Informationsmaterial C#

Arrays in C#

Arrays in C#

Arrays, auch Felder und häufig auch Vektoren genannt, sind zusammenhängende Datenobjekte eines Typs. Unter zusammenhängend ist hier zu verstehen, dass diese Datenobjekte sich direkt hintereinander im Speicher des Computers befinden. Die maximal mögliche Anzahl der Datenobjekte ist vom Datentyp und vom verfügbaren Speicherbereich abhängig.

In C# sind Arrays Objekte und nicht nur adressierbare, zusammenhängende Speicherbereiche wie in C und C++. ***Array*** ist der abstrakte Basistyp aller Arraytypen. Sie können die Eigenschaften sowie andere Klassenmember von **Array** verwenden. Ein Beispiel dafür wäre die Verwendung der Length-Eigenschaft, um die Länge eines Arrays abzurufen

Die Klasse Array

Eigenschaften und Methoden der Klasse Array (aus System)

`Array.BinarySearch(arr, obj)` `Array.BinarySearch(arr, o, ICompare)`	durchsucht das Feld `arr` nach dem Eintrag `obj`. Die Methode setzt voraus, dass das Feld sortiert ist, und liefert als Ergebnis die Indexnummer des gefundenen Eintrags. Optional kann eine eigene Vergleichsmethode angegeben werden.
`Array.Clear(arr, n, m)`	setzt `m` Elemente beginnend mit `arr[n]` auf den Standardwert des zugrunde liegenden Datentyps.
`arr2 = (Datentyp[])(arr1.Clone())`	weist `arr2` eine Kopie von `arr1` zu.
`Array.Copy(arr1, n1, arr2, n2, m)`	kopiert `m` Elemente vom Feld `arr1` in das Feld `arr2`, wobei `n1` der Startindex in `arr1` und `n2` der Startindex in `arr2` ist.
`arr = Array.CreateInstance ( Typ, n [,m [,o]] )`	erzeugt ein Feld der Größe `(n,m,o)`, wobei in den einzelnen Elementen Objekte des Typs `type` gespeichert werden können.
`arr.GetLength(dimension);`	ermittelt die Anzahl der Elemente der Dimension `dimension` des Arrays `arr`.
`arr.GetUpperBound(dimension);`	liefert die obere Grenze der Dimension `dimension` des Arrays `arr`.
`arr.GetLowerBound(dimension);`	liefert die untere Grenze der Dimension `dimension` des Arrays `arr`. Diese Funktion ist zwar selten nützlich, es kann jedoch vorkommen, dass ein Array nicht die Untergrenze 0 hat.
`Array.Reverse(arr)`	vertauscht die Reihenfolge der Elemente des Arrays `arr`.
`arr.SetValue(data, n [,m [,o]])`	speichert im Element `arr(n, m, o)` den Wert `data`.
`Array.Sort(arr [,ICompare] )`	sortiert `arr` (unter Anwendung der Vergleichsfunktion des `ICompare`-Objekts, falls angegeben).

Weitere Information zu Arrays:
http://www.microsoft-press.de/chapter.asp?cnt=getchapter&id=ms-5201.pdf
http://msdn.microsoft.com/de-de/library/aa288453%28v=vs.71%29.aspx
http://openbook.galileocomputing.de/csharp/kap16.htm

Eindimensionale Arrays

In C# können eindimensionale und mehrdimensionale Arrays von allen Datentypen angelegt werden.

Syntax:

Name des Arrays

Datentyp *[ ] Bezeichner =* *new* ***Datentyp****[Anzahl];*

Gibt den Datentyp des Array an

Dynamische Reservierung von Anzahl Objekten des Datentyps

Beispiel:

```
int [ ] feld = new int[5]  ;
```

Durch diese Definition wird ein zusammenhängender Speicherbereich für fünf Datenobjekte vom Typ ***int*** vom C-Compiler reserviert.

Gezählt wird von Null an, so dass die int-Datenelemente einen Index von ***0 bis 4*** erhalten.

Es werden insgesamt 40 Byte reserviert,
genau **Elementanzahl * Typgröße = 5 * 4 Byte = 40 Byte.**

Felder können auch während der Definition initialisiert werden:

```
int [ ] feld = new int[5]  { 1, 2, 3, 4, 5 } ;
```

Felder, die bei ihrer Definition beschrieben werden, nennt man **statische Felder**.

