

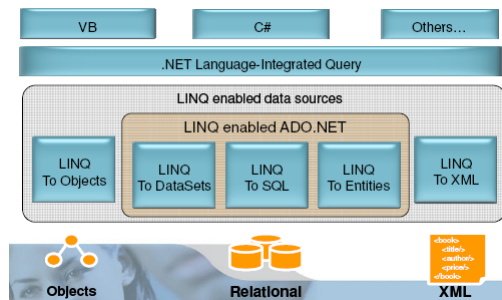
## Hoofdstuk 3 : LINQ

## Hoofdstuk 3 : LINQ

- ▶ Inleiding
- ▶ LINQ to Objects
  - Stap 1 : Een eerste LINQ voorbeeld
  - Stap 2 : LINQ en collections
  - Stap 3 : LINQ en anonymous types
  - Stap 4 : LINQ en extension methods
  - Stap 5 : Extension methods en lambda expressions
- ▶ Oefening

# Inleiding

- ▶ LINQ : Language Integrated Query
  - Querytaal geïntegreerd in de C# taal : dus ook Intellisense en compile time checking van de queries
  - 1 querytaal ongeacht de datasource : Objecten, SQL, XML,... -> Ontwikkelaar dient geen SQL, XPath, ... meer te kennen



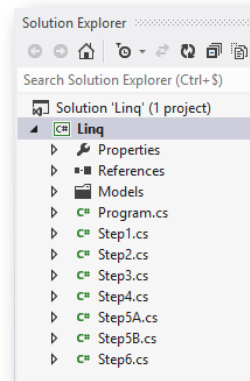
# Inleiding

- ▶ De core LINQ Assemblies

Assembly	Meaning in Life
System.Core.dll	Defines the types that represent the core LINQ API. This is the one assembly you must have access to.
System.Data.Linq.dll	Provides functionality for using LINQ with relational databases (LINQ to SQL).
System.Data.DataSetExtensions.dll	Defines a handful of types to integrate ADO.NET types into the LINQ programming paradigm (LINQ to DataSet).
System.Xml.Linq.dll	Provides functionality for using LINQ with XML document data (LINQ to XML).

## Inleiding

- ▶ Download Linq Starter applicatie, pak uit.
- ▶ Dubbelklik op de .sln file om het project in Visual Studio te openen.
- ▶ Het project is een Console applicatie, geen MVC applicatie . Het bevat
  - Program.cs : bevat de Main methode. Deze toont een menu in de Console. Je kan een stap kiezen
  - StepX.cs : 1 klasse per stap in de slides. Bevatten allen de methode Execute.
  - Models folder
    - Bevat de klassen nodig voor de applicatie



## LINQ to Objects

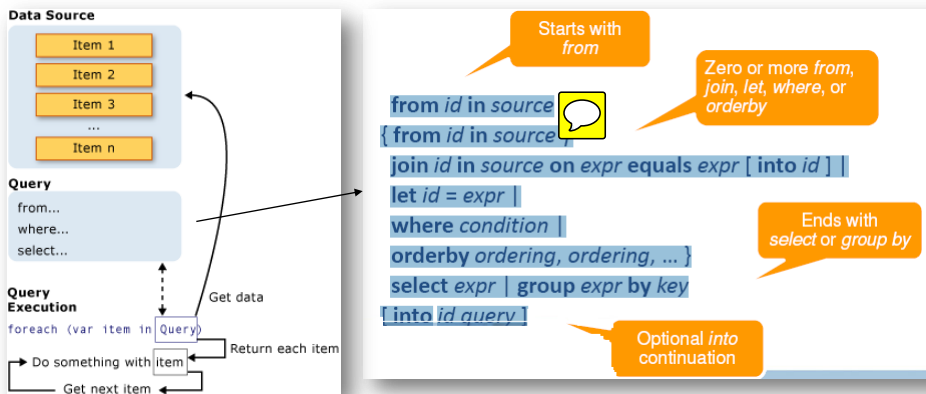
- ▶ Stap 1 : Een eerste LINQ voorbeeld
  - LINQ to Objects : raadplegen van collecties in het geheugen
  - Alle collecties die IEnumerable<T>- interface implementeren
    - Arrays, generieke collecties (List<T>), LinkedList<T>, ...)

```
namespace System.Collections {  
    public interface IEnumerable {  
        // Summary : Returns an enumerator that iterates through a collection.  
        // Returns: A System.Collections.IEnumerator object that can be used to iterate through  
        //           the collection.  
        IEnumerator GetEnumerator();  
    }  
}
```

