LINQ (Language-Integrated Query)

Using Automatic Type Inference (Using **var** to declare a variable)

```
int myInteger;
string myString;
  … myInteger = 5; myString = "Hi there";

var myInteger;
var myString;
  … myInteger = 5; myString = "Hi there";
```

var in a declaration means that the compiler will decide what type the variable is once something is assigned.

We don't usually use var unless we can't (or don't want to) name the type up front. We use var a lot within the context of LINQ.

Language-Integrated Query (LINQ)

LINQ is a set of features that extends powerful query capabilities to the language syntax of C#. (MSDN)

Add this to the top of your controller if you want to use LINQ:

using System.Linq;

Here is what LINQ code looks like:

What type is over250?

IEnumerable<Product>
But we used var so we didn't have to concern ourselves with the type..

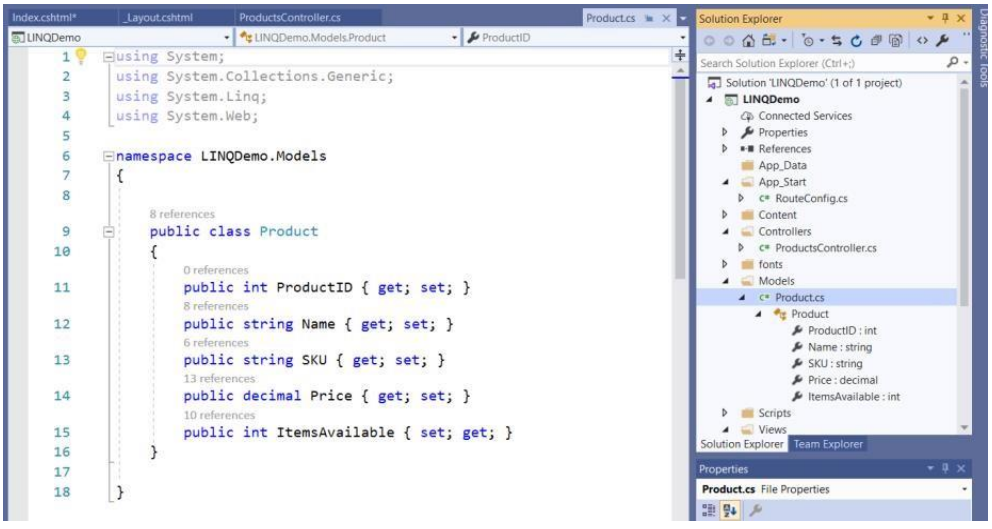
```
var over250 =
    from p in ProductArray   ← Look at each element in the collection called ProductArray, call it p.
    where p.Price > 250M     ← Only keep those where Price is greater than 250.
    orderby p.Price          ← Order the elements by Price.
    select p;                ← Include all of the properties for each element (since we used p and not specific property names).
```

The object called over250 will be a list of products from ProductArray with a price greater than 250 and ordered by price.

Download LINQDemoQuerySyntax.zip from Canvas, unzip, and open in Visual Studio.

Let's look around …

Product Class



ProductsController

```
0 references
public class ProductsController : Controller
{
    Product[] ProductArray = {
            new Product { Name = "3-Shelf Bookcase", SKU = "D34U65G9", Price = 275M, ItemsAvailable = 50},
            new Product { Name = "5-Shelf Bookcase", SKU = "L34K43F4", Price = 589M, ItemsAvailable = 25},
            new Product { Name = "Desk", SKU = "J45U65Y9", Price = 875M, ItemsAvailable = 4},
            new Product { Name = "Office Chair", SKU = "L34U65G9", Price = 125M, ItemsAvailable = 0},
            new Product { Name = "Table Lamp", SKU = "T76O65G9", Price = 24.99M, ItemsAvailable = 150}
        };

    // GET: Product
    0 references
    public ActionResult Index()
    {
        return View(ProductArray);
    }
    0 references
    public ActionResult SortedByPrice()
    {
        var sorted =
            from p in ProductArray
            orderby p.Price
            select p;
        return View("Index", sorted);
    }
```

Index.cshtml

```
@model IEnumerable<LINQDemo.Models.Product>

@{
    ViewBag.Title = "Index";
}

<h2>Products</h2>

<table class="table">
    <thead><tr><th align="left">Product</th><th align="left">Price</th><th>Items Available</th></tr></thead>
    <tbody>

            @foreach (var product in Model)
            {
                <tr>
                    <td> @product.Name </td>
                    <td> @product.Price.ToString("c") </td>
                    <td> @product.SKU</td>
                    <td> @product.ItemsAvailable</td>
                </tr>
            }
        </table>
```

7

All Products Price Low to High Over $250 Cheap Stuff Low Stock Items Out of Stock Summary

