

# Module 14

## "LINQ to Objects"



**TEKNOLOGISK**  
**INSTITUT**

# Agenda

- ▶ **Introducing LINQ**
- ▶ LINQ Query Keywords
- ▶ LINQ Query Operator Methods
- ▶ Lab 14
- ▶ Extra: LINQ to Entities
- ▶ Extra: LINQ to XML
- ▶ Discussion and Review

# Motivation for LINQ

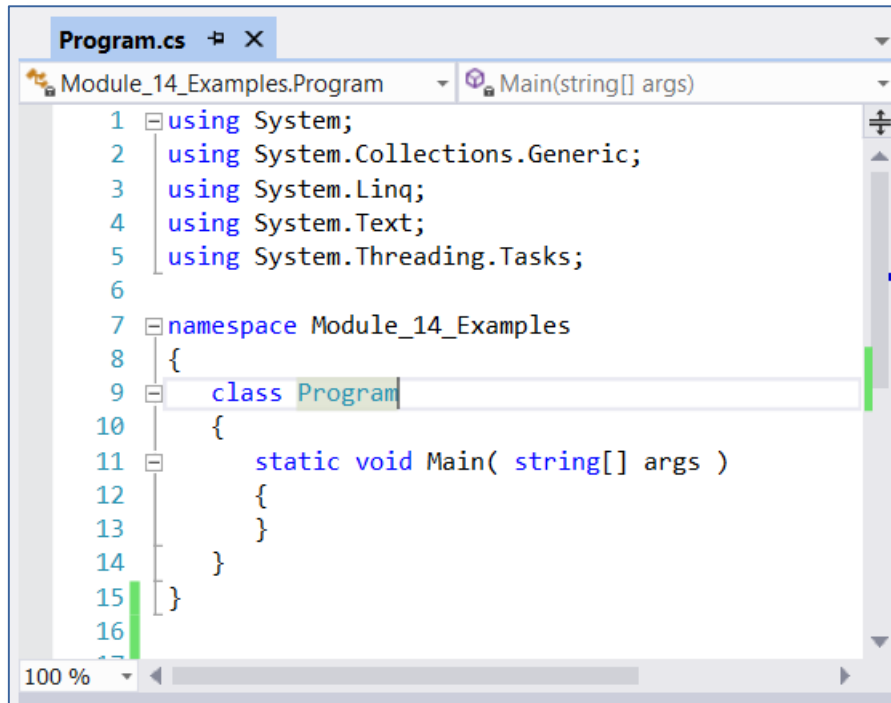
- ▶ LINQ = Language **IN**tegrated **Q**uery
  
- ▶ Several distinct motivations for LINQ
  - Uniform programming model for any kind of data
  - A better tool for embedding SQL queries into type-safe code
  - Another data abstraction layer
  - ...
  
- ▶ All of these descriptions to some extent hold true

# LINQ Components

- ▶ LINQ to Objects
  - ▶ LINQ to XML
  - ▶ LINQ to Entities
  - ▶ Parallel LINQ
  - ▶ ...
- 
- ▶ Later we will see a little bit of
    - LINQ to Entities
    - LINQ to XML

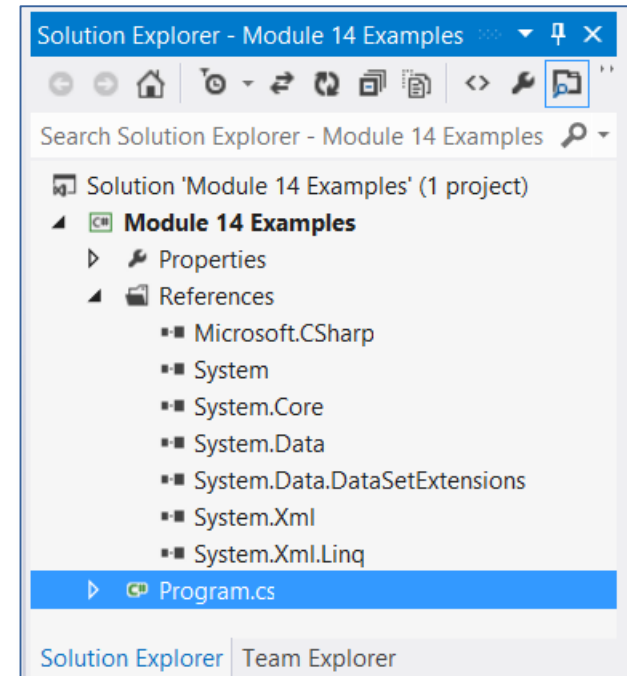
# Starting LINQ to Objects

- ▶ Main LINQ features live in **System.Core.dll** in the **System.Linq** namespace



```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace Module_14_Examples
8 {
9     class Program
10     {
11         static void Main( string[] args )
12         {
13         }
14     }
15 }
16
17
```

The screenshot shows a C# code editor with a file named Program.cs. The code defines a namespace Module\_14\_Examples containing a class Program with a static Main method. The Main method is currently empty. The editor includes a toolbar at the top with icons for file operations and a status bar at the bottom showing 100% zoom.



