sumar(2, 3, 4))    'Devuelve 9
        Dim i As Integer = 1
        System.Console.WriteLine(contador(i))       'Devuelve 2
        System.Console.WriteLine(contador(i, 3))    'Devuelve 4

        'Ejemplo de paso por referencia en tipos de referencia
        Dim cls As New clase()
        cls.valor = 2
        prueba(cls)
        System.Console.WriteLine(cls.valor)         'Devuelve 3
        System.Console.ReadLine()
    End Sub
    …
```

Paso de argumentos (IV)

```
    'Ejemplo de uso de ParamArray
    Function Sumar(ByVal ParamArray elem()) As Integer
        Dim suma As Integer = 0
        Dim i As Integer
        For i = elem.GetLowerBound(0) To elem.GetUpperBound(0)
            suma += elem(i)
        Next
        Return suma
    End Function
    'Ejemplo de uso de argumentos opcionales
    Function contador(ByVal x As Integer, _
                      Optional ByVal conta As Integer _
                      = Integer.MinValue) As Integer
        If conta = Integer.MinValue Then
            Return x + 1
        Else
            Return x + conta
        End If
    End Function
    'Ejemplo de paso por valor en tipos de referencia
    Sub prueba(ByVal cls As MiClase)
        cls.valor = contador(cls.valor)
    End Sub
End Module
```

Procedimientos de evento

- Procedimientos que se ejecutan en respuesta a un evento desencadenado por una acción del usuario o un suceso del programa.
  - No se les suele llamar por el método normal.



Arrays

- Se trata de datos de referencia que pertenecen a la clase `System.Array`.
- Declaración.

```
[{ Public | Protected | Friend | Protected Friend |
Private }] [Static | Shared ]
Dim identificador ([listaLímites])
[ As [ New ] tipoDatos ] [=ExprInic ]
```

  - Los modificadores de acceso tienen el mismo significado que en la declaración de variables normales.
  - La declaración no requiere la especificación del tamaño, sino únicamente de su rango.

```
Dim arr1() As Integer  'Declara una matriz de enteros de 1 dimensión
Dim arr2(,) As Byte    'Declara una matriz de bytes de 2 dimensiones
Dim arr3()() As Object 'Declara una matriz de objetos "escalonada"
                       '(Matriz de matrices)
```

Arrays (II)

- Inicialización (instanciación).
  - En la declaración indicando el/los índices superiores del array.
    - ✓ Los elementos se inicializa a su valor por omisión.

```
Dim arr1(5) As Integer  'La matriz tendría índices entre 0 y 5
                        'y se rellenaría de 0
```

  - Asignar sus valores iniciales mediante una expresión de inicialización.
    - ✓ El tamaño de la matriz dependería del número de elementos de la inicialización.

```
Dim arr2() As Integer = {1, 2, 3, 4, 5}
```

  - Asignando un objeto de la clase `Array` mediante el constructor `New` *`tipoDato`*`() {[`*`valores`*`]}`

```
Dim arr4() As Byte = New Byte(3) {}
Dim arr5() As Byte = New Byte() {0,1,2}   ' Desde 0 hasta 2
```

Arrays (III)

- Inicialización *(continuación).*
  - Para matrices multidimensionales.

```
Dim arr6(3, 5) As Integer
Dim arr7(,) As Integer = {{1, 2, 3}, {4, 5, 6}}
Dim arr8(,) As Byte = New Byte(,) {}
'arr8 sería un array sin elementos, un array dinámico
Dim arr9(,) As Byte = New Byte(,) {{1, 2}, {3, 4}}
Dim arr10(,) As Byte = New Byte(1, 1) {{5, 6}, {7, 8}}
```

  - Para matrices escalonadas (arrays de arrays).

```
Dim arr11(1)() As Byte
Dim arr12()() As Byte = {New Byte() {}, New Byte() {}}
Dim arr13()() As Byte = {New Byte() {1}, _
                         New Byte() {2, 3}, _
                         New Byte() {4, 5, 6}}
```

Arrays (IV)

- Límites de los índices de una matriz.
  - El límite inferior normalmente es 0.
  - Los índices debe ser enteros de 64 bits.
  - La propiedad `.Length`, devuelve el número de elementos.
  - Métodos `GetLowerBound(`*`dimensión`*`)` y `GetUpperBound(`*`dimensión`*`)`.

```
For x As Integer = 0 To arr10.GetUpperBound(0)
   For y As Integer = 0 To arr10.GetUpperBound(1)
      System.Console.Write(arr10(x, y) & " ")
   Next
   System.Console.WriteLine()
Next
For x As Integer = 0 To arr13.GetUpperBound(0)
   For y As Integer = 0 To arr13(x).GetUpperBound(0)
      System.Console.Write(arr13(x)(y) & " ")
   Next
   System.Console.WriteLine()
Next
```

Arrays (V)

- Redimensión de una matriz.
  - Variables de tipo matriz e instancias de esa variable.
  - Cambio del tamaño asignando otra matriz del mismo tipo y del mismo número de dimensiones.

```
arr6 = New Integer(,) {{1, 2}, {3, 4}}
arr7 = arr6  'arr7 se ha declarado de dos dimensiones anteriormente
```

  - Cambio de tamaño mediante la instrucción `ReDim`.
    - ✓ Reasigna el número de elementos creando una nueva matriz.
    - ✓ No puede cambiar el rango ni el tipo de elementos.
    - ✓ Los elementos de la matriz redimensionada se pierden.
      - Cláusula Preserve.
        - Mantiene los elementos de la **última** dimensión de la matriz, truncando o rellenando elementos si es necesario.

