The Missing LINQ

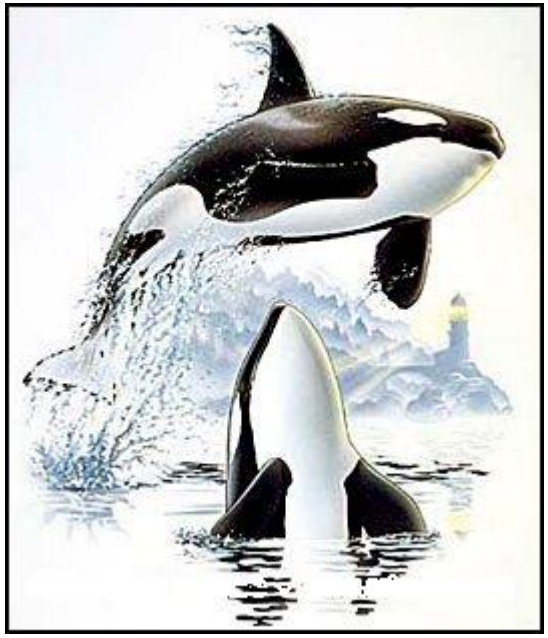
- og relaterede features

C# version

Henrik Lykke Nielsen / Captator
www.captator.dk
Softwarearkitekt, Microsoft Regional Director for Denmark
lykke@captator.dk
+45 2237 3311
twitter.com/dothenrik
dk.linkedin.com/in/henriklykkenielsen

Agenda

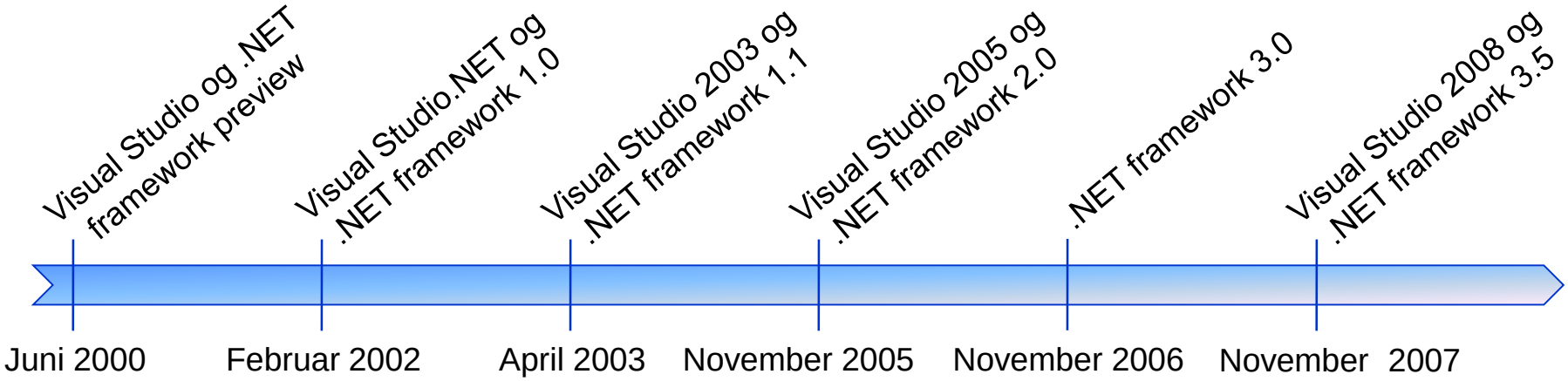
Captator

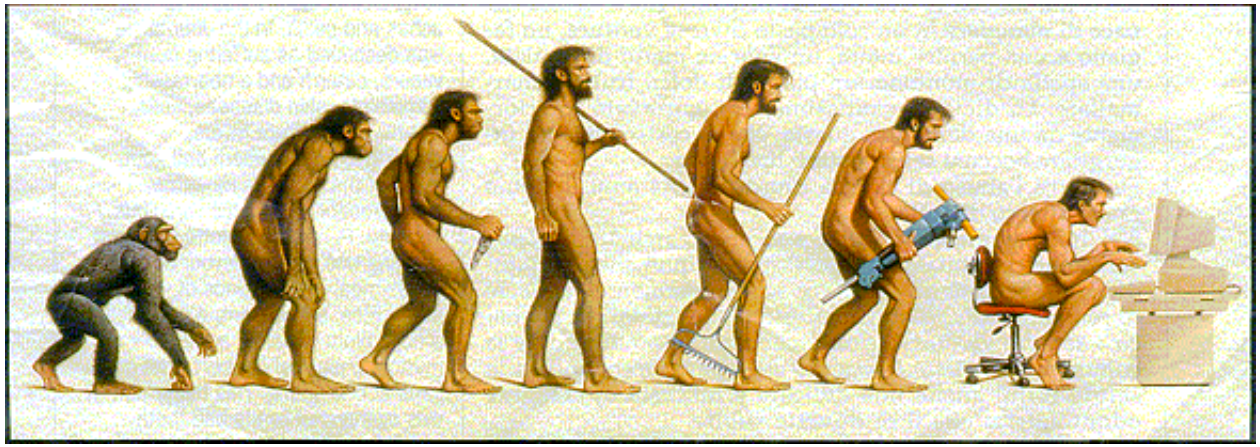


- **Sprog features**
  - Objekt og collection initializers
  - Implicit typede variable og arrays
  - Anonyme typer
  - Anonyme metoder, lambda udtryk
  - Extension metoder
- **The Missing LINQ**
  - Standard query operatorer, query expressions
  - LINQ to Objects
  - LINQ to DataSets
  - LINQ to databaser
  - LINQ to XML

