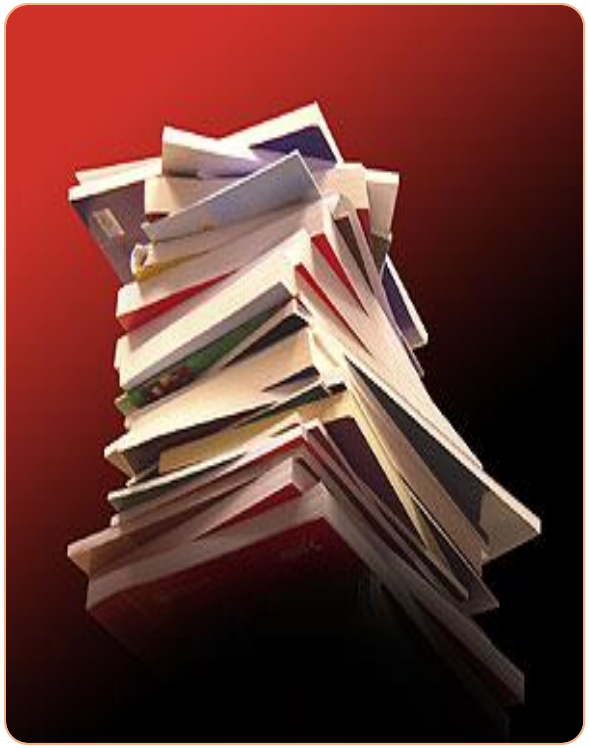
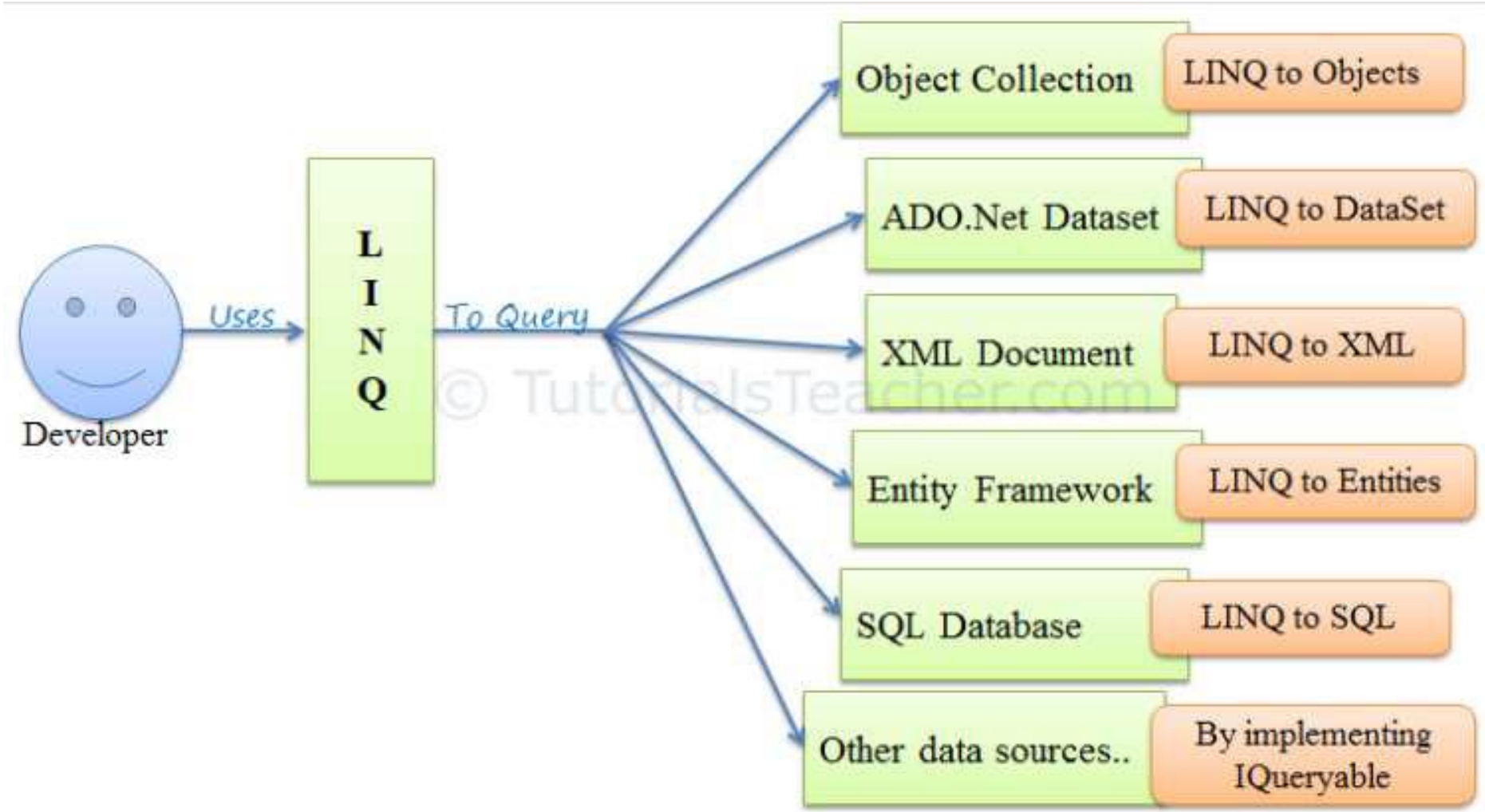
LINQ in C#