```
Dim arr14() As Byte = {1, 2, 3}
ReDim Preserve arr14(6)        '{1,2,3,0,0,0,0}
ReDim Preserve arr14(1)        '{1,2}
```

Arrays (VI)

- Mediante arrays dinámicos y la orden `Redim Preserve` es posible añadir elementos a un array en tiempo de ejecución.

```
'El array a está declarado como un array dinámico
'Al arrancar tiene 0 elementos
Dim a() as Integer = new Integer(){}
...

Sub añadirElementoAUnArray(ByRef a() As Integer, ByVal i As Integer)
        'Si el array tiene n elementos,
        'al redimensionar a(n) hace que el índice mayor sea n,
        'por lo que el array tendrá n+1 elemento
        Dim n As Integer = a.Length
        ReDim Preserve a(n)
        a(n) = i
End Sub
```

Arrays en procedimientos

- Matrices como argumentos.

```
Dim v() As String = {"hola", "adios", "pepe", "manolo"}
System.Console.WriteLine(buscar(v, "manolo")) 'Devuelve 3
System.Console.WriteLine(buscar(v, "xxx"))    'Devuelve -1
…
Function buscar(ByVal a() As Object, ByVal e As Object) As Long
  For i As Long = 0 To a.GetUpperBound(0)
    If a(i).Equals(e) Then
       Return i
    End If
  Next
  Return -1
End Function
```

Arrays en procedimientos (II)

- Paso por valor y paso por referencia.
  - El paso por valor impide cambiar la asignación de la instancia, pero si modificar sus miembros.

```
Dim v() As String = {"hola", "adios", "pepe", "manolo"}
PasoPorValor1(v) 'v = {"hola", "adios", "pepe", "manolo"}
PasoPorValor2(v) 'v = {"aaa", "adios", "pepe", "manolo"}
...
Sub PasoPorValor1(ByVal a() As Object)
    a = New String() {"a", "b"}
End Sub

Sub PasoPorValor2(ByVal a() As Object)
    a(0) = "aaa"
End Sub
```

Arrays en procedimientos (III)

- Matrices como valores de retorno en funciones.

```
Dim arr15() As Integer = MatrizDePares(4) '{0,2,4,6,8}
...
'Rellena una matriz con n números pares
Function MatrizDePares(ByVal n As Integer) As Integer()
    Dim v(n) As Integer
    For i As Integer = 0 To n
        Dim par As Integer
        v(i) = par
        par += 2
    Next
    Return v
End Function
```

Copia y borrado de matrices

- Borrado de matrices.
  - Método estático `Array.Clear(`*`array`*`,`*`indiceInicio`*`,`*`longitud`*`)`
  - Asignación del literal `Nothing`.
- Asignación de matrices.

```
Dim v() As String = {"hola", "adios", "pepe", "manolo"}
Dim v2() As String
v2 = v
v2(0) = "xxx"       'v={"xxx", "adios", "pepe", "manolo"}
                    'v2={"xxx", "adios", "pepe", "manolo"}
```

- Copia de matrices.
  - Método `Clone`.
    - ✓ Hace una copia superficial del array:
      - Si los elementos son tipos de referencia, se copia la referencia, no los objetos de la matriz.

```
Dim v4() As String = v.Clone
v4(0) = "yyy"       'v={"xxx", "adios", "pepe", "manolo"}
                    'v4={"yyy", "adios", "pepe", "manolo"}
```

Copia y borrado de matrices (II)

- Método `CopyTo`.

  arrayOrigen`.CopyTo(`*arrayDestino*`,`*desplazamiento*`)`

```
Dim v5() As Byte = {1, 2, 3, 4}
Dim v6(10) As Byte
v5.CopyTo(v6, 5)    'v6={0,0,0,0,1,2,3,4,,0,0}
```

- Método estático `Copy`.

  `Array.Copy(`*arrayOrig*`,`*inicioArrayOrig*`,`*arrayDest*`,`*inicioArrayDest*`,`*long*`)`

```
v5 = New Byte() {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
Dim v9(10) As Byte
'Copia desde el elemento 7 de v5 2 elementos
'a partir del elemento 3 de v9
Array.Copy(v5, 7, v9, 3, 2) 'v9={0,0,0,8,9,0,0,0,0,0,0}
```

Búsqueda de elementos

- Método estático `IndexOf`.
  `Array.IndexOf(`*`array`*`,`*`elemento`*`)`
  `Array.IndexOf(`*`array`*`,`*`elemento`*`,`*`inicio`*`)`
  `Array.IndexOf(`*`array`*`,`*`elemento`*`,`*`inicio`*`,`*`último`*`)`
  - Devuelve la posición de *`elemento`* o -1 si no lo encuentra.

```
Dim nombres() As String = {"Juana", "Pepe", "Luisa", "Pepe", "Ana"}
System.Console.WriteLine(Array.IndexOf(nombres, "Pepe"))    '1
System.Console.WriteLine(Array.IndexOf(nombres, "Manolo"))  '-1
System.Console.WriteLine(Array.IndexOf(nombres, "Pepe", 2)) '3
System.Console.WriteLine(Array.IndexOf(nombres, "Ana", 1, 3)) '-1
'Busca todas las apariciones de Pepe en el array
Dim pos As Integer = -1
Do
    pos = Array.IndexOf(nombres, "Pepe", pos + 1)
    If pos <> -1 Then
        System.Console.WriteLine(pos)  'Escribe 1 y 3
    End If
Loop Until pos < 0
```

- El método `LastIndexOf`, realiza la búsqueda del último al primer elemento.

Búsqueda de elementos (II)

- Búsqueda binaria: método estático `BinarySearch`.

  `Array.BinarySearch(`*`array`*`,`*`elemento`*`)`

  `Array.BinarySearch(`*`array`*`,`*`índice`*`,`*`longitud`*`,`*`elemento`*`)`

  - Devuelve el índice de *`elemento`* si el elemento está.
  - Si el elemento no está devuelve un número negativo.