Visual Studio og .NET frameworket

- **En platform i udvikling**





Objekt initializers

Captator

- **Kan direkte sætte properties og fields ved instansiering**

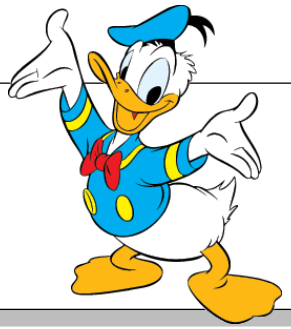
```
class Person {
  public string Navn;
  public string CprNr;
  public string Adresse { get { ... } set { ... }
}
```

- **Klassen skal i dette første eksempel have en public, parameterløs konstruktør (paranteserne kan udelades)**

```
Person p = new Person() { Navn="Anders", Adresse="Andeby" };
```

- **Objekt initializeren svarer til:**

```
Person temp = new Person();
temp.Navn = "Anders";
temp.Adresse = "Andeby”;
p = temp;
```



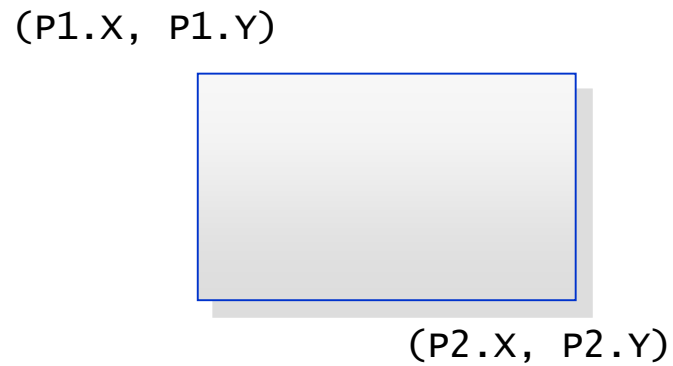
- **Kan kombineres med konstruktørparametre**

```
Person2 p2 = new Person2("Andersine") { Adresse = "Andeby" };
```

Objekt initializers

- **Objekt initializers kan nestes**

```
class Point
{
  public int X;
  public int Y;
}

class Rectangle
{
  public Point P1;
  public Point P2;
}
```




```
Point a = new Point { X=7, Y=42 };
Point b = new Point { X=17, Y=117 };
Rectangle r1 = new Rectangle { P1=a, P2=b };
int right1X = r1.P2.x;

Rectangle r2 = new Rectangle { P1 = new Point { X=10, Y=100 },
                               P2 = new Point { X=20, Y=200 }};
int right2X = r2.P2.X;
```

Collection initializers

Captator

- **Allerede eksisterende array initialisering**

```
int[] primesArray = new int[] { 2, 3, 5, 7, 11, 13, 17 };
```

- **Collectionen skal implementere System.Collections.Generic.ICollection<T>**
  - Kalder Add(T)

```
List<int> primtal = new List<int> { 2, 3, 5, 7, 11, 13, 17 };

List<Person> ænder = new List<Person> {
  new Person {Navn = "Anders And", Adresse = "Andeby"},
  new Person {Navn = "Andersine And", Adresse = "Andeby"},
  new Person {Navn = "Fætter Vims", Adresse = "Andeby"} };

System.Collections.ArrayList primesArrayList =
  new System.Collections.ArrayList { 2, 3, 5, 7, 11, 13, 17 };

var værdier = new Dictionary<int, string>
  {{0, "Første"}, {1, "Anden"}}
```

Implicitly typed local variables

Captator

- **Typen kan udledes af kompileren**

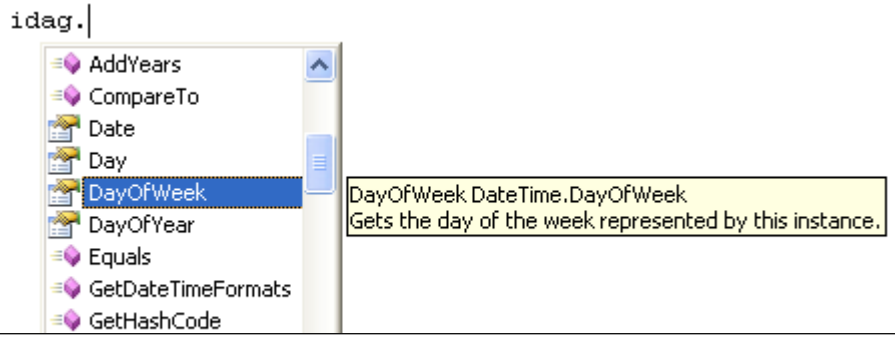
```
var pi = 3.14;
var navn = "Anders And";
var idag = System.DateTime.Today;

var tal = new int[] {10, 4, 17, 42};
var duck = new Person {Navn="Anders", Adresse="Andeby", Alder=71 };
```

Double

String

DateTime

- **Statisk typet!**
- **Fuld intellisense**
- **Kompilercheck**




```
System.Collections.Generic.List<Person> personer =
    new System.Collections.Generic.List<Person>();

var personer = new System.Collections.Generic.List<Person>();
```