- LINQ maakt gebruik van extension methods (zie verder)

# LINQ to Objects

## C# LINQ Syntax



# LINQ to Objects

## Voorbeeld Step1.cs

```
using System.Linq;
using System.Collections.Generic;

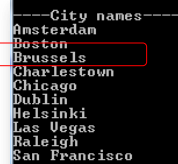
public void Execute()
{
    string[] cities = {"London", "Amsterdam", "San Francisco", "Las Vegas", "Boston",
        "Raleigh", "Chicago", "Charlestown", "Helsinki", "Nice", "Dublin"};
    //steden waarvan de naam langer dan 5 posities, gesorteerd op naam
    IEnumerable<string> places = from city in cities
                                where city.Length > 5
                                orderby city ascending
                                select city;
    foreach (string city in places)
        Console.WriteLine(city);
}
```

```
----City names----
Amsterdam
Boston
Charlestown
Chicago
Dublin
Helsinki
Las Vegas
London
Raleigh
San Francisco
```

## LINQ to Objects

- Eigenschappen LINQ
  - Compile time checking van queries
  - **Differed execution** : LINQ queries worden pas geëvalueerd als de resultaatset overlopen wordt (foreach)

```
public void Execute() {  
    string[] cities = {"London", "Amsterdam", "San Francisco", "Las Vegas", "Boston",  
        "Raleigh", "Chicago", "Charlestown", "Helsinki", "Nice", "Dublin" };  
    IEnumerable<string> places = from city in cities  
        where city.Length > 5  
        orderby city ascending  
        select city;  
  
    cities[0] = "Brussels";  
    Console.WriteLine("\n----City names----");  
    foreach (string city in places)  
        Console.WriteLine(city); }  
}
```



```
----City names----  
Amsterdam  
Boston  
Brussels  
Charlestown  
Chicago  
Dublin  
Helsinki  
Las Vegas  
Raleigh  
San Francisco
```

Linq query wordt pas uitgevoerd bij de foreach en bevat dus Brussels

## LINQ to Objects

- Eigenschappen LINQ
  - **Immediate execution** enkel bij aanroepen van LINQ conversie operatoren To... (vb ToList<>) of als query 1 waarde retourneert (Sum,...)

```
public void Execute() {  
    string[] cities = {"London", "Amsterdam", "San Francisco", "Las Vegas", "Boston",  
        "Raleigh", "Chicago", "Charlestown", "Helsinki", "Nice", "Dublin" };  
    IEnumerable<string> places = (from city in cities  
        where city.Length > 5  
        orderby city ascending  
        select city).ToList();  
  
    cities[0] = "Brussels";  
    Console.WriteLine("\n----City names----");  
    foreach (string city in places)  
        Console.WriteLine(city); }  
}
```



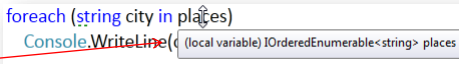

```
----City names----  
Amsterdam  
Boston  
Charlestown  
Chicago  
Dublin  
Helsinki  
Las Vegas  
London  
Raleigh  
San Francisco
```

Immediate execution door ToList dus Brussels komt niet voor.

## LINQ to Objects

- Implicitly typed variables : keyword **var**
  - Compiler detecteert datatype adhv toegewezen waarde.
    - Toekenning moet gebeuren bij declaratie van variabele
    - De variabele kan NOOIT van type veranderen! Alle types zijn mogelijk.
  - var is strongly typed! (var javascript != var C#) -> IntelliSense werkt.

```
public ActionResult Index() {  
    string[] cities = { "London", "Amsterdam", "San Francisco", "Las Vegas", "Boston",  
                        "Raleigh", "Chicago", "Charlestown", "Helsinki", "Nice", "Dublin" };  
  
    var places = from city in cities  
                  where city.Length > 5  
                  orderby city ascending  
                  select city;  
  
    foreach (string city in places)  
        Console.WriteLine(city);  
}
```



## LINQ to Objects

### ► Stap 2 : LINQ en klassen/generic collections

- Voorbeeld (zie Models)

