SOFTWARE GUILD

LINQ
(Language Integrated Query)

.NET Cohort

Coding Bootcamp

SOFTWARE GUILD

Lesson Goals

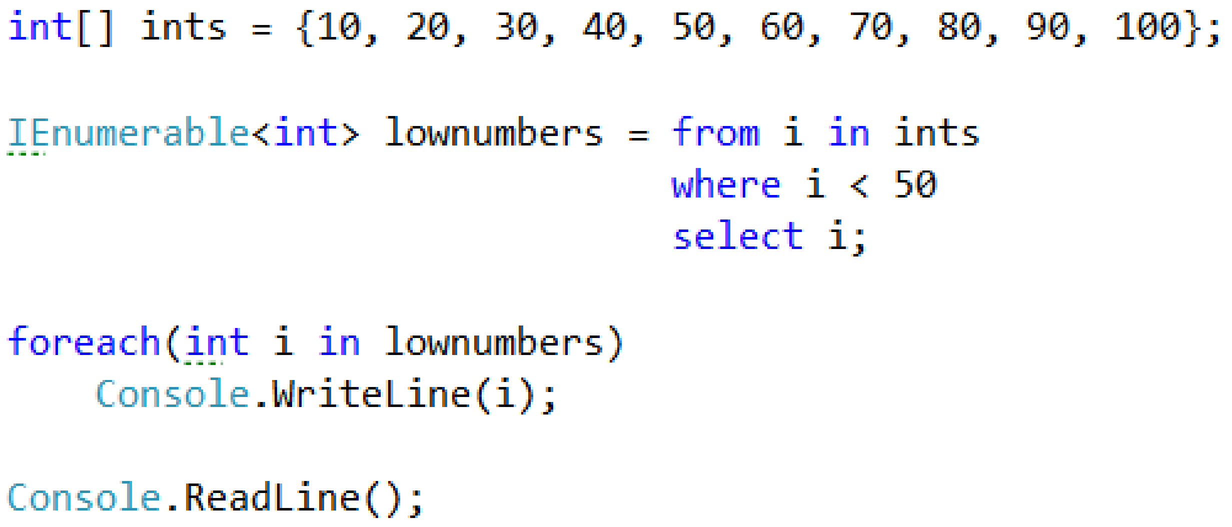
- We will learn why LINQ is awesome
- We will learn query syntax
- Also, we will show how delegates and lambdas can make your LINQ statements more concise

What is LINQ?

- LINQ stands for “Language Integrated Query.”
- It is an extension of the .NET framework that allows you to query collections (arrays, lists, etc.) in a manner similar to SQL.
- You can also use LINQ to query databases, XML Documents, and just about any other set that implements it.
- You can use Query syntax or Method syntax.

LINQ and IEnumerable

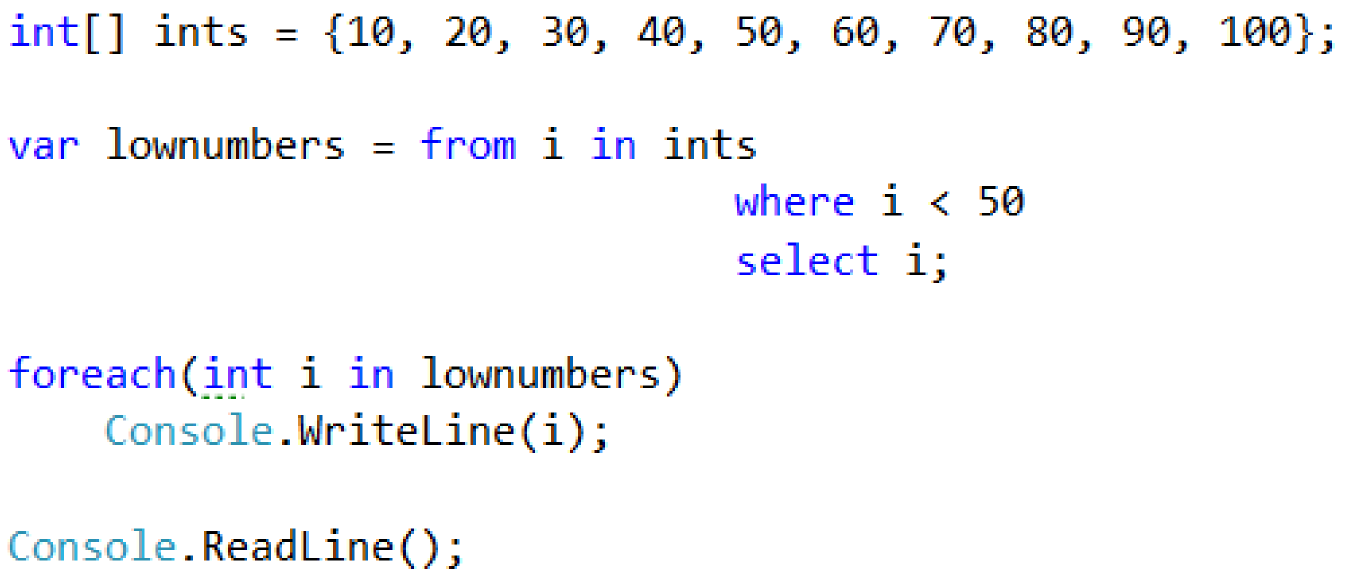
- LINQ queries always return an enumeration of results (in the case of no matches, it will return an empty set).



```
int[] ints = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};

IEnumerable<int> lownumbers = from i in ints
                              where i < 50
                              select i;

foreach(int i in lownumbers)
    Console.WriteLine(i);

Console.ReadLine();
```

