Programación con **Visual Basic .NET**

2 – El Lenguaje Visual Basic .NET

Francisco Ruiz
Manuel Ángel Serrano

Escuela Superior de Informática
Universidad de Castilla-La Mancha

Programación con **Visual Basic .NET**

Contenidos sesión 2

- Aplicaciones de consola
- Conceptos básicos
  - Estructura de una Aplicación
  - Variables y constantes
  - Arrays
  - Depuración de Código
  - Operadores
  - Funciones predefinidas
  - Sentencias y líneas múltiples
- Procedimientos
  - Tipos
    - Sub
    - Function
  - Paso de parámetros
  - Sobrecarga
- Estructuras de control
  - Selección
  - Repetición
- Ámbito
  - De procedimientos
  - De variables
  - Vida de las variables
- Organización del código
  - Contenedores de código
  - Opciones del VS.NET
- Control de errores
- Práctica 1
  - Resolver ecuación 2do grado

Aplicaciones de consola (i)

- Se ejecutan dentro de una ventana de línea de comandos (estilo DOS).
- Clase Console.
  - Clase preconstruida del namespaces System.
  - No es necesario crear una instancia previa.
  - Métodos:
    - WriteLine: Escribir línea.
      - Textos, números, expresiones,

```
Console.WriteLine("Hola " & nombre)
Console.WriteLine(5>2)
```

      - Uso de parámetros

```
Nombre="Luis"
Console.WriteLine("Hola {0}, que tal?", Nombre)
```

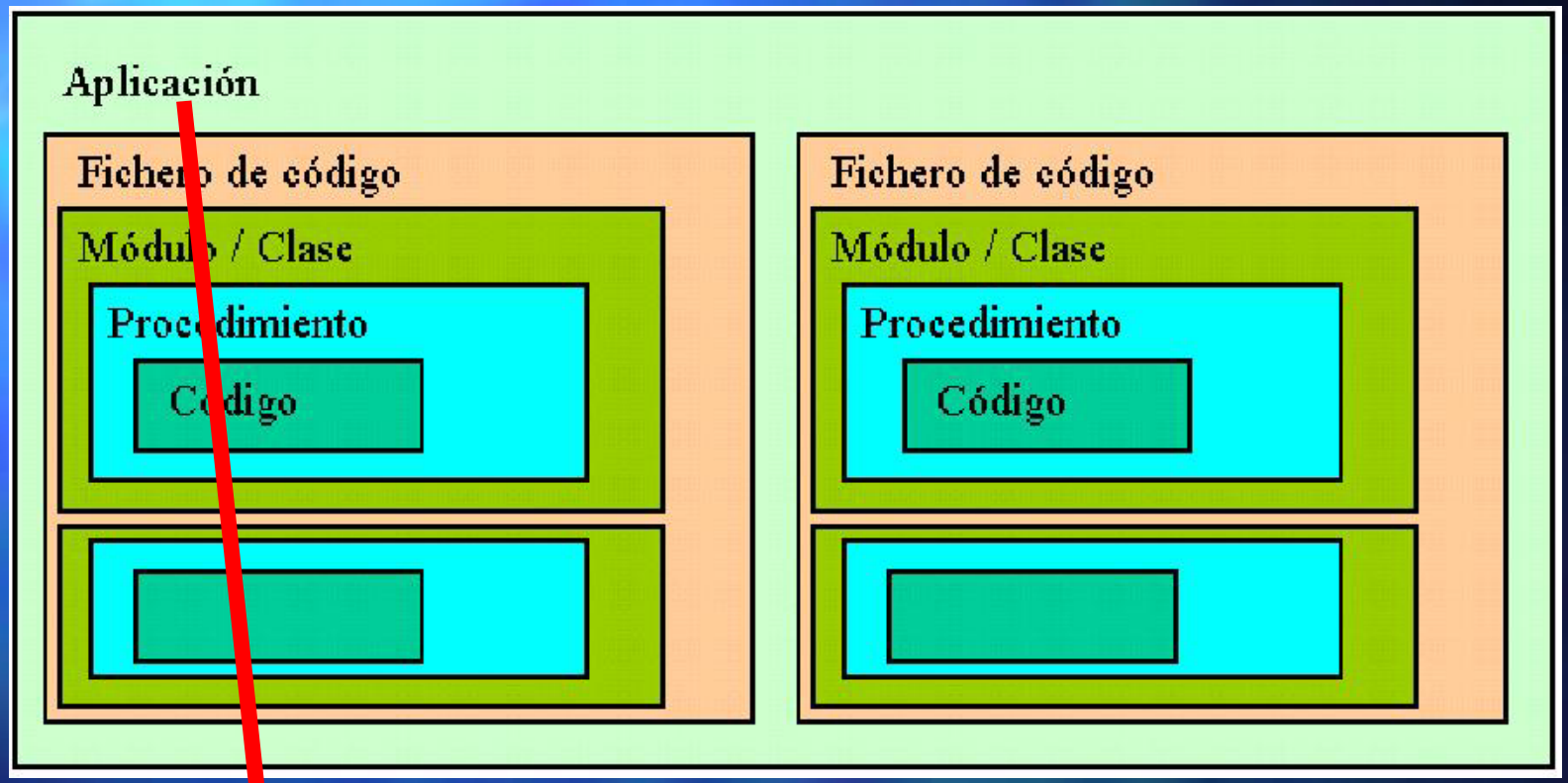
    - Write: Escribir sin salto de línea.
    - Readline: Leer texto tecleado.
      - Sirve para esperar hasta que usuario pulsa INTRO.
    - Read: leer una tecla o carácter tecleado.

