06 – Collections

Bildungszentrum Uster
Höhere Berufsbildung
Uster
HBU

© D. Pfulg & M. Sabbatella

Agenda

- Collections
- Generics

Bildungszentrum Uster
Höhere Berufsbildung
Uster
HBU

© D. Pfulg & M. Sabbatella

Collections

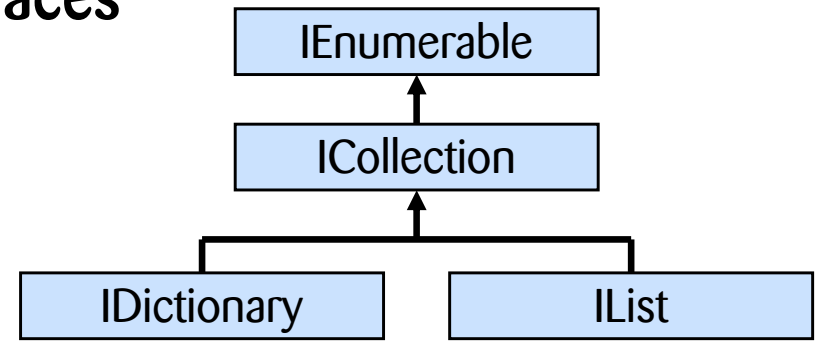
Grundlagen

- Collections, auch Auflistungen genannt, werden zur Verwaltung von Daten eingesetzt
- Der Zugriff erfolgt meistens über einen Index oder Schlüssel (Key)
- Beispiele sind das Verwalten der Inhalte von Listboxen oder Comboboxen
- Vorteile der Collections gegenüber Arrays:
  - Variable Grösse
  - Komfortable Methoden wie `Add()`, `Remove()`, `RemoveAt()` usw.
- Namensraum: `System.Collections`

Bildungszentrum Uster
Höhere Berufsbildung Uster
HBU

© D. Pfulg & M. Sabbatella

Collection-Interfaces

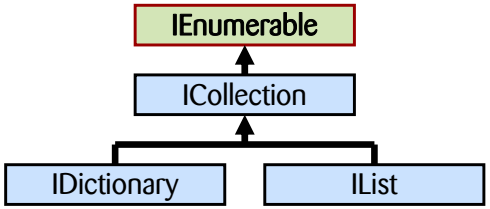


- Klassen, die das Interface `IList` implementieren, beschreiben Objektauflistungen. Über einen Index kann auf die Einträge zugegriffen werden.
- Klassen, die das Interface `IDictionary` implementieren, verwalten ihre Einträge über eine Schlüssel-Wert-Kombination (Key/Value-Pair) in einem Verzeichnis.

© D. Pfulg & M. Sabbatella

Collections

Interface - `IEnumerable`



- Eigenschaften:
  - Keine
- Methoden:
  - `GetEnumerator()` gibt ein Enumerator-Objekt zurück, welches die Fähigkeit hat, eine Collection elementweise zu durchlaufen. Der Enumerator positioniert sich vor dem ersten Element der Collection, besitzt die Eigenschaft `Current` und die Methoden `Reset()` und `MoveNext()`.
- Anwendung:
  - Mit der `foreach`-Schleife können alle Elemente durchlaufen werden.

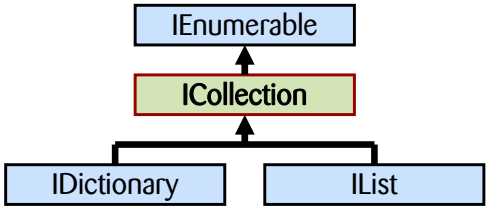
```
foreach(Typ Variable in Auflistung) {
  // Anweisungen
}
```

Bildungszentrum Uster
Höhere Berufsbildung Uster
HBU

© D. Pfulg & M. Sabbatella

Collections

Interface – `ICollection`



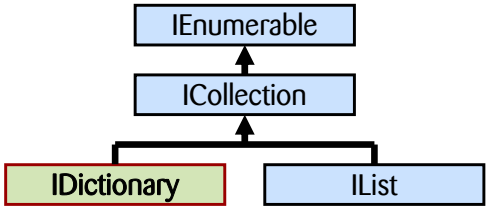
- Das Interface `ICollection` erweitert das Interface `IEnumerable` mit folgenden Members:

Eigenschaften	Beschreibung
Count	Liefert die Anzahl Elemente in der Collection.
IsSynchronized	Liefert `true`, wenn die Collection synchronisiert, also thread-sicher ist.
SyncRoot	Liefert eine Objektreferenz, die den Zugriff auf das Objekt synchronisiert.