Inhoud

1. Wat is LINQ?
2. Voordelen van LINQ
3. LINQ Operators
   - Filtering Operators
   - Grouping Operators
   - Concatenation
   - Sorting Operators
   - Join Operators
   - Equality
   - Projection Operations
   - Aggregation
   - Quantifier Operations
   - Partition Operations
   - Generation Operations
   - Set Operations
   - Conversions
   - Element Operators




Developer
Uses
L
I
N
Q
To Query
Object Collection
LINQ to Objects
ADO.Net Dataset
LINQ to DataSet
XML Document
LINQ to XML
Entity Framework
LINQ to Entities
SQL Database
LINQ to SQL
Other data sources..
By implementing IQueryable

LINQ Usage

Toepassingen van Linq

- **Linq to objects**

  Linq queries op collections/arrays van objects

- **Linq to XML**

  Queries on XML data en XML documents

- **Linq to DataSet**

  Toepassen van Linq queries op ADO.NET DataSet objecten

- **Linq to Entities**

  Linq queries voor ADO.Net Entity Framework API

- **Parallel Linq (PLINQ)**

  Parallelle verwerking van data die teruggegeven door een Linq query

Linq – Groepeer-operators

1. GroupBy
2. GroupBy x by y into z

Linq - groepeeroperators – 1. groupby

GroupBy

Voorbeeld:

```
class Dier {
      public string Name { get; set; }
      public double Leeftijd { get; set; }
}
List<Dier> dieren = new List<Dier> (){
new Dier { Name="Bobby", Leeftijd ge=8.3 },
new Dier { Name="Balou", Leeftijd =4.9 },
new Dier { Name="Wiske", Leeftijd =1.5 },
new Dier { Name="Dolly", Leeftijd =4.3 } };
var query = dieren.GroupBy( dier => Math.Floor(dier. Leeftijd), dier
=> dier. Leeftijd, (LeeftijdKey, leeftijden) => new { Key = LeeftijdKey,
Count = leeftijden.Count(), Min = leeftijden.Min(), Max =
leeftijden.Max() });
```

Linq - groepeeroperators – 2. Group x by y into z

Voorbeeld:

```
int[] getallen = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };
var getallenGroepen = from getal in getallen
   group getal by getal % 5 into groep
   select (Rest: groep.Key, Getallen: groep);

foreach (var groep in getallenGroepen){
      Console.WriteLine($"Getallen met rest
{groep.Rest} bij deling door 5:");
      foreach (var getal in groep. Getallen){
            Console.WriteLine(getal);
       }
}
```

Linq - Where

Where clause

Voorbeeld:

```
List<Product> products = GetProductList();


var expensiveInStockProducts =
from prod in products
where prod.UnitsInStock > 0 && prod.UnitPrice > 3.00M
select prod;

Console.WriteLine("In-stock products that cost more than 3.00:");
foreach (var product in expensiveInStockProducts)
{
    Console.WriteLine($"{product.ProductName} is in stock and
    costs more than 3.00.");
}
```