Aplicaciones de consola (ii)

- Ejemplo: Mostrar código y carácter de las teclas pulsadas acabando al pulsar INTRO

```
Dim CodTecla as Integer
Dim NomTecla as String
Console.WriteLine("Pulsar teclas, acabar con INTRO")
Console.WriteLine() ' escribe línea en blanco
While True
   CodTecla=Console.Read() ' lee una tecla pulsada
   If CodTecla=13 then ' si pulsado INTRO
      Exit While
   End If
   Console.WriteLine("código de tecla: {0}", CodTecla)
   NomTecla=Chr(CodTecla)
   ConsoleWriteLine ("carácter de tecla: {0}", NomTecla)
End While
Console.WriteLine("Ejemplo acabado")
Console.ReadLine()
```

Estructura de una aplicación



Proyecto: ficheros de código, recursos y referencias a clases globales

Conceptos básicos de VB.NET
Variables y constantes (i)

- Declaración de Variables:

```
Dim MiVar As Integer
```

  - Al comienzo del procedimiento
  - Tipos en VB.NET *(equivalentes en .NET Framework)*
    - Boolean, Byte, Char, Date (DateTime), Decimal, Double, Integer (Int32), Long (Int64), Short (Int16), Single, String,
    - Object [cualquier tipo, tipo por defecto]
    - Definido por el usuario
  - Obligatoriedad de la declaración
    - **`Option Explicit {Off|On}`**
    - A Nivel de Proyecto / de Fichero
  - Obligatoriedad de la tipificación
    - **`Option Strict {Off|On}`**
    - A Nivel de Proyecto / de Fichero

Conceptos básicos de VB.NET
Variables y constantes (ii)

- Asignación

```
MiVar = 6
Dim Nombre As String = “Luis”
```

- Valor por defecto
  - 0 ; “” ; 01/01/0001 ; False ; Nothing
- Declaración de Constantes:

```
Const Color As String = “Azul”
```

Conceptos básicos de VB.NET
Arrays

- Declaración:

```
Dim Colores() As String
Dim Nombres(3) As String  'crea 4: 0,1,2,3
Dim Frutas() As String ={"Manzana","Pera"}
```