Methoden	Beschreibung
CopyTo	Kopiert die Elemente in ein Array beginnend bei einem Index.

© D. Pfulg & M. Sabbatella

Interface – `IDictionary` (1/2)

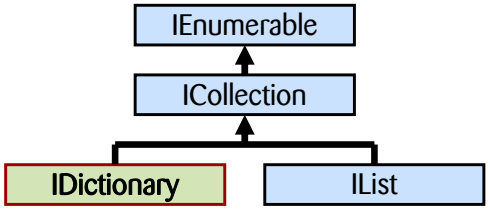


- Das Interface `IDictionary` erweitert das Interface `ICollection` mit folgenden Members:

Eigenschaften	Beschreibung
IsFixedSize	Gibt `true` zurück, falls die Grösse der Collection sich nicht dynamisch vergrössert.
IsReadOnly	Gibt `true` zurück, falls die Auflistung schreibgeschützt ist.
Keys	Liefert alle in der Liste verwendeten Schlüssel zurück.
Values	Liefert alle in der Liste verwendeten Werte zurück.

© D. Pfulg & M. Sabbatella

Interface – `IDictionary` (2/2)



- Das Interface `IDictionary` erweitert das Interface `ICollection` mit folgenden Members:

Methoden	Beschreibung
Add	Addiert ein Element zur Auflistung mit dem angegebenen Schlüssel.
Clear	Löscht alle Elemente aus der Auflistung.
Contains	Gibt `true` zurück, falls ein bestimmtes Objekt in der Collection bereits vorhanden ist.
Remove	Löscht ein Objekt mit dem angegebenen Schlüssel.

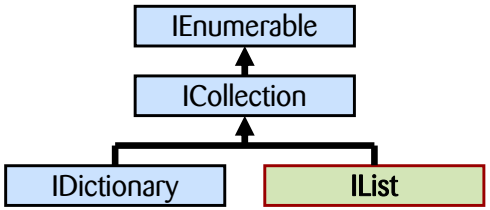
© D. Pfulg & M. Sabbatella

Interface – `IList`

- Das Interface `IList` erweitert das Interface `ICollection` mit:

Eigenschaften	Beschreibung
IsFixedSize	Gibt `true` zurück, falls die Grösse der Collection sich nicht dynamisch vergrössert.
IsReadOnly	Gibt `true` zurück, falls die Auflistung schreibgeschützt ist.
Item	Ruft das Element am angegebenen Index ab oder legt dieses fest.

Methoden	Beschreibung
Add	Fügt ein Element hinzu.
Clear	Löscht alle Elemente.
Contains	Gibt `true` zurück, falls ein bestimmtes Objekt in der Collection bereits vorhanden ist.
IndexOf	Liefert den Index eines bestimmten Objekts.
Insert	Fügt ein Objekt an einer bestimmten Position ein.
Remove	Löscht ein Objekt mit der angegebenen Referenz.
RemoveAt	Löscht ein Objekt an einer bestimmten Position.



© D. Pfulg & M. Sabbatella

Collections

Übersicht der Klassen aus `System.Collections`

Collection-Klasse	Beschreibung
ArrayList	Universelle Liste zum speichern aller Objektarten.
BitArray	Speichert ein Array aus Bit-Werten als boolsche Werte.
Hashtable	Speichert Schlüssel/Werte-Paare (Key/Value-Pairs). Der Zugriff erfolgt mittels Schlüssel. Zugriff mittels Index nicht möglich.
Queue	Repräsentiert eine Warteschlange, resp. FIFO (First In / First Out).
SortedList	Kombination aus Array und Hashtable. Die Liste ist immer sortiert und der Zugriff ist via Schlüssel oder Index möglich.
Stack	Repräsentiert einen Stapelspeicher, resp. LIFO (Last In / First Out).

