**Wichtigste Collections – List, Dictionary und foreach**

**Collections**

* In Programmen werden oft mehrere Daten gespeichert und verarbeitet. Collections helfen dabei, diese Daten effizient zu verwalten.
* Eine Collection ist eine Datenstruktur, die **mehrere Werte speichern kann, ohne dass man für jede Variable einen separaten Namen vergeben muss**.

**Vergleich: Arrays vs. Collections**

|  |  |  |
| --- | --- | --- |
| Eigenschaft | Arrays | Collections (List, Dictionary) |
| Größe | Feste Größe bei Deklaration | Dynamisch erweiterbar |
| Speicherort | Stack oder Heap | Heap |
| Methoden | Keine oder wenige Methoden | Viele nützliche Methoden |
| Zugriff | Über Index | Über Methoden oder Schlüssel |

Collections bieten also mehr Flexibilität als Arrays, was sie für viele Anwendungen nützlicher macht.

**Was ist List<T>?**

* Eine List<T> ist eine **generische** Liste, die dynamisch wächst und schrumpft.
* Sie ist im Namespace System.Collections.Generic enthalten.
* Sie speichert Elemente des Typs T und bietet viele hilfreiche Methoden zur Verwaltung der Elemente.

**Wichtige Methoden von List<T>**

|  |  |
| --- | --- |
| Methode | Beschreibung |
| Add(T item) | Fügt ein Element am Ende der Liste hinzu. |
| Remove(T item) | Entfernt das erste Vorkommen eines Elements. |
| RemoveAt(int index) | Entfernt das Element an einer bestimmten Position. |
| Contains(T item) | Prüft, ob ein Element in der Liste enthalten ist. |
| IndexOf(T item) | Gibt den Index des ersten Vorkommens eines Elements zurück. |
| Count | Gibt die Anzahl der Elemente zurück. |
| Sort() | Sortiert die Elemente in der Liste. |
| Reverse() | Kehrt die Reihenfolge der Elemente um. |

**Beispiel**

// List von Datentyp String instanzieren und Elemente hinzufügen

List<string> namen = new List<string>();

namen.Add("Anna");

namen.Add("Ben");

namen.Add("Chris");

Console.WriteLine("Anzahl der Namen: " + namen.Count);

// Überprüfung nach bestimmtem Element

if (namen.Contains("Ben"))

{

Console.WriteLine("Ben ist in der Liste enthalten.");

}

// Entfernen von betseimmtem Element

namen.Remove("Chris");

Console.WriteLine("Nach dem Entfernen von Chris: " + namen.Count);

// Entfernen von Element an einer bestimmten Stelle

namen.RemoveAt(0);

Console.WriteLine("0te Stelle wurde gelöscht. Jetzt hat die Liste " + namen.Count + " Elemente.");

// Hinzufügen von Element am Listende

namen.AddRange(namen);

Console.WriteLine("Nach Verdoppelung haben wir: " + namen.Count + " Elemente.");

for (int i = 0; i <= 100; i++)

{

namen.Add(Guid.NewGuid().ToString());

}

Console.WriteLine("100 neue Elemente hinzugefügt. Jetzt haben wir: " + namen.Count + " Elemente.");

// Ausfsteigendes Sortieren der Liste

namen.Sort();

Console.WriteLine("Nach Sortierung");

for (int i = 0; i <= 100; i++)

{

Console.WriteLine(namen[i]);

}

// Umkehr der aktuellen Sortierung

namen.Reverse();

Console.WriteLine("Nach Umkehr der Sortierung");

for (int i = 0; i <= 100; i++)

{

Console.WriteLine(namen[i]);

}

// Löschen von allen Elementen

namen.Clear();

Console.WriteLine("Nach Clear(): Anzahl Elemente: " + namen.Count);

**Dictionary<TKey, TValue> – Schlüssel-Wert-Paare**

* Ein Dictionary<TKey, TValue> speichert Daten in Form von **Schlüssel-Wert-Paaren**.
* Es ist besonders nützlich, wenn man Elemente anhand eines eindeutigen Schlüssels effizient abrufen möchte.
* Es gehört ebenfalls zum Namespace System.Collections.Generic.

**Wichtige Methoden von Dictionary<TKey, TValue>**

|  |  |
| --- | --- |
| Methode | Beschreibung |
| Add(TKey key, TValue value) | Fügt ein Schlüssel-Wert-Paar hinzu. |
| Remove(TKey key) | Entfernt ein Element anhand seines Schlüssels. |
| ContainsKey(TKey key) | Prüft, ob ein Schlüssel existiert. |
| TryGetValue(TKey key, out TValue value) | Versucht, einen Wert sicher zu lesen. |
| Count | Gibt die Anzahl der Einträge zurück. |

**Beispiel**

// Instanzieren und befüllen mit Schlüssel-Werte-Paaren

Dictionary<int, string> studenten = new Dictionary<int, string>();

studenten.Add(101, "Anna");

studenten.Add(102, "Ben");

Console.WriteLine("Student mit ID 101: " + studenten[101]);

// Ob ein Elemenet mit gegebenem Schlüssel existiert

if (studenten.ContainsKey(103))

Console.WriteLine("Student mit ID 103 existiert.");

else

Console.WriteLine("Student mit ID 103 existiert nicht.");

// Entfernen eines Elements nach Key

studenten.Remove(101);

**foreach – Die einfachste Art, Collections zu durchlaufen**

**Was ist foreach?**

* Die foreach-Schleife ermöglicht das einfache Durchlaufen einer Collection ohne Verwendung eines Indexes.
* Sie ist besonders nützlich für List<T> und Dictionary<TKey, TValue>.

**Anwendungsbeispiele**

**Durchlaufen einer List<T> mit foreach**

foreach (var name in namen)

{

Console.WriteLine(name);

}

**Durchlaufen eines Dictionary<TKey, TValue> mit foreach**

foreach (KeyValuePair<int, string> eintrag in studenten)

{

Console.WriteLine("ID: " + eintrag.Key + " , Name: " + eintrag.Value);

}

**Übungen**

1. **List-Übung:**
   * Erstelle eine List<int> mit den Zahlen 10, 20, 30, 40, 50.
   * Berechne die Summe aller Zahlen und gib sie aus.
2. **Dictionary<TKey, TValue>-Übung:**
   * Implementiere ein Dictionary<string, int> für Namen und Alter.
   * Die Liste soll 50 Personen beinhalten
   * Name soll eine auf 5 Stellen zugeschnittene GUID sein.
   * Alter ist zufällig aus dem Bereich 1-50
   * Gib alle Namen mit einem Alter über 18 aus.
3. **foreach-Übung:**
   * Speichere eine Liste von Städten (List<string>) und gib sie mit foreach aus.