Implicitly typed arrays

Captator

- **Arrayets type fastlægges ud fra elementerne**

```
var a = new[] { 2, 4, 17, 42 };                    // Integer-array
var b = new[] { 1, 1.5, 2, 2.5 };                  // Double-array
var c = new[] { "hello", null, "world" };          // String-array

var d = new[] { 1, "one", 2, "two" };

var f = new[] { (object) 1, "one", 2, "two" };     // Object-array
```

✘ Compiler fejl

- **Alle elementer skal implicit kunne castes til typen**
- **Kan kombineres med fx anonyme objekt initializers**

```
var anonymeÆnder = new[]
  {
    new {Navn="Anders And", Adresse="Andeby"},
    new {Navn="Andersine And", Adresse="Andeby"},
    new {Navn="Fætter Vims", Adresse="Andeby"}
  };

int andTal = anonymeÆnder.Length;
string andeNavn = anonymeÆnder[1].Navn;
```

Implicitly typed arrays

- **Kan udlede for-variables type**

```
foreach (var x in new[] { 2, 4, 17, 42 })        // Integer-array
{
  // ...
}
```

```
var a = new[] { 2, 4, 17, 42 };                  // Integer-array

// ...

foreach (var x in a)
{
  // ...
}
```

Anonyme typer

Captator

- **En type behøver ikke at have noget (kendt) navn**

```
var b1 = new { Titel=".NET for hackere", Pris = 17.42 };
var b2 = new { Titel=".NET for plattenslagere", Pris = 42.17 };

b2 = b1;
var b3 = b2;
string bogensTitel = b1.Titel;

MessageBox.Show("Første bogs titel = " + b1.Titel);
MessageBox.Show("Tredje bogs titel = " + b3.Titel);
```

```
Person p = new Person()
 { Navn="Anders", Adresse="Andeby", Alder=71, CprNr="123456-7890" };

var addressInfo1 = new { p.Navn, p.Adresse };
var addressInfo2 = new { Navn = p.Navn, Adresse = p.Adresse };
var addressInfo3 = new { Name = p.Navn, Address = p.Adresse };

addressInfo2 = addressInfo1;

addressInfo3 = addressInfo2;
```

✘ Compiler fejl

Anonyme typer – casting by example

Captator

- **Følgende funktion returnerer en anonym type**

```
public object ReturnAnonymous()
{
  var anonymTypeInstans = new { Navn = "Anders And", Adresse = "Andeby" };
  return anonymTypeInstans;
}
```

- **Inferens af generiske typer gør, at ved kald af "Cast" kan typen T fastlægges ud fra parameteren "type"**

```
public T Cast<T>(object obj, T type)
{
  return (T)obj;
}
```

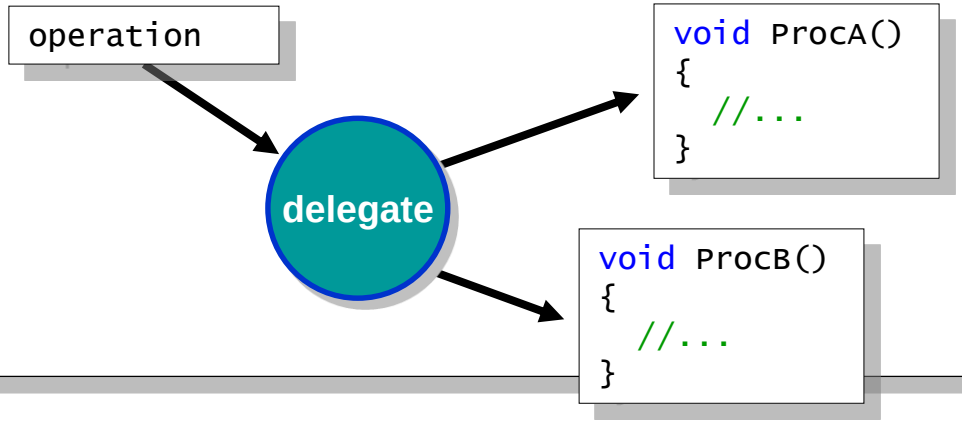
- **Parameteren "type" til "Cast" metoden angives som et eksempel på den anonyme type, der castes til**

```
object obj = ReturnAnonymous();

var typed = Cast(obj, new { Navn = "", Adresse = "" });

MessageBox.Show(typed.Navn + " - " + typed.Adresse, "Data fra funktion");
```

Delegates

```
private delegate int MathOperation(int a, int b);

private int Add(int a, int b) {
  return a + b;
}

private int Subtract(int a, int b) {
  return a - b;
}

private int Multiply(int a, int b) {
  return a * b;
}

private void Calculate(MathOperation operation, int a, int b) {
  int result = operation(a, b);
}
```

```
MathOperation operation;

if (radAdd.Checked)
  operation = new MathOperation(Add);
else if (radSubtract.Checked)
  operation = (MathOperation) Subtract;
else if (radMultiply.Checked)
  operation = Multiply;

Calculate(operation, 42, 17);
```



Anonyme metoder

- **Delegates kan erstattes med inline kodeblokke**

```
delegate [(parameter-list)] { anonymous-method-block }
```