This is Where var Comes in

- The *var* keyword is commonly used in LINQ, especially when we return anonymous objects.



```
int[] ints = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};

var lownumbers = from i in ints
                              where i < 50
                              select i;

foreach(int i in lownumbers)
    Console.WriteLine(i);

Console.ReadLine();
```

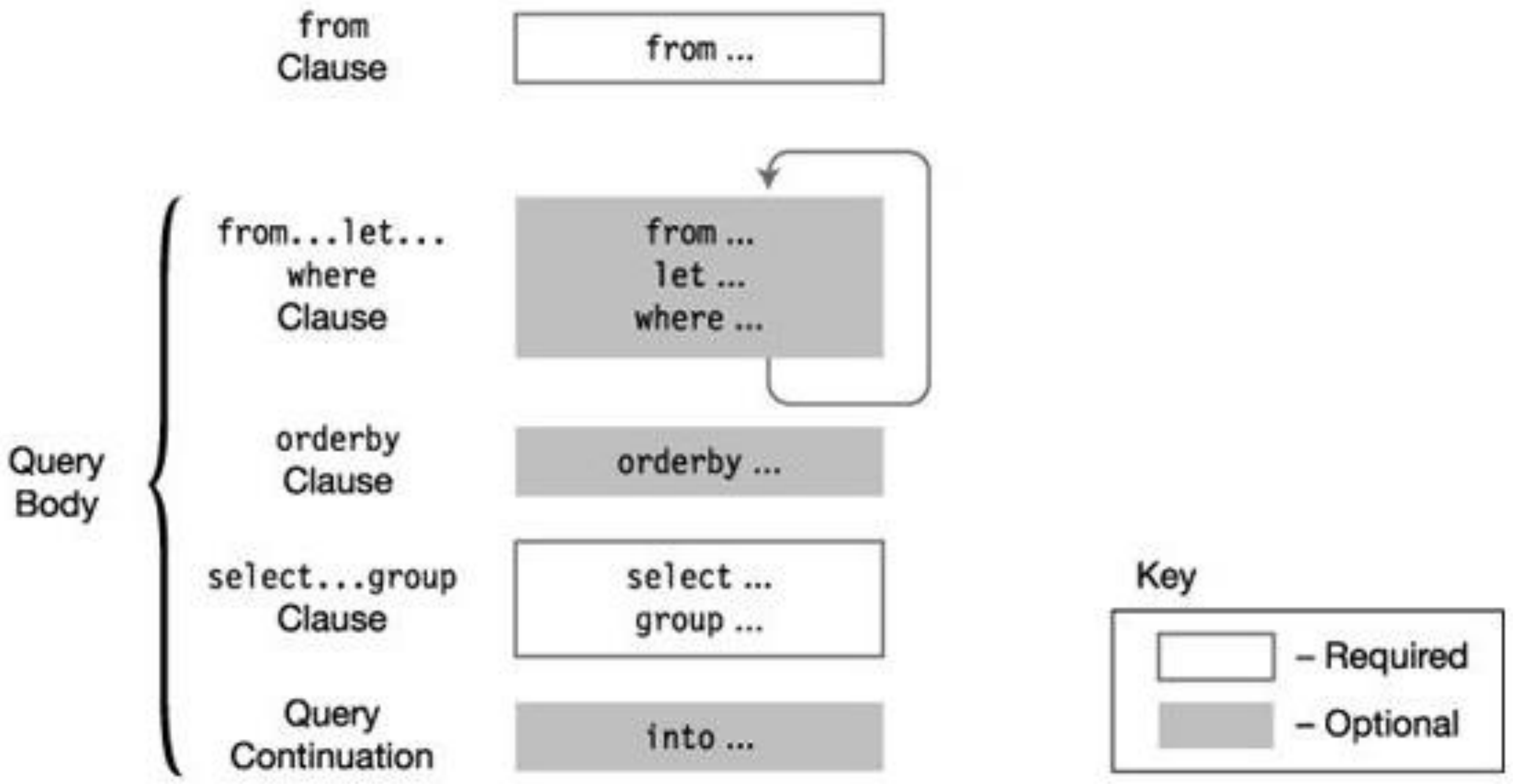
Anonymous Types

- Sometimes in LINQ we want to create an object “on the fly” that contains a subset of properties or combinations thereof.
- In this case, we must use the var keyword because a type doesn’t exist until the statement is executed.

LINQ Anonymous Types

DEMO

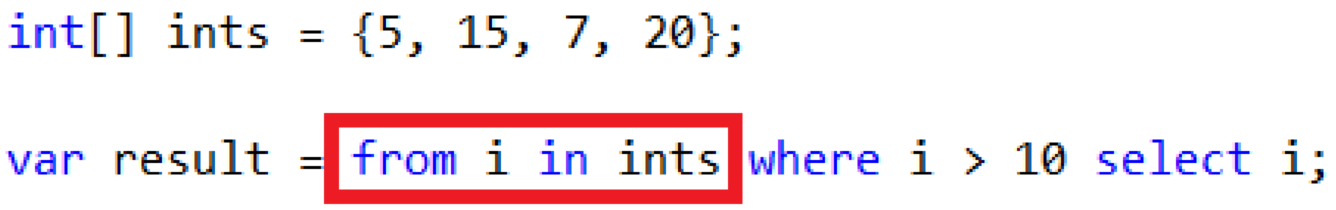
LINQ Query Structure



The *from* clause

- We use the *from* clause to specify which collection we will use in the query.
- It also defines the variable alias that represents one item in the collection.

from item in items



```
int[] ints = {5, 15, 7, 20};

var result = from i in ints where i > 10 select i;
```

SOFTWARE GUILD

The *join* clause