```
'nombre = {"Ana","Juana","Luisa","Pepe","Pepe","Ramón"}
'Devuelve 2
System.Console.WriteLine(Array.BinarySearch(nombres, "Luisa"))
'Devuelve 3
System.Console.WriteLine(Array.BinarySearch(nombres, "Pepe"))
'Devuelve -2
System.Console.WriteLine(Array.BinarySearch(nombres, "Bartolo"))
'Devuelve -7
System.Console.WriteLine(Array.BinarySearch(nombres, "Zacarías"))
```

Ordenación de matrices

- Método estático `Sort`.

  `Array.Sort(`*`array`*`)`

  `Array.Sort(`*`array`*`,`*`inicio`*`,numE`*`lementos`*`)`

  - El método estático `Reverse`.

    `Array.Reverse(`*`array`*`)`

    `Array.Reverse(`*`array`*`, `*`inicio`*`,numE`*`lementos`*`)`

  - Ordenación de clases o estructuras por una clave.

    `Array.Sort(`*`arrayDeClaves`*`,`*`array`*`)`

```
'Se rellena un array de Personas
Dim a() As Personas = {New Personas("Pepe", 23), _
                       New Personas("Ana", 30), _
                       New Personas("Juan", 18)}
'Se declara y rellena un array con la clave a ordenar
Dim edades(a.GetUpperBound(0)) As Byte
For i As Integer = 0 To a.GetUpperBound(0)
    edades(i) = a(i).edad
Next
'Se ordena a partir del segundo array
Array.Sort(edades, a)
```

```
Structure Personas
    Dim nombre As String
    Dim edad As Byte
    Sub New(ByVal nom, ByVal ed)
        nombre = nom
        edad = ed
    End Sub
End Structure
```

Ordenación y búsqueda por varios criterios: interfaz IComparer

```
Class Persona
    Public nombre As String
    Public edad As Byte
    Sub New(ByVal nom As String, ByVal ed As Integer)
        Me.nombre = nom
        Me.edad = ed
    End Sub
    Class CompararPorNombre
        Implements IComparer
        Public Function Compare(ByVal x As Object, ByVal y As Object) _
              As Integer Implements System.Collections.IComparer.Compare
            If (x Is Nothing) And (y Is Nothing) Then Return 0
            If (x Is Nothing) Then Return 1
            If (y Is Nothing) Then Return -1
            Dim p1 As Persona = CType(x, Persona)
            Dim p2 As Persona = CType(y, Persona)
            Return StrComp(p1.nombre, p2.nombre, CompareMethod.Text)
        End Function
    End Class
    Class CompararPorEdad
        Implements IComparer
        Public Function Compare(ByVal x As Object, ByVal y As Object) _
              As Integer Implements System.Collections.IComparer.Compare
              ...
        End Function
    End Class
End Class
...
Dim a() As Persona = {New Persona("Pepe", 23), New Persona("Ana", 30), _
                                New Persona("Juan", 18)}
Array.Sort(a, New Persona.CompararPorNombre) //Ordena por nombre
Array.Sort(a, New Persona.CompararPorEdad)   //Ordena por edad
//Busca una persona de 23 años por edad
Dim p = Array.BinarySearch(a, New Persona("", 23), New Persona.CompararPorEdad)
```

Otras clases relacionadas

- Clase `LinkedList`.
  - Proporciona una lista enlazada con métodos para añadir nodos al comienzo o al final de la lista, o antes o después de un nodo determinado.
- Clase `ArrayList`.
  - Proporciona un array de una dimensión al que se pueden añadir y eliminar elementos en tiempo de diseño.
- Clase `Dictionary`.
  - Proporciona una colección de parejas formadas por una clave y un valor.
- Clase `HashTable`.
  - Proporciona una colección de parejas formadas por una clave hash y un valor.
- Clase `SortedList`.
  - Proporcina una colección de parejas formadas por una clave y un valor ordenadas por el valor de la clave.

Otras clases relacionadas (II)

- Clase `Stack`.
  - Representa una estructura LIFO.
  - Constructor: `New Stack(`*`dimensión`*`)`.
  - Propiedad `Count`.
  - Métodos:
    - ✓ `Push(`*`objeto`*`)`. Inserta un elemento en la cima.
    - ✓ `Pop(`*`objeto`*`)`. Extrae el elemento cima.
    - ✓ `Peek()`. Devuelve el elemento cima sin sacarlo.
- Clase `Queue`.
  - Representa una estructura FIFO.
  - Constructor: `New Queue(`*`dimensión`*`)`.
  - Propiedad `Count`.
  - Métodos:
    - ✓ `Enqueue(`*`objeto`*`)`. Inserta un elemento al final de la cola.
    - ✓ `Dequeue(`*`objeto`*`)`. Extrae el primer elemento de la cola.
    - ✓ `Peek()`. Devuelve el primer elemento sin sacarlo.

Otras clases relacionadas (III)