# A First Example

- ▶ Find all games with more than 18 characters in the title

```
string[] wiiGames = {  
    "Super Mario Galaxy",  
    "FIFA 09",  
    "Guitar Hero III",  
    "Wii Sports",  
    "Wii Fit",  
    "Legend of Zelda: Twilight Princess"  
};
```

```
IEnumerable<string> query = from g in wiiGames  
                             where g.Length >= 18  
                             select g;
```

```
foreach( string s in query )  
{  
    Console.WriteLine( s );  
}
```



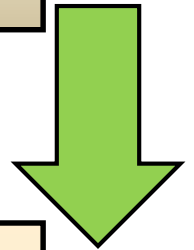
# Implicitly Typed Variables

- ▶ Query results can be of a multitude of types

```
int[] numbers = {10, 20, 30, 40, 1, 2, 3, 8};  
IEnumerable<int> query = from i in numbers  
                        where i < 10 select i;  
foreach( int i in query )  
{  
    Console.WriteLine( i );  
}
```

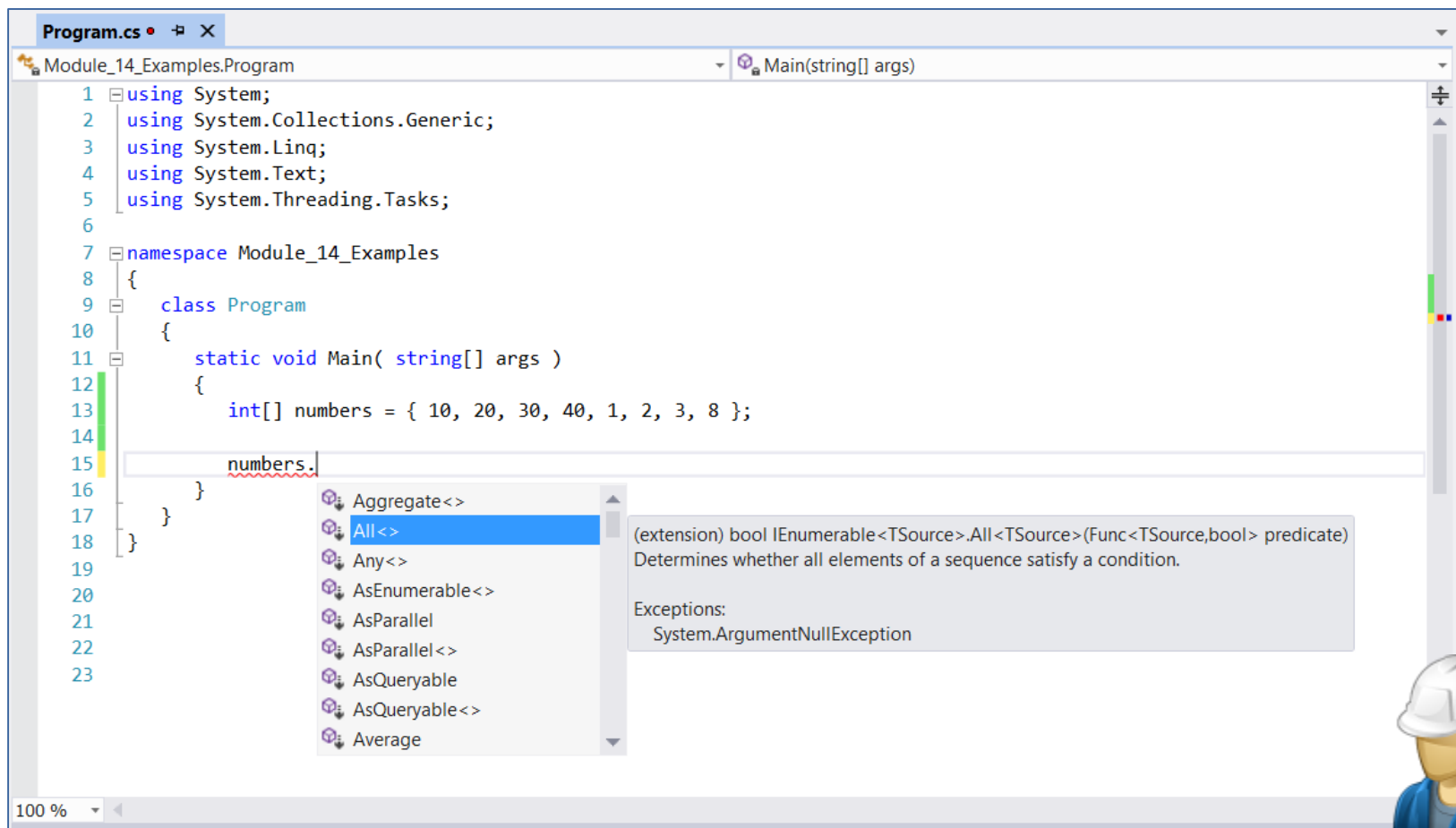
- ▶ Innocently-looking modifications might change underlying type
- ▶ Make all query variables implicitly typed...!

```
int[] numbers = {10, 20, 30, 40, 1, 2, 3, 8};  
var query = from i in numbers where i < 10 select i;  
foreach( var i in query )  
{  
    Console.WriteLine( i );  
}
```



