

.NET Advanced

LINQ introductie



AP HOGESCHOOL
ANTWERPEN

AP.BE

Topics

- Inleiding
- C# features gebruikt in LINQ
 - Extension methods
 - Lambda expressions
 - Initializers
 - Implicit typing en type inference
 - Anonymous types
- LINQ basics

LINQ introductie

Inleiding

LINQ introductie - Inleiding

Leerdoelen

- Het concept van LINQ kennen
- De LINQ enabled data sources kennen
- De voordelen van LINQ kennen

LINQ Introductie - Inleiding

Theorie

- LINQ = **L**anguage **I**Ntegrated **Q**uery
- Set van .NET Framework extensions
- Beschikbaar sinds .NET Framework versie 3.5
- Query's schrijven in C# i.p.v. SQL
- Ook bruikbaar voor andere data sources
- Drie categorieën van LINQ enabled data sources

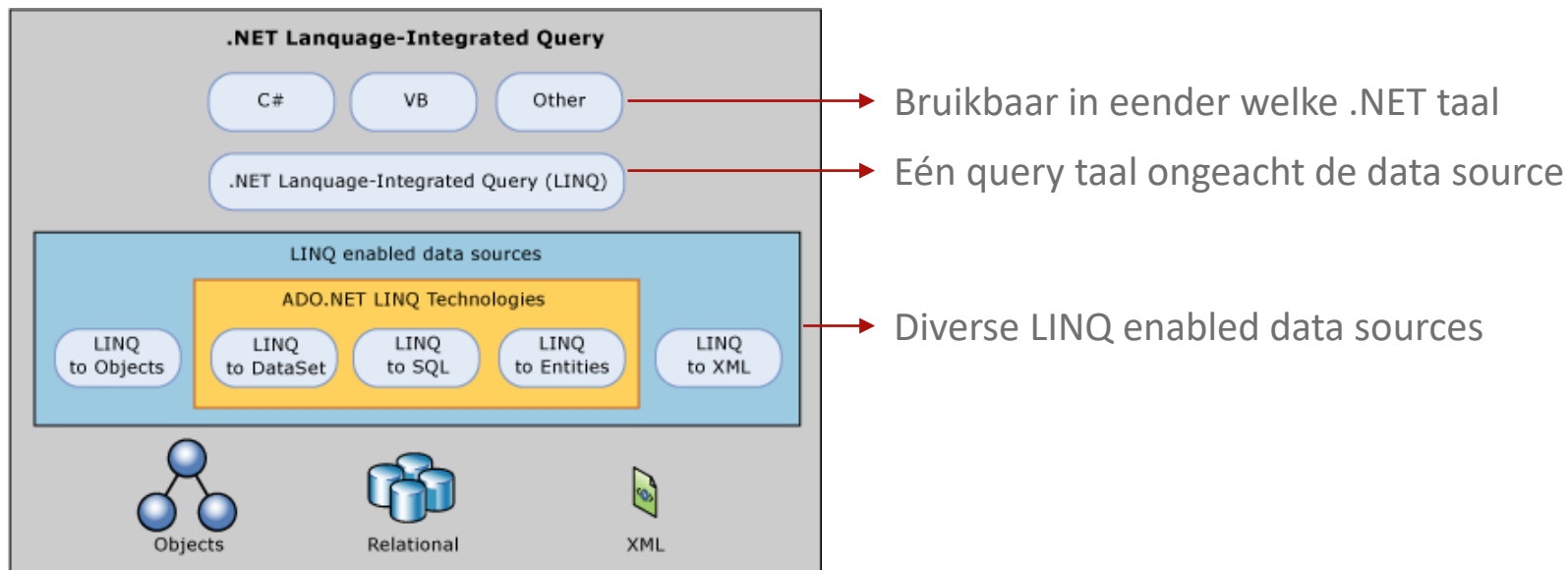
LINQ Introductie - Inleiding

Theorie

- LINQ to Objects
 - Query's schrijven tegen eender welke IEnumerable of IEnumerable<T> collection
 - Bv. array, list, ...
- LINQ to ADO.NET
 - LINQ To SQL
 - LINQ To DataSet
 - LINQ To Entities ⇨ Wordt gebruikt bij Entity Framework
- LINQ to XML
 - Query's schrijven tegen bronnen in XML-formaat

LINQ Introductie - Inleiding

Theorie



LINQ Introductie - Inleiding

Theorie

Waarom LINQ gebruiken?