- Asignar y obtener valores:

```
Nombres(3)="Pepe"
Var = Nombres(3)
```

- Modificar Tamaño:
  - Perdiendo los valores anteriores

```
ReDim Nombres(6)
```

  - Conservándolos

```
ReDim Preserve Nombres(6)
```

- Saber el número de elementos:

```
UBound(Nombres)
```

VS.NET

Depuración de código – ejemplo

- Demo con programa "PreguntarUsuario"
  - Quitar "Dim Nombre As String" y probar efecto de
    - `Option Explicit {Off|On}`
  - Quitar sólo "As String" y probar efecto de
    - `Option Strict {Off|On}`
  - Dejar "Dim Nombre As String" bien
  - Probar depuración paso a paso
  - Consultar valores de variables
  - Ventana de comandos
    - Permite ejecutar órdenes interactivas

Conceptos básicos de VB.NET
Operadores

- Aritméticos
  `^ * / \ Mod + -`
- Concatenación de strings
  `&`
- Asignación
  `=` **abreviada:** `A += B` **equiv.** `A = A+B`
- Comparación
  `< <= > >= = <>`
  - De cadenas `Option Compare {Binary|Text}`
    - Con patrón `"Bonito2" Like "B*to#"`
    - Comodines: `? * # [lista] [!lista]`
  - De Objetos `ObjetoA Is ObjetoB`
- Lógicos y manejo de bits
  `And Not Or Xor AndAlso OrElse`

Conceptos básicos de VB.NET
Funciones predefinidas

- Códigos ASCII
  `Asc Chr`
- Comprobación de Tipos
  `IsNumeric IsDate IsArray`
- Numéricas
  `Int Fix Randomize Rnd`
- Cadenas
  `Len Space InStr Left Right Mid Replace LTrim Rtrim`
  `Trim UCase Lcase Format StrConv`
- Tiempo (fecha y hora)
  `Now DateAdd DateDiff DatePart`

Conceptos básicos de VB.NET

Sentencias y líneas múltiples

- Sentencia multilínea
  - Acabada en un subrayado _
- Línea multisentencia
  - Separadas por :

Procedimientos
Tipos

- Todo el código ejecutable está contenido en rutinas, llamadas procedimientos, de tres tipos:
  - Sub (*procedimientos*), no devuelven valor
  - Function (*funciones*), devuelven un valor
  - Property (*para manejar propiedades de objetos*)
- Llamada
  - Sub

```
ProcPrueba()
```

  - Function

```
CalcRaiz()
Resultado = CalcRaiz() + 5
```

Procedimientos
Declaración y código

- Sub

```
[Ámbito] Sub NombreProcedimiento[(ListaParámetros)]
    [CódigoEjecutable]
    [Exit Sub | Return]
    [CódigoEjecutable]
End Sub
```