- The *join* clause in LINQ takes two collections and creates a new collection where each element has members from both original collections.

from type in types join type2 in types2

on type.property equals type2.property

LINQ Joins

DEMO

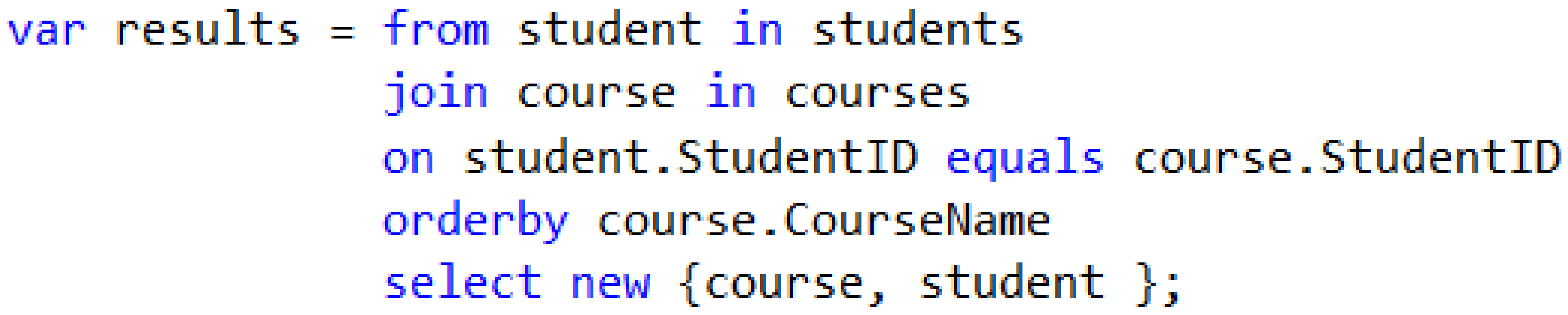
The *where* clause

- The *where* clause in LINQ works much like the where clause in SQL. We can chain together boolean expressions to filter the list.

```
var results = from student in students
              join course in courses on student.StudentID equals course.StudentID
              where student.StudentID == 1
              select new {course, student };
```

The *orderby* Clause

- The *orderby* clause can be ascending or descending.



```
var results = from student in students
              join course in courses
              on student.StudentID equals course.StudentID
              orderby course.CourseName
              select new {course, student };
```

The *group* clause

- The *group* clause takes a field and creates a wrapper object for each unique field, called a *key.*
- You can enumerate the keys to list out the members of each group.