Object Initializers

```
public class Location  
{  
    public string Country { get; set; }  
    public string City { get; set; }  
    public int Distance { get; set; }  
  
    public override string ToString() {  
        return City + " in " + Country; }  
}
```

```
public class CityDistance  
{  
    public string Country { get; set; }  
    public string Name { get; set; }  
    public int DistanceInKm { get; set; }  
}
```

HoGent  
Automatic properties

```
public class TravelOrganizer{  
    public static IList<Location> PlacesVisited  
    {  
        get {  
            IList<Location> cities = new List<Location>{  
                new Location { City="London", Distance=4789, Country="UK" },  
                new Location { City="Amsterdam", Distance=4869, Country="Netherlands" },  
                new Location { City="San Francisco", Distance=684, Country="USA" },  
                new Location { City="Las Vegas", Distance=872, Country="USA" },  
                new Location { City="Boston", Distance=2488, Country="USA" },  
                new Location { City="Raleigh", Distance=2363, Country="USA" },  
                new Location { City="Chicago", Distance=1733, Country="USA" },  
                new Location { City="Charleston", Distance=2421, Country="USA" },  
                new Location { City="Helsinki", Distance=4771, Country="Finland" },  
                new Location { City="Nice", Distance=5428, Country="France" },  
                new Location { City="Dublin", Distance=4527, Country="Ireland" }  
            };  
            return cities;  
        }  
    }  
}
```

## LINQ to Objects

- Voorbeeld : objecten zelf retourneren

```
public void Execute()
{
    IEnumerable<Location> cities = TravelOrganizer.PlacesVisited;

    //Steden waarvan afstand tot Seattle > 1000; gesorteerd op land/stad
    IEnumerable<Location> places = from city in cities
                                   where city.Distance > 1000
                                   orderby city.Country, city.City
                                   select city;
    foreach (Location l in places)
        Console.WriteLine(String.Format("{0}\t{1}\t{2}", l.Country, l.City, l.Distance.ToString()));
}
```

----Cities and their distances----

Country	City	Distance
Finland	Helsinki	4771
France	Nice	5428
Ireland	Dublin	4527
Netherlands	Amsterdam	4869
UK	London	4789
USA	Boston	2488
USA	Charleston	2421
USA	Chicago	1733
USA	Raleigh	2363

## LINQ to Objects

- Object & collection initializers
  - Retourneren van CityDistance ipv Location objecten
  - Instantiatie en toekennen van eigenschappen in 1 enkel statement
  - Voorbeeld

```
public void Execute() {
    IList<Location> cities = TravelOrganizer.PlacesVisited;
    IEnumerable<CityDistance> places = from city in cities
                                        where city.Distance > 1000
                                        orderby city.Country, city.City
                                        select new CityDistance () {
                                            Name = city.City,
                                            Country = city.Country,
                                            DistanceInKm = (int)(city.Distance * 1.61) };
    foreach (CityDistance l in placesInKm)
        Console.WriteLine(String.Format("{0}\t{1}\t{2}",
                                         l.Country, l.Name, l.DistanceInKm.ToString()));
}
```

----Cities and their distances in km----

Country	City	Distance in Km
Finland	Helsinki	7633
France	Nice	8684
Ireland	Dublin	7243
Netherlands	Amsterdam	7790
UK	London	7662
USA	Boston	3980
USA	Charleston	3873
USA	Chicago	2772
USA	Raleigh	3780

## LINQ to Objects

- ▶ Stap 3 : LINQ en anonymous types.
  - Tijdelijke types zonder naam en zonder klassedefinitie. Compiler maakt zelf dit type aan. Dus gebruik var keyword!
  - Gebruik new keyword en object initializers bij instantiatie
  - Eigenschappen zijn alleen-lezen => achteraf niet wijzigbaar

```
var stad = new
{
    Name = "Gent",
    Country = "Belgium"
};
```

## LINQ to Objects

- ▶ Stap 3 : LINQ en anonymous types.

```
public void Execute(){
    IList<Location> cities = TravelOrganizer.PlacesVisited;
    var places = from city in cities
        where city.Distance > 1000
        orderby city.Country, city.City
        select new {
            Name = city.City,
            city.Country,
            DistanceInKm = city.Distance * 1.61 };
    foreach (var l in placesInKm)
        Console.WriteLine(l.ToString());
}
```