- Function

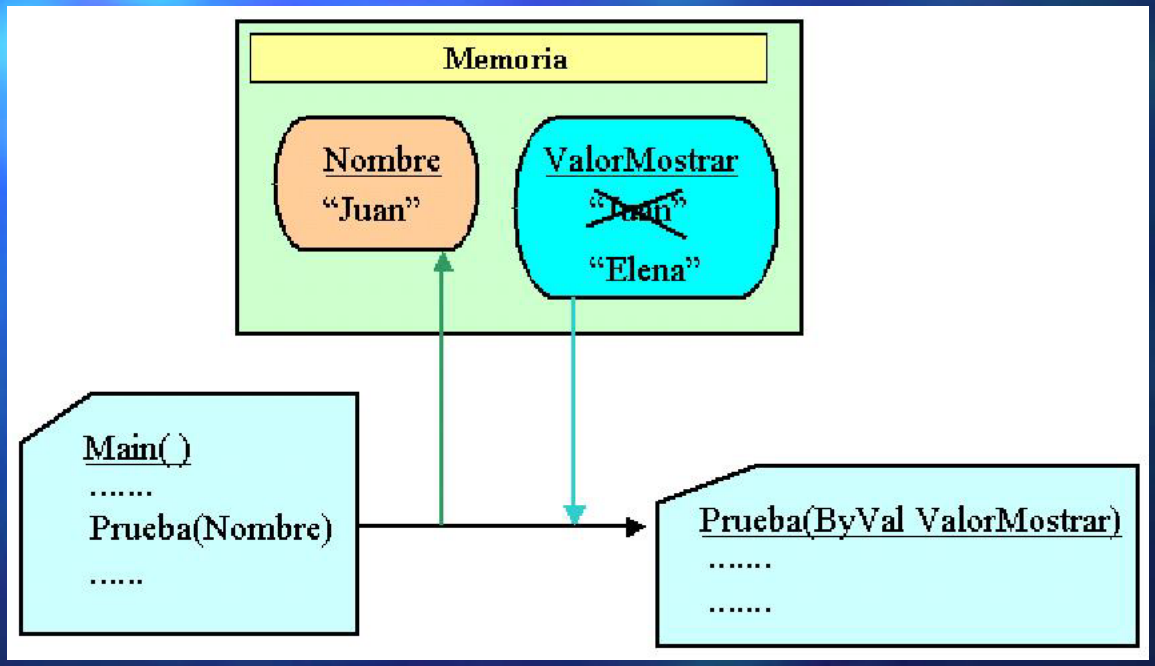
```
[Ámbito] Function NombreFunción[(ListaParámetros)] As
            TipoDato
    [CódigoEjecutable]
    [Return Valor]
    [NombreFunción = Valor]
    [Exit Function]
    [CódigoEjecutable]
End Function
```

Procedimientos

Paso de parámetros (i)

```
[Optional] [ByVal|ByRef] [ParamArray] Nombre As
   TipoDato
```

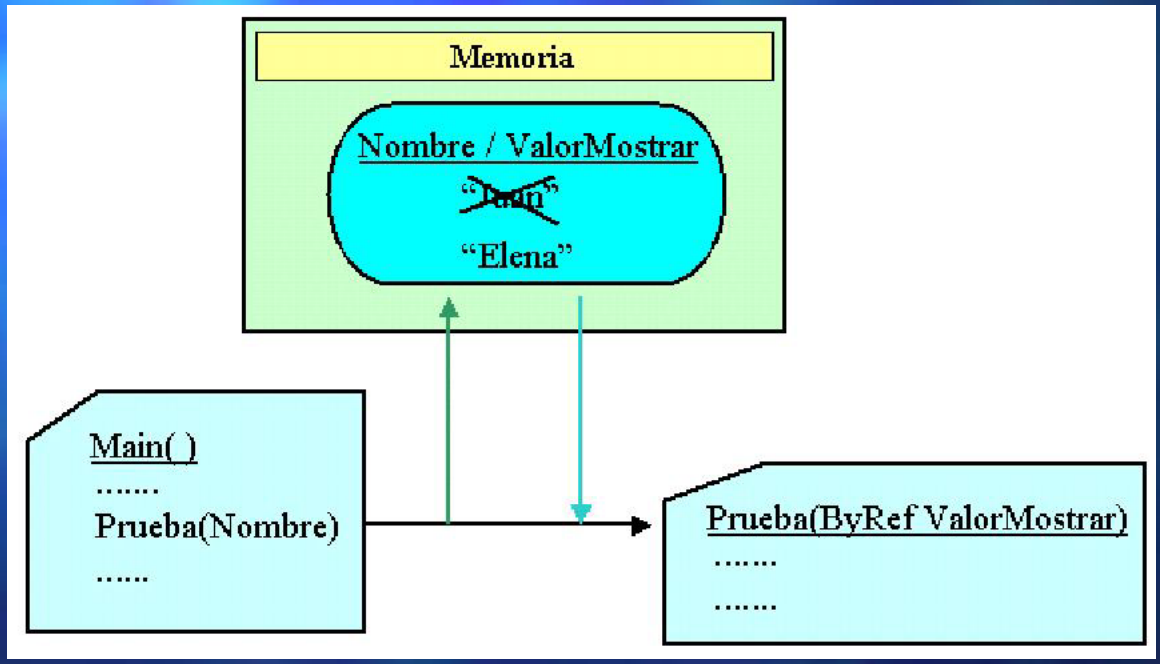
- Paso por Valor
  - Opción por defecto, Se crea una nueva variable



