LẬP TRÌNH TRÊN MÔI TRƯỜNG WINDOWS

***

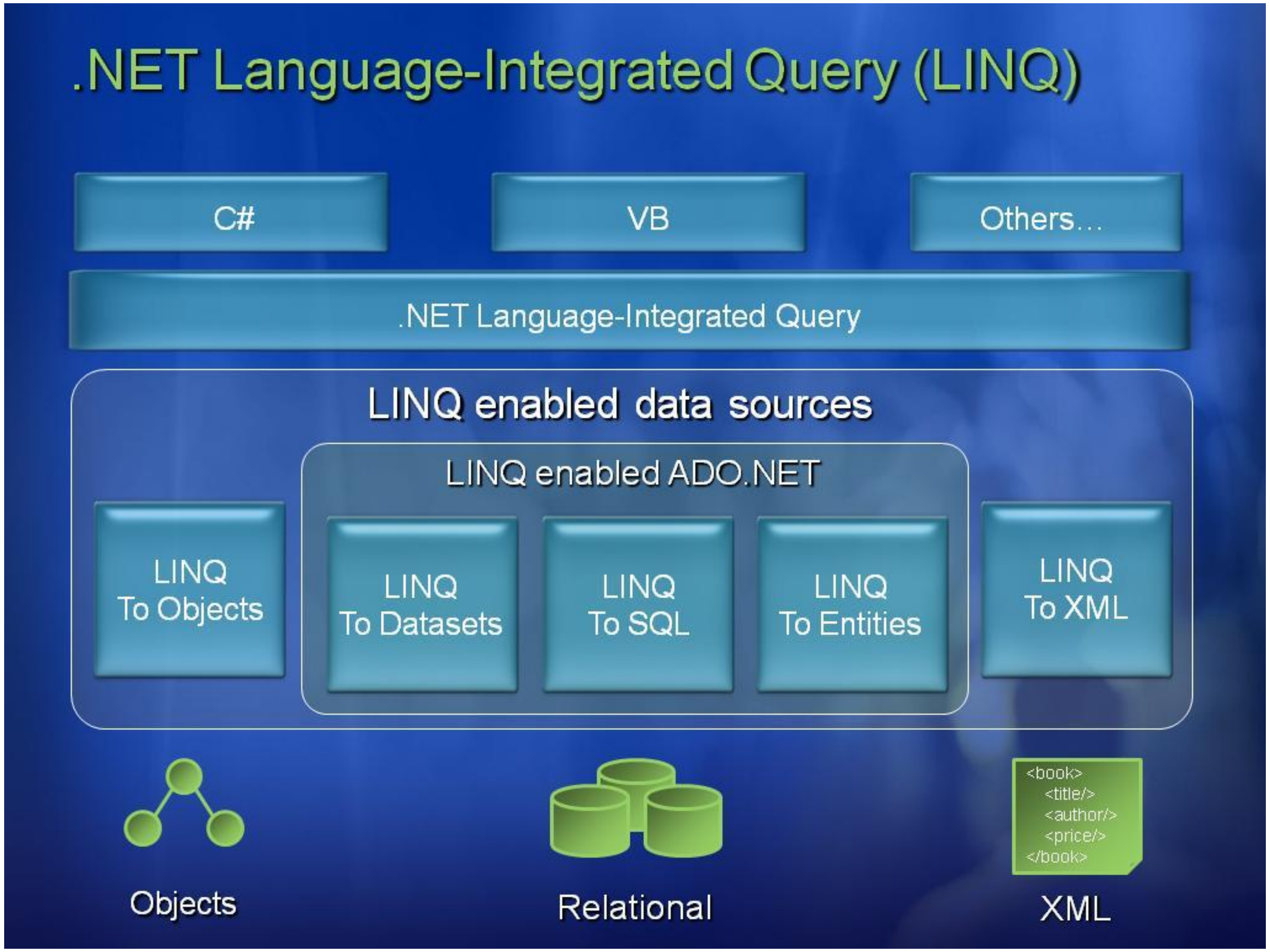
LINQ

Content

Introduction

- It is a set of language changes and API's that allow you to write SQL-like queries natively in a .NET programming language.
- LINQ allows you to obtain data in a consistent manner.
- Apply to all sources of information, not just relational or XML data.

Introduction



Introduction

- Some LINQ Namespaces
  - System.Linq
  - System.Linq.Expressions
  - System.Data.Linq
  - System.Xml.Linq
  - …

Introduction

- Example

```
public void Linq1()
{
        int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };
        var lowNums = from n in numbers where n < 5 select n;
        Console.WriteLine("Numbers < 5:");
        foreach (var x in lowNums)
        {
                Console.WriteLine(x);
        }
}
```