Zugriff auf Arrays

Als Hilfsmittel Vereinbarung einer INDEX-Variablen i:

```
unsigned char i = 0;
float uebername;
float [ ] feld = new float[5]  {8.0, 12.1, 6.23, 9.12, 7.201};
```

Durch **feld[i]** und **i++** kann man auf alle Feldelemente zugreifen:

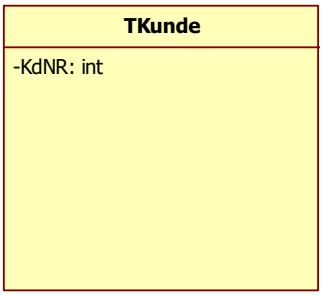
uebername = feld[i].

Inhalt in Abhängigkeit der Indexvariablen:

feld[0] : 8.0	*i = 0*	uebernahme	= 8.0
feld[1] : 12.1	*i = 1*	uebernahme	= 12.1
feld[2] : 6.23	*i = 2*	uebernahme	= 6.23
feld[3] : 9.12	*i = 3*	uebernahme	= 9.12
feld[4] : 7.201	*i = 4*	uebernahme	= 7.201

Objekte:

Es können genauso Array mit Objekte gebildet werden:




```
public class TKunde
{
    privat int KdNR;

    public string Nachname
    {
        get;
        set;
    }
   .....
}
```

TKunde [ ] Kunden;
Kunden = new TKunde[5];

oder

TKunde [ ] Kunden = new TKunde[5];

Außerdem muss nun jedes neue Objekt instanziert werden:

i ++;
Kunden[i] = new TKunde();

Allerdings ist dabei zu beachten, dass diese Objekte nicht einfach ausgegeben werden können. Der Zugriff auf Elemente des Objektes geschieht mit dem „.“-Operator:

textBox1.Text = Kunden[i].Nachname

Foreach-Schleife:

Die *foreach* Schleife iteriert über jedes Element eines Arrays oder eines Objektes. Als Beispiel nehmen wir das Integer Array *iarray* und die Integer Variable *i*. Die einzelnen Elemente von *iarray* werden nach und nach bei jedem Schleifendurchlauf an die Variable *i* übergeben.

```
int[] iarray = new int[] { 1, 2, 3, 4, 5 };
foreach (int i in iarray)
{
    Console.WriteLine(i);
}
```

Beispiel:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace arraybsp
{
  class Program
  {
    static void Main(string[] args)
    {
      int[] myArr;
      // Eingabe der Array-Größe
      Console.Write("Geben Sie die Anzahl der Elemente ein: ");
      int number = Convert.ToInt32(Console.ReadLine());
      // Initialisierung des Arrays
      myArr = new int[number];

      // jedes Element des Arrays in einer Schleife durchlaufen
      for(int i=0; i<number; i++)
      {
        // dem Array-Element einen Wert zuweisen
        myArr[i] = i * i;
        // das Array-Element an der Konsole ausgeben
        Console.WriteLine("myArr[{0}] = {1}",i, myArr[i]);
      }

      foreach (int i in myArr)
      {
         Console.WriteLine(i);
      }

    }
    Console.ReadLine();
  }
}
```

Mehrdimensionale Arrays

Syntax:

```
Datentyp [, ... ,] Bezeichner = new Datentyp[Dim1, .... , DimN];
```

Beispiele:

```
double [ , , ] Dimension3 = new double[3, 4, 5];// Dreidimensionales Array

string [ , ] Dimension2 = new string[ 10,10 ];// Zweidimensionales Array
```

Anzahl der maximalen Elemente in der angegebenen Dimension mit der Methode GetLength(Dimension) abfragen:

```
int maxLaenge;
maxLaenge = Dimension3.GetLength(2);

Ergebnis: maxLaenge = 5;
```

Kopieren von Arrays:

Arrays sind Verweistypen und müssen deshalb mit einer „tiefen Kopie“ kopiert werden. Das bedeutet, dass eine Methode der Klasse Array benutzt werden muss:

```
int [ ] iX = new int [3] {1,2,3};
int [ ] iY = new int[3];

System.Array.Copy(iX, iY, iX.Length);
foreach(int k in iY) Console.WriteLine(k);
```

Sortieren von Arrays:

Mit der Klasse Array können Arrays einfach sortiert werden.

```
int [ ] iX = new int [5] {7,5,2,3,1};

System.Array.Sort(iX);
foreach(int k in iY) Console.WriteLine(k);
```

Ausgabe:

1
2
3
4
5