7.4.4 Eindimensionale und mehrdimensionale Arrays – Array-Dimensionen

Wenn man als Kriterium die Anzahl der Dimensionen eines Arrays zugrunde legt, so kann man ein- und mehrdimensionale Arrays unterscheiden.

Ein Array bezeichnet man als

- *eindimensional*, wenn es über die Syntax Type[Dim1] oder Type[ ] erstellt wurde.
- *abgeleitet eindimensional*, wenn es über die Syntax ObjektType[Dim1] oder ObjektType[ ] erzeugt wurde.
- *echt mehrdimensional* oder *nativ mehrdimensional*, wenn es über die Syntax Type[Dim1, Dim2, ..., DimN] (2 <= N <= 8) erstellt wurde oder
- *abgeleitet mehrdimensional*, wenn es über die Syntax Type[ ][ ]...[ ] erstellt wurde.

Die *Anzahl* der Dimensionen eines Arrays muss bei der Kompilierung festgelegt sein. Die Größe der Dimension können Sie direkt oder durch einen beliebigen numerischen Ausdruck festlegen.

Beispiele

Eindimensionale Arrays:

```
Dim myArrayS As New String[3]
Dim myArrayF As New Float[]
```

Echt mehrdimensionale Arrays:

```
Dim myArray As String[10, 5] ' → FEHLER! New fehlt
Dim a2DArrayB As New String[10, C_MAX * 5]
Dim a3DArray as New Integer[8, 8, 8]
Dim myArray4 As New String[10, 5]    *
```

Eine Alternative zur letzten Zeile sind die folgenden 2 Zeilen:

```
Dim myArray4 As New String[]
myArray4 = New String[10, 5] **
```

In beiden Fällen * und ** werden die beiden Dimensionen statisch festgelegt und können zur Laufzeit nicht mehr geändert werden!

Abgeleitete eindimensionale Arrays:

```
Dim aLabels As New Label[]
Dim aArrayComponents As New Components[]
Dim aClassArray As New CDS[] ' Array, dessen Elemente CDS-Objekte der Klasse CDS sind
```

Abgeleitetes mehrdimensionales Array:

```
Dim aLabels As New Label[][]
```

Beispiel 1 – Einsatz eines eindimensionalen, abgeleiteten Arrays

Das eingesetzte Array 'aLabels' ist einerseits eindimensional und anderseits ein abgeleitetes Array, weil 'Label' eine Klasse ist.

```
[1]  Public Sub btnShowLabels_Click()
[2]    Dim aLabels As New Label[]
[3]    Dim hControl As Control
[4]    Dim iCount As Integer
[5]
[6]    If aLabels <> Null And aLabels.Count = 0 Then
[7]       Message.Info("Das Array 'aLabels' existiert - ist aber leer.")
[8]    Endif
[9]
[10]   For Each hControl In ME.Children
[11]     If Object.Type(hControl) = "Label" Then aLabels.Add(hControl)
[12]   Next ' hControl
[13]
[14]   For iCount = 0 To aLabels.Max
```

```
[15]     aLabels[iCount].Tag = Str(iCount)
[16]     Print "Label-Name = "; aLabels[iCount].Name;
[17]     Print " | Label-Text = "; aLabels[iCount].Text;
[18]     Print " | Label-Tag = "; aLabels[iCount].Tag
[19]   Next ' iCount
[20]
[21] End ' btnShowLabels_Click()
```

Ausgabe in der IDE-Konsole:

```
Label-Name = lblCArray     | Label-Text = LabelArrays | Label-Tag = 0
Label-Name = lblACaption   | Label-Text = Demonstration Klasse ARRAY | Label-Tag = 1
Label-Name = lblMultiArray | Label-Text = Label1 | Label-Tag = 2
Label-Name = lblBArray     | Label-Text = Class Array | Label-Tag = 3
```

Kommentare:

- In der Zeile 2 wird das Arrays 'aLabels' erzeugt.
- Das Array 'aLabels' existiert als Array-Objekt – ist aber leer. Mit der Methode *aLabels.Resize(4)* könnten Sie dynamisch 4 leere Elemente einfügen. Das ist aber ungünstig, weil Sie nicht wissen, ob und wenn ja, wie viele Labels sich im (Haupt-)Fenster befinden.
- Daher werden in den Zeilen 10 bis 12 alle (sichtbaren) Komponenten im Programmfenster untersucht und nur jene Komponenten vom Objekt-Typ 'Label' werden dem Array 'aLabels' über die Add-Methode hinzugefügt. Diese Add-Methode ruft *automatisch* die Resize()-Methode auf, bevor ein Element in das Array eingefügt wird.
- Die (erweiterte) Ausgabe des Inhalts von Array 'aLabels' wird in der FOR-NEXT-Kontroll-Struktur in den Zeilen 14 bis 19 angeschoben, nachdem zuvor noch die *Tag-Eigenschaft* für alle Label gesetzt wurde.