# Enumerable Extension Methods

- ▶ The **System.Linq.Enumerable** class provides a lot of extension methods



The screenshot shows a Visual Studio editor window with a file named **Program.cs**. The code defines a namespace **Module\_14\_Examples** and a class **Program** with a static **Main** method. Inside **Main**, an array of integers is declared: `int[] numbers = { 10, 20, 30, 40, 1, 2, 3, 8 };`. The cursor is positioned at the end of the `numbers` array, and an IntelliSense dropdown menu is displayed, listing various extension methods from **System.Linq.Enumerable**. The **All<>** method is currently selected. A tooltip for **All<>** is visible, providing its signature and description.

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace Module_14_Examples
8  {
9      class Program
10     {
11         static void Main( string[] args )
12         {
13             int[] numbers = { 10, 20, 30, 40, 1, 2, 3, 8 };
14
15             numbers.
16         }
17     }
18 }

```

IntelliSense Dropdown:

- Aggregate<>
- All<>**
- Any<>
- AsEnumerable<>
- AsParallel
- AsParallel<>
- AsQueryable
- AsQueryable<>
- Average

Tooltip for **All<>**:

(extension) bool IEnumerable<TSource>.All<TSource>(Func<TSource,bool> predicate)  
Determines whether all elements of a sequence satisfy a condition.

Exceptions:  
System.ArgumentNullException





# Deferred Execution

- ▶ Query expressions are not evaluated until they're enumerated!
- ▶ This is called *Deferred Execution*

```
int[] numbers = { 10, 20, 30, 40, 0, 1, 2, 3, 8 };  
var query = from i in numbers where i < 10 select 87 / i;  
  
foreach( var i in query )  
{  
    Console.WriteLine( i );  
}
```

- ▶ You can force evaluation through the Visual Studio debugger
  - Use the Results View of the query variable



# Immediate Execution

- ▶ You can force evaluation by using conversion extension methods

```
int[] numbers = { 10, 20, 30, 40, 0, 1, 2, 3, 8 };  
var query = from i in numbers where i < 10 select i;  
  
int[] intNumbers = query.ToArray();  
List<int> listNumbers = query.ToList();
```

- ▶ There are other such extension methods, e.g.
  - **ToDictionary<T,K>**



# LINQ and Generic Collections

- ▶ LINQ can query data in various members of `System.Collections.Generic`

```
Stack<int> stack = new Stack<int>( new int[]{ 42, 87, 112, 255 } );  
var query = from i in stack where i < 100 select i;
```

```
List<Car> cars = new List<Car>() {  
    new Car{ PetName="Henry", Color="Silver", Speed=100, Make="VW" },  
    new Car{ PetName="Daisy", Color="Tan", Speed=90, Make="BMW" },  
    new Car{ PetName="Mary", Color="Black", Speed=55, Make="VW" },  
    new Car{ PetName="Clunker", Color="Rust", Speed=5, Make="Yugo" },  
    new Car{ PetName="Melvin", Color="White", Speed=43, Make="Ford" }  
};
```