```
'Comprueba si una cadena es un palíndromo
Function EsPalindromo(ByVal str As String) As Boolean
    Dim p As New Stack(str.Length)
    Dim c As New Queue(str.Length)
    Dim car1 As Char
    Dim car2 As Char
    'Llenar los elementos en la pila y en la cola
    For i As Integer = 0 To str.Length - 1
        p.Push(str.Chars(i))
        c.Enqueue(str.Chars(i))
    Next
    'Desapilar y desencolar hasta encontrar un carácter distinto
    'o que alguna de las estructuras esté vacía
    car1 = p.Pop()
    car2 = c.Dequeue()
    Do While car1 = car2 And p.Count > 0
        car1 = p.Pop()
        car2 = c.Dequeue()
    Loop
    Return car1 = car2
End Function
```

Entrada/salida de secuencias

- Una secuencia o *stream* permite leer o escribir desde un almacén.
  - Representa una secuencia de bytes que residen en un almacén.
- La clase `Stream`.
  - Realiza todo el manejo de datos desde secuencias de dispositivos externos físicos (discos, puerto paralelo) o virtuales (memoria, sockets TCP/IP).
  - Se trata de una clase abstracta.
    - ✓ No se trabajará con ella, sino con clases derivadas.
  - Todas las clases para el manejo de secuencias residen en el espacio de nombres `System.IO`.
    - ✓ Será necesario colocar la declaración `Imports System.IO` al principio del código o utilizar el nombre totalmente cualificado para utilizarlas.

Operaciones básicas con secuencias

- Las operaciones básicas para trabajar con secuencias serán:
  - Lectura. Método `Read`.
    - ✓ Lee una serie de bytes de la secuencia y avanza el puntero de la secuencia.
  - Escritura.Método `Write`.
    - ✓ Escribe un número de bytes en la secuencia y avanza el puntero de la misma.
  - Búsqueda. Método `Seek`.
    - ✓ Obtiene o establece el puntero de la secuencia a una posición determinada.
  - No todos las secuencias admiten estas operaciones.
    - ✓ Propiedades `CanRead`, `CanWrite` y `CanSeek`.
- Método `Flush()`.
  - Vuelca el contenido del buffer en el almacenamiento.
- Propiedad `Length`.
  - Devuelve el tamaño total del Stream.
- Propiedad `Position`.
  - Determina la posición actual de la secuencia.

Lectores y escritores

- La clase `Stream` sólo puede leer bytes individuales o grupos de bytes.
  - Los objetos lectores y escritores de secuencias permiten trabajar con datos más estructurados.
- Clases `StreamReader` y `StreamWriter`.
  - Trabajan con cadenas de texto ANSI.
- Clases `BinaryReader` y `BinaryWriter`.
  - Trabajan con datos primitivos en formato binario.
- Clases `TextReader` y `TextWriter`.
  - Trabajan con cadenas de texto Unicode.
- Clases `XMLReader` y `XMLWriter`.
  - Trabajan con texto XML.

Lectura de archivos de texto

- Se utiliza un objeto `StreamReader`.
- Será necesario obtener una referencia a este objeto y asociarle el archivo de texto:
  - Pasando el la referencia al archivo en el constructor.

```
Dim sr As New StreamReader("MiArchivo.txt")
```

  - Creando un objeto de la clase `FileStream` y pasándolo al constructor.

```
Dim fs as New
  FileStream("MiArchivo.txt",FileMode.open)
Dim sr as New StreamReader(fs).
```

Lectura de archivos de texto (II)

- Lectura de datos.
  - Método `Read()`.
    - ✓ Devuelve un número entero que representa a un carácter de la secuencia y hace avanzar la posición del puntero en un carácter.

      ```
      Dim unCarácter As Char = sr.Read()
      ```

      - Si no hay más caracteres devuelve -1.
  - Método `ReadLine()`.
    - ✓ Devuelve una cadena con todos los caracteres de una secuencia hasta el final de línea y avanza la posición del puntero hasta la línea siguiente.

      ```
      Dim línea as String = sr.ReadLine()
      ```

      - Devuelve una referencia nula (`Nothing`) si no hay más datos.
  - Método `ReadToEnd()`.
    - ✓ Devuelve una cadena con toda la secuencia desde la posición actual hasta el final del archivo.

      ```
      Dim contenidoArchivo as String = sr.ReadToEnd()
      ```

      - Si no hay más caracteres devuelve una cadena vacía.

Lectura de archivos de texto (III)

- Método `Peek()`.
  - Lee el siguiente carácter y devuelve el código del mismo.
  - No avanza al siguiente carácter.
  - Devuelve -1 si no hay más caracteres.
- Mover el puntero de la secuencia sin leer.
  - El método `Seek` de la clase `Stream` permite mover el puntero de la secuencia una serie de bytes.

    `Seek(offset,inicio)`

    ✓ `offset` es el número de bytes a desplazar.

    ✓ `inicio` es un miembro de la enumeración SeekOrigin que puede tomar los valores…
    - `SeekOrigin.Begin`, mueve un número de bytes desde el inicio de la secuencia.
    - `SeekOrigin.Current`, mueve un número de bytes desde la posición actual de la secuencia.
    - `SeekOrigin.End`, mueve un número de bytes desde el final de la secuencia.
- Método `Close`.
  - Es necesario cerrar tanto el `StreamReader`, como el *stream* base si es necesario.

Lectura de archivos de texto (IV)

- Ejemplo: leer el contenido del archivo miArchivo.txt carácter a carácter.

```
Dim sr As New StreamReader("MiArchivo.txt")
Do While sr.Peek <> -1
    Dim carácter As Integer = sr.Read()
    System.Console.WriteLine(Chr(carácter))
Loop
sr.Close
```

- Ejemplo: leer el contenido del archivo miArchivo.txt línea a línea.

