Chapter 9
Introduction to LINQ and the `List` Collection

Visual C# 2012 How to Program

OBJECTIVES

In this chapter you'll:

- Learn basic LINQ concepts.
- Query an array using LINQ.
- Learn basic .NET collections concepts.
- Create and use a generic `List` collection.
- Query a generic `List` collection using LINQ.

9.1 Introduction

- Arrays hebben beperkte mogelijkheden, maar zijn wel veel gebruikt.
- Een ***`List`*** is gelijkaardig aan een array, maar voorziet extra functionaliteiten. Bijvoorbeeld: **dynamic resizing**.
- Een taal, SQL genaamd, is de standaard om **queries** uit te voeren (~ vragen stellen volgens bepaalde criteria) en data te manipuleren.
- C#'s **LINQ** (**Language-Integrated Query**) mogelijkheden laten u toe om queries te schrijven voor verschillende data sources, niet alleen databases.
- **LINQ to Objects** kan gebruikt worden om array en lists te filteren, zodat je enkel de gevraagde info terugkrijgt.

9.1 Introduction (Cont.)

- Figuur 9.1 geeft een overzicht van waar LINQ in het boek gebruikt wordt.
- Een **LINQ provider** is een verzameling van klasses die LINQ implementeren zodat programma's kunnen interageren met data sources.

Chapter	Used to
Chapter 9, Introduction to LINQ and the `List` Collection	Query arrays and `Lists`.
Chapter 16, Strings and Characters: A Deeper Look	Select GUI controls in a Windows Forms app (located in the online section of the chapter).
Chapter 17, Files and Streams	Search a directory and manipulate text files.
Chapter 22, Databases and LINQ	Retrieve information from a database.
Chapter 23, Web App Development with ASP.NET	Retrieve information from a database to be used in a web-based app.
Chapter 24, XML and LINQ to XML	Query an XML document.
Chapter 30, Web Services	Query and update a database. Process XML returned by WCF services.

Fig. 9.1 | LINQ usage throughout the book.

9.2 Querying an Array of int Values Using LINQ

- Figuur 9.2 demonstreert ‘querying an array of integers using LINQ’.


```
1  // Fig. 9.2: LINQWithSimpleTypeArray.cs
2  // LINQ to Objects using an int array.
3  using System;
4  using System.Linq;
5
6  class LINQWithSimpleTypeArray
7  {
8     public static void Main( string[] args )
9     {
10       // create an integer array
11       int[] values = { 2, 9, 5, 0, 3, 7, 1, 4, 8, 5 };
12
13       // display original values
14       Console.Write( "Original array:" );
15       foreach ( var element in values )
16          Console.Write( " {0}", element );
17
18       // LINQ query that obtains values greater than 4 from the array
19       var filtered =
20          from value in values
21          where value > 4
22          select value;
23
```

Fig. 9.2 | LINQ to Objects using an `int` array. (Part 1 of 4.)

```
24         // display filtered results
25         Console.Write( "\nArray values greater than 4:" );
26         foreach ( var element in filtered )
27            Console.Write( " {0}", element );
28
29         // use orderby clause to original values in ascending order
30         var sorted =
31            from value in values
32            orderby value
33            select value;
34
35         // display sorted results
36         Console.Write( "\nOriginal array, sorted:" );
37         foreach ( var element in sorted )
38            Console.Write( " {0}", element );
39
40         // sort the filtered results into descending order
41         var sortFilteredResults =
42            from value in filtered
43            orderby value descending
44            select value;
45
```

Fig. 9.2 | LINQ to Objects using an `int` array. (Part 2 of 4.)


```
46         // display the sorted results
47         Console.Write(
48            "\nValues greater than 4, descending order (separately):" );
49         foreach ( var element in sortFilteredResults )
50            Console.Write( " {0}", element );
51
52         // filter original array and sort results in descending order
53         var sortAndFilter =
54            from value in values
55            where value > 4
56            orderby value descending
57            select value;
58
59         // display the filtered and sorted results
60         Console.Write(
61            "\nValues greater than 4, descending order (one query):" );
62         foreach ( var element in sortAndFilter )
63            Console.Write( " {0}", element );
64
65         Console.WriteLine();
66      } // end Main
67   } // end class LINQWithSimpleTypeArray
```