- Eén querytaal voor verschillende data sources (SQL, XML, ...) geïntegreerd in .NET
- Traditionele query's bieden geen type checking @ compile time
 - M.b.v. LINQ schrijven we query's tegen strongly typed collections
 - Intellisense
- Biedt voordelen t.o.v. traditionele foreach loops in code
 - Beknoper en beter leesbaar, zeker bij meerdere filtercondities
 - Krachtige filter-, sorteer- en groeperingsmogelijkheden met minimale code
- Porteerbaar naar andere data sources met weinig tot geen wijzigingen

LINQ Introductie - Inleiding

Voorbeelden

```
static void QuerySyntax()
{
    int[] numbers = { 5, 10, 8, 3, 6, 12 };

    IEnumerable<int> evenNumbers =
        from number in numbers
        where number % 2 == 0
        orderby number
        select number;

    foreach(int number in evenNumbers)
    {
        Console.WriteLine($"{number} ");
    }

    Console.ReadLine();
}
```

```
static void MethodSyntax()
{
    int[] numbers = { 5, 10, 8, 3, 6, 12 };

    IEnumerable<int> evenNumbers =
        numbers.Where(n => n % 2 == 0).OrderBy(n => n);

    foreach (int number in evenNumbers)
    {
        Console.WriteLine($"{number} ");
    }

    Console.ReadLine();
}
```

Resultaat: 6 8 10 12 _

LINQ introductie

C# Features - Extension methods

LINQ introductie - C# Features - Extension methods

Leerdoelen

- Het concept van extension methods kennen
- Het verband tussen extension methods en LINQ kennen
- De syntax voor extension methods kennen
- Extension methods kunnen schrijven
- Extension methods kunnen aanroepen

LINQ introductie - C# Features - Extension methods

Theorie

- “Toevoegen” van extra methods aan bestaande datatypes zonder daarbij
 - Een nieuw afgeleid datatype te moeten maken
 - Het originele datatype te moeten wijzigen of hercompileren
- Zijn static methods die lijken alsof ze instance methods van het extended type zijn
- Bruikbaar om
 - Custom validations te schrijven
 - Bestaande class of interface uit te breiden met functionaliteit zonder deze te overriden

LINQ introductie - C# Features - Extension methods

Theorie

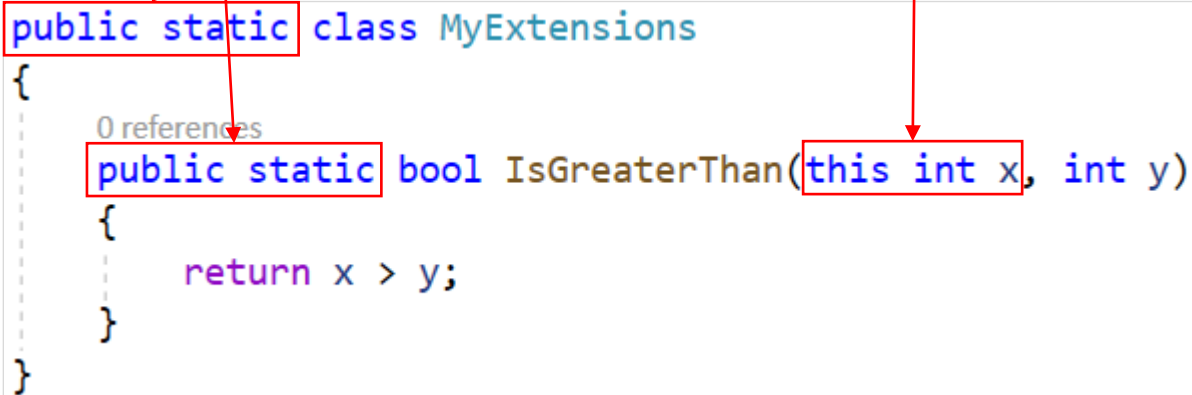
- Vereisten
 - Aanmaken in een static non-generic public class
 - Method moet als static worden gedefinieerd
 - Toevoegen van `this` modifier voor eerste parameter van method
 - Datatype van deze parameter bepaalt voor welk datatype de extension method wordt aangemaakt
- Aanroepen
 - Static method aanroepen met instance syntax

LINQ introductie - C# Features - Extension methods

Voorbeelden

Definieer de class als public static
Definieer de method als public static

this modifier geeft aan voor welk datatype
de extension method bruikbaar wordt



```
public static class MyExtensions
{
    0 references
    public static bool IsGreaterThan(this int x, int y)
    {
        return x > y;
    }
}
```


