Programas de Aplicación III

Tema 2. Elementos del lenguaje (Parte 2)

Luis Rodríguez Baena y María Dorrego Luxán

Universidad Pontificia de Salamanca (campus Madrid)
Facultad de Informática

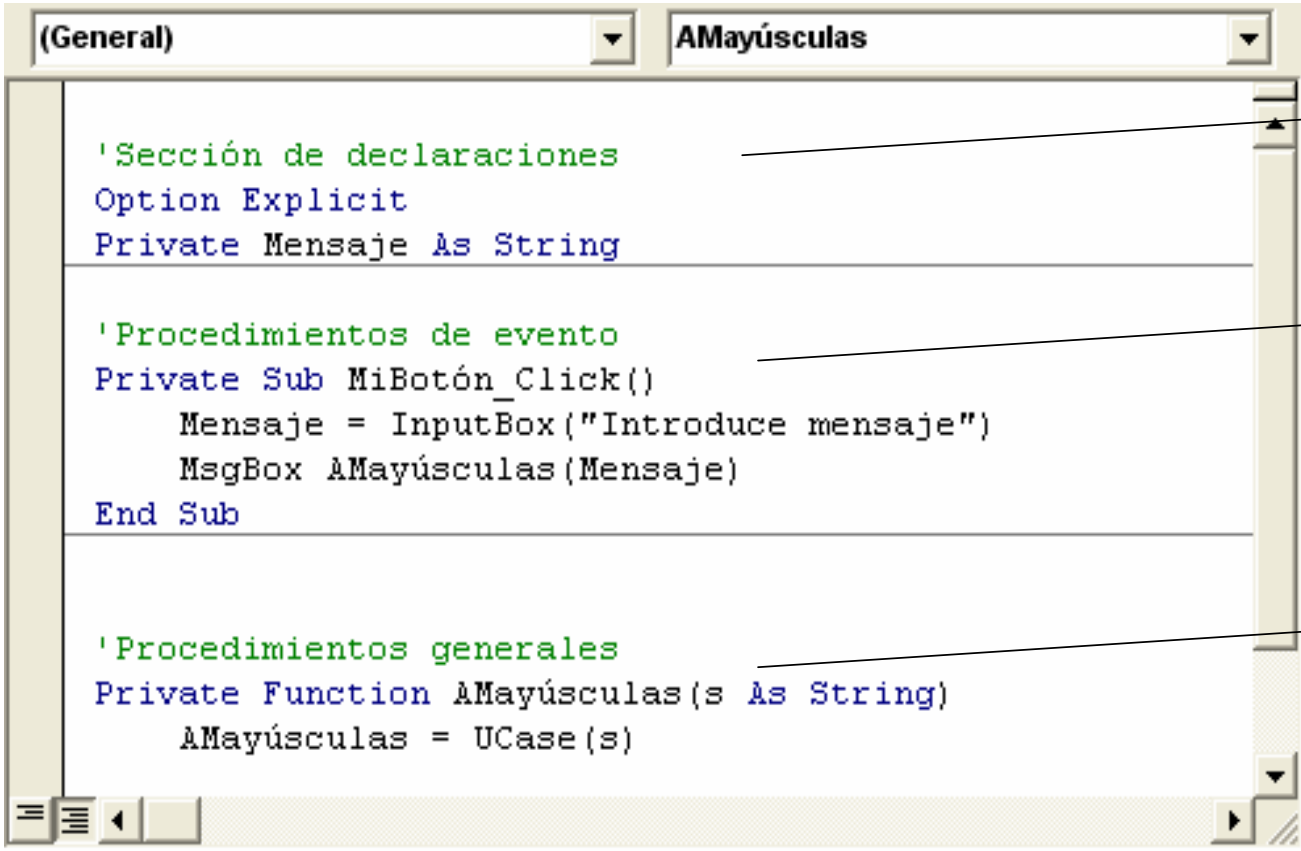
Módulos (I)

- Tipos de módulos:
  - Módulos de código (*.bas).
  - Módulos de formulario o formularios (*.frm).
  - Módulos de clase (*.cls).
- Estructura de un módulo.
  - Sección de declaraciones.
  - Procedimientos de eventos (sólo los formularios).
  - Procedimientos generales.
- Sección de declaraciones.
  - Aparece antes de cualquier instrucción ejecutable.
  - Incluye declaraciones a nivel de módulo de variables, constantes, DLL, etc.