Fig. 9.2 | LINQ to Objects using an `int` array. (Part 3 of 4.)

```
Original array: 2 9 5 0 3 7 1 4 8 5
Array values greater than 4: 9 5 7 8 5
Original array, sorted: 0 1 2 3 4 5 5 7 8 9
Values greater than 4, descending order (separately): 9 8 7 5 5
Values greater than 4, descending order (one query): 9 8 7 5 5
```

Fig. 9.2 | LINQ to Objects using an `int` array. (Part 4 of 4.)

9.2 Querying an Array Using LINQ

- Herhalingsopdrachten die arrays moeten filteren om het gevraagde resultaat te bekomen heet: '**imperative programming**'.
- LINQ queries specifieëren de voorwaardes waaraan het resultaat moet voldoen. Dit heet: '**declarative programming**'.
- The ***`System.Linq`*** namespace bevat de LINQ to Objects provider.

9.2 Querying an Array Using LINQ

```
var filtered =
    from value in values
    where value > 4
    select value;
```

- Je kan lokale variabelen maken, waarbij je de compiler laat uitzoeken welk type de variabele moet hebben. Hiervoor gebruik je het ***var*** keyword.
- Een LINQ query begint met ***from***. Hierin wordt een ***range*** variable (value) en de ***data source*** (values) gespecifieerd.
- De ***range*** variabele vertegenwoordigt elk element in de data source. Te vergelijken met een lusvariabele in een ***foreach*** – lus.
- Als de conditie in de ***where*** waar is, wordt het element geselecteerd.

9.2 Querying an Array Using LINQ

- Door de ***select*** optie, wordt bepaald welke waarde(s) in het resultaat geplaatst wordt.
- Gebruik ***order by*** om het resultaat te sorteren.
- Gebruik het keyword ***descending*** voor het aflopend sorteren.

9.2 Querying an Array Using LINQ

- Een LINQ query geeft een object terug dat ***IEnumerable<T>*** implementeert.
- ***IEnumerable<T>*** beschrijft de functionaliteit van objecten waardoor je kan itereren.
- Arrays and collections implementeren ***IEnumerable<T>.***
- Met LINQ is er een opsplitsing tussen de code die de elementen selecteert en de code die de elementen toont (display't). Dat maakt de code gemakkelijker te begrijpen en te onderhouden.

9.3 Querying an Array of Employee Objects Using LINQ

- LINQ is niet beperkt tot het bevragen/manipuleren van arrays van simple types.
- Om een LINQ query uit te voeren moet de compiler kunnen vergelijken. Daarom moeten de objecten waarop je een LINQ query wil uitvoeren, de ***IComparable<T>*** implementeren.
- Alle ingebouwde types zoals ***string***, ***int*** and ***double*** implementeren ***IComparable<T>.***
- Figuur 9.3 toont de Employee class. Figure 9.4 gebruikt 'LINQ to query an array of Employee objects'.

```
1  // Fig. 9.3: Employee.cs
2  // Employee class with FirstName, LastName and MonthlySalary properties.
3  public class Employee
4  {
5     private decimal monthlySalaryValue; // monthly salary of employee
6
7     // auto-implemented property FirstName
8     public string FirstName { get; set; }
9
10    // auto-implemented property LastName
11    public string LastName { get; set; }
12
13    // constructor initializes first name, last name and monthly salary
14    public Employee( string first, string last, decimal salary )
15    {
16       FirstName = first;
17       LastName = last;
18       MonthlySalary = salary;
19    } // end constructor
20
```

