7.4.2 Dynamische Arrays

Ein dynamisches Array ist ein Array, das ein echtes Gambas-Objekt ist. Die Größe eines dynamischen Arrays kann zur Laufzeit abgefragt werden. Ein dynamisches Array mit *einer Dimension* kann jederzeit in seiner Größe geändert werden.

Beispiel

```
[1]  Public Sub btnShowExample_Click()
[2]    Dim i, j As Integer
[3]    Dim a2DDateArray As New Date[][]
[4]    Dim myArray As Date[]
[5]
[6]    a2DDateArray.Resize(10) ' *
[7]
[8]    For i = 0 To 9
[9]      a2DDateArray[i] = New Date[]
[10]     a2DDateArray[i].Resize(5) ' **
[11]     For j = 0 To 4
[12]         a2DDateArray[i][j] = Date(Now() + j - i) ' Unterschiedliche Datumswerte generieren
[13]     Next ' j
[14]   Next ' i
[15]
[16] ' Ausgabe IDE-Konsole (Beispiel Terminplan):
[17]   For Each myArray In a2DDateArray
[18]     For j = 0 To myArray.Max
[19]       Print Format(myArray[j], "dd. mm. yyyy"); " | ";
[20]       If (j + 1) Mod (myArray.Max + 1) = 0 Then Print
[21]     Next ' j
[22]   Next ' myArray
[23]
[24]   Print "Dimension von a2DDateArray: ", a2DDateArray.Dim
[25]   Print "Anzahl Elemente von 'a2DDateArray': ", a2DDateArray.Count
[26]   Print "Anzahl Elemente von 'a2DDateArray': ", a2DDateArray.Bounds[0] ' Alternative
[27]
[28] End ' btnShowExample_Click()
```

- Ein (leeres) Array für Datumswerte 'a2DDateArray' vom Typ Date[ ] wird in der Zeile 3 erzeugt.
- In der Zeile 6 wird Platz für 10 Elemente auch vom Daten-Typ Date[ ] dynamisch reserviert.
- In den beiden FOR-NEXT-Kontroll-Strukturen (Zeilen 8 bis 14) werden Date[ ]-Arrays erzeugt (Zeile 9), die Anzahl der Elemente im Date[ ]-Array mit der Resize(..)-Methode ** auf 5 festgelegt und jeweils mit 5 unterschiedlichen Datumswerten gefüllt.
- In den Zeilen 17 bis 22 wird jedes Element vom Array 'a2DDateArray' ausgelesen und danach werden die 5 Datumswerte – die in jedem Element gespeichert sind – ausgelesen und gruppiert angezeigt. Dafür sorgen die Zeilen 19 und 20.

Ausgabe in der IDE-Konsole:

```
29. 04. 2014 | 30. 04. 2014 | 01. 05. 2014 | 02. 05. 2014 | 03. 05. 2014 |
28. 04. 2014 | 29. 04. 2014 | 30. 04. 2014 | 01. 05. 2014 | 02. 05. 2014 |
27. 04. 2014 | 28. 04. 2014 | 29. 04. 2014 | 30. 04. 2014 | 01. 05. 2014 |
26. 04. 2014 | 27. 04. 2014 | 28. 04. 2014 | 29. 04. 2014 | 30. 04. 2014 |
25. 04. 2014 | 26. 04. 2014 | 27. 04. 2014 | 28. 04. 2014 | 29. 04. 2014 |
24. 04. 2014 | 25. 04. 2014 | 26. 04. 2014 | 27. 04. 2014 | 28. 04. 2014 |
23. 04. 2014 | 24. 04. 2014 | 25. 04. 2014 | 26. 04. 2014 | 27. 04. 2014 |
22. 04. 2014 | 23. 04. 2014 | 24. 04. 2014 | 25. 04. 2014 | 26. 04. 2014 |
21. 04. 2014 | 22. 04. 2014 | 23. 04. 2014 | 24. 04. 2014 | 25. 04. 2014 |
20. 04. 2014 | 21. 04. 2014 | 22. 04. 2014 | 23. 04. 2014 | 24. 04. 2014 |

Dimension von 'a2DDateArray':           1

Anzahl Elemente von 'a2DDateArray':     10
Anzahl Elemente von 'a2DDateArray':     10 ' Alternative
```

Die Struktur der gruppierten Anzeige entspricht auch der Speicher-Struktur im Array 'a2DDateArray', da Sie zum Beispiel auf das 4. Datum im 7. Element so zugreifen können:

```
Print Format(a2DDateArray[6][3], "dd. mm yyyy")
```

Beachten Sie, dass sowohl 'a2DDateArray' als auch alle Arrays 'a2DDateArray[i]' (→ Zeile 9) mit dem Index i aus dem Intervall von 0 bis 9 *eindimensionale* Arrays sind und Sie deshalb die Möglichkeit haben, die Dimension zur Laufzeit zu ändern! In den Zeilen * und ** wird das auch praktiziert, was Ihnen mit der folgenden (statischen) Deklaration nicht möglich wäre:

```
Dim a2DDateArray As New Date[10][5]
```

Auf das (Hilfs-)Array 'myArray' könnten Sie verzichten, wenn Sie den Quelltext für die *Ausgabe* so ändern:

```
  For i = 0 To a2DDateArray.Max
    For j = 0 To a2DDateArray[i].Max
        If (j + 1) Mod (a2DDateArray[i].Max + 1) = 0 Then
           Print Format(a2DDateArray[i][j], "dd. mm. yyyy"); " | "
        Else
           Print Format(a2DDateArray[i][j], "dd. mm. yyyy"); " | ";
        Endif
    Next ' j
  Next ' i
```

Dynamische Arrays von Strukturen werden zur Zeit nicht unterstützt (→ Kapitel 7.2.1.2 Arrays von Strukturen), weil 'Struct' als strukturierter Daten-Container gedacht ist. Wenn Sie dynamische Arrays brauchen, dann verwenden Sie eine *Klasse* statt einer *Struktur*, denn in Gambas können Sie von jeder Klasse einen dynamischen Array-Typ ableiten.

Struktur:

```
Public Struct Schueler
   JGS As Integer
   GebDatum As Date
   Nachname As String
   DG1Kurs As String
   DG2Kurs As String
End Struct
```

Quelltext CDS.class:

```
' Gambas class file

' Diese Klasse ist eine (reine) Datenstruktur ohne eigene Methoden.
' Sie repräsentiert zum Beispiel einen Datensatz

Public JGS As Integer
Public GebDatum As Date
Public Nachname As String
Public DG1Kurs As String
Public DG2Kurs As String
```

Im Hauptprogramm können Sie durch CDS[ ] einen dynamischen Array-Typ erzeugen, dessen Elemente Objekte vom Typ CDS sind → Kapitel 7.4.3.2 Abgeleitete Arrays. Sie können mit diesem abgeleiteten Array so arbeiten, wie Sie es auch von den Strukturen gewohnt sind – jedoch ohne die (statischen) Schranken einer Struktur.