- **Benytter delegate inferens**
- **Kan bruges hvor og som delegates kan bruges**
  - EventHandlers, callback-delegates
  - Eksplicit assignment, delegate-parameter i metodekald
- **Kode blokken kan undlade eller medtage delegate-typens parameterliste efter behov**
  - Undladt parameterliste er forskellig fra den tomme ( )
  - Returtypen og en eventuel parameterliste skal være kompatibel med delegatetypen
- **Kan ”capture” omkringliggende metodes variable**
  - Forlænget levetid af disse lokale variable

Anonyme metoder

```
private delegate int MinDelegateType(int a, int b);
```

```
MinDelegateType delCalc;

delCalc = delegate { return 17 + 42; };
MessageBox.Show("17 + 42 = " + delCalc(2, 2).ToString());

delCalc = delegate(int x, int y) { return x + y; };
MessageBox.Show("2 + 2 = " + delCalc(2, 2).ToString());

int j = 42;
delCalc = delegate(int x, int y) { return x + j; };
MessageBox.Show("2 + 42 = " + delCalc(2, 2).ToString());
```

```
private int i = 119;
private void btnAnonymeMetoder_Click(object sender, EventArgs e)
{
  MinDelegateType delCalc;

  int k = 7;
  delCalc = delegate(int x, int y) { return i / k + y; };
  MessageBox.Show("119 / 7 + 2 = " + delCalc(2, 2).ToString());
}
```

Lamba udtryk

Captator

- **Komprimerede anonyme metoder**

```
private delegate int MinDelegateType(int a);

private void ShowResult(MinDelegateType del, int i)
{
  MessageBox.Show(del(i).ToString());
}
```

```
MinDelegateType delCalc = delegate(int x) { return x + 42; };
ShowResult(delCalc, 17);

ShowResult(delegate(int x) { return x + 42; }, 18);

ShowResult((int x) => x + 42, 19);

ShowResult(x => x + 42, 20);
```

x er eksplicit typet

x er implicit typet

- **Parametre til lambda udtryk kan være eksplicit eller implicit typede**

Lamba udtryk

Captator

- **Predefinerede generiske delegate typer: Func**
- **Ligger i System namespacet (System.Core.dll)**

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

- **Eksempel – f3 svarer til f2 svarer til f1:**

```
private int NavnGivetFunktion(int x)
{
  return x + 42;
}
```

```
System.Func<int, int> f1 = NavnGivetFunktion;

System.Func<int, int> f2 = delegate(int x) { return x + 42; };
System.Func<int, int> f3 = (x => x + 42);
```

- **Action delegate typerne svarer til Func blot "void"**

Lamba udtryk

- **Predefineret generisk predikats-funktion**

```
delegate bool Predicate<T>(T obj)
```

- **Eksempel:**

```
System.Predicate<int> selector = ( x => x > 0);

if (selector(42))
{
  MessageBox.Show("Tallet kommer med.");
}
else
{
  MessageBox.Show("Tallet kommer ikke med.");
}
```

- **Expression trees kan repræsentere lambda udtryk som data**

Extension metoder

Captator

- **Kan udvide udvalgte typer med ekstra metoder**
- **Lav en extension metode**
  - Statisk metode i statisk klasse (C#), metode i module (VB)
  - Det første argument til metoden udpeger typen, som extension metode virker på
- **Importer namespacet (using (C#) / Imports (VB))**
  - Extension metoderne i det importerede namespace kan nu kaldes
- **Metoden kaldes som en instansmetode**
  - Kan naturligvis også kaldes via typen som normalt
- **Kan være almindelige såvel som generiske metoder**
- **Instansmetoder ”vinder over” extension metoder**

Extension metoder - eksempel

Captator

```
Namespace SuperExtensions
{
  public static class UserInterface
  {
    public static void Show(this string s)
    {
      System.Windows.Forms.MessageBox.Show(s);
    }
  }
}
```

this angiver objektet metoden kaldes på

- **I C# angiver ”this” som parameter-modifier, at dette er en extension metode**

```
Using SuperExtensions;
```

```
SuperExtensions.UserInterface.Show("Hej statiske verden");

string str = "Hej verden";
str.Show();
```

Et problem … der måske eksisterer

```
string sql = "SELECT * FROM Authors";

var cmd = new System.Data.SqlClient.SqlCommand(sql, conn);
var adap = new System.Data.SqlClient.SqlDataAdapter(cmd);
var tbl = new System.Data.DataTable();
adap.Fill(tbl);

string navn = (string) tbl.Rows[0]["Name"];
```

- **Query, parametre og resultat er ikke stærkt typet**
  - Queryen er kun en streng
  - Parametre er svagt typede
  - Resultatet (f.eks. DataTable) er en collection af svagt typede objekter
- **Query sproget er typisk koblet til databaser**
  - Ikke-trivielt at lave querysprog

LINQ arkitektur

Captator

C#

VB

Andre sprog …

.NET Language Integrated Query (LINQ)

