329-Windows Programlama

Bil. Müh. S. Kıvanç EKİCİ

Implicit Typing of Local Variables

```
static void DeclareImplicitVars()
{
        // Implicitly typed local variables.
        var myInt = 0;
        var myBool = true;
        var myString = "Time, marches on...";

        // Print out the underlying type.
        Console.WriteLine("myInt is a: {0}", myInt.GetType().Name);
        Console.WriteLine("myBool is a: {0}", myBool.GetType().Name);
        Console.WriteLine("myString is a: {0}", myString.GetType().Name);
}
```

Extension Methods

```
namespace MyExtensions
{
    static class ObjectExtensions
    {
        // Define an extension method to System.Object.
        public static void DisplayDefiningAssembly(this object obj)
        {
            Console.WriteLine("{0} lives here:\n\t->{1}\n", obj.GetType().
            Name,
            Assembly.GetAssembly(obj.GetType()));
        }
    }
}
```

Extension Methods 2

```
static void Main(string[] args)
{
        // Since everything extends System.Object, all classes and structures
        // can use this extension.
        int myInt = 12345678;
        myInt.DisplayDefiningAssembly();

        System.Data.DataSet d = new System.Data.DataSet();
        d.DisplayDefiningAssembly();

        Console.ReadLine();
}
```

Anonymous Types

```
var purchaseItem = new {
TimeBought = DateTime.Now,
ItemBought = new {Color = "Red", Make = "Saab", CurrentSpeed = 55},
Price = 34.000};
```

LINQ

- Veri işlemek için oluşturulmuş bir kütüphane.
- Veri sadece veritabanından alınan veriler olarak değil, listeler, xml dosyaları ve diziler şeklinde olabilir.

- LINQ to Objects: This term refers to the act of applying LINQ queries to arrays and collections.

- LINQ to XML: This term refers to the act of using LINQ to manipulate and query XML documents.

- LINQ to DataSet: This term refers to the act of applying LINQ queries to ADO.NET DataSet objects.

- LINQ to Entities: This aspect of LINQ allows you to make use of LINQ queries within the ADO.NET Entity Framework (EF) API.

- Parallel LINQ (a.k.a. PLINQ): This allows for parallel processing of data returned from a LINQ query.

LINQ to primitive arrays

```
static void QueryOverStrings()
{
    // Assume we have an array of strings.
    string[] currentVideoGames = {"Morrowind", "Uncharted 2",
    "Fallout 3", "Daxter", "System Shock 2"};

    // Build a query expression to find the items in the array
    // that have an embedded space.
    IEnumerable<string> subset = from g in currentVideoGames
    where g.Contains(" ") orderby g select g;

    // Print out the results.
    foreach (string s in subset)
    Console.WriteLine("Item: {0}", s);
}
```

LINQ and implicit typing

```
static void QueryOverInts()
{
	int[] numbers = {10, 20, 30, 40, 1, 2, 3, 8};

	// Use implicit typing here...
	var subset = from i in numbers where i < 10 select i;

	// ...and here.
	foreach (var i in subset)
		Console.WriteLine("Item: {0} ", i);

	ReflectOverQueryResults(subset);
}
```

The Role of Deferred Execution

- LINQ sorgularının sonuçları hemen alınıp saklanmak yerine kullanıldığı anda çalıştırılır.
- Bu sayede en güncel bilgi edinilmiş olur.
- Eğer sorgu yazıldığı andaki sonuç alınmak istenirse bir diziye eklenmesi yoluyla yapılabilir.