Products

Product	Price	Items Available	
3-Shelf Bookcase	$275.00	D34U65G9	50
5-Shelf Bookcase	$589.00	L34K43F4	25
Desk	$875.00	J45U65Y9	4
Office Chair	$125.00	L34U65G9	0
Table Lamp	$24.99	T76O65G9	150

```
        new Product { Name = "3-Shelf Bookcase", SKU = "D34U65G9", Price = 275M, ItemsAvai
        new Product { Name = "5-Shelf Bookcase", SKU = "L34K43F4", Price = 589M, ItemsAvai
        new Product { Name = "Desk", SKU = "J45U65Y9", Price = 875M, ItemsAvailable = 4},
        new Product { Name = "Office Chair", SKU = "L34U65G9", Price = 125M, ItemsAvailabl
        new Product { Name = "Table Lamp", SKU = "T76O65G9", Price = 24.99M, ItemsAvailabl
    };

// GET: Product
0 references
public ActionResult Index()
{
    return View(ProductArray);
}
0 references
public ActionResult SortedByPrice()
{
    var sorted =
       from p in ProductArray
       orderby p.Price
       select p;
    return View("Index", sorted);
}
```

ProductsController

Products

Product	Price	Items Available	
Table Lamp	$24.99	T76O65G9	150
Office Chair	$125.00	L34U65G9	0
3-Shelf Bookcase	$275.00	D34U65G9	50
5-Shelf Bookcase	$589.00	L34K43F4	25
Desk	$875.00	J45U65Y9	4

```
public ActionResult Over250()
{
    var over250 =
        from p in ProductArray
        where p.Price > 250M
        orderby p.Price
        select p;
    return View("Index", over250);
}
```

Products

Product	Price	Items Available	
3-Shelf Bookcase	$275.00	D34U65G9	50
5-Shelf Bookcase	$589.00	L34K43F4	25
Desk	$875.00	J45U65Y9	4


```
public ActionResult CheapStuff()
{
    var under100 =
     from p in ProductArray
     where p.Price < 100M
     orderby p.Price
     select p;
    return View("Index", under100);
}
```

Products

Product	Price	Items Available	
Table Lamp	$24.99	T76O65G9	150

```
public ActionResult LowStock()
{
    var lowStock =
     from p in ProductArray
     where p.ItemsAvailable < 20 && p.ItemsAvailable > 0
     orderby p.Name
     select p;
    return View("Index", lowStock);
}
```

Products

Product	Price	Items Available
Desk	$875.00	J45U65Y9

```
public ActionResult OutOfStock()
{
    var noStock =
        from p in ProductArray
        where p.ItemsAvailable == 0
        orderby p.Name
        select p;
    return View("Index", noStock);
}
```

Products

Product	Price	Items Available
Office Chair	$125.00	L34U65G9

What will this code do?

```
var aQuery =
    from p in ProductArray
    orderby p.Price descending
    select new { p.Name, p.Price };
```

What type is actually returned?
IEnumerable<An Anonymous Class>
Good thing we can use var!

A new class is created, separate from the Product class

```
public ActionResult Index()
{
    var aQuery =
      from p in ProductArray
      orderby p.Price descending
      select new { p.Name, p.Price };

    return View(aQuery);
}
```

What if we send aQuery into this view? →

By the way, what happens when we add .ToString("c") to the end of Price below?

The decimal value is put in currency format (with a $ and just two decimal places).

```
@foreach (var product in Model)
{
  <tr>
    <td> @product.Name </td>
    <td> @product.Price.ToString("c") </td>
    <td> @product.SKU</td>
    <td> @product.ItemsAvailable</td>
  </tr>
}
```

We'll get an error. Why?

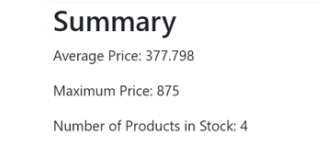
SKU and ItemsAvailable are not valid properties for our new class.

Other cool stuff we can do:

```
public ActionResult Summary()
{

    ViewBag.AvgPrice = (from p in ProductArray
                        select p.Price).Average();

    ViewBag.MaxPrice = (from p in ProductArray
                        select p.Price).Max();

    ViewBag.NumProducts = (from p in ProductArray
                           where p.ItemsAvailable > 0
                           select p).Count();

    return View("Summary");
}
```

Matching View →

```
@{
    ViewBag.Title = "Summary";
}

<h2>Summary</h2>

<p>Average Price: @ViewBag.AvgPrice</p>
<p>Maximum Price: @ViewBag.MaxPrice</p>
<p>Number of Products in Stock: @ViewBag.NumProducts</p>
```



Here is a good link for reference.
https://msdn.microsoft.com/en-us/library/bb397927.aspx

We have just learned LINQ using **Query Syntax**.