Linq – Join operator

1. **Inner join**
2. **Group join**

Linq – Join operator – inner join

Join

Voorbeeld:

```
 string[] categories = {
                    "Beverages",
                    "Condiments",
                    "Vegetables",
                    "Dairy Products",
                    "Seafood"};

List<Product> products = GetProductList();
var q = from c in categories
      join p in products on c equals p.Category
       select (Category: c, p.ProductName);
```

Linq – Join operator – group-join

Voorbeeld:

```
 string[] categories = {"Beverages","Condiments",
                        "Vegetables","Dairy Products",
                        "Seafood"};
List<Product> products = GetProductList();
var q = from c in categories
        join p in products on
        c equals p.Category into ps
       select (Category: c, Products: ps);

foreach (var v in q){
        Console.WriteLine(v.Category + ":");
        foreach (var p in v.Products){
         Console.WriteLine("    " + p.ProductName);} }
```

Linq – Sorteeroperatoren

1. Orderby …
2. Orderby …descending
3. Orderby…ThenBy
4. Reverse

Linq – sorteeroperators- 1.Orderby

Voorbeeld:

```
string[] woorden = { "kers", "appel", "banaan" };

var GesorteerdeWoorden = from woord in woorden
                         orderby woord.Length
                         select woord;

List<Product> products = GetProductList();
var sortedProducts = from prod in products
                           orderby prod.ProductName
                           select prod;
```

Linq – sorteeroperators-
2. Orderby …descending

Voorbeeld:

```
double[] doubles = { 1.7, 2.3, 1.9, 4.1, 2.9 };

var gesorteerdeDoubles = from d in doubles
                         orderby d descending
                         select d;

List<Product> products = GetProductList();
var sortedProducts = from prod in products
            orderby prod.UnitsInStock descending
            select prod;
```

Linq - sorteeroperatoren- 3. Orderby …ThenBy

```
string[] cijfers = { "nul", "een", "twee", "drie", "vier",
"vijf", "zes", "zeven", "acht", "negen" };
var gesorteerdOpLengte= from cijfer in cijfers
                        orderby cijfer.Length, cijfer
                        select cijfer;
 // Custom comparer voor sorteeroperator kan zelf gedefinieerd
//worden:
public class CaseInsensitiveComparer : IComparer<string>{
 public int Compare(string x, string y) =>
  string.Compare(x, y, StringComparison.OrdinalIgnoreCase);}
// gebruik van eigen Custom comparer:
string[] woorden = { "aPPEL", "AbAcUs", "bRaNcH", "BosBes",
"CiTRoEN", "keRSen" };

var gesorteerdeWoorden = woorden
                  .OrderBy(w => w.Length)
                  .ThenBy(w => w, new CaseInsensitiveComparer());
```

Linq – sorteer- operatoren- 4. Reverse

Voorbeeld:

```
 string[] cijfers = { "nul", "een", "twee", "drie",
"vier", "vijf", "zes", "zeven", "acht", "negen" };

var cijfersOmgekeerd = (
                  from cijfer in cijfers
                  where cijfers[1] == 'e'
                  select cijfers)
                  .Reverse();
 Console.WriteLine("Cijfers met tweede karakter =
‘e’ in omgekeerde volgorde:");
foreach (var c in cijfersOmgekeerd){
     Console.WriteLine(c);
}
```

Linq – Partitie operatoren

1. Take
2. Skip
3. TakeWhile
4. SkipWhile

Linq - partitie operators- 1.Take

Voorbeeld:

```
int[] getallen = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };
var eerste3 = getallen.Take(3);
Console.WriteLine("Eerste 3 getallen:");
foreach (var g in eerste3){
      Console.WriteLine(g);
}

List<Customer> customers = GetCustomerList();
var eerste2USbestellingen = ( from cust in customers
       from order in cust.Orders
       where cust.Region == "US"
      select (cust.CustomerID, order.OrderID,
      order.OrderDate)).Take(3);
```