Data sources tilgængelige fra LINQ

LINQ enabled ADO.NET

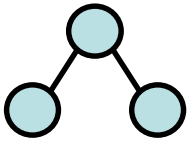
LINQ to Objects

LINQ to DataSets

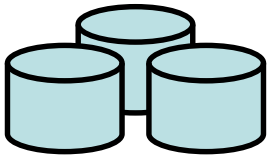
LINQ to SQL

LINQ to Entities

LINQ to XML



Objekter

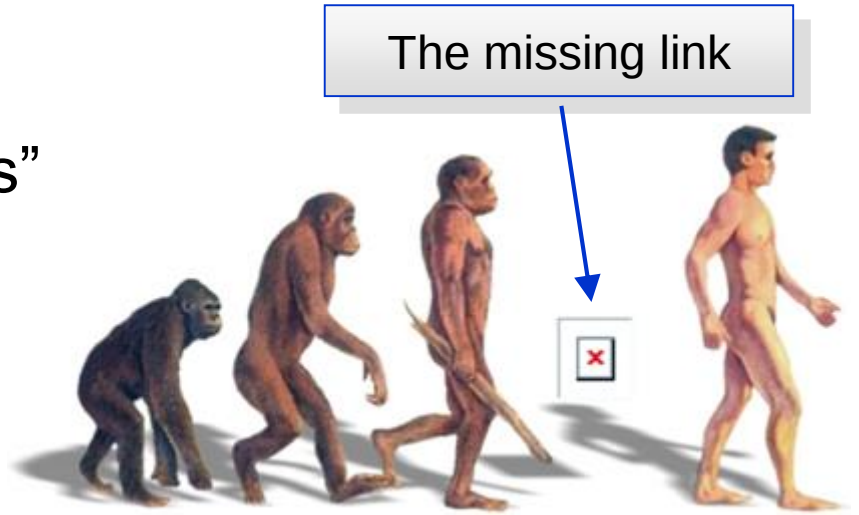


Relationel

XML

The Missing LINQ

- **LINQ danner bro mellem .NET sprog og ”LINQ to *”**
- **Generel query facilitet i .NET frameworket**
  - Definerer en række standard query operatorer
    - Gennemløb, filtrering, projektion, ...
  - Et API til at query ethvert .NET array eller collection
  - Opererer på og returnerer IEnumerable<T> (C#) / IEnumerable(Of T) (VB) objekter (inklusive arrays)
  - Metoderne eksekveres først ved gennemløb
  - De fleste metoder kan ”pipes”
  - Koden kan modulariseres
- **Deklarativ model**



Standard operatorer

- **Extension metoder defineret i System.Linq.Enumerable klassen**
- **De nuværende standard operatorer:**
  - Aggregate, All, Any, AsEnumerable, Average, Cast, Concat, Contains, Count, DefaultIfEmpty, Distinct, ElementAt, ElementAtOrDefault, Empty, Except, First, FirstOrDefault, GroupBy, GroupJoin, Intersect, Join, Last, LastOrDefault, LongCount, Max, Min, OfType, OrderBy OrderByDescending, Range, Repeat, Reverse, Select, SelectMany, SequenceEqual, Single, SingleOrDefault, Skip, SkipWhile, Sum, Take, TakeWhile, ThenBy, ThenByDescending, ToArray, ToDictionary, ToList, ToLookup, Union, Where

Standard query operatorer - Where

Captator

- **Filtrerer en sekvens udfra et predikat**

Extension metode

```
public static IEnumerable<T> Where<T>
                (this IEnumerable<T> source, Func<T,bool> predicate)
```

```
List<Person> ænder = new List<Person> {
    new Person {Navn = "Fætter Vims", Adresse = "Andeby"},
    new Person {Navn = "Andersine And", Adresse = "Andeby"},
    new Person {Navn = "Anders And", Adresse = "Andeby"} };

var res1 = System.Linq.Enumerable.Where(ænder,
    (obj => obj.Navn.StartsWith("Anders")));

foreach (Person obj in res1)
{
  MessageBox.Show(obj.Navn, "Kaldt som statisk metode");
}

var res2 = ænder.Where(obj => obj.Navn.StartsWith("Anders"));
foreach (Person obj in res2)
{
  MessageBox.Show(obj.Navn, "Kaldt som instans metode");
}
```

LINQ query expressions

Captator

```
List<Person> ænder = new List<Person> {
    new Person {Navn="Fætter Vims", Adresse="Andeby", Alder = 34},
    new Person ...} };
```

- **Sprogunderstøttelse for query udtryk**

```
var andeborgere = from duck in ænder
                  where duck.Adresse == "Andeby"
                  orderby duck.Alder, duck.Navn
                  select new {duck.Alder, duck.Navn};
```

Eksekveres som

```
var andeborgere = ænder.Where(duck => duck.Adresse == "Andeby")
                       .OrderBy(duck => duck.Alder)
                       .ThenBy(duck => duck.Navn)
                       .Select(duck => new {duck.Alder, duck.Navn});
```

```
foreach (var duck in andeborgere) {
  MessageBox.Show(duck.Navn + " " + duck.Alder);
}
```

- **Fleksible projektioner**

```
select new { duck.Navn, Født = System.DateTime.Now.Year-duck.Alder };
```

LINQ query operatorer

- **Clauses indbygget i VB (keywords)**
  - From, Select, Where, Order By, Join, Group By, Group Join, Aggregate, Let, Distinct, Skip, Skip While, Take, Take While
  - En query skal begynde med enten en *From* eller en *Aggregate* clause
- **Clauses indbygget i C# (keywords)**
  - from, where, select, group, into, orderby, join, let
  - En query skal begynde med en *From* clause
- **Øvrige LINQ operatorer kan kaldes som extension metoder (placeret I klasserne Enumerable og Queryable i System.Linq namespacet)**