```
Dim sr As New StreamReader("MiArchivo.txt")
Do While sr.Peek <> -1
    Dim carácter As Integer = sr.Read()
    System.Console.WriteLine(Chr(carácter))
Loop
sr.Close
```

Escritura en archivos de texto

- Será necesario crear una referencia a un objeto `Stream` y asignarlo al archivo.
  - Es necesario que el `Stream` permita la escritura.
    - ✓ Propiedad `CanWrite`.
- La escritura se realiza con un objeto `StreamWriter`.
  - Para crear una referencia al `StreamWriter` se procede de forma similar al `StreamReader`.
    - ✓ Pasando el la referencia al archivo en el constructor.

      ```
      Dim sw As New StreamWriter("MiArchivo.txt")
      ```
    - ✓ Creando un objeto de la clase `FileStream` y pasándolo al constructor.

      ```
      Dim fs as New FileStream("MiArchivo.txt",FileMode.Create)
      Dim sw as New StreamWriter(fs)
      ```
      - El modo de apertura `FileMode.Open`, requiere que el archivo exista.
      - El modo de apertura `FileMode.Create`, crea el archivo si este no existe.
      - El modo de apartura `FileMode.Append`, mueve el puntero de la secuencia al final si el archivo existe.
      - El modo de apertura `FileMode.CreateNew`, lanza una excepción si el archivo no existe.

Escritura en archivos de texto (II)

- Método `Write`.
  - Escribe en la posición que indique el puntero de la secuencia una representación textual de cualquier objeto.

```
sw.Write("Hola")         'Escribe la cadena Hola
sw.Write(33)             'Escribe una representación del entero 33
sw.Write(New Persona()) 'Escribe una representación del objeto
                            'Persona
```

- Método `WriteLine`.
  - Escribe una cadena e inserta después un salto de línea.

```
sw.WriteLine("Esta es la línea " & numLinea)
```

- Para escribir todo un archivo (pequeño) en otro…

```
sw.Write(sr.ReadToEnd)
```

Ejemplo

- Se desea crear un archivo de texto con los datos de una serie de personas (nombre y edad). Al escribirlos cada campo se separa por el carácter de dos puntos (:).

```
Imports System.IO
Module Module1
 Public p() As Personas = New Personas() {}
 Sub Main()
   'Se crea una referencia al archivo para añadir nuevos elementos y se crea un StreamWriter
   Dim fs As New FileStream("personas.txt", FileMode.Append)
   Dim sw As New StreamWriter(fs)

   Dim nombre As String
   Dim edad As Byte
   System.Console.Write("Nombre (FIN para terminar): ")
   nombre = System.Console.ReadLine()
   Do While nombre <> "FIN"
       System.Console.Write("Edad: ")
       edad = System.Console.ReadLine()
       'Se escriben los dos campos separados por dos puntos
       sw.Write(nombre) :sw.Write(":") : sw.Write(edad)
       'De forma alternativa se podrían escribir con sw.WriteLine(nombre & ":" & edad)
       System.Console.Write("Nombre (FIN para terminar): ")
       nombre = System.Console.ReadLine()
   Loop
   sw.Close()
   fs.Close()
 End Sub
End Module
```

Ejemplo (II)

- Cargar los datos del archivo personas.txt en un array de personas.

```
Sub CargarPersonasDeArchivo(ByVal p() As Personas)
  'Se crea una referencia al archivo personas.txt
  'para añadir nuevos elementos y se crea un StreamWriter
  Dim fs As New FileStream("personas.txt", FileMode.Open)
  Dim sr As New StreamReader(fs)

  Do While sr.Peek <> -1
      Dim línea As String = sr.ReadLine()
      'Se separan los datos por medio de los dos puntos
      'registro(0) tiene el nombre, registro(1) la edad
      Dim registro() As String = línea.Split(":")
      'Se aumenta en 1 posición el número de elementos del array
      ReDim Preserve p(p.GetUpperBound(0) + 1)
      p(p.GetUpperBound(0)) = New Personas(registro(0), registro(1))
  Loop
  sr.Close()
  fs.Close()
End Sub

Structure Personas
    Dim nombre As String
    Dim edad As Byte
    Sub New(ByVal nom, ByVal ed)
        nombre = nom
        edad = ed
    End Sub
End Structure
```

Programación orientada a objetos

Declaración de clases

- Una clase es un fragmento de código contenido entre las declaraciones `Class..End Class`.

```
[ modificadorDeAcceso ] Class nombreClase
   [declaraciones]
End Class
```

- El acceso predeterminado de una clase es `Friend`.
  - ✓ Sólo puede accederse desde el proyecto donde está declarada.
- Una clase `Public` puede ser utilizada en cualquier parte del proyecto o de otro proyecto (siempre que se haya agregado una referencia al proyecto donde se encuentra).
- Los modificadores `Private`, `Protected` y `Protected Friend` sólo pueden utilizarse si la clase está declarada dentro de otro tipo, es decir dentro de otra clase, módulo, interfaz, etc.

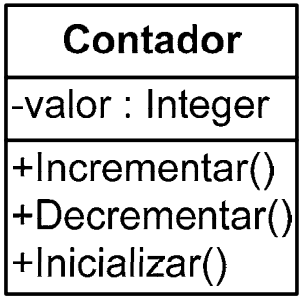
Programación orientada a objetos (II)

Declaración de miembros.

- Los atributos se pueden considerar como variables de la clase.
  - ✓ Se declaran de forma similar a otras variables en VB.NET
- Los métodos se declararán en forma de procedimientos `Sub` y `Function` dentro de la clase.
- Accesibilidad de los miembros.
  - ✓ De forma predeterminada la accesibilidad de los atributos es `Private` y de los métodos `Public`.
  - ✓ El modificador `Friend` permitirá utilizar dicho miembro en cualquier parte dentro del mismo proyecto.
  - ✓ El modificador `Protected` se utiliza para la herencia.
    - Un atributo `Proctected` podrá ser utilizado por la clase donde ha sido declarado y en todas sus clases derivadas.

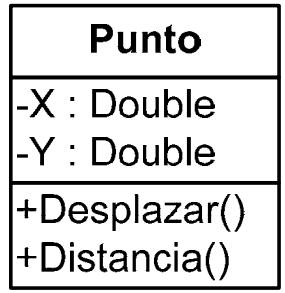
Programación orientada a objetos (III)