Fig. 9.3 | Employee class. (Part 1 of 2.)


```
21    // property that gets and sets the employee's monthly salary
22    public decimal MonthlySalary
23    {
24       get
25       {
26          return monthlySalaryValue;
27       } // end get
28       set
29       {
30          if ( value >= 0M ) // if salary is nonnegative
31          {
32             monthlySalaryValue = value;
33          } // end if
34       } // end set
35    } // end property MonthlySalary
36
37    // return a string containing the employee's information
38    public override string ToString()
39    {
40       return string.Format( "{0,-10} {1,-10} {2,10:C}",
41          FirstName, LastName, MonthlySalary );
42    } // end method ToString
43 } // end class Employee
```

Fig. 9.3 | Employee class. (Part 2 of 2.)

```
1  // Fig. 9.4: LINQWithArrayOfObjects.cs
2  // LINQ to Objects using an array of Employee objects.
3  using System;
4  using System.Linq;
5
6  public class LINQWithArrayOfObjects
7  {
8     public static void Main( string[] args )
9     {
10       // initialize array of employees
11       Employee[] employees = {
12          new Employee( "Jason", "Red", 5000M ),
13          new Employee( "Ashley", "Green", 7600M ),
14          new Employee( "Matthew", "Indigo", 3587.5M ),
15          new Employee( "James", "Indigo", 4700.77M ),
16          new Employee( "Luke", "Indigo", 6200M ),
17          new Employee( "Jason", "Blue", 3200M ),
18          new Employee( "Wendy", "Brown", 4236.4M ) }; // end init list
19
20       // display all employees
21       Console.WriteLine( "Original array:" );
22       foreach ( var element in employees )
23          Console.WriteLine( element );
24
```

'M' zorgt ervoor dat het getal als decimal verwerkt wordt. Dit is een kommagetal van 128 bit met grotere precisie, maar kleinere grenzen…

Fig. 9.4 | LINQ to Objects using an array of `Employee` objects. (Part 1 of 6.)


```
25       // filter a range of salaries using && in a LINQ query
26       var between4K6K =
27          from e in employees
28          where e.MonthlySalary >= 4000M && e.MonthlySalary <= 6000M
29          select e;
30
31       // display employees making between 4000 and 6000 per month
32       Console.WriteLine( string.Format(
33          "\nEmployees earning in the range {0:C}-{1:C} per month:",
34          4000, 6000 ) );
35       foreach ( var element in between4K6K )
36          Console.WriteLine( element );
37
38       // order the employees by last name, then first name with LINQ
39       var nameSorted =
40          from e in employees
41          orderby e.LastName, e.FirstName
42          select e;
43
44       // header
45       Console.WriteLine( "\nFirst employee when sorted by name:" );
46
```

Fig. 9.4 | LINQ to Objects using an array of `Employee` objects. (Part 2 of 6.)

```
47         // attempt to display the first result of the above LINQ query
48         if ( nameSorted.Any() )
49            Console.WriteLine( nameSorted.First() );
50         else
51            Console.WriteLine( "not found" );
52
53         // use LINQ to select employee last names
54         var lastNames =
55            from e in employees
56            select e.LastName;
57
58         // use method Distinct to select unique last names
59         Console.WriteLine( "\nUnique employee last names:" );
60         foreach ( var element in lastNames.Distinct() )
61            Console.WriteLine( element );
62
63         // use LINQ to select first and last names
64         var names =
65            from e in employees
66            select new { e.FirstName, Last = e.LastName };
67
```

Fig. 9.4 | LINQ to Objects using an array of `Employee` objects. (Part 3 of 6.)


```
68         // display full names
69         Console.WriteLine( "\nNames only:" );
70         foreach ( var element in names )
71            Console.WriteLine( element );
72
73         Console.WriteLine();
74      } // end Main
75   } // end class LINQWithArrayOfObjects
```

Fig. 9.4 | LINQ to Objects using an array of `Employee` objects. (Part 4 of 6.)

```
Original array:
Jason      Red         $5,000.00
Ashley     Green       $7,600.00
Matthew    Indigo      $3,587.50
James      Indigo      $4,700.77
Luke       Indigo      $6,200.00
Jason      Blue        $3,200.00
Wendy      Brown       $4,236.40

Employees earning in the range $4,000.00-$6,000.00 per month:
Jason      Red         $5,000.00
James      Indigo      $4,700.77
Wendy      Brown       $4,236.40

First employee when sorted by name:
Jason      Blue        $3,200.00

Unique employee last names:
Red
Green
Indigo
Blue
Brown
```