```
var query = from c in cars  
    where c.Speed > 90 && c.Make == "BMW"  
    select c;
```



# LINQ and Nongeneric Collections

- ▶ Nongeneric collections lack the **IEnumerable<T>** infrastructure for querying
- ▶ This can be provided using the **OfType<T>** extension method

```
ArrayList cars = new ArrayList() {  
    new Car{ PetName="Henry", Color="Silver", Speed=100, Make="BMW" },  
    new Car{ PetName="Daisy", Color="Tan", Speed=90, Make="BMW" },  
    new Car{ PetName="Mary", Color="Black", Speed=55, Make="VW" },  
    new Car{ PetName="Clunker", Color="Rust", Speed=5, Make="Yugo" },  
    new Car{ PetName="Melvin", Color="White", Speed=43, Make="Ford" }  
};
```

```
IEnumerable<Car> enumerableCars = cars.OfType<Car>();  
var query = from c in enumerableCars  
            where c.Speed > 90 && c.Make == "BMW"  
            select c;
```



# LINQ and Custom Collections

- ▶ LINQ queries can be performed directly on any **IEnumerable<T>** type
  - Even your own types!

```
class Garage : IEnumerable<Car>
{
    ...
}
```

```
Garage g = new Garage();
var query = from c in garage
            where c.PetName.StartsWith( "F" )
            select c;

foreach (var c in query)
{
    Console.WriteLine( c.PetName );
}
```



# Agenda

- ▶ Introducing LINQ
- ▶ **LINQ Query Keywords**
- ▶ LINQ Query Operator Methods
- ▶ Lab 14
- ▶ Extra: LINQ to Entities
- ▶ Extra: LINQ to XML
- ▶ Discussion and Review

# The **from** Clause

- ▶ Range variables and data source are specified in the **from** clause

```
Stack<int> stack = new Stack<int>( new int[]{ 42, 87, 112, 255} );  
var query = from i in stack where i < 10 select i;
```

- ▶ It can define the type of the range variable as well

```
ArrayList cars = new ArrayList {  
    new Car{ PetName="Henry", Color="Silver", Speed=100, Make="BMW" },  
    ...  
};  
var query = from Car c in cars  
            where c.Speed > 90 && c.Make == "BMW"  
            select c;
```

- ▶ Can in fact have multiple **from** clauses...



# The **where** Clause

- ▶ Filtering conditions are specified by a boolean expression in a **where** clause

```
List<Car> cars = new List<Car> {  
    new Car{ PetName="Henry", Color="Silver", Speed=100, Make="BMW" },  
    ...  
};  
var query = from c in cars  
             where c.Speed > 90 && c.Make == "BMW"  
             select c;
```

```
var query = from c in cars  
             where c.Speed > 90  
             where SomePredicate( c )  
             select c;
```

- ▶ Can have multiple **where** clauses also





# The **select** Clause

- ▶ Projections of results are done through the **select** clause

```
List<Car> cars = new List<Car> {  
    new Car{ PetName="Henry", Color="Silver", Speed=100, Make="BMW" },  
    ...  
};  
var query = from c in cars  
             where c.Speed > 90 && c.Make == "BMW"  
             select c.Make;
```

```
var query = from c in cars  
             where c.Speed > 90 && c.Make == "BMW"  
             select new { c.Make, c.Color };
```

- ▶ Projections can create new (anonymous) data types



# The **orderby** Clause

- ▶ Results can be sorted using the **orderby** clause

```
List<Car> cars = new List<Car> {  
    new Car{ PetName="Henry", Color="Silver", Speed=100, Make="BMW" },  
    ...  
};  
var query = from c in cars  
            where c.Speed >= 55  
            orderby c.PetName  
            select c;
```

- ▶ The order can be **ascending** (the default) or **descending**

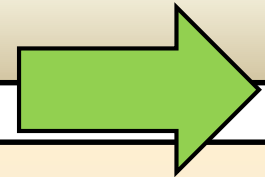
```
var query = from c in cars  
            where c.Speed >= 55  
            orderby c.PetName descending, c.Color  
            select c;
```



# Query Operators Resolution

- ▶ These query operators are keywords with syntax highlighting and IntelliSense
- ▶ But they are resolved as extension methods in the **Enumerable** class

```
var query = from g in wiiGames
             where g.Length >= 18
             orderby g.Length, g
             select g.ToUpper();
```