Procedimientos

Paso de parámetros (ii)

- Paso por Referencia
  - Ambas variables utilizan la misma memoria.
  - Los cambios afectan a la variable del código llamador.



Procedimientos

Paso de parámetros (iii)

- Asociación entre los parámetros escritos en la llamada y los escritos en la declaración del procedimiento
  - Por posición

    ```
    Prueba (Importe, DiaHoy)
    ```
  - Por nombre

    ```
    Prueba (Fecha:=DiaHoy, Cantidad:=Importe)
    ```

Procedimientos
Sobrecarga

- Varias versiones de un procedimiento (mismo nombre), pero con listas de parámetros diferenciadas en su número, orden o tipo.

```
Overloads Sub Datos()
    ' código del procedimiento
    ' ............
End Sub
Overloads Sub Datos(ListaParametrosA)
    ' código del procedimiento
    ' ............
End Sub
Overloads Function Datos(ListaParametrosB) As
   TipoDatoRetorno
    ' código del procedimiento
    ' ............
End Function
```

Estructuras de Control

- Permiten cambiar el flujo de ejecución a formas no secuenciales.
- Tipos
  - Selección *(decisión/selección)*
    - If ... Then ... End If
    - Select ... Case ... End Select
  - Repetición *(bucle/iteración)*
    - While ... End While
    - Do ... Loop
    - For ... Next
    - For Each ... Next

Estructuras de Control
Selección (i)

- If Simple

```
If Expresión Then
   Código
End If

If Expresión Then Instrucción
```

- If Doble

```
If Expresión Then
   Código
Else
   Código
End If

If Expresión Then Instrucción1 Else Instrucción2
```

Estructuras de Control
Selección (ii)

- If Múltiple

```
If Expresión1 Then
   Código
ElseIf Expresión2 Then
   Código
   ...
[ElseIf ExpresiónN Then]
   Código
Else
   Código
End If
```

- Select ... Case

```
Select Case Expresión
  Case Lista1
    Código
  [Case Lista2]
    Código
  [Case Else]
    Código
End Select
```

- ListaN:
  - Expresión
  - Expr1 To Expr2
  - Is OpComparación Expresión

Estructuras de Control
Repetición (i)

- While ... End While

```
While Expresión
  Código
End While
```

- For ... Next

```
For contador = inicio To fin [Step incremento]
  Código
  [Exit For]
  Código
Next

For Each elemento In Colección/Array
  Código
  [Exit For]
  Código
Next
```

Estructuras de Control
Repetición (ii)

- Do ... Loop
  - Condición al principio

```
Do { While | Until } Expresión
   Código
   [Exit Do]
   Código
Loop
```

  - Condición al final *=> se ejecuta al menos 1 vez*

```
Do
   Código
   [Exit Do]
   Código
Loop { While | Until } Expresión
```

  - Sin Condición
    - Peligro de bucle infinito

Ámbito
Procedimientos

- Capacidad de poder llamar a un procedimiento desde un punto dado del código.
  - Depende del nivel de acceso indicado en la declaración del Sub/Function:

```
Ámbito { Sub | Function } Nombre ([Parámetros])
```

  - **Public**: Público
    - Puede ser llamado desde cualquier módulo del proyecto.
  - **Private**: Privado
    - Sólo puede ser llamado desde el módulo en que se ha declarado.

