Generische Container-Klasse 'List'

Inhalt

Allgemeines

Die Klasse dient zur flexiblen Speicherung von Objekten.

Während für Arrays schon bei der Definition festgelegt wird, wie viele Elemente sie enthalten können, ist die generische Klasse 'List' flexibler. Ein 'List'-Objekt kann eine variable Anzahl von Objekten beinhalten.

Bei der Definition eines 'List'-Objekts wird in spitzen Klammern der Grundtyp der 'List' angegeben, d.h. von welchem Typ die zu speichernden Elemente sind. Für diesen Grundtyp sind sowohl Klassen als auch Basisdatentypen erlaubt.

Anwendungsbereich

Will man eine Serie von Daten mit dem gleichen Typ speichern, dann bieten sich dafür Arrays oder auch eine Reihe von spezialisierten Container-Klassen an. Arrays werden von allen gängigen Programmiersprachen unterstützt. Die Container-Klassen dienen ebenfalls dazu, Elemente aufzunehmen und bieten zusätzlich unterschiedliche Service-Funktionen an.

Generische Programmierung

In objektorientierten Programmiersprachen wie C# können Anwendungsprogrammierer mit Hilfe von Klassen eigene Datentypen erzeugen. Beim Schreiben von Systembibliotheken muss darauf reagiert werden, indem Klassen und Methoden bereitgestellt werden, die mit diesen Datentypen umgehen können. Dafür gibt es zwei unterschiedliche Vorgehensweisen, und zwar aus der objektorientierten Programmierung und mit Hilfe von generischen Template-Klassen.

Objektorientierter Ansatz

In der objektorientierten Programmierung gibt es die Möglichkeit, ein Objekt einer Unterklasse mit einer Referenz einer Oberklasse zu verwalten. Die Oberklasse zu allen anderen Klassen ist die Klasse 'Object'. Mit einer Referenz der Klasse 'Object' kann man also auf Objekte beliebiger Klassen verweisen, diese Objekte speichern und wieder zur Verfügung stellen.

Mit Hilfe dieser Vorgehensweise werden Objekte in WPF-Listboxen und Comboboxen gespeichert und zu Verfügung gestellt.

Ansatz mit generischen Klassen

Generische Klassen werden häufig als Container verwendet, um Objekte eines vorgegebenen Typs zu speichern. Dieser sogenannte Parameter-Typ wird dabei in spitzen Klammern hinter dem Grundtyp angegeben, z.B. `List<Person>` für eine Tabelle von Person-Objekten. Darin lassen sich dann Person-Objekte sowie Objekte von abgeleiteten Klassen der Klasse 'Person' speichern.

Beispielprogramm

```
using System;
using System.Collections.Generic;

namespace ListDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            //
            // Zu einer Listen-Referenz muss ein Listen-Objekt erzeugt werden.
            //
            List<string> arbeitstage = new List<string>();
            //
            // Mit der Methode 'Add' können einzelne Einträge hinzugefügt werden.
            //
            arbeitstage.Add("Montag");
            arbeitstage.Add("Dienstag");
            arbeitstage.Add("Mittwoch");
            arbeitstage.Add("Donnerstag");
            arbeitstage.Add("Freitag");
            //
            // 'Count' erlaubt das Abrufen der Anzahl der Elemente.
            // Der Index-Operator liefert ein einzelnes Element.
            //
            Console.WriteLine("Arbeitstage");
            for(int i = 0; i < arbeitstage.Count; i++)
                Console.WriteLine("\t" + arbeitstage[i]);
            //
            List<string> wochenende = new List<string>();
            wochenende.Add("Samstag");
            wochenende.Add("Sonntag");
            //
            // Beide Listen zusammenfügen.
            //
            List<string> wochentage = new List<string>(arbeitstage);
            wochentage.AddRange(wochenende);
            //
            // Die foreach-Schleife kann auch angewandt werden.
            //
            Console.WriteLine("Alle Wochentage");
            foreach(string tag in wochentage)
                Console.WriteLine("\t" + tag);
        }
    }
}
```

Ausgabe des Programms

```
Arbeitstage
        Montag
        Dienstag
        Mittwoch
        Donnerstag
        Freitag
Alle Wochentage
        Montag
        Dienstag
        Mittwoch
        Donnerstag
        Freitag
        Samstag
        Sonntag
```

Typische Operationen mit Arrays und Listen

Operation	Array	List
Anlegen eines Objekts	`string[] namen = new string[10];` Ein Array mit 10 Einträgen wird erstellt.	`List<string> namen = new List<string>();` Eine leere Liste wird erstellt.
Anzahl der Elemente abrufen	`int anzahl = namen.Length;`	`int anzahl = namen.Count;`
Ein Element anhängen.	in Standard-C# nicht möglich	`namen.Add("Mikey");`
Ein Element abrufen.	`string benutzer = namen[5];`	`string benutzer = namen[5];` wenn namen.Count >= 6
Ein Element setzen.	`namen[5] = "Donald";`	`namen[5] = "Donald";` wenn namen.Count >= 6
for-each-Schleife	`foreach(string name in namen)` `    Console.WriteLine(name);`	`foreach(string name in namen)` `    Console.WriteLine(name);`

Wichtige Methoden der generischen Klasse 'List'

`List<string>()`

erzeugt ein leeres List-Objekt für String-Objekte.

`List<string>(string[] array)`

erzeugt ein List-Objekt mit den Daten eines vorgegebenen Arrays.

`void Add(string s)`

fügt ein weiteres String-Objekt an das Ende der List an.

`void Clear()`

löscht alle Elemente in der List.

`bool Contains(string s)`

prüft, ob der übergebene String in der List enthalten ist.

`string liste[int index]`

liest oder schreibt das Element bei dem übergebenen Index. Der Index beginnt mit dem Wert 0.

`int IndexOf(string s)`

liefert den Index, wo der übergebene String in der List als erstes auftritt, oder -1, wenn der String nicht enthalten ist.

`bool Remove(string s)`

löscht den übergebenen String.

`void RemoveAt(int index)`

löscht den String an der angegebenen Stelle.

`void Sort()`

sortiert die in der List gespeicherten Objekte in ihrer natürlichen Reihenfolge.