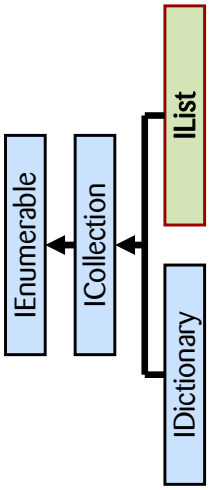
Bildungszentrum Uster
Höhere Berufsbildung Uster
HBU

© D. Pfulg & M. Sabbatella

Collections

Übersicht der Collection Klassen und den implementierten Interfaces

Collection-Klasse	IEnumerable	ICollection	IList	IDictionary
Array	x	x	x	
ArrayList	x	x	x	
BitArray	x	x		
Hashtable	x	x		x
Queue	x	x		
SortedList	x	x		x
Stack	x	x		



Bildungszentrum Uster
Höhere Berufsbildung
Uster
HBU

© D. Pfulg & M. Sabbatella

ArrayList (Feldliste)

- Merkmale:
  - Die Grösse des Arrays ist dynamisch und passt sich automatisch an
- Zugriff via:
  - Indexer
- Wichtigste Methoden:
  - Elemente hinzufügen: `Add(), AddRange(), Insert()`
  - Elemente löschen: `Remove(), RemoveAt(), Clear()`
- Wichtigste Eigenschaften:
  - `Count`

© D. Pfulg & M. Sabbatella

ArrayList (Feldliste)

- Beispiel: Ein Element hinzufügen

```
// Ein Element (z.B. string) hinzufügen:
string stadt = "Zürich";
ArrayList arrList = new ArrayList();
int index = arrList.Add(stadt);
```

- Beispiel: Einen Bereich hinzufügen

```
// Ein Array aus Primzahlen hinzufügen:
ArrayList arrList = new ArrayList();
int[] primZahlenArray = {2, 3, 5, 7, 11};
arrList.AddRange(primZahlenArray);
arrList.RemoveAt(3);
```


© D. Pfulg & M. Sabbatella

BitArray (Bit-Feldliste)

- Merkmale:
  - Speichert Bit-Werte als bool
- Zugriff via:
  - Indexer
  - Methoden (siehe unten)
- Wichtigste Methoden:
  - Bit-Werte setzen: `Set()`
  - Werte lesen: `Get()`
  - Verknüpfen: `And(), Or(), Not()`
- Wichtigste Eigenschaften:
  - `Count`

© D. Pfulg & M. Sabbatella

Hashtable (Verzeichnis)

- Merkmale:
  - Speichert Schlüssel/Werte-Paare (Key/Value-Pairs).
- Zugriff via:
  - Key oder in Schleife via `IEnumerator`-Interface
- Wichtigste Methoden:
  - Elemente hinzufügen: `Add(), Insert()`
  - Elemente löschen: `Remove(), Clear()`
  - Elemente abfragen: `Contains(), ContainsKey(), ContainsValue()`
  - Kopieren: `Clone(), CopyTo()`
- Wichtigste Eigenschaften:
  - `Count, Keys, Values`

© D. Pfulg & M. Sabbatella

Queue (Warteschlange)

- Merkmale:
  - Speichert Objekte in einer Wartschlange (sequentiell, FIFO).
- Zugriff via:
  - Methoden
- Wichtigste Methoden:
  - Elemente hinzufügen: `Enqueue()`
  - Elemente entnehmen: `Dequeue()`
  - Löschen: `Clear()`
  - Elemente abfragen: `Contains(), Peek()`
  - Kopieren: `Clone(), CopyTo(), ToArray()`
- Wichtigste Eigenschaften:
  - `Count`

© D. Pfulg & M. Sabbatella

SortedList (Sortierte Liste)

- Merkmale:
  - Speichert Objekte in einer sortierten Liste.
- Zugriff via:
  - Index
- Wichtigste Methoden:
  - Elemente hinzufügen: `Add()`
  - Elemente entfernen: `Remove(), RemoveAt(), Clear()`
  - Elemente abfragen: `Contains(), ContainsKey(), ContainsValue(), GetKey(), GetKeyList(), GetValueList()`
  - Kopieren: `Clone(), CopyTo(), ToArray()`
- Wichtigste Eigenschaften:
  - `Capacity, Count, Keys, Values`