There is another syntax called **Method Syntax**. It is

- not as easy to read
- but is easier to type and remember.

Query Syntax

```
var over250 =
     from p in ProductArray
     where p.Price > 250M
     orderby p.Price
     select p;
```

Corresponding Method Syntax

Condition Expression

```
var over250 = ProductArray.Where(p => p.Price > 250M).OrderBy(p => p.Price);
```

Query Syntax

```
var justNamesAndPricesInDescendingOrderByPrice =
        from p in ProductArray
        orderby p.Price descending
        select new { p.Name, p.Price };
```

Corresponding Method Syntax

```
var justNamesAndPricesInDescendingOrderByPrice =
      ProductArray.OrderByDescending(p => p.Price)
      .Select(p => new { p.Name, p.Price });
```

Query Syntax

```
var avgPrice = (from p in ProductArray
                select p.Price).Average();
```

Corresponding Method Syntax

```
var avgPrice = ProductArray.Select(p => p.Price).Average();
```

```
var productsSorted =
      ProductArray.OrderByDescending(p => p.Price)
      .Select(p => new { p.Name, p.Price }).Take(3);
```

```
 var productsSorted = ProductArray.OrderByDescending(p => p.Price)
                      .Skip(5);
```

`All`	Returns `true` if all the items in the source data match the predicate
`Any`	Returns `true` if at least one of the items in the source data matches the predicate
`Contains`	Returns `true` if the data source contains a specific item or value
`Count`	Returns the number of items in the data source
`First`	Returns the first item from the data source
`FirstOrDefault`	Returns the first item from the data source or the default value if there are no items
`Last`	Returns the last item in the data source
`LastOrDefault`	Returns the last item in the data source or the default value if there are no items
`Max` `Min`	Returns the largest or smallest value specified by a lambda expression
`OrderBy` `OrderByDescending`	Sorts the source data based on the value returned by the lambda expression
`Reverse`	Reverses the order of the items in the data source
`Select`	Projects a result from a query
`SelectMany`	Projects each data item into a sequence of items and then concatenates all of those resulting sequences into a single sequence
`Single`	Returns the first item from the data source or throws an exception if there are multiple matches
`SingleOrDefault`	Returns the first item from the data source or the default value if there are no items, or throws an exception if there are multiple matches
`Skip` `SkipWhile`	Skips over a specified number of elements, or skips while the predicate matches
`Sum`	Totals the values selected by the predicate
`Take` `TakeWhile`	Selects a specified number of elements from the start of the data source or selects items while the predicate matches

`ToArray`
`ToDictionary`
`ToList`
	Converts the data source to an array or other collection type

`Where`	Filters items from the data source that do not match the predicate

Let's go back and rewrite all of our LINQ code in **method syntax**.
Write the results on your notes if you have them printed.

Query Syntax

```
public ActionResult SortedByPrice()
{
    var sorted =
       from p in ProductArray
       orderby p.Price
       select p;
    return View("Index", sorted);
}
```

Method Syntax

```
public ActionResult SortedByPrice()
{


    return View("Index", sorted);
}
```

Query Syntax

```
public ActionResult Over250()
{
    var over250 =
        from p in ProductArray
        where p.Price > 250M
        orderby p.Price
        select p;
    return View("Index", over250);
}
```

Method Syntax

```
public ActionResult Over250()
{


    return View("Index", over250);
}
```

Query Syntax

```
public ActionResult CheapStuff()
{
    var under100 =
     from p in ProductArray
     where p.Price < 100M
     orderby p.Price
     select p;
    return View("Index", under100);
}
```

Method Syntax

```
public ActionResult CheapStuff()
{


    return View("Index", under100);
}
```

Query Syntax

```
public ActionResult LowStock()
{
    var lowStock =
     from p in ProductArray
     where p.ItemsAvailable < 20 && p.ItemsAvailable > 0
     orderby p.Name
     select p;
    return View("Index", lowStock);
}
```

Method Syntax

```
public ActionResult LowStock()
{



   return View("Index", lowStock);
}
```

Query Syntax

```
public ActionResult OutOfStock()
{
    var noStock =
        from p in ProductArray
        where p.ItemsAvailable == 0
        orderby p.Name
        select p;
    return View("Index", noStock);
}
```

Method Syntax

```
public ActionResult OutOfStock()
{



    return View("Index", noStock);
}
```

Query Syntax

```
public ActionResult Summary()
{
    ViewBag.AvgPrice = (from p in ProductArray
                        select p.Price).Average();
    ViewBag.MaxPrice = (from p in ProductArray
                        select p.Price).Max();
    ViewBag.NumProducts = (from p in ProductArray
                           where p.ItemsAvailable > 0
                           select p).Count();
    return View("Summary");
}
```

Method Syntax

```
public ActionResult Summary()
{



  return View("Summary");
}
```