Fig. 9.4 | LINQ to Objects using an array of `Employee` objects. (Part 5 of 6.)


```
Names only:
{ FirstName = Jason, Last = Red }
{ FirstName = Ashley, Last = Green }
{ FirstName = Matthew, Last = Indigo }
{ FirstName = James, Last = Indigo }
{ FirstName = Luke, Last = Indigo }
{ FirstName = Jason, Last = Blue }
{ FirstName = Wendy, Last = Brown }
```

Fig. 9.4 | LINQ to Objects using an array of `Employee` objects. (Part 6 of 6.)

9.3 Querying an Array of Employee Objects Using LINQ

- De ***where*** optie kan ook de properties van de ***range*** variabele bekijken.
- Voorwaarden combineren kan ook: ***&&.***
- De ***orderby*** optie kan ook sorteren op meedere cirteria, volgens een 'comma-separated list'.

9.3 Querying an Array of Employee Objects Using LINQ

- Methods die op een LINQ result kunnen toegepast worden:
  - ***Any***()
  - ***First***()
  - ***Count***()
  - ***Distinct***()

Extention methods op de IEnumerable Interface!
Zie object browser!

- Creating new types in the ***select*** clause of a LINQ query.
  - Uitbreiding…

9.4 Introduction to Collections

- .NET heeft verschillende klasses die ‘collections’ ondersteunen. In zo’n collection zit data die nauw met elkaar verbonden is.
- Een collection kan je efficiënt organiseren, opslaan en bevragen zonder dat je moet weten hoe de data opgeslaan wordt.
- List<T> is één van die klasses. Je kan ‘on the fly’ zijn grootte wijzigen.
- Het is een generic class, je kan er elk type object in plaatsen.
- Zie figuur 9.5 voor veel gebruikte methods.

Method or property	Description
`Add`	Adds an element to the end of the `List`.
`Capacity`	Property that gets or sets the number of elements a `List` can store without resizing.
`Clear`	Removes all the elements from the `List`.
`Contains`	Returns `true` if the `List` contains the specified element and `false` otherwise.
`Count`	Property that returns the number of elements stored in the `List`.
`IndexOf`	Returns the index of the first occurrence of the specified value in the `List`.
`Insert`	Inserts an element at the specified index.
`Remove`	Removes the first occurrence of the specified value.
`RemoveAt`	Removes the element at the specified index.
`RemoveRange`	Removes a specified number of elements starting at a specified index.
`Sort`	Sorts the `List`.
`TrimExcess`	Sets the `Capacity` of the `List` to the number of elements the `List` currently contains (`Count`).

Fig. 9.5 | Some methods and properties of class `List<T>`.

```
1  // Fig. 9.6: ListCollection.cs
2  // Generic List<T> collection demonstration.
3  using System;
4  using System.Collections.Generic;
5
6  public class ListCollection
7  {
8     public static void Main( string[] args )
9     {
10       // create a new List of strings
11       List< string > items = new List< string >();
12
13       items.Add( "red" ); // append an item to the List
14       items.Insert( 0, "yellow" ); // insert the value at index 0
15
16       // display the colors in the list
17       Console.Write(
18          "Display list contents with counter-controlled loop:" );
19       for ( int i = 0; i < items.Count; i++ )
20          Console.Write( " {0}", items[ i ] );
21
```

Fig. 9.6 | Generic `List<T>` collection demonstration. (Part 1 of 4.)


```
22       // display colors using foreach
23       Console.Write(
24          "\nDisplay list contents with foreach statement:" );
25       foreach ( var item in items )
26          Console.Write( " {0}", item );
27
28       items.Add( "green" ); // add "green" to the end of the List
29       items.Add( "yellow" ); // add "yellow" to the end of the List
30
31       // display the List
32       Console.Write( "\nList with two new elements:" );
33       foreach ( var item in items )
34          Console.Write( " {0}", item );
35
36       items.Remove( "yellow" ); // remove the first "yellow"
37
38       // display the List
39       Console.Write( "\nRemove first instance of yellow:" );
40       foreach ( var item in items )
41          Console.Write( " {0}", item );
42
43       items.RemoveAt( 1 ); // remove item at index 1
44
```