© D. Pfulg & M. Sabbatella

Stack (Stapelspeicher)

- Merkmale:
  - Speichert Objekte auf einem Stapel (LIFO).
- Zugriff via:
  - Methoden
- Wichtigste Methoden:
  - Elemente hinzufügen: `Push()`
  - Elemente entnehmen: `Pop()`
  - Löschen: `Clear()`
  - Elemente abfragen: `Contains(), Peek()`
  - Kopieren: `Clone(), CopyTo(), ToArray()`
- Wichtigste Eigenschaften:
  - `Count`

© D. Pfulg & M. Sabbatella

Übung 6.1

Mitarbeiterverzeichnis (Hashtable)

- 1) Schreibe eine Klasse `Mitarbeiter`, welche folgende Daten speichert:
  - Personalnummer, Name, Vorname, Telefonnummer
- 2) Schreibe eine Klasse `Mitarbeiterverzeichnis`, welche Mitarbeiter in einer Hashtable verwalten soll. Überlege welche Methoden die Klasse braucht, um Telefon und Mitarbeiter zu suchen, sowie mit welchem Schlüssel die Mitarbeiter im Mitarbeiterverzeichnis eingefügt werden sollen.
- 3) Schreibe eine Klasse `Testtreiber`, um Mitarbeiter zu erfassen und dem Mitarbeiterverzeichnis hinzuzufügen und abzufragen.

Bildungszentrum Uster
Höhere Berufsbildung Uster
HBU

© D. Pfulg & M. Sabbatella

Generics

Grundlagen

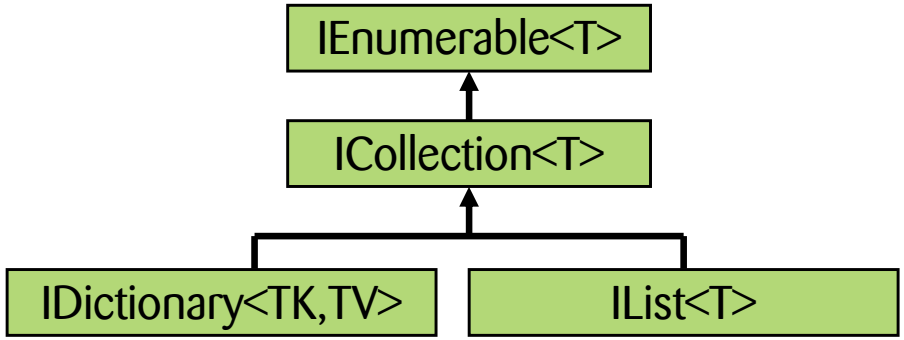
- Generics funktionieren auf Basis generischer Datentypen
- Generics sind Platzhalter für Datentypen (analog einer Variable)
- Haupteinsatzgebiet für Generics sind Collections
- Sie sind leistungsfähiger als normale Collection-Klassen und reduzieren den zu erstellenden Code
- Häufig ist es der Fall, dass die Funktionalität und die Daten einer Klasse nur durch den Datentyp voneinander unterscheiden.
- In .NET 1.1 musste pro Datentyp eine entsprechende Klasse implementiert werden. Mit Generics muss die Klasse nur einmal implementiert werden, und für den Datentyp wird ein Platzhalter verwendet.

© D. Pfulg & M. Sabbatella

Generische Collections

Collection-Interfaces

- Die generischen Collection-Klassen implementieren die generischen Interfaces. Diese sind analog den üblichen Collection-Interfaces.