LINQ Group

DEMO

Standard Operators

- *Count*
- *Select*
- *SelectMany*
- *Take*
- *Skip*
- *TakeWhile*
- *SkipWhile*
- *Join*
- *GroupJoin*
- *Concat*
- *OrderBy*
- *Reverse*
- *GroupBy*
- *Distinct*
- *Union*
- *Intersect*

More Operators

- *Except*
- *AsEnumerable*
- *ToArray*
- *ToList*
- *ToDictionary*
- *ToLookup*
- *OfType*
- *Cast*
- *SequenceEqual*
- *First*
- *FirstOrDefault*
- *Last*
- *LastOrDefault*
- *Single*
- *SingleOrDefault*
- *ElementAt*

Even More!

- *ElementAtOrDefault*
- *DefaultIfEmpty*
- *Range*
- *Repeat*
- *Empty*
- *Any*
- *All*

- *Contains*
- *Count*
- *LongCount*
- *Sum*
- *Min*
- *Max*
- *Average*
- *Aggregate*

Needless to Say…

There are way too many to go over without turning your brains into mush.

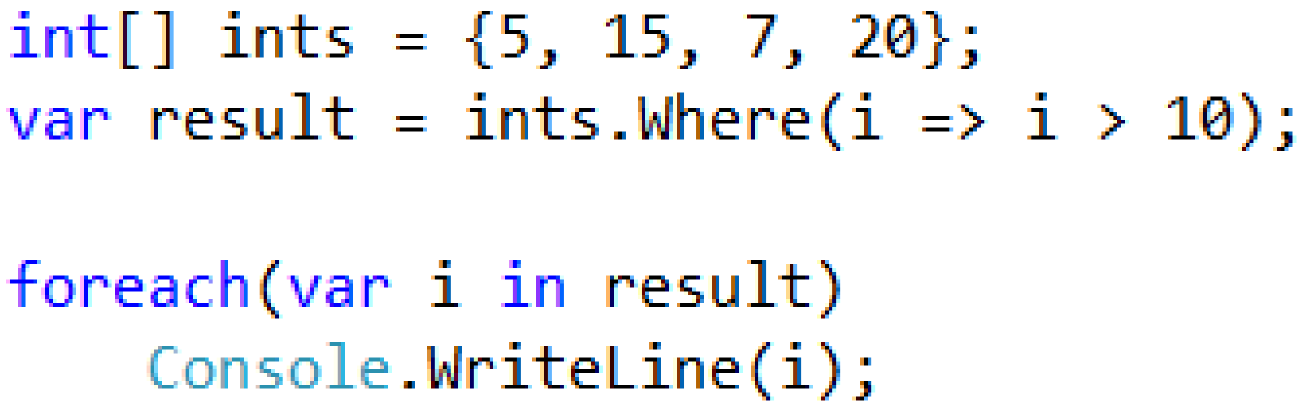
Find all the examples on MSDN:

http://msdn.microsoft.com/en-us/library/bb397896.aspx

Use your intellisense!

Remember Delegates?

- We can use delegates and lambda expressions as parameters to standard operators:



```
int[] ints = {5, 15, 7, 20};
var result = ints.Where(i => i > 10);

foreach(var i in result)
    Console.WriteLine(i);
```

Using Func<>

- *Func* is a built-in generic delegate. Let's say we want to count only odd numbers:

```
static bool IsOdd(int x)
{
    return x % 2 == 1;
}

static void Main()
{
    int[] ints = {5, 15, 7, 20};

    Func<int, bool> myDelegate = IsOdd;

    var result = ints.Count(myDelegate);

    Console.WriteLine("There are {0} odd numbers", result);

    Console.ReadLine();
}
```

Conclusion on LINQ

- LINQ is everywhere. It is the de facto method for filtering data in C# these days.
  - Just remember that with every abstraction, there is a performance cost. Don't use LINQ to stuff every round peg into a square hole.
  - Databases are optimized to perform sorting and filtering functions. A good architect will determine whether to do filtering and sorting on the client via LINQ or on the server via SQL.