Fig. 9.6 | Generic `List<T>` collection demonstration. (Part 2 of 4.)

```
45         // display the List
46         Console.Write( "\nRemove second list element (green):" );
47         foreach ( var item in items )
48            Console.Write( " {0}", item );
49
50         // check if a value is in the List
51         Console.WriteLine( "\n\"red\" is {0}in the list",
52            items.Contains( "red" ) ? string.Empty : "not " );
53
54         // display number of elements in the List
55         Console.WriteLine( "Count: {0}", items.Count );
56
57         // display the capacity of the List
58         Console.WriteLine( "Capacity: {0}", items.Capacity );
59      } // end Main
60   } // end class ListCollection
```

Fig. 9.6 | Generic `List<T>` collection demonstration. (Part 3 of 4.)


```
Display list contents with counter-controlled loop: yellow red
Display list contents with foreach statement: yellow red
List with two new elements: yellow red green yellow
Remove first instance of yellow: red green yellow
Remove second list element (green): red yellow
"red" is in the list
Count: 2
Capacity: 4
```

Fig. 9.6 | Generic `List<T>` collection demonstration. (Part 4 of 4.)

9.4 Introduction to Collections

- De property ***`Capacity`*** geeft aan hoeveel items in de ***`List`*** kunnen, zonder dat de lijst moet groeien.
- Groeien is 'lastig'. Er moet een groter array aangemaakt worden en alle elementen moeten gekopieerd worden.
- Een ***`List`*** groeit alleen als er geen plaats meer is voor extra items.

9.5 Querying a Generic Collection Using LINQ

- Je kan LINQ to Objects gebruiken om ***`List`*** te 'bevragen' (to query Lists).
- In Fig. 9.7, een ***`List`*** of strings wordt geconverteerd naar hoofdletters en onderzocht op woorden die starten met "R".

```
1  // Fig. 9.7: LINQWithListCollection.cs
2  // LINQ to Objects using a List< string >.
3  using System;
4  using System.Linq;
5  using System.Collections.Generic;
6
7  public class LINQWithListCollection
8  {
9     public static void Main( string[] args )
10    {
11       // populate a List of strings
12       List< string > items = new List< string >();
13       items.Add( "aQua" ); // add "aQua" to the end of the List
14       items.Add( "RusT" ); // add "RusT" to the end of the List
15       items.Add( "yElLow" ); // add "yElLow" to the end of the List
16       items.Add( "rEd" ); // add "rEd" to the end of the List
17
18       // convert all strings to uppercase; select those starting with "R"
19       var startsWithR =
20          from item in items
21          let uppercaseString = item.ToUpper()
22          where uppercaseString.StartsWith( "R" )
23          orderby uppercaseString
24          select uppercaseString;
```

Fig. 9.7 | LINQ to Objects using a `List<string>`. (Part 1 of 2.)


```
26       // display query results
27       foreach ( var item in startsWithR )
28          Console.Write( "{0} ", item );
29
30       Console.WriteLine(); // output end of line
31
32       items.Add( "rUbY" ); // add "rUbY" to the end of the List
33       items.Add( "SaFfRon" ); // add "SaFfRon" to the end of the List
34
35       // display updated query results
36       foreach ( var item in startsWithR )
37          Console.Write( "{0} ", item );
38
39       Console.WriteLine(); // output end of line
40    } // end Main
41 } // end class LINQWithListCollection
```

```
RED RUST
RED RUBY RUST
```

Fig. 9.7 | LINQ to Objects using a `List<string>`. (Part 2 of 2.)