Stap 3 : LINQ en anonymous types

```
< Name = Helsinki, Country = Finland, DistanceInKm = 7633 >
< Name = Nice, Country = France, DistanceInKm = 8684 >
< Name = Dublin, Country = Ireland, DistanceInKm = 7243 >
< Name = Amsterdam, Country = Netherlands, DistanceInKm = 7790 >
< Name = London, Country = UK, DistanceInKm = 7662 >
< Name = Boston, Country = USA, DistanceInKm = 3980 >
< Name = Charleston, Country = USA, DistanceInKm = 3873 >
< Name = Chicago, Country = USA, DistanceInKm = 2772 >
< Name = Raleigh, Country = USA, DistanceInKm = 3780 >
```



## LINQ to Objects

- ▶ Stap 4 : LINQ en extension methods.
  - Linq alleen is onvoldoende : enkel filteren en sorteren
  - Wat als we het aantal steden willen opvragen, of de gemiddelde afstand willen berekenen?
    - => Extensiemethodes
  - C# kent heel wat voorgedefinieerde extensie methodes
  - We bekijken eerst hoe we zelf zo'n methode kunnen aanmaken en oproepen.

## LINQ to Objects

- ▶ Stap 4 : LINQ en extension methods.
  - Mogelijke extension methods

Type	Methodes
Restrict	s.Where(...)
Project	s.Select(...), s.SelectMany(...)
Order	s.OrderBy(...).ThenBy(...) ...
Group	s.GroupBy(...)
Quantify	s.Any(...), s.All(...)
Partition	s.TakeFirst(...), s.SkipFirst(...)
Set	s.Distinct(), s.Union(...), s.Intersect(...), s.Except(...)
Singleton	s.Element(...), s.ElementAt(...)
Aggregate	s.Count(), s.Sum(), s.Min(), s.Max(), s.Average(), s.Aggregate()
Convert	s.Reverse<>(), s.ToArray<>(), s.ToList<>(), s.ToDictionary<>()
Cast	s.OfType<T>(), s.Cast<T>

- Opm : Extension methods voor IEnumerable<T> :  
klasse System.Linq.Enumerable.

# LINQ to Objects

- Uitgebreide voorbeelden op <http://code.msdn.microsoft.com/101-LINQ-Samples-3fb9811b>

<b>Restriction Operators</b> <ul style="list-style-type: none"><li>Where - Simple 1</li><li>Where - Simple 2</li><li>Where - Simple 3</li><li>Where - Drilldown</li><li>Where - Indexed</li></ul>	<b>Grouping Operators</b> <ul style="list-style-type: none"><li>GroupBy - Simple 1</li><li>GroupBy - Simple 2</li><li>GroupBy - Simple 3</li><li>GroupBy - Nested</li><li>GroupBy - Comparer</li><li>GroupBy - Comparer, Mapped</li></ul>	<b>Aggregate Operators</b> <ul style="list-style-type: none"><li>Count - Simple</li><li>Count - Conditional</li><li>Count - Indexed</li><li>Count - Nested</li><li>Count - Grouped</li><li>Sum - Simple</li><li>Sum - Projection</li><li>Sum - Grouped</li><li>Min - Simple</li><li>Min - Projection</li><li>Min - Grouped</li><li>Min - Elements</li><li>Max - Simple</li><li>Max - Projection</li><li>Max - Grouped</li><li>Max - Elements</li><li>Average - Simple</li><li>Average - Projection</li><li>Average - Grouped</li><li>Fold - Simple</li><li>Fold - Seed</li></ul>
<b>Projection Operators</b> <ul style="list-style-type: none"><li>Select - Simple 1</li><li>Select - Simple 2</li><li>Select - Transformation</li><li>Select - Anonymous Types 1</li><li>Select - Anonymous Types 2</li><li>Select - Anonymous Types 3</li><li>Select - Indexed</li><li>Select - Filtered</li><li>SelectMany - Compound from 1</li><li>SelectMany - Compound from 2</li><li>SelectMany - From Assignment</li><li>SelectMany - Multiple from</li><li>SelectMany - Indexed</li></ul>	<b>Set Operators</b> <ul style="list-style-type: none"><li>Distinct - 1</li><li>Distinct - 2</li><li>Union - 1</li><li>Union - 2</li><li>Intersect - 1</li><li>Intersect - 2</li><li>Except - 1</li><li>Except - 2</li></ul>	<b>Miscellaneous Operators</b> <ul style="list-style-type: none"><li>Concat - 1</li><li>Concat - 2</li><li>EqualAll - 1</li><li>EqualAll - 2</li></ul>
<b>Partitioning Operators</b> <ul style="list-style-type: none"><li>Take - Simple</li><li>Take - Nested</li><li>Skip - Simple</li><li>Skip - Nested</li><li>TakeWhile - Simple</li><li>SkipWhile - Simple</li><li>SlipWhile - Simple</li></ul>	<b>Conversion Operators</b> <ul style="list-style-type: none"><li>To Array</li><li>To List</li><li>To Dictionary</li><li>OfType</li></ul>	<b>Custom Sequence Operators</b> <ul style="list-style-type: none"><li>Combine</li></ul>
<b>Ordering Operators</b> <ul style="list-style-type: none"><li>OrderBy - Simple 1</li><li>OrderBy - Simple 2</li><li>OrderBy - Simple 3</li><li>OrderBy - Comparer</li><li>OrderByDescending - Simple 1</li><li>OrderByDescending - Simple 2</li><li>OrderByDescending - Comparer</li><li>ThenBy - Simple</li></ul>	<b>Element Operators</b> <ul style="list-style-type: none"><li>First - Simple</li><li>First - Indexed</li><li>FirstOrDefault - Simple</li><li>FirstOrDefault - Condition</li><li>FirstOrDefault - Indexed</li><li>ElementAt</li></ul>	<b>Query Execution</b> <ul style="list-style-type: none"><li>Deferred</li><li>Immediate</li><li>Query Reuse</li></ul>
<b>Quantifiers</b> <ul style="list-style-type: none"><li>Any - Simple</li><li>Any - Indexed</li><li>Any - Grouped</li><li>All - Simple</li><li>All - Indexed</li></ul>	<b>Generation Operators</b> <ul style="list-style-type: none"><li>Range</li><li>Repeat</li></ul>	