The Role of Deferred Execution

```
static void QueryOverInts()
{
    int[] numbers = { 10, 20, 30, 40, 1, 2, 3, 8 };
    // Get numbers less than ten.
    var subset = from i in numbers where i < 10 select i;

    // LINQ statement evaluated here!
    foreach (var i in subset)
        Console.WriteLine("{0} < 10", i);

    // Change some data in the array.
    numbers[0] = 4;

    // Evaluated again!
    foreach (var j in subset)
    Console.WriteLine("{0} < 10", j);
}
```

LINQ immediate execution

```
static void ImmediateExecution()
{
	int[] numbers = { 10, 20, 30, 40, 1, 2, 3, 8 };
	// Get data RIGHT NOW as int[].

	int[] subsetAsIntArray =
	(from i in numbers where i < 10 select i).ToArray<int>();
	// Get data RIGHT NOW as List<int>.

	List<int> subsetAsListOfInts =
	(from i in numbers where i < 10 select i).ToList<int>();
}
```

returing LINQ results 1

```
static IEnumerable<string> GetStringSubset()
{
    string[] colors = {"Light Red", "Green",
    "Yellow", "Dark Red", "Red", "Purple"};

    // Note subset is an IEnumerable<string>-compatible object.
    IEnumerable<string> theRedColors = from c in colors
    where c.Contains("Red") select c;

    return theRedColors;
}
```

returing LINQ results 2

```
static string[] GetStringSubsetAsArray()
{
    string[] colors = {"Light Red", "Green",
    "Yellow", "Dark Red", "Red", "Purple"};

    var theRedColors = from c in colors

    where c.Contains("Red") select c;

    // Map results into an array.
    return theRedColors.ToArray();
}
```

Applying LINQ Queries to Collection Objects

```
class Car
{
        public string PetName {get; set;}
        public string Color {get; set;}
        public int Speed {get; set;}
        public string Make {get; set;}
}
static void Main(string[] args)
{
        // Make a List<> of Car objects.
        List<Car> myCars = new List<Car>() {
                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"}
        };

}
```

Applying LINQ Queries to Collection Objects 2

```
static void GetFastCars(List<Car> myCars)
{
        // Find all Car objects in the List<>, where the Speed is
        // greater than 55.
        var fastCars = from c in myCars where c.Speed > 55 select c;

        foreach (var car in fastCars)
        {
                Console.WriteLine("{0} is going too fast!", car.PetName);
        }
}
```

Filtering Data Using OfType<T>()

```
static void OfTypeAsFilter()
{
	// Extract the ints from the ArrayList.
	ArrayList myStuff = new ArrayList();

	myStuff.AddRange(new object[] { 10, 400, 8, false, new Car(), "string data"
	});

	var myInts = myStuff.OfType<int>();

	// Prints out 10, 400, and 8.
	foreach (int i in myInts)
	{
	Console.WriteLine("Int value: {0}", i);
	}
}
```

LINQ operators

- Table 12-3. Common LINQ Query Operators Sayfa 458.

basic selection syntax

var result = from matchingItem in container select matchingItem;

```
static void SelectEverything(ProductInfo[] products)
{
        // Get everything!
        Console.WriteLine("All product details:");
        var allProducts = from p in products select p;

        foreach (var prod in allProducts)
        {
                Console.WriteLine(prod.ToString());
        }
}
```

Obtaining Subsets of Data

var result = from item in container where BooleanExpression select item;

```
static void GetOverstock(ProductInfo[] products)
{
	Console.WriteLine("The overstock items!");
	// Get only the items where we have more than
	// 25 in stock.
	var overstock = from p in products where p.NumberInStock > 25 select p;

	foreach (ProductInfo c in overstock)
	{
		Console.WriteLine(c.ToString());
	}
}
```

Obtaining Counts Using Enumerable

```
static void GetCountFromQuery()
{
    string[] currentVideoGames = {"Morrowind", "Uncharted 2",
    "Fallout 3", "Daxter", "System Shock 2"};

    // Get count from the query.
    int numb =
    (from g in currentVideoGames where g.Length > 6 select g).Count();

    // Print out the number of items.
    Console.WriteLine("{0} items honor the LINQ query.", numb);
}
```