© D. Pfulg & M. Sabbatella

Generische Collections

Generische Collection-Klassen und impl. Interfaces

Collection Klasse	IEnumerable<T>	ICollection<T>	IList<T>	IDictionary <TKey, TValue>
Dictionary<TKey, TValue>	x	x		X
List<T>	x	x	x	
Queue<T>	x	(ICollection)		
SortedDictionary<TKey, TValue>	x	x		X
Stack<T>	x	(ICollection)		
SortedList<TKey, TValue>	x	x		x
LinkedList<T>	x	x		

© D. Pfulg & M. Sabbatella

Deklarieren

- Syntax:
  - Der Platzhalter wird hinter dem Klassennamen in spitzer Klammer angegeben.

```
public class GenClassName<T> {...}
```

  - Es können auch mehrere generische Datentypen angegeben werden.

```
public class GenClassName<T, U> {...}
```


© D. Pfulg & M. Sabbatella

Beispiel: Generischer Stack

- Implementierung

```
class GenericStack<T> {
  private readonly int size;
  private T[] elements;
  private int pointer = 0;   // Zeiger auf nächstes Element
  public int Length {get};

  public GenericStack(int size) {
    this.size = size;
    elements = new T[size];
  }
  public void Push(T element) { ... }
  public T Pop() { ... }
}
```


© D. Pfulg & M. Sabbatella

Beispiel: Generischer Stack

- Anwendung

```
Static void Main() {
  GenericStack<int> stack = new GenericStack<int>(5);
  stack.Push(32);
  stack.Push(983);
  stack.Push(44);
  for (int inx = stack.Length; inx >= 1; inx--) {
    Console.WriteLine(stack.Pop());
  }
}
```


© D. Pfulg & M. Sabbatella

Generische Collection-Klassen

- Generische Collection-Klassen ermöglichen auf elegante Weise die Verwendung von typsicheren Auflistungen
- Namespace: `System.Collections.Generic`
- Anwendungsbeispiele:
  - Eine typsichere `SortedList`, in welcher nur Objekte der Klasse Mitarbeiter verwaltet werden können.
  - Eine typsichere `Hashtable` (Verzeichnis), in welcher nur Haustiere verwaltet werden können.

© D. Pfulg & M. Sabbatella

Beispiel mit generischer Collection-Klasse:

- Generische `SortedList` für String/String-Paare

```
// Neue SortedList mit string Werten und string Schlüssel
SortedList<string, string> openWith =
  new SortedList<string, string>();

// Einige Element hinzufügen
openWith.Add("txt", "notepad.exe");
openWith.Add("bmp", "paint.exe");
openWith.Add("dib", "paint.exe");
openWith.Add("rtf", "wordpad.exe");

Console.WriteLine("For key = \"txt\", value = {0}.",
              openWith["txt"]);
// Ausgabe: For key = "txt„, value = notepad.exe
```


© D. Pfulg & M. Sabbatella

Generische Methoden

- Generics können auch auf Parametern für Methoden angewandt werden
- Die Klasse, in welcher die Methode deklariert wird, muss nicht generisch sein.
- Beispiel einer generischen Methode:

```
public static void Swap<T>(ref T firstValue, ref T secondValue)
{
  T tempValue = firstValue;
  firstValue = secondValue;
  secondValue = tempValue;
}
```


© D. Pfulg & M. Sabbatella

Generische Methoden

- Beispiel:
  - Aufruf einer generischen Methode mit Angabe des Datentyps

```
int a = 50;
int b = 100;
Swap<int>(ref a, ref b);

double c = 100.6;
double d = 50.3;
Swap<double>(ref c, ref d);
```

  - Der Aufruf funktioniert auch ohne Angabe des Datentyps, wenn sich dieser aus dem Parametern ergibt.

```
int a = 50;
int b = 100;
Swap(ref a, ref b);
```


© D. Pfulg & M. Sabbatella

Übung 6.2

Mitarbeiterverzeichnis (Generic Dictionary<T, G>)

- 1) Modifiziere die Klasse `Mitarbeiterverzeichnis` aus Übung 6.1 so, dass die generische Hashtable `Dictionary<T, G>` aus dem Namensraum `System.Collections.Generic` verwendet wird.
- 2) Implementiere eine Klasse `Kunde`, welche folgende Daten speichert:
  - Kundennummer, Name, Vorname, Firma, Telefonnummer
- 3) Erweitere die Klasse `Testtreiber` und teste, ob ein Objekt der Klasse `Kunde` in das `Mitarbeiterverzeichnis` eingefügt werden kann.

Bildungszentrum Uster
Höhere Berufsbildung Uster
HBU

© D. Pfulg & M. Sabbatella