```
var query = wiiGames.Where( g => g.Length >= 18 )
                    .OrderBy( g => g.Length )
                    .ThenBy( g => g )
                    .Select( g => g.ToUpper() );
```

- ▶ You can use either syntax or use delegates instead of anonymous methods etc.

# Agenda

- ▶ Introducing LINQ
- ▶ LINQ Query Keywords
- ▶ **LINQ Query Operator Methods**
- ▶ Lab 14
- ▶ Extra: LINQ to Entities
- ▶ Extra: LINQ to XML
- ▶ Discussion and Review

# Count<T>

- ▶ You can compute the number of items in the result set with **Count<T>**

```
string[] wiiGames = {  
    "Super Mario Galaxy",  
    "FIFA 09",  
    "Guitar Hero III",  
    "Wii Sports",  
    "Wii Fit",  
    "Legend of Zelda: Twilight Princess"  
};  
var query = from g in wiiGames  
            where g.Length >= 18  
            select g;  
Console.WriteLine( "{0} games match the query", query.Count() );
```

- ▶ This forces an evaluation of the query expression!



# Set Operations: **Except<T>**

- ▶ Differences between queries can be computed with **Except<T>**

```
string[] wiiGames = {  
    "Super Mario Galaxy", ...  
};  
string[] xbox360Games = {  
    "Halo", ...  
};  
  
var query = ( from ... where ... select ... ).Except(  
    ( from ... where ... select ... );  
var query2 = wiiGames.Except( xbox360Games );
```

- ▶ Do you think this will evaluate the query expression? ☺
- ▶ **Union<T>**, **Intersect<T>**, and **Except<T>** constitute the set operations (**Distinct<T>** is also helpful!)



# Singleton Operations

- ▶ A single element can be retrieved from a query result

- `First<T>`
- `Last<T>`
- `Single<T>`

```
var query = wiiGames.Intersect( xbox360Games );  
  
var first = query.First();  
var last = query.Last();  
var theOnlyOne = query.Single();  
  
Console.WriteLine( first );  
Console.WriteLine( last );  
Console.WriteLine( theOnlyOne );
```

- ▶ Each of these has an `...OrDefault<T>` version
  - `FirstOrDefault<T>`
  - `LastOrDefault<T>`
  - `SingleOrDefault<T>`



# Partitioning Operators

- ▶ **Take()** and **Skip()**

```
string[] wiiGames = {  
    "Super Mario Galaxy", ...  
};  
string[] xbox360Games = {  
    "Halo", ...  
};  
  
var query1 = wiiGames.Union( xbox360Games ).Take( 7 );  
var query2 = wiiGames.Union( xbox360Games ).Skip( 3 );
```

- ▶ There are also
  - **TakeWhile()**
  - **SkipWhile()**







# Lab 14: Creating LINQ Queries



# Agenda

- ▶ Introducing LINQ
- ▶ LINQ Query Keywords
- ▶ LINQ Query Operator Methods
- ▶ Lab 14
- ▶ **Extra: LINQ to Entities**
- ▶ Extra: LINQ to XML
- ▶ Discussion and Review

# ADO.NET Entity Framework

- ▶ The de-facto standard for disconnected data access providing
  - Entity Data Models (EDM)
  - Entity SQL
  - Object Services
- ▶ It supports
  - Writing code against a conceptual model
  - Type-safe data access
  - Robustness and indepedance across storage systems
  - Maintainability
- ▶ Tools and wizards supporting
  - Database-first design
  - Code-first design



# Querying and Updating Data

- ▶ Using LINQ to Entities to query data

```
using( ShopEntities entities = new ShopEntities() )  
{  
    var query = from c in entities.Customers  
                 where c.Orders.Count > 0  
                 select c;  
  
    ...  
}
```

- ▶ DbContext-generated class
  - keeps tracks of updates
  - saves back to database

```
using( ShopEntities entities = ... )  
{  
    ...  
    entities.SaveChanges();  
}
```



# Customizing Classes

- ▶ Never modify the auto-generated classes!!
  - Instead, augment the auto-generated partial classes

```
public partial class Customer
{
    public string FullName => $"{FirstName} {LastName}";

    public int Age
    {
        get { return ...; }
    }
}
```



# Agenda

- ▶ Introducing LINQ
- ▶ LINQ Query Keywords
- ▶ LINQ Query Operator Methods
- ▶ Lab 14
- ▶ Extra: LINQ to Entities
- ▶ **Extra: LINQ to XML**
- ▶ Discussion and Review

# Extra: LINQ to XML Example

- ▶ LINQ to XML once again uses same keywords and expressions

