Fachhochschule Nordwestschweiz
Hochschule für Technik

LINQ & LAMBDAS

Martin Kropp, Yves Senn
University of Applied Sciences Northwestern Switzerland

Learning Targets

- You
  - can explain the purpose and concepts of Lambda expressions
  - know what closures are and their side effects
  - can write own lambda expressions
  - can explain the purpose and concepts of LINQ
  - can apply LINQ efficiently for programming

Content

- Lambdas
  - What is a Lambda?
  - How to use?
  - Execution Context
- LINQ Basics
  - Why?
  - LINQ Queries
  - LINQ Operators

Lambda expressions

Fachhochschule Nordwestschweiz
Hochschule für Technik

Just another way to write methods:

```
static bool SomePredicate(Point p)
{
    return p.X * p.Y > 100000;
}
Predicate<Point> d = SomePredicate;
```

```
Predicate<Point> d = delegate(Point p)
{
    return p.X * p.Y > 100000;
};
```

```
Predicate<Point> d = p => p.X * p.Y > 100000;
```

Lambda expressions

Fachhochschule Nordwestschweiz
Hochschule für Technik

```
Func<Point,bool> d = p => p.X * p.Y > 100000;
```

Type inferred automatically

```
Func<Point,bool> d = (Point p) => p.X * p.Y > 100000;
```

Manual typing

```
Func<Point,bool> d = p =>
{
    var r = p.X * p.Y;
    return r > 100000;
};
```

Complex functions

Lambda expressions


```
Func<Point,int,bool> d = (p, n) => p.X * p.Y > n;
```

Type inferred automatically

```
Func<Point,int,bool> d = (Point p, int n) => p.X * p.Y > n;
```

Manual typing

```
Func<Point,int,bool> d = (p, n) =>
{
    var r = p.X * p.Y;
    return r > n;
};
```

Complex functions

- Check back lesson two on switch expressions

Expression bodied functions


```
public class Car
{
    public string Model;
    public int Year;

    public override string ToString()
    {
        return $"{Year} {Model}";
    }
    /// or
    public override string ToString() => $"{Year} {Model}";
}
```

The “traditional” way

The lambda way

Expression bodied functions

Fachhochschule Nordwestschweiz
Hochschule für Technik

Any method can also be written as lambda expression:

```
public class Car
{
    //...

    public void Stop()
    {
        SetSpeed(0);
    }
    /// or
    public void Stop() => SetSpeed(0);
}
```

The “traditional” way

The lambda way

Properties the lambda way

Fachhochschule Nordwestschweiz
Hochschule für Technik

Any **readonly** property can be written as lambda expression:

```
public class Car
{
    private string brand;

    public string Brand {
      get { return brand; }
    }
    /// or
    public string Brand => brand;
}
```

The “traditional” way

The lambda way

More about Expression Bodies

- On Operators

```
public static Complex
                operator +(Complex a, Complex b) => new
                            Complex(a.Re + b.Re, a.Im + b.Im);

public static implicit
        operator string(Person p) => "${p.First} {p.Last}";
```

- On Properties and Indexers

```
public string Name => First + " " + Last;

public Customer this[long id] => store.LookupCustomer(id);
```

For getter-only properties and indexers

Closures

A lambda expression may use variables defined outside its context («outer variables»):

Variable declared in the method context

Lambda uses this variable

```
void doSomething()
{
    var factor = 2;
    Func<int, int> multiplier = n => n * factor;

    Console.WriteLine(multiplier(3)); //6
}
```

See Jon Skeets https://csharpindepth.com/Articles/Closures

Caveat: Closures


```
var actions = new List<Action>();

for (var i = 0; i < 10; i++)
    actions.Add( () => Console.WriteLine(i) );

foreach (var action in actions)
    action();
```

What is the result?

Explanation

```
() => Console.WriteLine(i)
```

writes the **current** value of `i` and **not** the value of `i` back when the delegate was created. When the `Action` runs, the last value assigned to `i` was 10.

→ Closures close over variables, not over values.

Solution

Declare an new “inner variable" for each iteration to be captured, instead of a single “outer variable" which is captured only once.

```
//...
for (var i = 0; i < 10; i++)
{
  var j = i; //← each iteration gets its own,
                 new variable j
  actions.Add(() => Console.WriteLine(j));
}
//...
```

Worksheet – Part 1

Language INtegrated Query

- Functions to transform sequences:

```
string[] names = {"Hans", "Peter", "Hanspeter", "Heiri"};

// filter
var longNames = names.Where(n => n.Length > 5);

// reorder
var orderedByLength = names.OrderBy(n => n.Length);
```

- Implemented as **Extension Methods**:

```
using System.Linq;
```

Why LINQ?