Ámbito
Variables (i)

- Capacidad de poder usar una variable desde un punto dado del código.
  - Depende del nivel de acceso indicado en la declaración de la variable y del lugar donde está dicha declaración:

```
Ámbito [Dim] Nombre As TipoDato
```

  - A nivel de procedimiento (sub/function)
    - `Dim Nombre ...`
  - A nivel de bloque (estructura de control)
    - `Dim Nombre ...`
  - A nivel de módulo (module)
    - `Private Nombre ...`
  - A nivel de proyecto (module)
    - `Public Nombre ...`

Ámbito
Variables (ii)

```
Module A
    Public V1 As String
    Private V2 As Integer
    Public Sub Main()
       Dim V3 As Char
       ...
    End Sub
    Private Sub Proc1()
       If V2>5 then
          Dim V4 as string
          ...
       End If
    End Sub
End Module

Module B
    Public Sub Proc2()
       ...
    End Sub
End Module
```

V3

ámbito de procedimiento

V4

ámbito de bloque

Vida de las variables

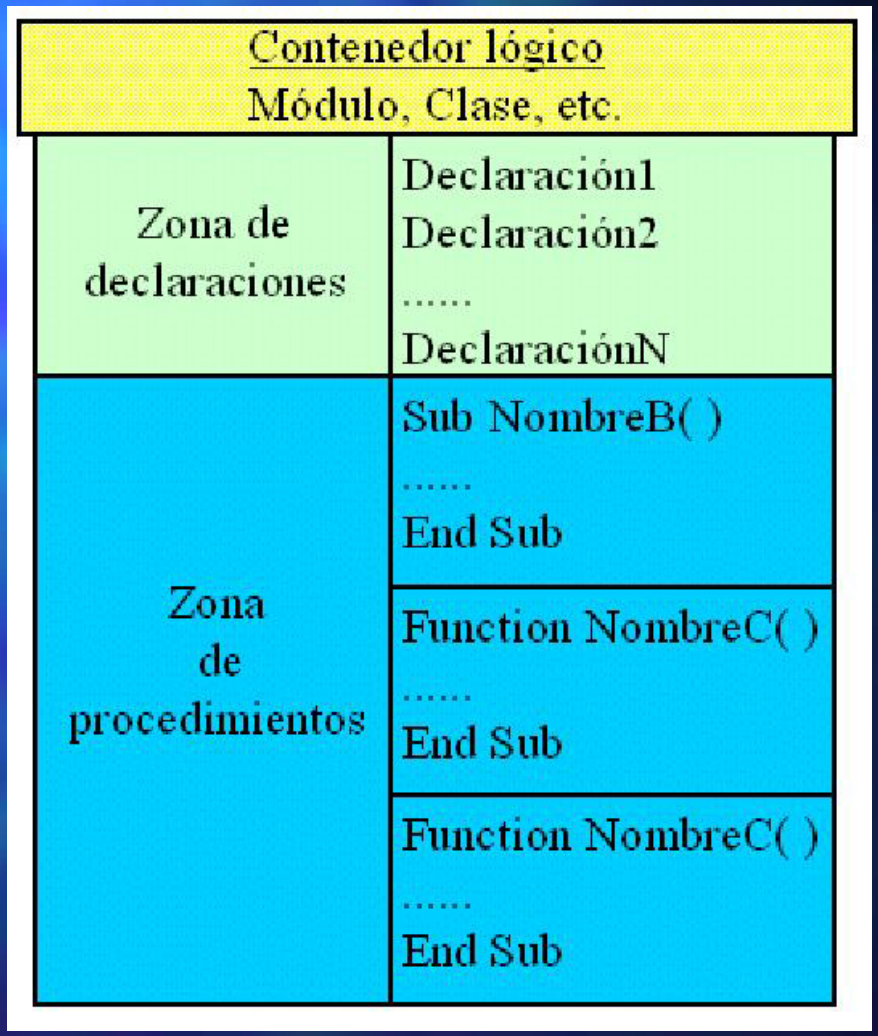
- El periodo de vida (cuando está activa) depende de su ámbito:
  - De bloque: desde que se declaran hasta que acaba la ejecución del bloque.
  - De procedimiento: desde que se declaran hasta que termina la ejecución del procedimiento.
  - De módulo o proyecto: ejecución de la aplicación.
- Variables Static
  - Retienen su valor al finalizar el procedimiento/bloque donde se declaran.

```
Static [Dim] Nombre As TipoDato
```

