Collections
Aufzählungen bzw. Listen

Ingo Köster

Diplom-Informatiker (FH)

Arrays vs. Collections

› Arrays sind statische Pools freier und belegter Elementpositionen

› Arrays besitzen keine Möglichkeit, sich dynamisch an Änderungen anzupassen

› Collections sind Klassen, die ähnlich den Arrays als Container meist typgleicher Elemente dienen

Arrays vs. Collections

- Im Unterschied zu Arrays arbeiten diese Klassen dynamisch
  - Enthalten keine leeren Indizes
  - Vergrößern oder verkleinern die Kapazität entsprechend der Anzahl der Einträge

- Jede Klasse unterscheidet sich von der anderen durch besondere Fähigkeiten und Charakteristika
  - Interne Verwaltung der Objekte
  - Zugriff auf die Einträge
  - Geschwindigkeit, mit der nach einem bestimmten Eintrag gesucht werden kann

Non-Generic & Generic

- Nichtgenerischen Listen des Namespace `System.Collections`
- Generischen Listen des Namespace `System.Collections.Generic`
- Falls möglich, sollten die generischen Collections wegen der Typsicherheit verwendet werden

IEnumerable & IEnumerator

- `IEnumerable` ermöglicht es, eine Liste in einer `foreach`-Schleife zu durchlaufen

- Weist nur die Methode `GetEnumerator()` auf, die ein Objekt zurückliefert, welches die Schnittstelle `IEnumerator` implementiert

- Das Enumerator-Objekt verfügt über die Fähigkeit, eine Auflistung elementweise zu durchlaufen
  - Enumerator-Objekt ist ein Positionszeiger mit 3 Methoden
  - `Current`, `MoveNext` und `Reset`

IEnumerable & IEnumerator

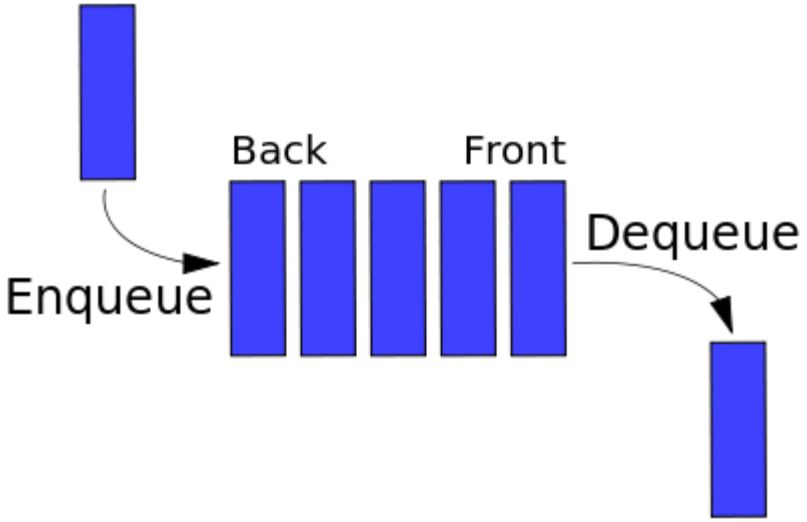
- Der Enumerator wird vor den ersten Eintrag einer Auflistung positioniert
- Um ihn auf den ersten Eintrag und anschließend auf alle Folgeeinträge zeigen zu lassen, wird die Methode `MoveNext` ausgeführt
- Mit `Current` wird auf den Eintrag zugegriffen, auf den der Enumerator zeigt
- `Reset` setzt den Enumerator an seine Ausgangsposition zurück
- Alle Collections implementieren die Schnittstelle `IEnumerable`

ArrayList - Methoden

Einfügen in die Liste	
`Add`	Einzelnen Wert einfügen
`AddRange`	Ein Array einfügen
`Insert`	Add an bestimmtem Index
`InsertRange`	AddRange an bestimmtem Index
Entfernen aus der Liste	
`Remove`	Entfernt das übergebene Element
`RemoveAt`	Entfernt am Index n (n ist int)
`RemoveRange`	Entfernt einen Bereich (index, anzahl)
Weitere Methoden	
`IndexOf, Sort , Clear`	…