# LINQ to Objects

- Wat is een Extension method ?
  - Voegt nieuwe methodes toe aan bestaande (CLR of andere) klassen.
  - Zonder te subclassen of hercompileren
  - Is static methode met this argument in een static klasse
- Hoe een extension method aanmaken en gebruiken?
  - Definieer een static klasse, vaak binnen een aparte namespace
  - Voeg static methodes toe met als eerste parameter een instance van de klasse die wordt uitgebreid (gebruik hiervoor this keyword).
    - extension methode heeft geen toegang tot private members van die klasse!
  - Extension methodes zijn achteraf ook in IntelliSense zichtbaar

## LINQ to Objects

- Ga naar de Models folder, klasse StringExtension.
- Namespace StringExtension
- De klasse bevat momenteel 1 static methode HerhaalText

```
namespace StringExtensions{  
    public static class StringExtension {  
        public static string HerhaalTekst(string s, int aantal)  
        {  
            string resultaat = String.Empty;  
            for (int i = 1; i <= aantal; i++)  
                resultaat += s;  
            return resultaat;  
        }  
    }  
}
```

- Pas de code in Step4 klasse RepeatText aan, voeg using toe en run

```
public void RepeatText(string text, int number)  
{  
    Console.WriteLine(StringExtension.HerhaalTekst(text, number));  
}
```

HoGent

Step 4 : LINQ and extension methods

```
-----Repeat text-----  
Enter the text to be repeated :  
Hello  
Enter how many times it has to be repeated :  
4  
HelloHelloHelloHello
```

## LINQ to Objects

- Ga naar de Models folder, klasse StringExtension aan.
- We maken van HerhaalTekst een extension methode
  - De eerste parameter is een instance van de klasse die wordt uitgebreid (gebruik hiervoor this keyword).

```
namespace StringExtensions{  
    public static class StringExtension {  
        public static string HerhaalTekst(this string s, int aantal){  
            string resultaat = String.Empty;  
            for (int i = 1; i <= aantal; i++)  
                resultaat += s;  
            return resultaat; } } }
```

- Pas de code in Step4 klasse RepeatText aan en run

```
public void RepeatText(string text, int number)  
{  
    Console.WriteLine(text.HerhaalTekst(number));  
}
```

HoGent

## LINQ to Objects

- LINQ maakt gebruik van extension methodes. Voorbeeld, zie Step4.cs. Wat doen deze methodes? Voeg query2 toe!

```
ICollection<Location> cities = TravelOrganizer.PlacesVisited;  
IEnumerable<Location> locations = (from city in cities  
    orderby city.Distance descending  
    select city).Skip(1).Take(5);
```

Skip : skipt een aantal items in de lijst  
Take retourneert een aantal items uit de lijst

```
int count = (from city in cities  
    where city.Country != "USA"  
    select city).Count();
```

Count : telt aantal items in lijst

```
Location farthestCity = (from city in cities  
    orderby city.Distance descending  
    select city).FirstOrDefault();
```

FirstOrDefault : retourneert eerste in lijst, retourneert null als lijst leeg.