LINQ query expressions

Captator

- **Når man definerer en LINQ query opbygges et expression tree**
- **Deferred (udskudt) eksekvering**
  - Normalt evalueres queryen først, når elementerne i resultatet tilgås (i eksempelvis en For Each-løkke)
  - Gør det muligt at kombinere queries
- **Immediate (øjeblikkelig) eksekvering**
  - Metoderne Count, Sum, Any etc. returnerer beregnet værdi
  - Metoderne First, Last, Single etc. returnerer et enkelt element
  - Metoderne ToList, ToArray etc. returnerer mængder - kan bruges til caching

LINQ to Objects

Captator

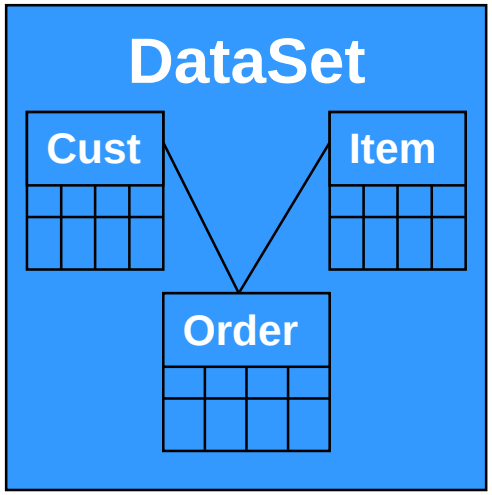
```
string[] dirPaths = Directory.GetDirectories(@"C:\CaptatorVss\Eifos2");

var foundDirs = from dirPath in dirPaths
                let dir = new System.IO.DirectoryInfo(dirPath)
                let fileCount = dir.GetDirectories().Count()
                where dir.Name.StartsWith("C") && fileCount > 3
                select dir;

foreach (var directory in foundDirs) {
  txtData.Text += directory.Name + "\r\n";
}
```

```
string[] dirPaths = Directory.GetDirectories(@"C:\CaptatorVss\Eifos2");

var foundDirs1 = from dirPath in dirPaths
                 let dir = new System.IO.DirectoryInfo(dirPath)
                 where dir.Name.StartsWith("C")
                select dir;

var foundDirs2 = from dirPath in foundDirs1
                 let fileCount = dir.GetDirectories().Count()
                 where fileCount > 3
                 select dir;

foreach (var directory in foundDirs2) {
  txtData.Text += directory.Name + "\r\n";
}
```

Queries kan kombineres, hvilket letter genbrug mærkbart

LINQ to DataSet

- DataSets er disconnectede datastrukturer
- Har en struktur der minder om relationelle databaser
- Standard LINQ syntax understøttelse
- Filtrering, projektion, joins
- Kan kombineres med andre in-memory datastrukturer
- Understøtter aggregering



LINQ to DataSet

Captator

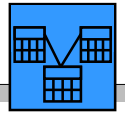
- **DataSets indeholder svagt typede data**
- **De svagt typede data skal gøres typestærke**
  - Kald System.Data.DataTableExtensions.AsEnumerable extension-metoden på DataTable-objekter

```
public static System.Data.EnumerableRowCollection<DataRow>
    AsEnumerable(this System.Data.DataTable source)
```

  - System.Data.DataRowExtensions.Field extension-metoderne (en del overloads) giver typestærk tilgang til de enkelte felter

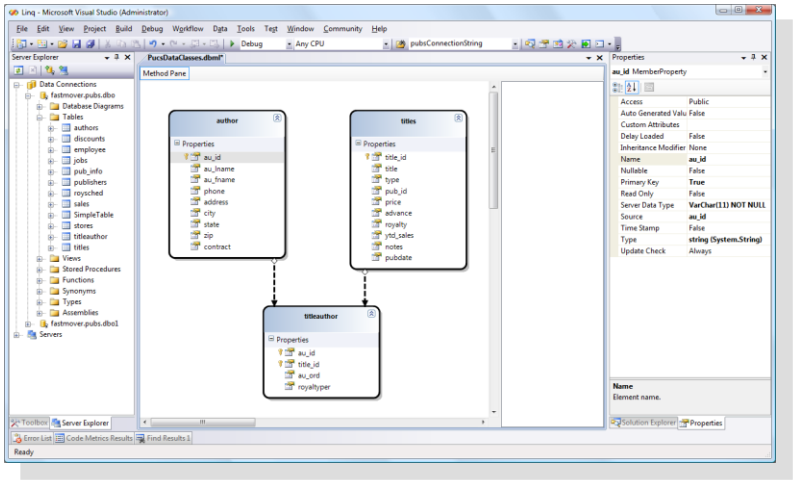
```
public static T Field<T>(this System.Data.DataRow row, string columnName,
    System.Data.DataRowVersion version)
```

```
var kundeNavne =
    from kunde in _ds.Tables["Customer"].AsEnumerable()
    where kunde.Field<string>("City") == "London"
    select new { CompanyName = kunde.Field<string>("CompanyName") };
```