Linq - partitie operators- 2. Skip

Voorbeeld:

```
int[] getallen = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };
var alleBehalveEerste4 = getallen.Skip(4);
Console.WriteLine("Alle behalve de eerste 4 getallen:");
foreach (var n in alleBehalveEerste4){
    Console.WriteLine(n);
}
List<Customer> customers = GetCustomerList();
var usOrders = from cust in customers
               from order in cust.Orders
               where cust.Region == "US"
               select (cust.CustomerID, order.OrderID,
               order.OrderDate);
var alleBehalveEerste2 = usOrders.Skip(2);
 Console.WriteLine("Alle bestellingen behalve de eerste 2
orders in US regio:");
foreach (var order in alleBehalveEerste2) {
                  Console.WriteLine(order); }
```

Linq - partitie operators- 3. TakeWhile

Voorbeeld:

```
 int[] getallen = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };
 Console.WriteLine("Eerste getallen Kleiner dan 6:");
 var eersteKleinerDan6 = getallen.TakeWhile(n => n < 6);


int[] getallen = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };
 Console.WriteLine("Eerste getallen niet Kleiner dan hun
positie(index):");

var eersteKleinerDan6 =
getallen.TakeWhile((n, index) => n >= index);
```

Linq - partitie operators- 4. SkipWhile

Voorbeeld:

```
int[] getallen = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };

Console.WriteLine("Alle elementen startend van het eerste
drievoud:");
var allButFirst3Numbers = getallen.SkipWhile(n => n % 3 != 0);

int[] getallen = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };
Console.WriteLine(" Alle elementen startend van het eerste dat
Kleiner is dan zijn positie(index):");
var laterNumbers = getallen.SkipWhile((n, index) => n >= index);
```

Linq – Quantificeerders

1. Any
2. All

Linq – Quantificeerder – 1. Any

Voorbeeld:

```
string[] woorden = { "eileider", "plezier", "reis" };
bool iNaE = woorden.Any(w => w.Contains("ei"));

List<Product> products = GetProductList();
var productGroups = from p in products
          group p by p.Category into g
          where g.Any(p => p.UnitsInStock == 0)
          select (Category: g.Key, Products: g);

foreach(var group in productGroups){
    Console.WriteLine(group.Category);
    foreach(var product in group.Products){
        Console.WriteLine($"\t{product}");
    }}
```

Linq – Quantificeerder – 2. All

Voorbeeld:

```
int[] getallen = { 1, 11, 3, 19, 41, 65, 19 };
bool enkelOnevenGetallen = getallen.All(n => n % 2 == 1);
Console.WriteLine($"De lijst bevat al de oneven getallen:
{enkelOnevenGetallen}");

List<Product> products = GetProductList();
var productGroups = from p in products
                                group p by p.Category into g
                                where g.All(p => p.UnitsInStock > 0)
                                select (Category: g.Key, Products: g);

foreach (var group in productGroups){
        Console.WriteLine(group.Category);
        foreach (var product in group.Products){
        Console.WriteLine($"\t{product}");
        }
}
```

Linq – Query Executie

1. **Uitgestelde (Deferred)**
2. **Onmiddellijke (Immediate) (Eager)**

Linq – Query executie - Uitgestelde

Voorbeeld:

```
int[] getallen = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };
int i = 0;
var q = from n in getallen
            select ++i;
// de locale variabele 'i’ wordt niet
//geïncrementeerd totdat elk element is
//geëvalueerd (hier in de foreach) :
foreach (var v in q)
{
     Console.WriteLine($"v = {v}, i = {i}");
}
```

Linq – Query executie - Onmiddellijke

Voorbeeld:

```
// Methoden zoals ToList(), ToArray() zorgen
//ervoor dat de query onmiddellijk wordt
//uitgevoerd en de restultaten worden gecached.
int[] getallen = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0
};
int i = 0;
var q = (from n in getallen
          select ++i).ToList();
// De locale variabele i wordt reeds volledig
//geïncrementeerd voordat we de resultaten
//itereren via foreach:
foreach (var v in q){
     Console.WriteLine($"v = {v}, i = {i}");
}
```

Lambda Expressions en LINQ

Vragen?

Referenties

- Telerik Software Academy
  - https://www.telerikacademy.com/

https://docs.microsoft.com/en-us/dotnet/api/system.linq?view=netcore-3.0