→ LINQ wordt onmiddellijk uitgevoerd!

## LINQ to Objects

- ▶ Stap 5 : Extension methods en lambda expressions
  - Extensie methodes worden vaak gebruikt in combinatie met lambda's. Stel je wenst de som van de afstanden te kennen, dan moet je op 1 of andere manier opgeven aan de Sum extensiemethode hoe hij de afstand kan bekomen van elke stad
  - Een lambda expression is een anonieme methode
    - Lambda expression bestaat uit parameterlijst, =>, instructies

```
// Calculate total city distances of all cities outside US  
int totalDistance = (from city in cities  
    where city.Country != "USA"  
    select city).Sum(loc => loc.Distance);
```

```
// Calculate average city distances of each city trip  
double averageDistance = cities.Average(loc => loc.Distance);
```

## LINQ to Objects

- Wat is een Lambda expression ?
  - Om de betekenis te verstaan, dienen we terug te gaan naar delegates (zie Step5A.cs)
- **A. Declaratie van delegate**
  - Delegate = blauwdruk (of definitie) van een methode.
  - Zo kan ook een methode als parameter worden meegegeven aan een andere methode.
  - Definitie van een delegate :
    - Definitie kan binnen een namespace, buiten of binnen een klasse
    - Definieert signatuur(return type en parameters) van een methode

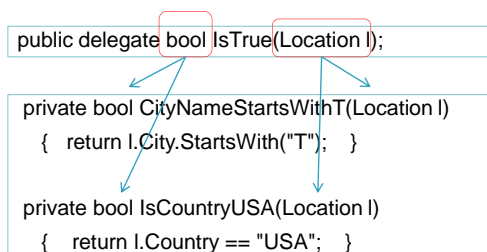
```
public delegate string GetAString();  
public delegate bool IsTrue(Location l);
```

- Delegat IsTrue beschrijft een methode die een Location object als parameter verwacht en een boolean teruggeeft.

## LINQ to Objects

- Wat is een Lambda expression ?
- **B. Instantiatie van delegate**
  - Vervolgens kan je 1 of meerdere methodes aanmaken die met de signatuur van de delegate overeenkomen
  - Voorbeeld

```
public delegate bool IsTrue(Location l);  
  
private bool CityNameStartsWithT(Location l)  
{ return l.City.StartsWith("T"); }  
  
private bool IsCountryUSA(Location l)  
{ return l.Country == "USA"; }
```



- Beide methodes hebben 1 parameter van type Location en retourneren een bool, zoals opgegeven in delegate

## LINQ to Objects

- Lambda expressions?
  - Vervolgens een instantie aanmaken van een delegate
    - Constructor van een delegate bevat steeds 1 parameter nl. de naam van de methode die zal worden uitgevoerd bij uitvoeren van delegate
    - Parameter kan om het even welke instance of statische methode van om het even welk object zijn, op voorwaarde dat de signatuur met delegate overeenkomt.

```
Location l = TravelOrganizer.PlacesVisited[0];
IsTrue method1 = new IsTrue(CityNameStartsWithT);
IsTrue method2 = new IsTrue(IsCountryUSA);
Console.WriteLine("City " + l.City + " starts with T? " + method1(l).ToString());
Console.WriteLine("City " + l.City + " in the USA? " + method2(l).ToString());
```

Voert de code uit de methode  
CityNameStartsWithT uit

Voert de code uit de methode  
IsCountryUSA uit

City London starts with T? False  
City London in the USA? False

## LINQ to Objects

- Lambda expressions?
  - Vervolgens een instantie aanmaken van een delegate
    - Wat is het resultaat van

```
GetAString methodGetAString = new GetAString(l.ToString);
Console.WriteLine("GetAString : " + methodGetAString());
```

- Run de code (code heeft nog runtimefout, code moet nog worden aangevuld)