Módulos (II)

- Procedimientos de eventos.
  - Se ejecutan en respuesta a los eventos de un determinado control.
  - Sólo existen en los módulos de formulario.
  - Utiliza plantillas predefinidas que presenta cada control.
  - Sub *nombreControl_nombreEvento*([*argumentos*]).
- Procedimientos generales.
  - Aparecen en la sección general de un módulo de código o formulario.
  - Contienen los subprogramas de uso general.
    - ✓ No están asociados a ningún control de ningún formulario.

Módulos (III)



Sección de declaraciones

Procedimientos de evento

Procedimientos generales

Procedimientos Sub

- Declaración.

```
[Private|Public][Static] Sub nombreProc [(argumentos)]
   [instrucciones]
   [Exit Sub]
End Sub
```

- Llamada a un procedimiento

```
Call nombreProc[(argumentos)]

nombreProc argumentos
```

Procedimientos Function

- Declaración.

```
[Private|Public][Static] Fuction nombreFunc
        ([argumentos]) [As TipoDato]
   [instrucciones]
   [Exit Function]
   [nombreFunc = expresión]
End Function
```

  - Tanto la llamada como la declaración incluyen los paréntesis.
  - El tipo de dato que devuelve por omisión es `Variant`.
  - Si no se incluye o no se ejectua la expresión de retorno devuelve 0, cadena nula, `Empty` o `Nothing`.

Ámbito de los procedimientos

- Ámbito de módulo.
  - Se declaran con `Private`.
- Ámbito de aplicación.
  - Se declaran con `Public`.
  - Llamada a procedimientos públicos.
    - ✓ Declarados en un módulo de código: `nombreProc`.
    - ✓ Declarados en un módulo de formulario: *`nombreForm.nombreProc`*.
- Procedimientos estáticos (modificador `Static`).
  - Todas sus variables son estáticas.

```
Private Static Sub p1()
    Dim i
    i = i + 1
    Debug.Print i
End Sub
```

Incrementa i cada vez que se llama al procedimiento

Paso de argumentos (I)

- El formato de cada argumento que pasamos es:

```
[Optional] [ByVal | ByRef] [ParamArray]
  nombreArgumento[( )] [As tipo] [= valorPredeterminado]
```

  - Por omisión los argumentos son de tipo `Variant`.
  - Si se indica el tipo es necesario que coincida con el de los parámetros actuales.

```
Private Function f1(a as String)
   …
End Function
…
Dim var as Variant
var = “Prueba”
```

La llamada `f(var)`, da error de tipo.

La llamada `f((var))`, no da error. Al pasar una expresión realiza la conversión de `Variant` a `String`

Paso de argumentos (II)

- Paso por referencia y paso por valor.
  - Por omisión, las expresiones se pasan por valor y las variables por referencia.
  - Se pueden utilizar los modificadores `ByVal` y `ByRef`.
- Parámetros opcionales.
  - `Optional` indica que el argumento es opcional. Deben ser los últimos argumentos de la lista.
  - Para detectar la presencia de argumentos opcionales se puede utilizar la función *`IsMissing(argumento)`*.

```
Function DevolverDos(Optional A)
   If IsMissing(A) Then
      DevolverDos = Null
   Else
      DevolverDos = A * 2
   End If
End Function
```

Paso de argumentos (III)