Content

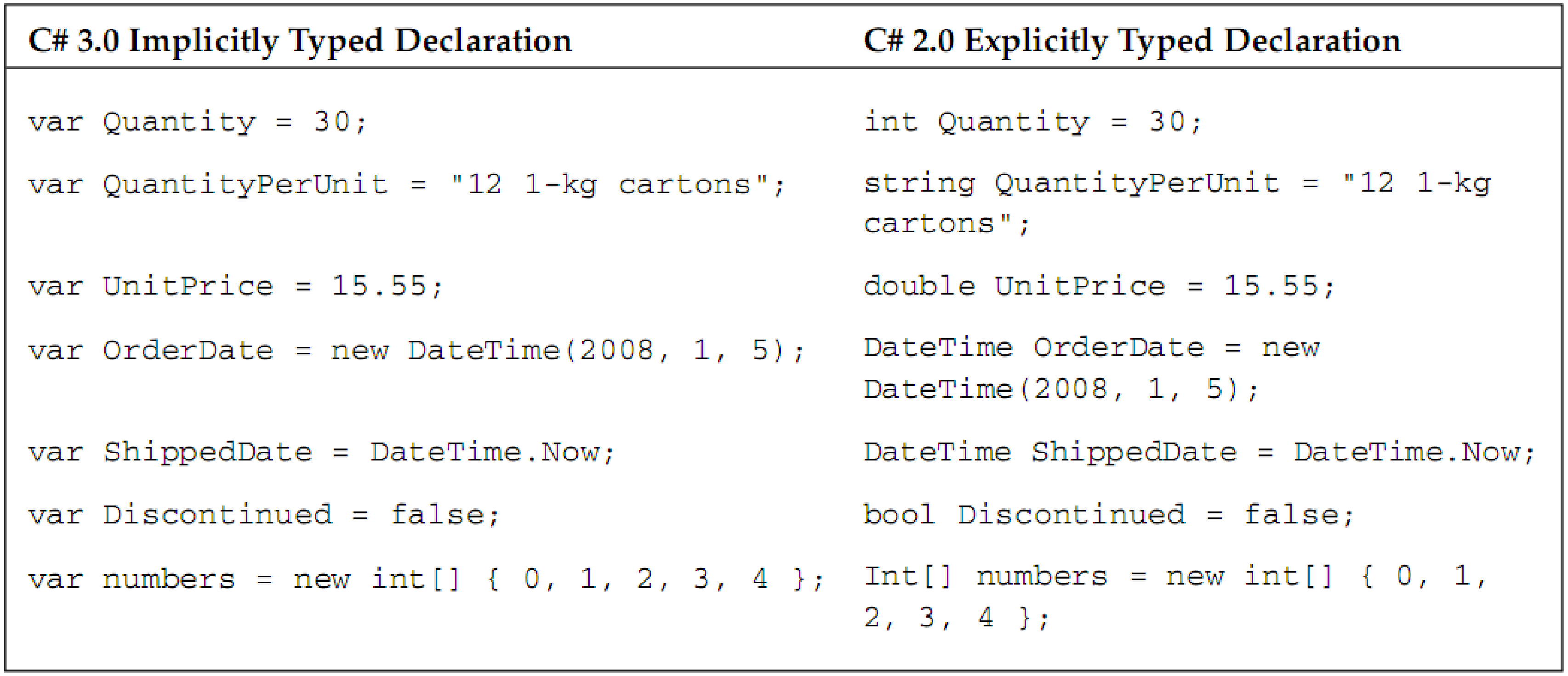
- Introduction
- **Language Extensions**
- Standard Query Operators
- LINQ providers

Language Extensions

- Implicitly typed local variables
- Object initializers
- Anonymous types
- Extension methods
- Lambda expressions
- Standard Query Operators
- Query expressions

Language Extensions

- Implicitly typed local variables



C# 3.0 Implicitly Typed Declaration	C# 2.0 Explicitly Typed Declaration
`var Quantity = 30;`	`int Quantity = 30;`
`var QuantityPerUnit = "12 1-kg cartons";`	`string QuantityPerUnit = "12 1-kg cartons";`
`var UnitPrice = 15.55;`	`double UnitPrice = 15.55;`
`var OrderDate = new DateTime(2008, 1, 5);`	`DateTime OrderDate = new DateTime(2008, 1, 5);`
`var ShippedDate = DateTime.Now;`	`DateTime ShippedDate = DateTime.Now;`
`var Discontinued = false;`	`bool Discontinued = false;`
`var numbers = new int[] { 0, 1, 2, 3, 4 };`	`Int[] numbers = new int[] { 0, 1, 2, 3, 4 };`

Language Extensions

- Object Initializers

C# 1.0 and 2.0

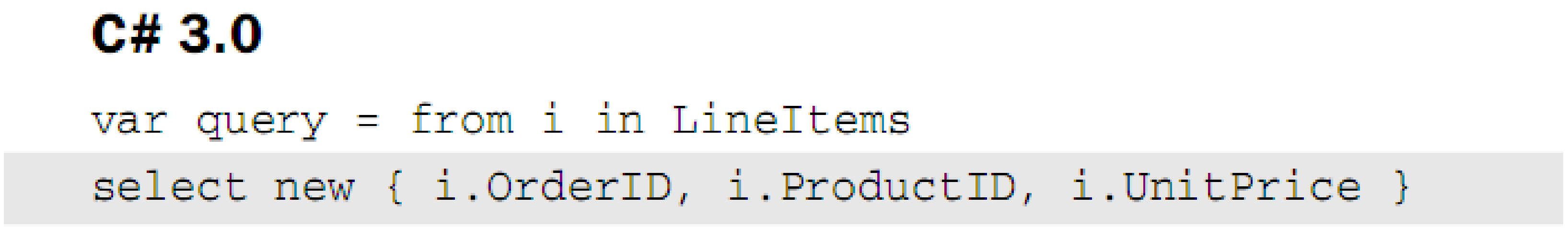
```
LineItem line2 = new LineItem();
         Line2.OrderID = 11000;
         line2.ProductID = 61;
         line2.Quantity = 30;
         line2.QuantityPerUnit = "12 1-kg cartons";
         line2.UnitPrice = 15.55M;
         line2.Discount = 0.15F;
```