## LINQ to Objects

- Lambda expressions?
  - **C. Anonymous methods**
    - Ipv bij instantiatie de naam van een methode mee te geven kan je ook een stukje code meegeven = anonymous methode
    - Bij instantiatie van de delegate, maak je dan gebruik van keyword delegate gevolgd door de parameter(s) en dan de implementatie code

```
IsTrue method3 = delegate(Location loc)
{
    return loc.City.StartsWith("L");
};
Console.WriteLine("City " + l.City + " starts with L? " + method3(l).ToString());
```

City London starts with L? True

## LINQ to Objects

- Lambda expressions?
  - **D. Verkorte notatie van een anonymous methode**
    - die kan gebruikt worden om delegates en expression trees te creëren
    - => : lambda operator, lezen als "goes to"
    - Linkerkant van lambda : de parameters van de anonieme methode, zonder vermelding van type (compiler vangt dit op)
    - Rechterkant van lambda : de implementatie waar je {} en het return statement mag weglaten indien code uit 1 instructie bestaat.

```
//anonieme methode
IsTrue method3 = delegate(Location loc)
{ return loc.City.StartsWith("L"); };
//anonieme methode maar nu gebruik makend van verkorte lambda notatie
IsTrue method5 = loc =>
{ return loc.City.StartsWith("L"); };
//of lambda verkort zonder {} en return statement
IsTrue method5 = loc => loc.City.StartsWith("L");
```

Parameter zonder vermelding van type

maakt nu gebruik makend van verkorte lambda notatie

## LINQ to Objects

- Lambda expressions?
  - Lambda expression voor City in USA?

```
//anonieme methode
IsTrue method4 = loc =>
{ return loc.Country == "USA"; };
Console.WriteLine("City " + l.City + " in USA? " + method4(l).ToString());

//of lambda verkort zonder {} en return statement
IsTrue method6 = loc => loc.Country == "USA";
Console.WriteLine("City " + l.City + " in USA? " + method6(l).ToString());
```

```
City London in USA? False
City London in USA? False
```

## LINQ to Objects

- Func delegates?
  - **E. Func – delegate** : verkorte manier om delegates te declareren.  
Is een generic delegate
  - vb Func<T,Tresult>
    - T is inputparameters delegate
    - Tresult = returntype delegate

Het .NET Framework heeft een aantal generische delegates gedefinieerd. Het *Func<>* type is een snelle manier om delegate te definiëren. De syntax is als volgt gedefinieerd in de MSDN library.

```
public delegate TResult Func<TResult> ()
public delegate TResult Func<T1, TResult> (T1)
public delegate TResult Func<T1, T2, TResult> (T1 arg1, T2 arg2 )
public delegate TResult Func<T1, T2, T3, TResult> (T1 arg1, T2 arg2,T3 arg3)
public delegate TResult Func<T1, T2, T3, T4, TResult> (T1 arg1, T2 arg2, T3 arg3, T4 arg4)
```



## LINQ to Objects

- Func delegates?
  - Zo kan de delegate definitie verwijderd worden en de code herschreven worden als

```
public delegate bool IsTrue(Location l);  
IsTrue method5 = loc => loc.City.StartsWith("T");
```



```
Func<Location, bool> method7 = loc => loc.City.StartsWith("T");  
Console.WriteLine("City " + l.City + " starts with T? " + method7(l).ToString());
```

## LINQ to Objects

- Func delegates?
  - Zo kan je ook in een methode een Func als parameter gebruiken

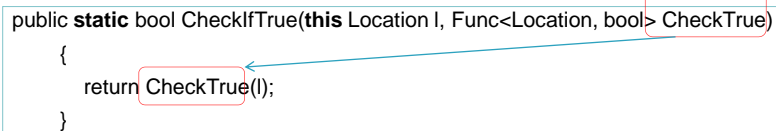
```
private bool CheckIfTrue(Func<Location, bool> CheckTrue, Location l)  
{  
    return CheckTrue(l);  
}
```

```
Console.WriteLine("City " + l.City + " starts with T? " +  
    CheckIfTrue(loc=>loc.City.StartsWith("T"),l).ToString();
```

## LINQ to Objects

- Func delegates?
  - **F. Func delegate kan je ook in Extension methodes als parameter gebruiken**
    - Zie de klasse StringExtension

```
public static bool CheckIfTrue(this Location l, Func<Location, bool> CheckTrue)
{
    return CheckTrue(l);
}
```