- Arrays de parámetros.
  - `ParamArray` indica que el argumento es un array de parámetros.
    - ✓ Array de `Variant` con un número variable de elementos.
  - Incompatible con `Optional`, `ByVal` o `ByRef`.
  - Debe ser el último elemento de la lista.

```
Function suma(ParamArray num()) As Integer
   Dim n as Integer
   Dim total as Integer
   For Each n In num
      total = total + n
   Next
   suma = total
End Function
```

Métodos

- Acciones asociadas a un objeto.
  - Cada control tiene su propia lista de métodos predefinida.
    - ✓ Para métodos que **no** devuelven valores:
      `NombreControl.NombreMétodo [argumentos]`
    - ✓ Para métodos que **si** devuelven valores:
      `NombreControl.NombreMétodo([argumentos]).`

Arrays (I)

- Declaración

  {Dim|Private|Public|Static} *nombreArray*([*límites*]) As *tipo*

  - ✓ Límites:

    [limInferior To] limSuperior [, [limInferior To] limSuperior]…

- Funciones UBound y LBound.
  - UBound(*array*[,*dimensión*]).
  - LBound(*array*[,*dimensión*]).

Arrays (II)

- Arrays de variant.
  - Dentro de un dato de tipo Variant también podemos meter arrays por lo que podríamos hacer un arrays de arrays de distintos tipos

```
Private intX As Integer                 ' Declara una variable contador.
Private Sub Command1_Click()
   'Declara y llena un array de enteros
   Dim countersA(5) As Integer
   For intX = 0 To 4
          countersA(intX) = 5
   Next intX
   ' Declara y llena un array de cadenas.
   Dim countersB(5) As String
   Dim strX As String
   For intX = 0 To 4
          countersB(intX) = "hello"
   Next intX
   Dim arrX(2) As Variant               ' Declara un array de dos miembros.
   arrX(1) = countersA()                ' Lo llena con otros arrays.
   arrX(2) = countersB()
   MsgBox arrX(1)(2)                    ' Muestra un elemento de cada array
   MsgBox arrX(2)(3)
End Sub
```

Arrays(III)

- Asignación de arrays.
  - ArrayA = ArrayB
    - ✓ Si el array de la izquierda es dinámico, ajusta las dimensiones.
    - ✓ Si el array de la izquierda es estático, ArrayB debe tener el mismo número de dimensiones y elementos.
- Arrays como argumento de funciones.

```
Dim b As Byte
Dim ReturnArray() As Byte
…
b  = CByte(100)
ReturnArray() = ArrayFunction(b)
…
Public Function ArrayFunction(b As Byte) As Byte()
   Dim x(2) As Byte
   x(0) = b
   x(1) = b + CByte(200)
   x(2) = b + b
   ArrayFunction = x
End Function
```

Arrays dinámicos

- Pueden redimensionarse en tiempo de ejecución.
- Declaración.

  `{Dim|Private|Public|Static}` *`nombreArray`*`() As` *`tipo`*
- Modificación del tamaño: instrucción `Redim`.
  - `Redim(`*`nuevosLímites`*`)`.
    - ✓ Instrucción ejecutable.
  - Elimina el contenido anterior del array.
  - `Redim Preserve` guarda el contenido anterior.
    - ✓ Sólo puede modificar el límite superior del array.

```
Dim a() As Integer
Private Sub p2()
    Static i As Integer
    i = i + 1
    ReDim Preserve a(i)
End Sub
```

Tipos definidos por el usuario

- Declaración.
  - Dentro de la sección de declaraciones de un módulo.

```
{Private|Public}Type NombreTipo
    NombreCampo = tipo
    …
End Type
```

- Tipos en los argumentos.
  - Siempre se pasan por referencia.

Enumerados

- Declaración.
  - Se debe hacer a nivel de módulo.

```
{Public|Private} Enum nombre
    nombreMiembro [=expresión]
    …
End Enum
```