Reversing Result Sets

```
static void ReverseEverything(ProductInfo[] products)
{
        Console.WriteLine("Product in reverse:");
        var allProducts = from p in products select p;

        foreach (var prod in allProducts.Reverse())
        {
                Console.WriteLine(prod.ToString());
        }
}
```

Sorting Expressions

```
static void AlphabetizeProductNames(ProductInfo[] products)
{
    // Get names of products, alphabetized.
    var subset = from p in products orderby p.Name select p;

    Console.WriteLine("Ordered by Name:");
    foreach (var p in subset)
    {
        Console.WriteLine(p.ToString());
    }
}


//
var subset = from p in products orderby p.Name ascending select p;
```

Except

```
static void DisplayDiff()
{
        List<string> myCars = new List<String> {"Yugo", "Aztec", "BMW"};
        List<string> yourCars = new List<String>{"BMW", "Saab", "Aztec" };

        var carDiff =(from c in myCars select c)
        .Except(from c2 in yourCars select c2);

        Console.WriteLine("Here is what you don't have, but I do:");
        foreach (string s in carDiff)
        Console.WriteLine(s); // Prints Yugo.
}
```

Intersect

```
static void DisplayIntersection()
{
        List<string> myCars = new List<String> { "Yugo", "Aztec", "BMW" };
        List<string> yourCars = new List<String> { "BMW", "Saab", "Aztec" };

        // Get the common members.
        var carIntersect = (from c in myCars select c)
        .Intersect(from c2 in yourCars select c2);

        Console.WriteLine("Here is what we have in common:");
        foreach (string s in carIntersect)
        Console.WriteLine(s); // Prints Aztec and BMW.
}
```

Union

```
static void DisplayUnion()
{
    List<string> myCars = new List<String> { "Yugo", "Aztec", "BMW" };
    List<string> yourCars = new List<String> { "BMW", "Saab", "Aztec" };

    // Get the union of these containers.
    var carUnion = (from c in myCars select c)
    .Union(from c2 in yourCars select c2);

    Console.WriteLine("Here is everything:");
    foreach (string s in carUnion)
    Console.WriteLine(s); // Prints all common members.
}
```

Concat

```
static void DisplayConcat()
{
        List<string> myCars = new List<String> { "Yugo", "Aztec", "BMW" };
        List<string> yourCars = new List<String> { "BMW", "Saab", "Aztec" };

        var carConcat = (from c in myCars select c)
        .Concat(from c2 in yourCars select c2);


        // Prints:
        // Yugo Aztec BMW BMW Saab Aztec.
        foreach (string s in carConcat)
        Console.WriteLine(s);
}
```

Distinct

```
static void DisplayConcatNoDups()
{
    List<string> myCars = new List<String> { "Yugo", "Aztec", "BMW" };
    List<string> yourCars = new List<String> { "BMW", "Saab", "Aztec" };
    var carConcat = (from c in myCars select c)
    .Concat(from c2 in yourCars select c2);

    // Prints:
    // Yugo Aztec BMW Saab Aztec.
    foreach (string s in carConcat.Distinct())
        Console.WriteLine(s);
}
```

LINQ Aggregation Operations

```
static void AggregateOps()
{
        double[] winterTemps = { 2.0, -21.3, 8, -4, 0, 8.2 };
        // Various aggregation examples.

        Console.WriteLine("Max temp: {0}",
                (from t in winterTemps select t).Max());

        Console.WriteLine("Min temp: {0}",
                (from t in winterTemps select t).Min());

        Console.WriteLine("Avarage temp: {0}",
                (from t in winterTemps select t).Average());

        Console.WriteLine("Sum of all temps: {0}",
                (from t in winterTemps select t).Sum());
}
```

Ödev

- Chapter 11: Advanced C# Language Features .....399 ve
- Chapter 12: LINQ to Objects .....439 özetlenecek 2*2= 4 sayfa olarak.

- Proje ödevi olarak moodle da yayınlanmış olan Maliyet Hesaplama projesi devam edilecek.