- Aanroepen van extension methode

```
Console.WriteLine("City " + l.City + " starts with T? " +  
    l.CheckIfTrue(loc => loc.City.StartsWith("T")).ToString());
```

## LINQ to Objects

- Enkele voorbeelden uit Linq (Stap5B.cs)

```
List<Location> cities = TravelOrganizer.PlacesVisited;  
int totalDistance = (from city in cities  
    where city.Country != "USA"  
    select city).Sum(loc => loc.Distance);  
  
double averageDistance = cities.Average(loc => loc.Distance);
```

## LINQ to Objects

- Nog enkele voorbeelden (Step5B.cs)
  - Where clause kan je ook als extension method met lambda noteren

```
IEnumerable<Location> places = from city in cities
                                where city.Country != "USA"
                                select city;
```

```
IEnumerable<Location> x = cities.Where(city=> city.Country != "USA")
                                .Select(city=>city);

int aantal= x.Count();
int totaal = x.Sum(loc => loc.Distance);
bool any = x.Any(city=>city.StartsWith("B")); //true als minstens 1 element
uit sequence aan voorwaarde voldoet
bool all = x.All(city=>city.StartsWith("B")); //true als alle elementen uit
sequence aan voorwaarde voldoen
bool contains = x.Contains("Brussels"); //true als Brussels in sequence
```

## LINQ to Objects

- Voorbeeld : Extension methode Where in MSDN

```
public static IEnumerable<T> Where<T>(
    this IEnumerable<T> source,
    Func<T, bool> predicate)
{
    foreach (T item in source)
        if (predicate(item))
            yield return item;
}
```

# LINQ to Objects

- Query syntax <-> Extension method syntax
  - Query syntax wordt vertaald door compiler naar methode calls

```
int[] numbers = { 5, 10, 8, 3, 6, 12};
```

//Query syntax:

```
IEnumerable<int> numQuery1 =  
    from num in numbers  
    where num % 2 == 0  
    orderby num  
    select num;
```

//Method syntax:

```
IEnumerable<int> numQuery2 =  
    numbers.Where(num => num % 2 == 0)  
    .OrderBy(n => n)  
    .Select(n=>n);
```

Beide geven hetzelfde resultaat. Query syntax is leesbaarder. Je kiest zelf de syntax.

# Oefening

- Oefening : Zie Step6.cs

Step 6 : Exercises

----Countries----

Finland  
France  
Ireland  
Netherland  
UK  
USA

Choose a country :

Maak eerst gebruik van Linq queries. Herschrijf dan met extension methods

----Cities in USA----

Boston  
Charleston  
Chicago  
Las Vegas  
Raleigh  
San Francisco  
Total distance : 10561

## Appendix : Reflection

### ► Reflection



In object oriented programming languages such as [Java](#), reflection allows inspection of classes, interfaces, fields and methods at runtime without knowing the names of the interfaces, fields, methods at compile time. It also allows instantiation of new objects and invocation of methods.

Reflection can also be used to adapt a given program to different situations dynamically. For example, consider an application that uses two different classes  $X$  and  $Y$  interchangeably to perform similar operations. Without reflection-oriented programming, the application might be hard-coded to call method names of class  $X$  and class  $Y$ . However, using the reflection-oriented programming paradigm, the application could be designed and written to utilize reflection in order to invoke methods in classes  $X$  and  $Y$  without hard-coding method names. Reflection-oriented programming

- EF en andere ORM tools maken daar gebruik van

## Appendix : Reflection

### ► Voorbeeld : Main methode in Program.cs

```
if (keuze != "99")
{
    Type type = Type.GetType("Linq.Step" + keuze);
    if (type != null)
    {
        Object o = Activator.CreateInstance(type);
        type.GetMethod("Execute").Invoke(o, null);
    }
}
```

Type discovery : reflection zoekt een klasse in de assembly met de naam Linq.Step1.

Creëert een instantie van die klasse

Voert de methode Execute van dit object uit (null : daar deze methode geen parameters vereist)

## Referenties

- ▶ Voorbeelden uit slides :  
<http://weblogs.asp.net/scottgu/archive/2006/05/14/Using-LINQ-with-ASP.NET-2800-Part-1-2900.aspx>
- ▶ LINQ : <http://msdn.microsoft.com/en-us/library/bb397926.aspx>
- ▶ Pluralsight

Opm. In de slides werd gebruik gemaakt van bovenstaande referenties