LINQ introductie - C# Features - Extension methods

Voorbeelden

Extension method aanroepen met instance syntax

```
static void Main(string[] args)
{
    int i = 7, j = 6;

    Console.WriteLine(i.IsGreaterThan(j));
    j = 8;
    Console.WriteLine(i.IsGreaterThan(j));
    Console.ReadLine();
}
```

Resultaat:

True
False

LINQ introductie - C# Features - Extension methods

Theorie

- Standard query operators in LINQ zijn geïmplementeerd als extension methods
- Terug te vinden in class Enumerable in System.Linq
- Extension methods enkel gekend in namespace waarin deze zijn gedeclareerd
- In scope brengen met using System.Linq directive

LINQ introductie - C# Features - Extension methods

Demo

- Demo 1
- Voorbeeldcode op digitap

LINQ introductie

C# Features - Lambda Expressions

LINQ introductie - C# Features - Lambda Expressions

Leerdoelen

- Het concept van Delegates kennen
- Het concept van Lambda expressions kennen
- De syntax van Lambda expressions kennen

LINQ introductie - C# Features - Lambda Expressions

Theorie

- Lambda Expression is een anonymous method \Rightarrow geen naam, wel implementatie
- Een anonymous method wordt toegewezen aan een delegate object
- Een delegate object is een reference type variabele die een referentie naar een method bevat
- Een delegate object is van een bepaald delegate type
- Het delegate type bepaalt het signatuur waaraan methods moeten voldoen om aan een delegate object van dat delegate type toegewezen te kunnen worden
- Query operators in LINQ zoals Where() hebben een delegate object als parameter
- Hieraan kunnen we dus een anonymous function toewijzen

LINQ introductie - C# Features - Lambda Expressions

Theorie

C# 1.0: Apart declareren van method en delegate

```
public delegate int MyDelegate(DateTime dateTime);

0 references
static void Main(string[] args)
{
    MyDelegate calculateAge = CalculateAge;
    Console.WriteLine(calculateAge(new DateTime(2000, 11, 5)));
}

1 reference
public static int CalculateAge(DateTime DateOfBirth)
{
    return DateTime.Now.Year - DateOfBirth.Year;
}
```

We creëren een type MyDelegate. We kunnen dan variabelen declareren van dit type MyDelegate.

Aan dergelijke variabelen kunnen we dan methods koppelen die een DateTime als input parameter hebben EN een int als returnvalue opleveren

We creëren een variabele van het delegate type MyDelegate
We wijzen daarna de method CalculateAge toe aan deze variabele.

We kunnen nu de method uitvoeren via de variabele

LINQ introductie - C# Features - Lambda Expressions

Theorie

C# 2.0: Anonymous methods, apart declareren van method is weggevallen

```
public delegate int MyDelegate(DateTime dateTime);

References
static void Main(string[] args)
{
    MyDelegate calculateAge = delegate (DateTime DateOfBirth) { return DateTime.Now.Year - DateOfBirth.Year; };
    Console.WriteLine(calculateAge(new DateTime(2000, 11, 5)));
}
```

De implementatie van de method moet niet langer apart maar kan onmiddellijk worden toegewezen aan de variabele, zolang de implementatie voldoet aan het signatuur opgelegd door de delegate.

LINQ introductie - C# Features - Lambda Expressions

Theorie

C# 3.0: Lambda expressions

```
public delegate int MyDelegate(DateTime dateTime);  
  
0 references  
static void Main(string[] args)  
{  
    MyDelegate calculateAge = DateOfBirth => DateTime.Now.Year - DateOfBirth.Year;  
    Console.WriteLine(calculateAge(new DateTime(2000, 11, 5)));  
}
```

- Delegate keyword is niet meer nodig bij het toewijzen van een method implementatie aan de variabele.
- In geval de method exact 1 parameter verwacht hoeven er geen haakjes rond geplaatst te worden.
- In geval de implementatie uit 1 instructie bestaat moeten geen accolades getypt worden.
- Return statement is niet meer nodig tenzij je met accolades werkt.

LINQ introductie - C# Features - Lambda Expressions

Theorie

- Linkerkant van lambda expression is method signature
- Rechterkant is expression of statement block (function body)
- Daartussen de => operator (goes to)
- Elke lambda moet toegewezen worden aan een compatibel delegate type
- Hiervoor zijn ingebouwde delegates beschikbaar
 - Action ⇨ geeft geen return waarde terug
 - Func ⇨ geeft een return waarde terug waarbij laatste parameter de return waarde is
- Zichtbaar via intellisense bij query operator
- Lambda's zijn in het leven geroepen voor korte blokjes code ⇨ hou dit zo!