```
Public Class Contador
    Private valor As Integer = 0

    Public Sub Incrementar()
        valor += 1
    End Sub

    Public Sub Decrementar()
        valor -= 1
    End Sub
    Public Sub Inicializar(ByVal n As Integer)
        valor += n
    End Sub

    Public Function getValor() As Integer
        Return valor
    End Function
End Class
```



Programación orientada a objetos (III)

```
Public Class Punto
    Private x As Double
    Private y As Double
    Public Sub Desplazar(ByVal dx As Double, _
                ByVal dy As Double)
        x += dx
        y += dy
    End Sub
    Public Function Distancia( _
               ByVal p As Punto) As Double
        Return ((p.x - x) ^ 2 + (p.y - y) ^ 2)^0.5
    End Function
End Class
```



Programación orientada a objetos (IV)

Instanciación.

- Instanciar: crear un espacio de almacenamiento en una zona de memoria dinámica (reservar espacio para un puntero).
- Para usar un objeto es necesario reservar espacio en el montículo y asignar la posición de ese espacio a una referencia de la clase.

```
Dim conta As New Contador
conta.Inicializar(3)
conta.Incrementar()
conta.Decrementar()
System.Console.WriteLine(conta.getValor)

Dim p1 As New Punto
Dim p2 As New Punto
p1.Desplazar(0, 10)
System.Console.WriteLine(p1.Distancia(p2))
```

Constructores

- Métodos que se ejecutan cuando se crea una nueva instancia de la clase.
- Se suelen utilizar para inicializar los campos y otro tipo de acciones de inicialización (abrir archivos, conectar a bases de datos, etc.)
- En Visual Basic, dicho método se identifica mediante el nombre `New`.

```
'Para la clase Contador
Sub New()
    valor = 0
End Sub
'Para la clase Punto
Sub New()
        x = 0
        y = 0
End Sub
```

Constructores (II)

- Constructores con argumentos.
  - Los constructores pueden llevar argumentos para modificar la forma de inicialización.

```
'Para la clase Contador
Sub New(ByVal n As Integer)
    valor = n
End Sub
'Para la clase Punto
Sub New(ByVal x As Single, ByVal y As Single)
    Me.x = x     'Me se utiliza para evitar ambiguedad
    Me.y = y
End Sub
…
Dim otroContador = New Contador(5)
Dim p3 As New Punto(3.5, 2.6)
```

Constructores (III)

- Sobrecarga de constructores.
  - Sobrecargar un método supone la creación de otro método con el mismo nombre, pero con distintos argumentos.
    - ✓ Dependiendo de los argumentos con los que se mande el mensaje, en tiempo de ejecución, el compilador llamará a

```
'Crea un nuevo punto en la posición 0,y
Sub New(ByVal y As Single)
   'Llama al constructor sin argumentos: x=0, y=0
   'Sería equivalente a poner x=0
   Me.New()

   Me.y = y
End Sub
…
Dim p4 As New Punto(2.8) 'Crea un punto en 0,2.8
```

Tiempo de vida de un objeto

- VB.Net no dispone de destructores.
- Recolector de basura (recolector de elementos no utilizados, *Garbage Collector*, GC).
  - Rastrea la memoria y marca aquellos objetos que están referenciados por alguna variable.
  - Elimina de la memoria los objetos no marcados y compacta el montículo.

```
Dim p1 as New Punto(3.5,6.8)
Dim p2 as New Punto(1.4,6.3)
p1 = p2
'La referencia al punto 3.5,6.8 se pierde
'y queda disponible para el GC
```

Tiempo de vida de un objeto (II)

- El método `Finalize` de la clase `System.Object` se ejecuta justo ante de que el objeto se elimine de la memoria.
- Se puede sobrecargar para que se comporte de forma adecuada al objeto (liberar recursos, cerrar archivos, etc.).

```
Protected Overrides Sub Finalize()
    'Se puede incluir aquí el código que libere recursos
    Debug.WriteLine("El punto " & Me.x & "," & _
                Me.y & " está siendo destruido")
End Sub
```

- Algunas clases presentan el método `Dispose` que se puede llamar manualmente y que podrá ejecutar el código de finalización.

Procedimientos Property

- Acceso a atributos privados.
  - Mediante métodos ayudantes.

```
Public Function GetY() As Double
    Return y
End Function
Public Function GetX() As Double
    Return x
End Function
Public Sub SetY(ByVal y As Double)
    Me.y = y
End Sub
Public Sub SetX(ByVal x As Double)
    Me.x = x
End Sub
…
p4.SetX(3)
System.Console.WriteLine(p4.GetX)
```

Procedimientos Property (II)

- Los bloques `Property…End Property` permiten implementar métodos ayudantes.
- Dentro del bloque habrá otro bloque

```
'En la clase Contador
Private attValor as Integer
…
Property valor() As Integer
    Get
        Return attValor
    End Get
    Set(ByVal Value As Integer)
        attValor = Value
    End Set
End Property
…
System.Console.WriteLine(conta.valor)
```

Procedimientos Property (III)

- Atributos de sólo lectura.
  - Se pueden crear, omitiendo el bloque `Set` en la propiedad.
  - También se puede utilizar la palabra clave `ReadOnly` en la declaración del procedimiento `Property`.
  - Los valores sólo se podrán modificar en el constructor de la clase.
- Atributos de sólo escritura.
  - Se crean omitiendo el bloque `Get` o utilizando la palabra clave `WriteOnly` en la declaración del procedimiento.

Sobrecarga de métodos

- Un método sobrecargado es un método que tiene distintas versiones con el mismo nombre y con distintos argumentos.
  - Al mandar un mensaje al objeto, los argumentos del mensaje permitirán al compilador elegir qué procedimiento debe invocar.
  - Para diferenciar un método sobrecargado no se puede utilizar:
    - ✓ Diferencias en el paso de argumentos (`ByVal`, `ByRef`).
    - ✓ Los nombres de los parámetros.
    - ✓ El tipo de dato devuelto.
  - Sólo habrá que utilizar el número, tipo u orden de los parámetros.
- La palabra clave `Overloads` indica que un método está sobrecargado.