Im Zusammenhang mit Untersuchungen zu den Dimensionen von mehrdimensionalen Arrays leistet die virtuelle Klasse *Array.Bounds* gute Dienste, wie das folgende Beispiel zeigt:

```
[1]  Dim k As Integer
[2]  Dim a2DArray As New String[10, 5]
[3]
[4]  ' Statische Anzeige der Dimensionen
[5]    Print "Dimension = "; a2DArray.Dim ' Anzahl der Dimensionen
[6]    Print "Dimension 1 (Anzahl Zeilen)  = "; a2DArray.Bounds[0]
[7]    Print "Dimension 2 (Anzahl Spalten) = "; a2DArray.Bounds[1]
[8]
[9]    Print "Dimension = "; a2DArray.Dim
[10]   Print "DIM-Count = "; a2DArray.Bounds.Count ' Anzahl der Elemente im Bounds-Arrays
[11]
[12] ' Dynamische Anzeige der Grenzen der einzelnen Dimensionen
[13]   For k = 0 To a2DArray.Bounds.Count - 1
[14]     Print "Dimension "; k + 1; " = "; a2DArray.Bounds[k]
[15]   Next
```

Ausgabe in der IDE-Konsole:

```
Dimension = 2
Dimension 1 (Anzahl Zeilen)  = 10
Dimension 2 (Anzahl Spalten) = 5

Dimension = 2
DIM-Count = 2

Dimension 1 = 10
Dimension 2 = 5
```

Kommentare:

- Es wird gezeigt, das die beiden Eigenschaften *a2DArray.Dim* und *a2DArray.Bounds.Count* hier gleiche Ergebnisse liefern → Zeilen 9 und 10.
- Die Eigenschaft *a2DArray.Bounds.Count* gibt die Anzahl der Elemente im Bounds-Array an und repräsentiert so die Bounds-Array-Dimension.
- Die dynamische Anzeige der Dimensionen in den Zeilen 13 bis 15 ist der statischen Anzeige in den Zeilen 5 bis 7 klar vorzuziehen.

Exkurs – Ergänzende Bemerkungen zu mehrdimensionalen Arrays:

- String[ ][ ] ist ein Array von Arrays. Sie erhalten ein zweidimensionales Array, das aber in allen Dimensionen veränderbar ist, d.h. Sie können zur Laufzeit Elemente in jeder Dimension anhängen oder auch nur oder auch nur in einer Dimension. Bei einem zweidimensionalen Array mit String[x, y] wird das nicht funktionieren, weil die Dimensionen statisch festgelegt sind.
- Ein Array ist vom Typ *aTyp*, wenn die Elemente im Array vom Typ *aTyp* sind. String[ ] enthält nur Zeichenketten, ein String[ ][ ] wird als Elemente String-Arrays vom Typ String[] enthalten, die wiederum Strings enthalten. Sie können den Interpreter von jedem Datentyp und von jeder Klasse *aTyp* eine Array-Klasse *aTyp[ ]* ableiten lassen. Ist *aTyp* ein nativer Datentyp, dann nennt man *aTyp[ ]* ein natives Array. Beachten Sie: *aTyp[ ]* ist wiederum eine Klasse und Sie können mit *aTyp[ ][ ]* die Klasse "Array von aTyp-Arrays" ableiten. Auf diese Weise haben Sie die Möglichkeit, n-dimensionale Arrays realisieren. Ein Limit für n existiert nicht und die Ausdehnung einer jeden Dimension ist nicht statisch.
- Schreiben Sie aTyp[ ][ ], so haben Sie ein Array von Arrays vom Typ aTyp – was immer aTyp auch für ein Typ ist. aTyp[ ][ ] ist also eine zweidimensionale Anordnung von aTyp-Variablen. Sie greifen auf eine dieser Variablen mit MyaTypArray[i][j] zu. Ob Sie zum Beispiel i und j als 2 Integer-Schlüssel interpretieren wollen, liegt an der Art der Verwendung des Arrays myaTypArray[i][j] in Ihrer Anwendung.
- Eine andere Möglichkeit der Deklaration von n-dimensionalen Arrays (1 <= n <= 8) besteht darin, eine "echte" multidimensionale Array-Klasse zu verwenden. Diese Arrays haben die Form aTyp[n1, n2, ..., nN], wobei N (<= 8) die Anzahl der Dimensionen ist und n1 die Ausdehnung der ersten Dimension, n2 die der zweiten, usw.. Hier sei bemerkt, dass diese echt-multidimensionalen Arrays den abgeleiteten mehrdimensionalen Arrays jedoch in mindestens drei Punkten unterlegen sind:

  (1) Die Anzahl der Dimensionen ist auf 8 nach oben beschränkt.
  (2) Die Ausdehnung einer jeden Dimension ist statisch.
  (3) Sie können nur Matrizen bilden.

  Was den Punkt (3) betrifft, so ist er wohl der subtilste unter den dreien. Abgeleitete mehrdimensionale Arrays erlauben es, Systeme zu erstellen, in denen eine Dimension nicht überall gleichmäßig ausgedehnt sein muss – wovon Matrizen ein Spezialfall sind – denn dort muss gerade das gelten.