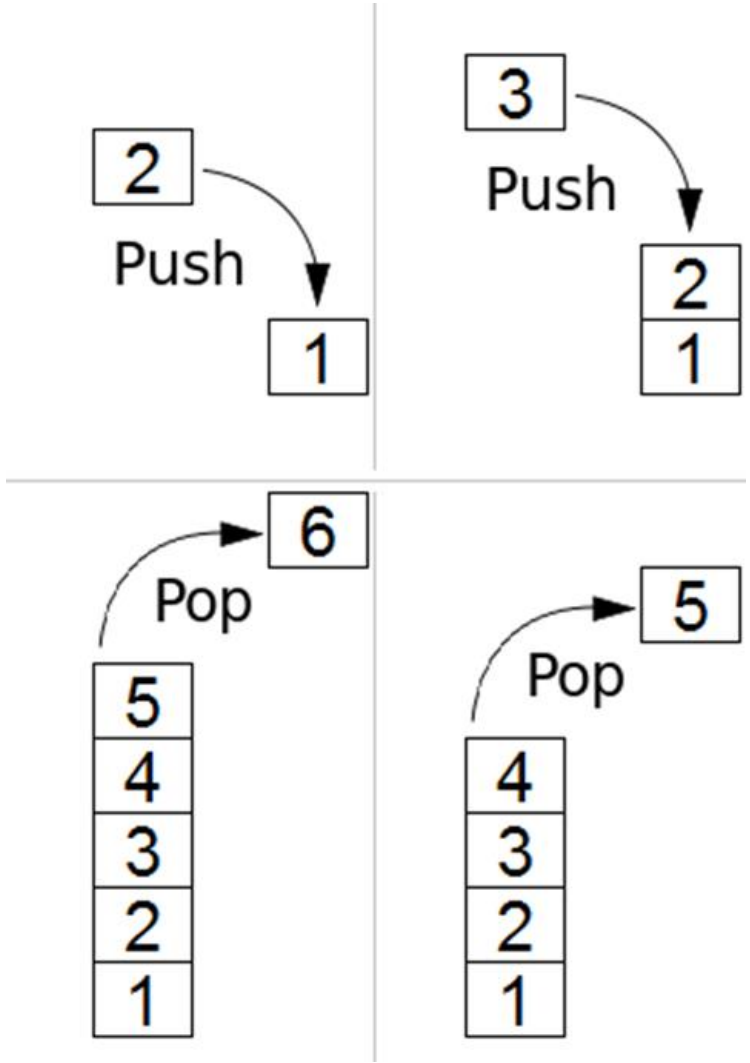
Klasse Queue

- Ist ein FiFo (First In, First Out)
- Enqueue()
  - Objekt in die Queue schreiben
- Dequeue()
  - Objekt aus der Queue lesen
  - Objekt wird dann aus der Queue gelöscht
- Peek()
  - Ein Objekt nur lesen, nicht löschen



Klasse Stack

- Ist ein LiFo (Last In, First Out)
- `Push()`
  - Objekt in den Stack schreiben
- `Pop()`
  - Objekt von dem Stack lesen
  - Objekt wird dann von dem Stack gelöscht
- `Peek()`
  - Ein Objekt nur lesen, nicht löschen

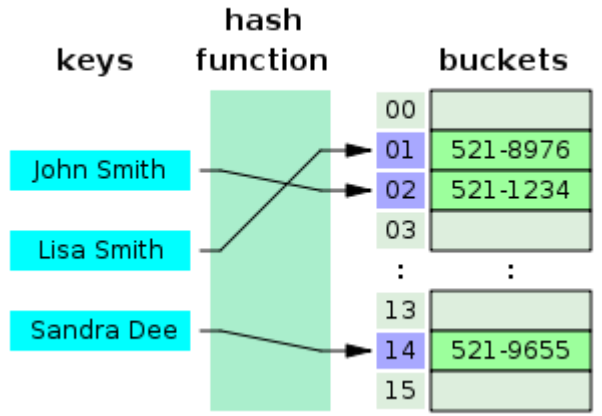


Hashtable

› Liste von Elementen, die nicht durch einen Index verwaltet werden, sondern durch ein Schlüssel-Werte-Paar

› Für die Verwaltung der Elemente wird ein Hashcode als Schlüssel verwendet

› Der Hashcode ist ein Wert, der aus den Daten eines Objekts gebildet wird bzw. werden kann



Hashtable

- Zugriff auf die Elemente in einer Hashtable erfolgt über Schlüsselwert
  - Kann grundsätzlich eine beliebige Information sein
  - Meist wird dazu eine Zeichenfolge benutzt
- Vorteil einer Hashtable
  - Innerhalb der Liste kann sehr schnell nach bestimmten Objekten gesucht werden
- Schlüsselwerte dürfen nicht doppelt vorkommen
  - Führen zu Ausnahmen

Hashtable - Einfügen

- `void Add(object key, object value);`
- Der erste Parameter wird als Schlüssel für das hinzuzufügende Element verwendet
- Der zweite Parameter ist die Referenz auf das hinzuzufügende Element

Hashtable

```
Hashtable hash = new Hashtable();

Artikel artikel1 = new Artikel(11, "Wurst", 1.98);
Artikel artikel2 = new Artikel(45, "Käse", 2.98);

hash.Add(artikel1.Bezeichner, artikel1);
hash.Add(artikel2.Bezeichner, artikel2);

foreach (DictionaryEntry item in hash) {
  Console.WriteLine("{0} – {1} ", item.Key, item.Value);
}
```

Generische Collections

Collection

- `ArrayList`
- `Stack`
- `Queue`
- `Hashtable`

Generische Collection

- `List<T>`
- `Stack<T>`
- `Queue<T>`
- `Dictionary<TKey, Tvalue>`

Klasse HashSet

- Klasse für Mengenoperationen
- Eine Auflistung ohne doppelte Elemente
- Die Elemente liegen in keiner bestimmten Reihenfolge vor
- HashSets können mit anderen HashSets vereinigt werden

```
HashSet<int> numbers = new HashSet<int>();
numbers.Add(2);
numbers.Add(2);   // doppelt, wird ignoriert
numbers.Add(3);
```

Spezielle Wörterbücher

- `SortedList<TKey, Tvalue>`
  - In Liste wird sortiert eingefügt
- `ListDictionary`
  - Sehr kleine Hashtable (< 10 Elemente)
- `HybridDictionary`
  - Ist bei wenig Elementen ein ListDictionary, bei Vergrößerung eine Hashtable
- `OrderedDictionary`
  - Ähnelt Hashtable; erlaubt Zugriff über Indices wie bei ArrayList