```
Overloads Public Function MiFunción() As Integer
```

  - La palabra clave `Overloads` es opcional.
    - ✓ Si se utiliza, es necesario utilizarla en todos los métodos sobrecargados.

Sobrecarga de métodos (II)

```
Public Sub Incrementar()
    valor += 1
End Sub
Public Sub Incrementar(ByVal n As Integer)
    valor += n
End Sub
Public Sub Decrementar()
    valor -= 1
End Sub
Public Sub Decrementar(ByVal n As Integer)
    valor -= 1
End Sub
Public Sub Inicializar(ByVal n As Integer)
    valor += n
End Sub
…
Dim conta as New Contador(1)
conta.Incrementar()   'valor = 2
conta.Incrementar(5)  'valor = 7
conta.Decrementar()   'valor = 6
conta.Decrementar(3)  'valor = 3
```

Miembros compartidos

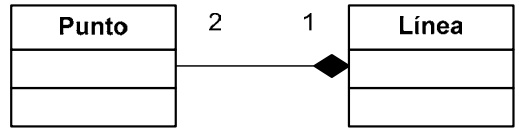
- Métodos o propiedades que no pertenecen a ninguna instancia en concreto, sino a la clase.
  - Pueden ser accedidos por todas las instancias de la clase.
  - La palabra reservada `Shared` después del modificador de accesibilidad indica que el campo es compartido.
  - Pueden ser llamados sin necesidad de crear una instancia de la clase.
  - La llamada se hace mediante *`nombreClase.nombreMiembro`*.
- Campos compartidos.
- Métodos compartidos.

Miembros compartidos (II)

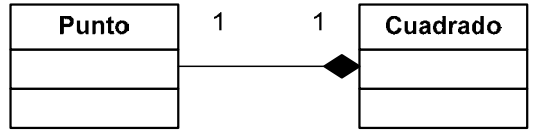
```
Private Shared NumPuntos = 0
…
Sub New()
    NumPuntos += 1
End Sub
Sub New(ByVal x As Single, ByVal y As Single)
    Me.New() 'Añadir esta llamada a todos los constructores
             'para que siempre incremente NumPuntos
    Me.x = x
    Me.y = y
End Sub
Public Shared Function GetNumPuntos() As Integer
    Return NumPuntos
End Function
…
System.Console.WriteLine(Punto.GetNumPuntos)
Dim p1 As New MisClases.Punto
Dim p2 As New MisClases.Punto
Dim p4 As New MisClases.Punto(2.8)
System.Console.WriteLine(p4.GetNumPuntos)
```

Relaciones entre clases

- Agregación y composición:
  - Se establece una relación entre el todo y la parte.
  - Los elementos agregados serían atributos del *todo*.
  - La clase Línea contiene dos objetos de la clase Punto: el punto del origen y el punto del final.
  - La clase Cuadrado tiene un Punto de origen y un ancho.




```
Public Class Línea
    Private origen As Punto
    Private final As Punto
    …
End Class
```




```
Public Class Cuadrado
    Private origen As Punto
    Private ancho as Double
    …
End Class
```

Relaciones entre clases (II)

```
Public Class Cuadrado
    Protected origen As Punto
    Protected ancho As Double
    'Crea un cuadrado nulo en el punto 0,0
    Sub New()
        origen = New Punto
        ancho = 0
    End Sub
    'Crea un cuadrado en un punto dado con un ancho determinado
    Sub New(ByVal p As Punto, ByVal a As Double)
        origen = p
        ancho = a
    End Sub
    'Devuelve el área de un cuadrado
    Public Function Área() As Double
        Return ancho * ancho
    End Function
    'Desplaza el punto de origen del cuadrado
    Public Sub Desplazar(ByVal dx As Double, ByVal dy As Double)
        origen.Desplazar(dx, dy)
    End Sub
    'Escala el cuadrado en un porcentaje determinado
    Public Sub Escalar(ByVal porc As Double)
        ancho += ancho * porc / 100
    End Sub
End Class
```

Herencia

- Generalización y especialización.
  - Relación entre dos clases de forma que una clase (subclase, clase derivada, clase hija) comparte la estructura o comportamiento de otra clase (superclase, clase padre).
  - Para describir la clase hija, no es necesario describir todos sus atributos y comportamientos, sino sólo los que difieran de la superclase.
  - La clase derivada:
    - ✓ Hereda todos los miembros que sean visibles de la superclase.
    - ✓ Redefine los métodos con el mismo nombre (redefinición de métodos).
    - ✓ Añade nuevos miembros.
  - La clase `Rectángulo`:
    - ✓ Hereda los atributos `origen` y `ancho` (`tienen una accesibilidad Protected`), y el método `Desplazar`.
    - ✓ Modifica los métodos `Área` y `Escalar`.
    - ✓ Añade el atributo `alto`.

Herencia (II)

- La declaración `Inherits` se utiliza para heredar una clase derivada a partir de la clase base.