LINQ introductie - C# Features - Lambda Expressions

Demo

- Demo 2
- Voorbeeldcode op digitap

LINQ introductie

C# Features - Initializers

LINQ introductie - C# Features - Initializers

Leerdoelen

- Het concept van object initializers kennen
- De syntax van object initializers kennen
- Het concept van collection initializers kennen
- De syntax van collection initializers kennen
- Object- en Collection initializers op een correcte manier kunnen toepassen

LINQ introductie - C# Features - Initializers

Theorie

- Object initializers maken het onnodig om
 - Eerst een object te instantiëren
 - Daarna per property een waarde toe te wijzen op een nieuwe regel
- Collection initializers
 - Maken het onnodig om de `Add()` method x-aantal keer te gebruiken om object instances toe te voegen
 - Maken het mogelijk om object instances toe te voegen tussen accolades

LINQ introductie - C# Features - Initializers

Demo

- Demo 3
- Voorbeeldcode op digitap

LINQ introductie

C# Features - Implicit typing

LINQ introductie - C# Features - Implicit typing

Leerdoelen

- Het concept van implicit typing en type inference kennen
- De syntax voor implicit typing te gebruiken kennen
- Implicit typing op een correct manier kunnen toepassen

LINQ introductie - C# Features - Implicit typing

Theorie


- Normaal steeds expliciet datatype van een variabele/parameter opgeven

```
string myString = "mystring";
```

- Mogelijkheid om datatype te laten detecteren door de compiler
⇒ Type inference!
- Toe te passen met het `var` keyword
 - Variabelen van dit type moeten onmiddellijk geïnitieerd worden

```
var myString;  
myString= "mystring";
```



 (local variable) var myString

CS0818: Implicitly-typed variables must be initialized

```
var myString = "mystring";
```




LINQ introductie - C# Features - Implicit typing

Theorie

- Geen meerdere declaraties tegelijkertijd mogelijk

```
var myString, myString2 = "mystring";
```



 (local variable) `string myString2`

CS0819: Implicitly-typed variables cannot have multiple declarators

```
var myString = "mystring";  
var myString2 = "mystring";
```



- `null` niet toewijsbaar

```
var myString = null;
```



CS0815: Cannot assign <null> to an implicitly-typed variable


LINQ introductie - C# Features - Implicit typing

Theorie

- Types mixen bij bewerkingen niet altijd mogelijk

```
var myValue = "42";  
int y = myValue + 1;
```




 (local variable) `string myValue`
CS0029: Cannot implicitly convert type 'string' to 'int'

- Het `var` keyword is niet hetzelfde als in javascript
 - Geen loose typing (`var b = 1; b="s";`)
 - Eens datatype is toegekend kan je geen value van een ander datatype meer toewijzen

```
var myValue = "42";  
myValue = 42;
```



 `readonly struct System.Int32`
Represents a 32-bit signed integer.
CS0029: Cannot implicitly convert type 'int' to 'string'

LINQ introductie

C# Features - Anonymous types

LINQ introductie - C# Features - Anonymous types

Leerdoelen

- Het concept van anonymous types m.b.t. LINQ kennen
- Het doel van anonymous types kennen
- Anonymous types op een correcte manier kunnen gebruiken

LINQ introductie - C# Features - Anonymous types

Theorie

- Naamloze classes die aangemaakt worden met een object initializer
- Volgende regels gelden
 - Moet steeds toegewezen worden aan het **var** type
 - Properties moeten gespecificeerd worden MET hun initiële waarden!
 - Eens aangemaakt kunnen waarden van properties niet meer aangepast worden (immutable type)
 - Kunnen enkel als variabele in een functie gedefinieerd worden, niet in een class
 - Erft altijd over van System.Object
 - Kunnen niet gebruikt worden als return value of parameter, indien toch gewenst kan dit enkel door als type System.Object te gebruiken
 - Kunnen gebruikt worden bij projectie!

