LINQ-Operatoren

LINQ enthält ca. 50 Operatoren zum Filtern, Projizieren, Sortieren, Konvertieren, Aggregation und Element-Auswahl.
Die wichtigsten davon sind:

Kategorie	Operator	Beschreibung
Filtern	`Where`	Liefert eine Untermenge von Elementen zurück, die eine gegebene Bedingung erfüllen
	`Distinct`	Liefert eine Collection ohne Duplikate zurück
	`OfType<T>`	Alle Elemente mit dem angegebenen Typ incl. automatischem Cast
	`Take` `TakeWhile`	Liefert die ersten x Elemente zurück Liefert Elemente zurück, solange die Bedingung erfüllt ist
	`Skip` `SkipWhile`	Ignoriert die ersten x Elemente und liefert die restlichen zurück Ignoriert Elemente, solange die Bedingung erfüllt ist
Projizieren	`Select`	Transformiert jedes Element mithilfe des angegebenen Lamda-Ausdrucks
Sortieren	`OrderBy` `ThenBy`	Sortiert eine Sequenz aufsteigend Weitere Sortierungen mit ThenBy
	`OrderByDescending` `ThenByDescending`	Sortiert eine Sequenz absteigend Weitere Sortierungen mit ThenByDescending
	`Reverse`	Liefert eine Sequenz in umgekehrter Reihenfolge zurück
Konvertieren	`ToArray`	IEnumerable<T> → T[]
	`ToList`	IEnumerable<T> → List[]
	`ToDictionary`	IEnumerable<T> → Dictionary<TKey,TValue>
Aggregation	`Average`	Durchschnitt
	`Count`	Liefert die Anzahl der Elemente der Sequenz zurück, die optional eine Bedingung erfüllen
	`Sum`	Summe
	`Max`	größtes Element
	`Min`	kleinstes Element
Element-Auswahl	`First`, `FirstOrDefault`	Liefert das erste Element der Sequenz, das optional eine Bedingung erfüllt. Default: geben den Default-Wert des Element-Typs zurück, falls kein Element die Bedingung erfüllt.
	`Last`, `LastOrDefault`	Liefert das letzte Element der Sequenz, das optional eine Bedingung erfüllt.
	`Single`, `SingleOrDefault`	Wie First, wirft aber Exception, wenn es mehr als ein passendes Element gibt.
	`ElementAt`, `ElementAtOrDefault`	Liefert das Element an der angegebenen Position zurück

GroupBy

GroupBy teilt die Collection in Teil-Collections anhand des angegenben Keys.
Die entry-Objekte in dieser Collection sind dann vom Typ **`System.Linq.Lookup.Grouping`**, die außerdem die Property **`Key`** (für den Wert, nach dem man aufgeteilt hat) hat.
Man kann daher über das Ergebnis wieder iterieren

```
string[] names = { "Franz", "Udo", "Hans", "Susi", "Tom", "CSI", "Hugo", "Anton", "Fritzi", "Heinz" };
foreach (var entry in names.GroupBy(x => x[0]))
{
  Console.Write(entry.Key + " --> ");
  foreach (var s in entry) Console.Write(s + " ");
}
```

ToDictionary

Die Collection, die man durch **`GroupBy`** erhält, kann man leicht in ein Dictionary umwandeln:

- Die Keys dieser Collection werden zum Key des Dictionarys
- Die Teil-Collections selbst kann man wieder mit einem Lambda-Ausdruck auf ein beliebiges Objekt umwandeln

```
var index = names.GroupBy(x => x[0])
                 .ToDictionary(
                                x => x.Key,
                                x => x.ToList()
                              );
```