  - Su periodo de vida es el de ejecución de la aplicación

Organización del código

- Contenedores de código
  - Físicos: archivos .VB
  - Lógicos: elementos con declaraciones y procedimientos
    - Módulos, Clases, Interfaces, Estructuras, ...
    - Namespaces (metacontenedores)



Organización del código - demo

- Probar opciones de VS.NET para organizar el código:
  1. Listas desplegables del editor de código:
     - De Clases, para elegir el módulo/clase.
     - De Métodos, para elegir el procedimiento/método.
  2. Agregar nuevo módulo (y fichero).
  3. Crear módulo dentro de un fichero existente.
  4. Cambiar nombre de un fichero.
  5. Excluir un fichero de código.
  6. Añadir un fichero de código ya existente.

Manejo de Errores

- 2 conceptos relacionados
  - Error: Evento que se produce durante el funcionamiento de un programa, provocando una interrupción en su flujo de ejecución. Al producirse esta situación, el error genera un objeto excepción.
  - Excepción: Un objeto generado por un error, que contiene información sobre las características del error que se ha producido.
- 2 técnicas de gestión de errores:
  - Estructurada: mediante excepciones y una estructura de control para detectar las excepciones producidas.
  - No estructurada: mediante detección y captura de errores y saltos no estructurados (GO TO) en el código.

Manejo de Errores
Estructurado (i)

```
Try
  Código sensible a errores
  [Exit Try]
  Código sensible a errores
[Catch [Excepción [As Tipo1]] [When Expresión]
  Código respuesta a error de tipo 1
  [Exit Try]]
...
[Catch [Excepción [As TipoN]] [When Expresión]
  Código respuesta a error de tipo N
  [Exit Try]]
...
[Finally
  Código posterior al control de errores]
End Try
```

Manejo de Errores
Estructurado (ii)

- Clase Exception
  - Message: descripción del error.
  - Source: objeto/aplicación que originó el error.
  - StackTrace: Ruta o traza del código donde se produjo el error.
  - ToString(): Devuelve información detallada en un string.
- Captura

```
Try
  x=x/y
Catch ex As OverFlowException When y=0
  MsgBox(ex.toString)
Catch ex As Exception
  Console.Writeline(ex.Message)
  Console.Writeline(ex.Source)
End Try
```

Manejo de Errores
No Estructurado

- Objeto Err
  - Proporciona información sobre los errores que se producen.
    - Number
    - Description
    - Clear() *inicializa el objeto*
    - Raise() *genera un error provocado*
- Captura

```
On Error GoTo EtiquetaLínea
On Error Resume Next
```

- Reanudación

```
Resume [Next]
```

- Desactivación

```
On Error GoTo 0
```

Práctica 1

Resolver ecuación 2do grado (i)

- Mostrar un formulario para preguntar los tres coeficientes de una ecuación de segundo grado:

  A*x^2 + B*x + C = 0

  - Preguntar cada coeficiente en un control de tipo TextBox
- Añadir un botón "Calcular" para obtener las soluciones invocando al procedimiento de igual nombre.
  - Indicar si las soluciones son reales o imaginarias con un control CheckBox.
  - Mostrar las 2 soluciones en un control etiqueta "Solución" con texto azul si son reales y rojo si son imaginarias.
  - Ejemplo formato 2 soluciones reales: "2'45 y 78'23"
  - Ejemplo formato 1 solución real: "-9'06"
  - Ejemplo formato 2 soluciones imaginarias: "1'48+2'1i y 0'63-1'86i"
- Añadir un botón "Salir" para acabar.