```

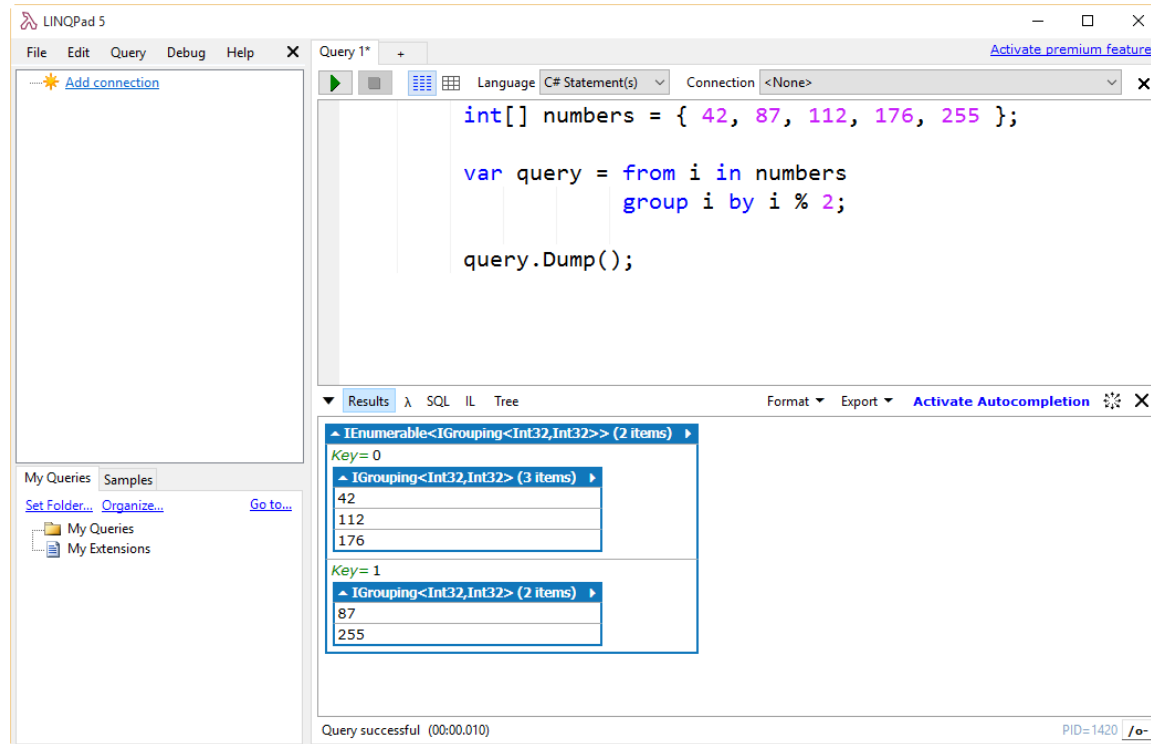
Movies.xml
<?xml version="1.0" encoding="utf-8" ?>
<Movies xmlns="">
  <Movie Type="Comedy">
    <Title>The Hangover</Title>
    <Tagline>Some guys just can't handle Vegas.</Tagline>
  </Movie>
  <Movie Type="Comedy">
    <Title>Forgetting Sarah Marshall</Title>
    <Tagline>From the guys who brought you "The 40-Year-Old Virgin" and "Knocked Up".</Tagline>
  </Movie>
  <Movie Type="Action">
    <Title>The Matrix</Title>
    <Tagline>Free your mind.</Tagline>
  </Movie>
  <Movie Type="Thriller">
    <Title>Shutter Island</Title>
    <Tagline>Someone is missing.</Tagline>
  </Movie>
</Movies>

```



# LINQPad

- ▶ LINQPad by Joseph Albahari is indispensable!



- ▶ Get it from <http://www.linqpad.net>





# Discussion and Review

- ▶ Introducing LINQ
- ▶ LINQ Query Keywords
- ▶ LINQ Query Operator Methods
- ▶ Extra: LINQ to Entities
- ▶ Extra: LINQ to XML



WINCUBATE

***Jesper Gulmann Henriksen***

PhD, MCT, MCSD, MCPD

Phone : +45 22 12 36 31

Email : [jgh@wincubate.net](mailto:jgh@wincubate.net)

WWW : <http://www.wincubate.net>

Hasselvangelen 243

8355 Solbjerg

Denmark