LINQ introductie - C# Features - Anonymous types

Demo

- Demo 4
- Voorbeeldcode op digitap

LINQ introductie

Query basics

LINQ introductie - Query basics

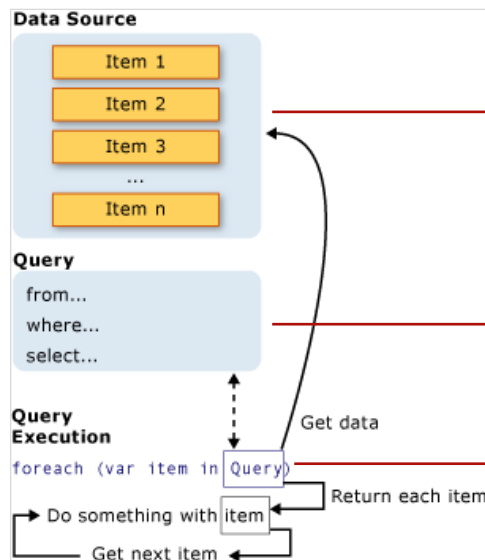
Leerdoelen

- Het concept van een LINQ query kennen
- De basiselementen van een LINQ query kennen
- Een basis LINQ query kunnen uitwerken in de query syntax
- Een basis LINQ query kunnen uitwerken in de extension method syntax
- Het resultaat van een LINQ query kunnen tonen

LINQ introductie - Query basics

Theorie

- Een typische LINQ query in query syntax bestaat uit 3 acties



De data source instellen

De query opstellen

De query uitvoeren

LINQ introductie - Query basics

Theorie

- Actie 1: Data source instellen
- De data source moet
 - Een object zijn dat de IEnumerable<T> of IQueryable<T> interface implementeert
 - Waarbij <T> primitive types (zoals int, double, ...) of custom types kunnen zijn

```
string[] names = {  
    "Nelson Muntz", "Milhouse Van Houten",  
    "Ralph Wiggum", "Dr. Nick Riviera", "Reverend Lovejoy"  
};
```

- In-memory zijn
 - Indien niet (bv. XML-bestand) moet het eerst in-memory geladen worden

LINQ introductie - Query basics

Theorie

- Actie 2: Query opstellen
- Specificiëren welke informatie op te halen uit de data source
- Optioneel sortering/groepering/transformatie specificiëren
- Wordt bewaard in een query variabele
 - Bevat enkel de definitie van de query
 - De query in de variabele wordt niet onmiddellijk uitgevoerd
 - We noemen dit concept deferred execution (uitgestelde uitvoering) (analoog aan query typen in MySQL Workbench maar niet uitvoeren)
 - Bevat drie clauses: **from** clause, **where** clause en **select** clause

LINQ introductie - Query basics

Voorbeelden

Query syntax

from clause:

- De data source (names)
- Een range variabele (name). Deze variabele zal dienen als een reference naar elk object in de datasource wanneer de query uitgevoerd wordt

De query variabele

```
IEnumerable<string> simpsons = from name in names  
                                where name.Length <= 16  
                                select name;
```

where clause:

De filter die je wilt toepassen (analoog aan SQL)

select clause:

Bepaalt wat je van het object wenst op te vragen

LINQ introductie - Query basics

Voorbeelden

Extension method syntax met lambda expressions

De query variabele

from clause:

- Niet meer expliciet te definiëren met het from keyword
- Geen range variabele meer nodig

```
var simpsons = names  
    .Where(n => n.Length <= 16);
```

where clause:

De filter die je wilt toepassen via een lambda expression

select clause:

Niet nodig in deze syntax vorm tenzij projectie nodig is.

LINQ introductie - Query basics

Theorie

- Actie 3: Query uitvoeren
- Effectieve uitvoering gebeurt pas bij het itereren over de query variabele
- Itereren gebeurt steeds in een **foreach** statement
- Uitzonderingen zijn “greedy” operatoren
 - Aggregatie operatoren zoals Count, Max, ...
 - Worden onmiddellijk uitgevoerd (impliciete **foreach**)
 - Dit soort query's geeft geen IEnumerable terug maar een scalaire waarde

LINQ introductie - Query basics

Voorbeelden

```
string[] names = {  
    "Nelson Muntz", "Milhouse Van Houten",  
    "Ralph Wiggum", "Dr. Nick Riviera", "Reverend Lovejoy"  
};
```

```
var simpsons = names  
    .Where(n => n.Length <= 16);
```

- Iteratie over de query m.b.v. een **foreach** loop
- Pas bij iteratie wordt query uitgevoerd

```
foreach (string s in simpsons)  
{  
    Console.WriteLine($"Character fullname: {s}");  
}
```

LINQ introductie - Query basics

Demo

- Demo 5 (Deferred execution)
- Demo 6 (Example query's)
- Voorbeeldcode op digitap

Oefeningen

Deel 4 - Oefenbundels 1 - 4

- Surf naar Digitap
- Download de opgave
- Los de vragen op

LINQ introductie



Bronnen

- Pluralsight
 - [LINQ Fundamentals](#) (Scott Allen)
- Microsoft
 - <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/extension-methods>
 - <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/object-and-collection-initializers>