```
Public Class Rectángulo
    Inherits Cuadrado
    Private alto as Double   'Añade el miembro alto
…
End Class
```

  - En VB.NET todas las clases de pueden heredar de forma predeterminada.
    - ✓ El modificador `NotInheritable` en la declaración de la clase evita que una clase se pueda derivar.
  - VB.NET no admite la herencia múltiple.
  - El tipo de acceso de una clase derivada debe ser igual o más restrictivo que el de la clase base.

Herencia (III)

- Sobreescritura de métodos de la clase base.
  - El modificador `Overridable` en un método de la clase base permite sobreescribir el método en la clase derivada.
  - El modificador `Overrides` en un método de la clase derivada modifica el método definido en la clase base.
  - El modificador `NotOverridable` en la clase base impide que el método se sobreescriba en la clase derivada.
    - ✓ Los métodos `Public` son de forma predeterminada `NotOverridables`.
    - ✓ E… …icador O…

```
'Calcula el área del rectángulo
Public Overrides Function Área() As Double
    Return getAncho() * alto
End Function
```

Herencia (IV)

- Palabra clave `MyBase`.
  - Hace referencia al miembro de la clase base.

```
'Crea un rectángulo en un punto dado con un ancho y alto determinado
Sub New(ByVal p As Punto, ByVal ancho As Double, ByVal alto As Double)
   MyBase.New(p, ancho)
   Me.alto = alto
End Sub
'Escala un rectángulo en un porcentaje determinado
Public Overrides Sub Escalar(ByVal porc As Double)
    MyBase.Escalar(porc)    'Llama al método Escalar de la
                            'clase Cuadrado
    alto += alto * porc / 100
End Sub
```

- Palabra clave `MyClass`.
  - Permite llamar a un método `Overridable` en la clase base y asegurarse de que se llama a la implementación del método en esta clase y no a la de un método reemplazado en una clase derivada

Métodos y clases abstractos

- Un método abstracto es un método sin implementación.
  - La declaración del método sólo precisa su firma.
- Una clase que contenga algún método abstracto será una clase abstracta.
  - Una clase abstracta no se puede instanciar (le falta implementación).
  - Sólo se puede utilizar a partir de una clase derivada.
- El modificador `MustOverride` en la declaración de un método le convierte en un método abstracto.
- El modificador `MustInherit` en la declaración de una clase le convierte en una clase abstracta.

Métodos y clases abstractos (II)

```
Public MustInherit Class Figura
    Protected origen As Punto
    Protected Shared numFiguras As Integer = 0

    Public Sub New()
        numFiguras += 1
    End Sub
    Public Shared Function getNumFiguras()
        Return numFiguras
    End Function

    'Desplaza una Figura
    Public Sub Desplazar(ByVal dx As Single, ByVal dy As Single)
        origen.Desplazar(dx, dy)
    End Sub
    'Calcula el área de una figura
    Public MustOverride Function Area() As Single
    'Calcula el perímetro de una figura
    Public MustOverride Function Perimetro() As Single
    'Modifica el tamaño de una figura en un porcentaje determinado
    Public MustOverride Sub Escalar(ByVal n As Single)
End Class
```

Métodos y clases abstractos (III)

```
Public Class Círculo :Inherits Figura
    Protected radio As Single
    'Crea un círculo nulo en el punto 0,0
    Public Sub New()
        MyBase.New()
        origen = New Punto
        radio = 0
    End Sub
    'Crea un círculo en el punto p con un radio determinado
    Public Sub New(ByVal p As Punto, ByVal r As Single)
        Me.New()
        origen = p
        Me.radio = r
    End Sub
    'Calcula el área de un círculo
    Public Overrides Function Area() As Single
        Return Math.PI * radio ^ 2
    End Function
    'Calcula el perímetro de un círculo
    Public Overrides Function Perimetro() As Single
        Return 2 * Math.PI * radio
    End Function
    'Modifica el radio de un círculo en un porcentaje
    Public Overrides Sub Escalar(ByVal porc As Single)
        radio += radio * porc / 100
    End Sub
End Class
```

Métodos y clases abstractos (III)

```
Public Class Cuadrado : Inherits Figura
    Protected ancho As Single
    'Crea un Cuadrado nulo en el punto 0,0
    Public Sub New()
        MyBase.New()
        origen = New Punto
        ancho = 0
    End Sub
    'Crea un Cuadrado en el punto p con un ancho determinado
    Public Sub New(ByVal p As Punto, ByVal ancho As Single)
        Me.New()
        origen = p
        Me.ancho = ancho
    End Sub
    'Calcula el área de un Cuadrado
    Public Overrides Function Area() As Single
        Return ancho * ancho
    End Function
    'Calcula el perímetro de un Cuadrado
    Public Overrides Function Perimetro() As Single
        Return ancho * 4
    End Function
    'Modifica el ancho y el alto de un en un porcentaje determinado
    Public Overrides Sub Escalar(ByVal porc As Single)
        ancho += ancho * porc / 100
    End Sub
End Class
```

Polimorfismo

- Capacidad de un método de tomar distintas formas en virtud del objeto que lo llame.
- Enlace estático y enlace dinámico del código.

```
Dim dibujo(4) As Figura
dibujo(0) = New Círculo(New Punto(200, 200), 150)
dibujo(1) = New Cuadrado(New Punto(150, 150), 100)
dibujo(2) = New Círculo(New Punto(200, 200), 20)
dibujo(3) = New Círculo(New Punto(200, 100), 50)
dibujo(4) = New Círculo(New Punto(200, 300), 50)
'Desplaza el dibujo
For i As Integer = 0 To 4
    'Llamada no polimórfica
    'Utiliza el método de la clase base
    dibujo(i).Desplazar(10, 10)
Next
'Escala el dibujo
For i As Integer = 0 To 4
    'Llamada polimórfica
    'Utiliza una forma diferente según el objeto
    dibujo(i).Escalar(50)
Next
```