Fachhochschule Nordwestschweiz
Hochschule für Technik

```
if (array.Length == 0)
{
   throw new InvalidOperationException();
}

var max = float.NegativeInfinity;
for(var x=0; x<array.Length; x++)
{
   if(array[x].Value > max)
   {
      max = array[x].Value;
   }
}
```

```
var max = array.Max(i => i.Value);
```

More LINQ examples


```
var min = array.Min();
var avg = array.Average();
var sum = array.Sum();

var filtered = array.Where(i => i > 55);

var last = array.Last();

if(array.Any(i => i % 2 == 0))
  //…
```

Some LINQ operators

`x.Select(a => a*2)`

- Returns a list where the numbers are doubled

`x.Distinct()`

- Returns a list of unique numbers in x

`x.Count()`

- Count the elements in list x

`x.Sum()`

- Sum of the numbers in x

`x.First()`

- Returns the first number in x

`x.OrderBy(a => Math.Abs(a))`

- Returns a copy of list x, orderd by |a|

Implementation Details:
http://referencesource.microsoft.com/#System.Core/System/Linq/Enumerable.cs

Caveat: Deferred execution

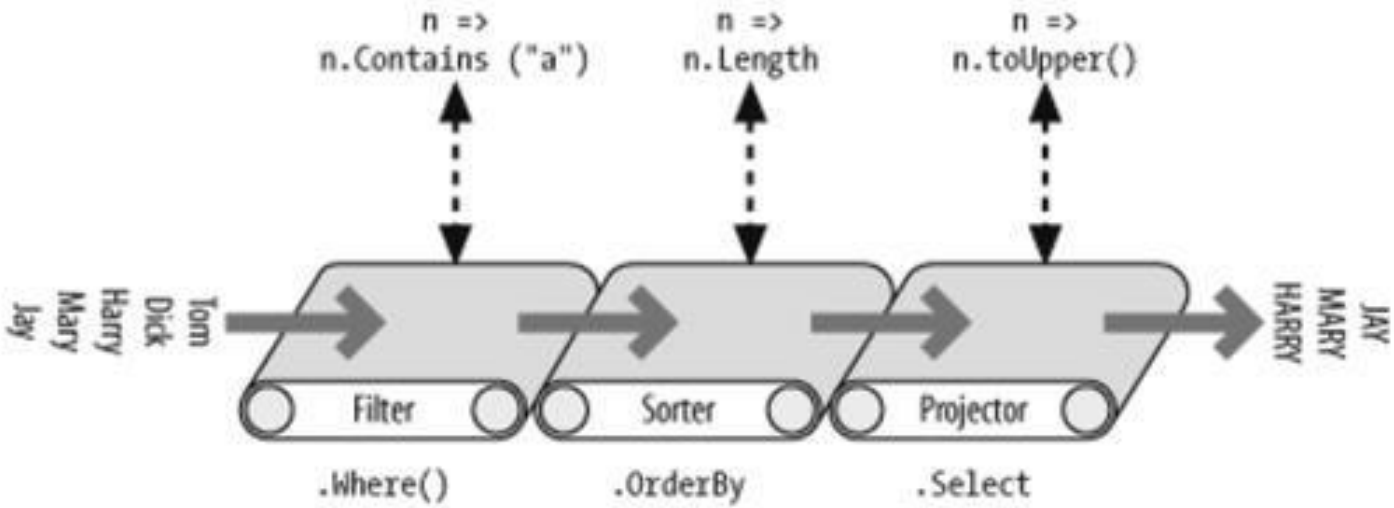
LINQ is executed when **results are accessed**, not when the query is created.

Execution happens when…

- Iterating over the LINQ results (e.g. with `foreach`)
- Calling immediate execution methods:
  - `.ToList()`, `.ToArray()`, `.ToDictionary()`, …
  - `.Count()`, `.Average()`, `.First()`, `.Sum()`, …

Chaining LINQ operators

```
var u = users.Where(n => n.Contains("a"))
             .OrderBy(n => n.Length)
             .Select(n => n.ToUpper())
```



More LINQ Examples


```
var x = array.Single(i => i.FirstName == "Peter");

var x = list.OrderBy(i => i.Name)
            .ThenBy(i => i.FirstName)
            .Take(10)
            .Average(i => i.Mass / i.Height / i.Height);

var x = table.Select(i => i.FirstName.Trim())
             .Sum(i => i.Length);
```

```
//.ForEach is *NOT LINQ*
//persons.ForEach(p => Console.WriteLine(p.FirstName));
```

More LINQ Examples

Fachhochschule Nordwestschweiz
Hochschule für Technik

```
var x = persons
    .Except(actors)
    .Select(p => new
        {
            Name = $"{p.FirstName} {p.LastName}",
            p.Age
        })
    .Distinct()
    .OrderBy(p => p.Age);
```

Makes use of anonymous types

Anonymous Types

```
var inferredName = 7;
var a = new
{
    someKey = "someValue",
    otherKey = 7,
    inferredName
};

Console.WriteLine($"{a.someKey}
                {a.inferredName}");
```

No class name

LINQ examples - Exercise


```
IEnumerable<Order> orders;

class Order
{
    public DateTime OrderDate;
    public string Customer;
    public IEnumerable<Article> OrderedArticles;
}
```

<u>Formulate the following LINQ queries</u>

1. Who was the first-ever customer?
2. Total number of ordered articles
3. Number of customers?
4. Number of different articles

More on LINQ operators

- Filtering
- Projecting
- Joining
- Ordering
- Grouping
- Set operators
- Conversion methods
- Element operators
- Aggregation methods
- Quantifiers
- Generation methods

https://code.msdn.microsoft.com/101-LINQ-Samples-3fb9811b

LINQ language features summary

Type inference

Lambda expressions

Operator overloading

```
var contacts =
    customers
    .Where(c => c.City == "Windisch")
    .Select(c => new { c.Name, c.Phone });
```

Extension methods

Anonymous types

Object initializers

LINQ guidelines

- Think functional
  - ***NEVER modify*** state
  - Use immutable objects

- Don’t repeat queries unnecessarily
  Store results in arrays or lists

- Don’t store arbitrarily large results
  How many results can there be? How much memory will be required? Do I have to cache?

Worksheet, Part 2