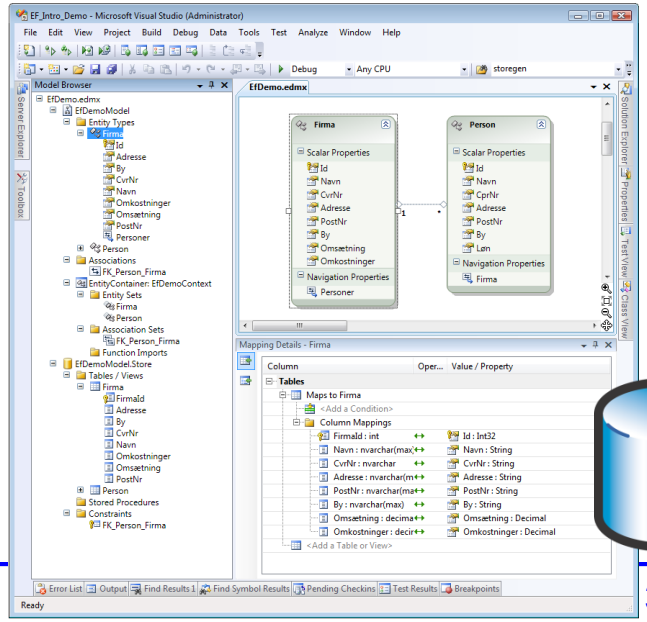
LINQ to SQL

- Supporterer Microsoft SQL Server
- En simpel mappingmekanisme mellem relationel database og objektmodel
- Tager udgangspunkt i en én-til-én mapping mellem SQL Server database og objektmodel
- Mappinger kan foretages via attributter eller XML fil
- Designer til Visual Studio
- Udnytter deferred execution



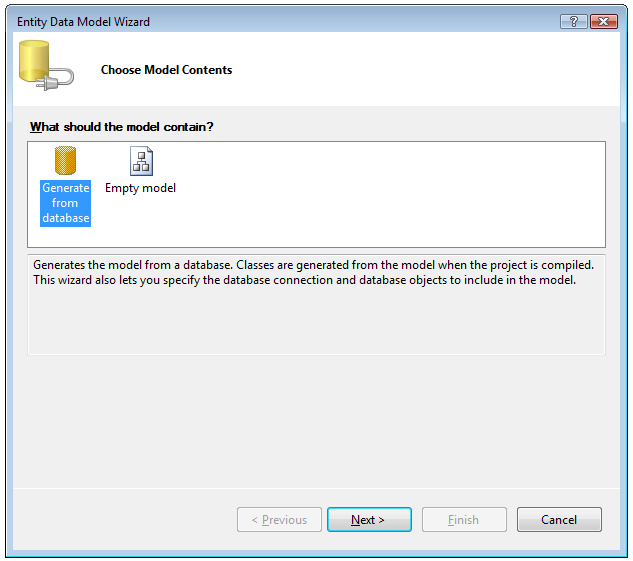
Entity frameworket og LINQ to Entities

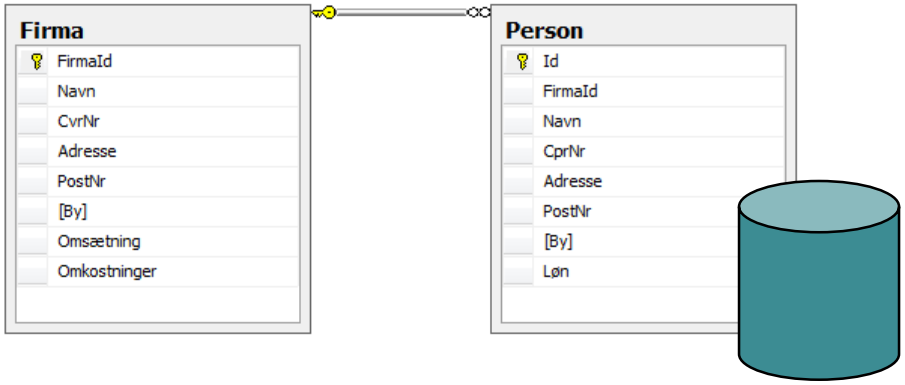
- **Database support via providermodel**
- **Entity Data Model (EDM)**
  - SSDL - Storage Schema Definition Language
    - Beskriver hvordan databasen ser ud
  - CSDL - Conceptual Schema Definition Language)
    - Definerer entiteter der anvendes i applikationen
  - MSL - Mapping Specification Language
    - Mapper SSDL laget til CSDL laget
- **Designer til Visual Studio**
- **Udnytter deferred execution**

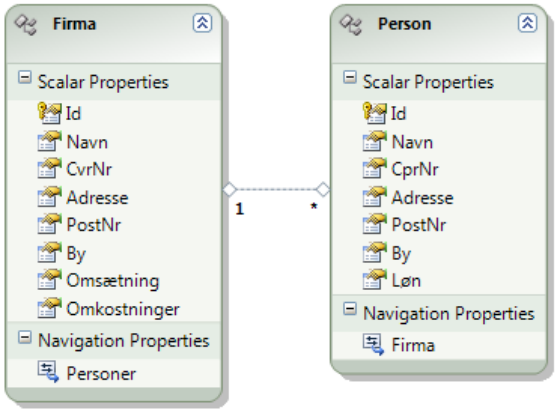


Træk og slip en Entity Framework model

- **Ud fra et database schema genereres en model og en række klasser**








```
var context = new EfDemoContext();

var firmaQuery = from f in context.Firma
                 select f;
Person and =
    firmaQuery.First().Personer.First(i=>i.Navn.StartsWith("And"));

string andensNavn = and.Navn;
```

