Tema 5: Arrays.

- Conceptos básicos.
- Copiado y borrado de matrices.
- Búsqueda de elementos.
- Ordenación de matrices.
- Otras clases relacionadas.

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 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
        Dim par As Integer
        For i As Integer = 0 To n
                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“        'v2={"xxx", "adios", "pepe", "manolo"}
```

- Copia de matrices.
  - Método Clone.

```
'v2={"xxx", "adios", "pepe", "manolo"}

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={0,0,0,0,0,1,2,3,4,0}
```

- Método estático Copy.

  Array.Copy(*arrayOrig*,*inicioArrayOrig,arrayDest*,*inicio,ArrayDest.longitud)*

```
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 4
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
```

Otras clases relacionadas

- 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 (II)

```
'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
```

Otras clases relacionadas (III)

- 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.