Chapter 8: Array

Section 8.1: Array definition

```
Dim array(9) As Integer ' Defines an array variable with 10 Integer elements (0-9).

Dim array = New Integer(10) {} ' Defines an array variable with 11 Integer elements (0-10)
                               'using New.

Dim array As Integer() = {1, 2, 3, 4} ' Defines an Integer array variable and populate it
                                      'using an array literal. Populates the array with
                                      '4 elements.

ReDim Preserve array(10) ' Redefines the size of an existing array variable preserving any
                         'existing values in the array. The array will now have 11 Integer
                         'elements (0-10).

ReDim array(10) ' Redefines the size of an existing array variable discarding any
                'existing values in the array. The array will now have 11 Integer
                'elements (0-10).
```

Zero-Based

All arrays in VB.NET are zero-based. In other words, the index of the first item (the lower bound) in a VB.NET array is always `0`. Older versions of VB, such as VB6 and VBA, were one-based by default, but they provided a way to override the default bounds. In those earlier versions of VB, the lower and upper bounds could be explicitly stated (e.g. `Dim array(5 To 10)`. In VB.NET, in order to maintain compatibility with other .NET languages, that flexibility was removed and the lower bound of `0` is now always enforced. However, the `To` syntax can still be used in VB.NET, which may make the range more explicitly clear. For instance, the following examples are all equivalent to the ones listed above:

```
Dim array(0 To 9) As Integer

Dim array = New Integer(0 To 10) {}

ReDim Preserve array(0 To 10)

ReDim array(0 To 10)
```

Nested Array Declarations

```
Dim myArray = {{1, 2}, {3, 4}}
```

Section 8.2: Null Array Variables

Since arrays are reference types, an array variable can be null. To declare a null array variable, you must declare it without a size:

```
Dim array() As Integer
```

Or

```
Dim array As Integer()
```

To check if an array is null, test to see if it `Is Nothing`:

```
Dim array() As Integer
If array Is Nothing Then
    array = {1, 2, 3}
End If
```

To set an existing array variable to null, simply set it to `Nothing`:

```
Dim array() As Integer = {1, 2, 3}
array = Nothing
Console.WriteLine(array(0))  ' Throws a NullReferenceException
```

Or use `Erase`, which does the same thing:

```
Dim array() As Integer = {1, 2, 3}
Erase array
Console.WriteLine(array(0))  ' Throws a NullReferenceException
```

Section 8.3: Array initialization

```
Dim array() As Integer = {2, 0, 1, 6}                  ''Initialize an array of four Integers.
Dim strings() As String = {"this", "is", "an", "array"} ''Initialize an array of four Strings.
Dim floats() As Single = {56.2, 55.633, 1.2, 5.7743, 22.345}
              ''Initialize an array of five Singles, which are the same as floats in C#.
Dim miscellaneous() as Object = { New Object(), "Hello", New List(of String) }
              ''Initialize an array of three references to any reference type objects
              ''and point them to objects of three different types.
```

Section 8.4: Declare a single-dimension array and set array element values

```
Dim array = New Integer() {1, 2, 3, 4}
```

or

```
Dim array As Int32() = {1, 2, 3, 4}
```

Section 8.5: Jagged Array Initialization

Note the parenthesis to distinguish between a jagged array and a multidimensional array SubArrays can be of different length

```
Dim jaggedArray()() As Integer = { ({1, 2, 3}), ({4, 5, 6}), ({7}) }
' jaggedArray(0) is {1, 2, 3} and so jaggedArray(0)(0) is 1
' jaggedArray(1) is {4, 5, 6} and so jaggedArray(1)(0) is 4
' jaggedArray(2) is {7} and so jaggedArray(2)(0) is 7
```

Section 8.6: Non-zero lower bounds

With `Option Strict On`, although the .NET Framework allows the creation of single dimension arrays with non-zero lower bounds they are not "vectors" and so not compatible with VB.NET typed arrays. This means they can only be seen as `Array` and so cannot use normal array `(index)` references.

```
Dim a As Array = Array.CreateInstance(GetType(Integer), {4}, {-1})
For y = LBound(a) To UBound(a)
    a.SetValue(y * y, y)
Next
For y = LBound(a) To UBound(a)
    Console.WriteLine($"{y}: {a.GetValue(y)}")
Next
```

As well as by using `Option Strict Off`, you can get the `(index)` syntax back by treating the array as an `IList`, but then it's not an array, so you can't use `LBound` and `UBound` on that variable name (and you're still not avoiding boxing):

```
Dim nsz As IList = a
For y = LBound(a) To UBound(a)
    nsz(y) = 2 - CInt(nsz(y))
Next
For y = LBound(a) To UBound(a)
    Console.WriteLine($"{y}: {nsz(y)}")
Next
```

Multi-dimensional non-zero lower bounded arrays *are* compatible with VB.NET multi-dimensional typed arrays:

```
Dim nza(,) As Integer = DirectCast(Array.CreateInstance(GetType(Integer),
                                    {4, 3}, {1, -1}), Integer(,))
For y = LBound(nza) To UBound(nza)
    For w = LBound(nza, 2) To UBound(nza, 2)
        nza(y, w) = -y * w + nza(UBound(nza) - y + LBound(nza),
                                 UBound(nza, 2) - w + LBound(nza, 2))
    Next
Next
For y = LBound(nza) To UBound(nza)
    Dim ly = y
    Console.WriteLine(String.Join(" ",
        Enumerable.Repeat(ly & ":", 1).Concat(
            Enumerable.Range(LBound(nza, 2), UBound(nza, 2) - LBound(nza, 2) + 1) _
            .Select(Function(w) CStr(nza(ly, w))))))
Next
```

MSDN reference: Array.CreateInstance

Section 8.7: Referencing Same Array from Two Variables

Since arrays are reference types, it is possible to have multiple variables pointing to the same array object.

```
Dim array1() As Integer = {1, 2, 3}
Dim array2() As Integer = array1
array1(0) = 4
Console.WriteLine(String.Join(", ", array2))  ' Writes "4, 2, 3"
```

Section 8.8: Multidimensional Array initialization

```
Dim array2D(,) As Integer = {{1, 2, 3}, {4, 5, 6}}
' array2D(0, 0) is 1 ; array2D(0, 1) is 2 ; array2D(1, 0) is 4

Dim array3D(,,) As Integer = {{{1, 2, 3}, {4, 5, 6}}, {{7, 8, 9}, {10, 11, 12}}}
' array3D(0, 0, 0) is 1 ; array3D(0, 0, 1) is 2
' array3D(0, 1, 0) is 4 ; array3D(1, 0, 0) is 7
```