LINQ to XML

- **System.Linq.Xml namespacet indeholder XML klasser til brug for LINQ to XML**
  - XML typer: XElement, XAttribute, …
  - System.Linq.Xml.Extensions indeholder extension metoder til brug for LINQ queries
- **Opbygning af XML**
- **Forespørgsler på og transformation af XML**
- **Manipulation af XML**

LINQ to XML

Captator

- **Læsning og transformation**

Source xml

```
<CustomerList>
  <Customer>
    <First>Anders</First>
    <Last>And</Last>
  </Customer>
  <Customer>
    <First>Joakim</First>
    <Last>von And</Last>
  </Customer>
</CustomerList>
```

Target xml

```
<Contacts>
  <Contact>
    <FirstName>Anders</FirstName>
    <LastName>And</LastName>
  </Contact>
  <Contact>
    <FirstName>Joakim</FirstName>
    <LastName>von And</LastName>
  </Contact>
</Contacts>
```

```
XElement kundeListe = XElement.Parse(xml);

XElement kontakter = new XElement("Contacts",
    from c in kundeListe.Elements("Customer")
    select new XElement("Contact",
    new XElement("FirstName", (string) c.Element("First")),
    new XElement("LastName", (string) c.Element("Last")) ));
```

LINQ to XML

Captator

- **Opbygning af XML vha. funktionel konstruktion**

```
System.IO.DirectoryInfo[] foundDirs = ...

System.Xml.Linq.XElement snippet =
    new System.Xml.Linq.XElement("directories",
        from dir in foundDirs
        select new System.Xml.Linq.XElement("directory",
            new System.Xml.Linq.XAttribute("fileCount",
                dir.GetDirectories().Count().ToString()),
            new System.Xml.Linq.XElement("fullName", dir.FullName)));

txtData.Text = snippet.ToString();
```

LINQ to XML i VB

Captator

- **Opbygning af XML vha. funktionel konstruktion**

```
Dim foundDirs As System.IO.DirectoryInfo() = ...

Dim snippet As System.Xml.Linq.XElement = _
    New System.Xml.Linq.XElement("directories", _
        From dir In foundDirs _
        Select New System.Xml.Linq.XElement("directory", _
            New System.Xml.Linq.XAttribute("fileCount", _
                dir.GetDirectories().Count().ToString()), _
            New System.Xml.Linq.XElement("fullName", dir.FullName)))

txtData.Text = snippet.ToString()
```

- **Opbygning af XML vha. VBs integrerede syntaks**

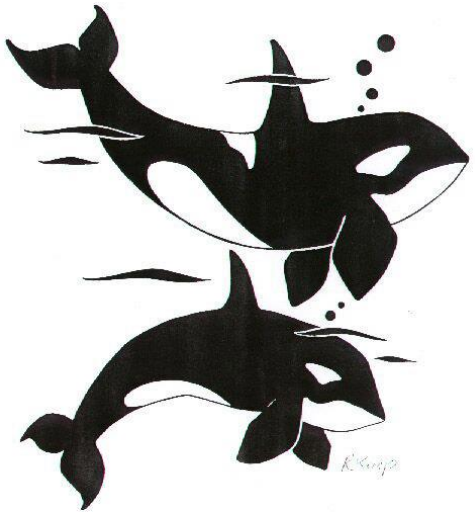
```
Dim foundDirs = ...

Dim snippet As System.Xml.Linq.XElement = _
<directories>
  <%= From dir In foundDirs _
    Select <directory fileCount=<%= dir.GetDirectories().Count().ToString() %>>
             <fullName><%= dir.FullName %></fullName>
           </directory> %>
</directories>

txtData.Text = snippet.ToString()
```

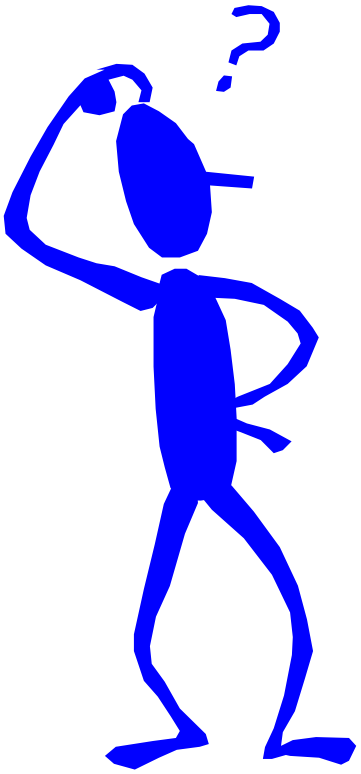
Indsætning af værdier

XML literals og latebinding i VB

```
Dim kundeliste As XElement = <CustomerList>
                               <Customer type="Privat">
                                 <First>Anders</First>
                                 <Last>And</Last>
                               </Customer>
                               <Customer type="Erhverv">
                                 <First>Joakim</First>
                                 <Last>von And</Last>
                               </Customer>
                             </CustomerList>

For Each Dim kunde In kundeliste.Customer
  MessageBox.Show(CStr(kunde.@type))
Next

Dim elementName = "vare"
Dim price = 42
Dim snippet As XElement = _
    <<%= elementName %>><pris><%= price %></pris></>

MessageBox.Show(snippet.ToString())
```

Spørgsmål





www.captator.dk

kurser, rådgivning og softwareudvikling