9.5 Querying a Generic Collection Using LINQ

- Er is een ***let*** optie, die lokaal in een LINQ query een nieuwe ‘range variabele’ kan maken.
- Die lokale variabele kan in de query gebruikt worden.
- Let op! LINQ gebruikt ‘uitgestelde uitvoering’ (deffered execution). De query wordt slechts uitgevoerd wanneer je de resultaten gebruikt, niet wanneer je de query definieert!
- Gebruik ***ToArray*** en ***ToList*** om onmiddellijk de query uit te voeren. Dat kan efficiënter zijn dan telkens opnieuw de query uitvoeren.

Extention Methods

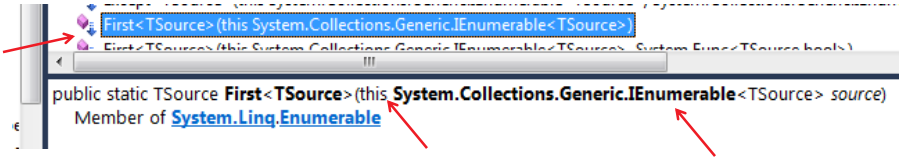
Intermezzo

Intermezzo: Extention methods

- Op een resultaat van LINQ kan je de methods ***Any()***, ***First()***, ***Last()*** en ***Distinct()*** toepassen… Dit zijn extention methods op de ***IEnumerable*** interface.
- Dus alle objecten die ***IEnumerable*** implementeren, krijgen er ook die methods bij.
- Ze zijn ***static public*** beschikbaar.
- Je moet er wel de ***namespace System.Linq*** voor toevoegen.

Intermezzo: Extention methods

- Bekijk in de object browser de ***System.Linq*** namespace…



- Let op het '***this***' keyword in de argumentlijst! Daarmee specificeer je dat je een extention method maakt.

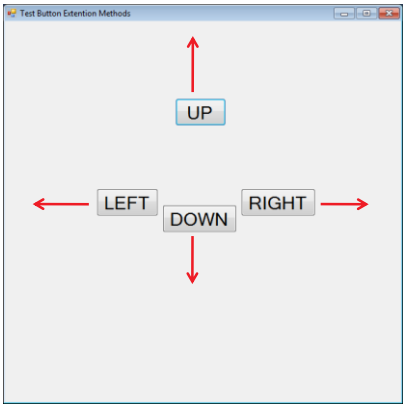
Intermezzo: Extention methods

- Extention methods zijn dus een uitbreiding op een bestaande klasse/type/interface, zonder de oorspronkelijke aan te passen!

- De methods kan je gebruiken net alsof ze al lid waren van de oorspronkelijke klasse.

Intermezzo: Extention methods

- Voorbeeld (demo):



Form1.cs

```
using Rubu.ButtonExtentions;

namespace ButtonExtentionMethods
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void btn_Click(object sender, EventArgs e)
        {
            switch (((Button)sender).Tag.ToString())
            {
                case "up": ((Button)sender).MoveUp(10); break;
                case "down": ((Button)sender).MoveDown(10);break;
                case "left": ((Button)sender).MoveLeft(10);break;
                case "right": ((Button)sender).MoveRight(10); break;
                default: break;
            }
        }
    }
}
```

Namespace met extention methods

Dit zijn nieuwe methods die we graag willen uitvoeren op een Button object…

ButtonExtentions.cs

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace Rubu.ButtonExtentions
{
    static class ButtonExtentions
    {
        public static void MoveRight(this Button btn, int distance)
        {
            btn.Location = new Point(btn.Location.X + distance, btn.Location.Y);
        }

        public static void MoveLeft(this Button btn, int distance)
        {
            btn.Location = new Point(btn.Location.X - distance, btn.Location.Y);
        }

        public static void MoveUp(this Button btn, int distance)
        {
            btn.Location = new Point(btn.Location.X, btn.Location.Y - distance);
        }

        public static void MoveDown(this Button btn, int distance)
        {
            btn.Location = new Point(btn.Location.X, btn.Location.Y + distance);
        }
    }
}
```

Intermezzo: Extention methods