C# 3.0

```
var line3 = new LineItem { OrderID = 11000, ProductID = 61,  Quantity = 30,
QuantityPerUnit = "12 1-kg cartons", UnitPrice = 15.55M, Discount = 0.15F };
```

Language Extensions

- Anonymous Types
  - Are an abbreviated form of object initializers that let you omit the type specification when initializing temporary objects or collections.

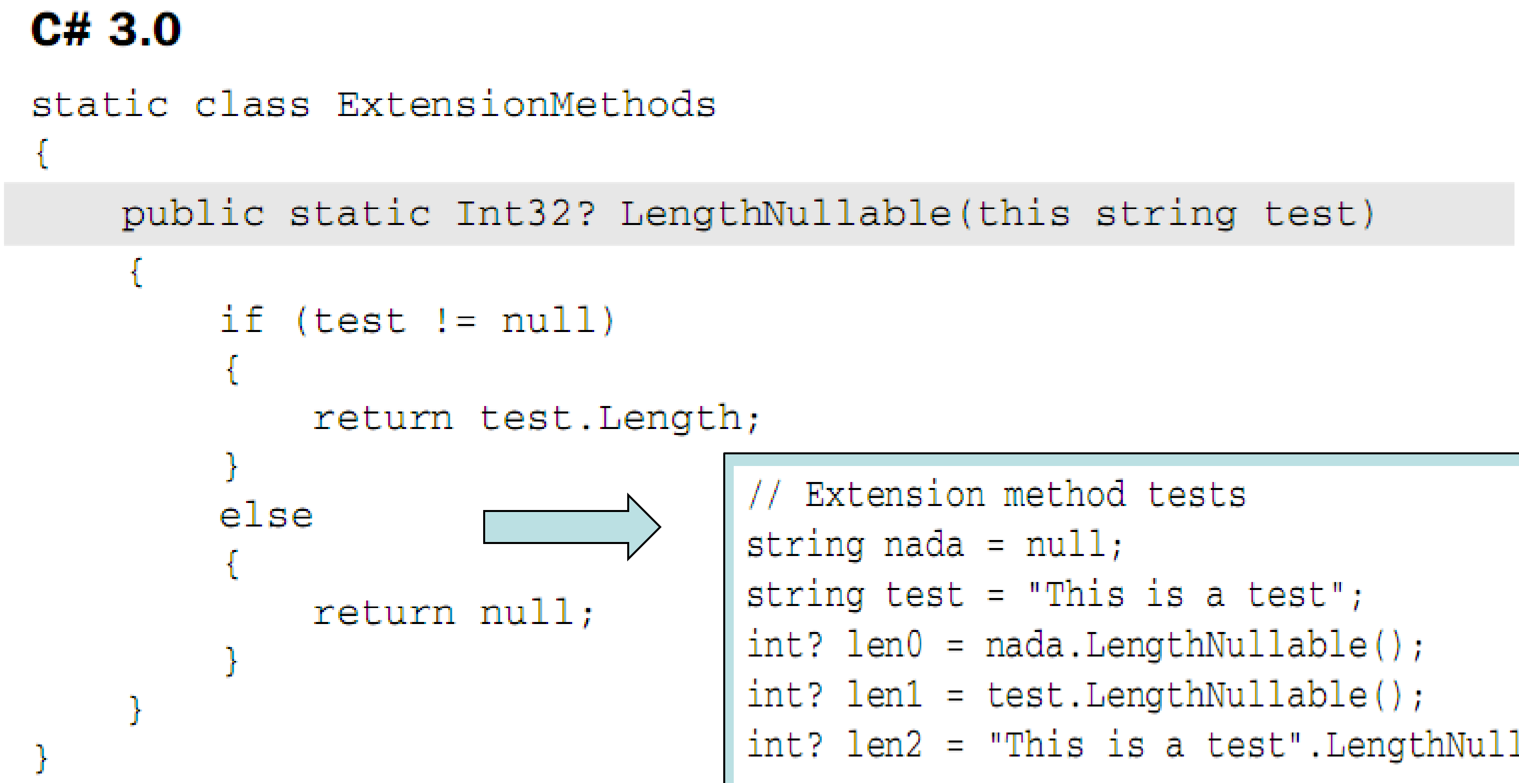


C# 3.0

```
var query = from i in LineItems
select new { i.OrderID, i.ProductID, i.UnitPrice }
```

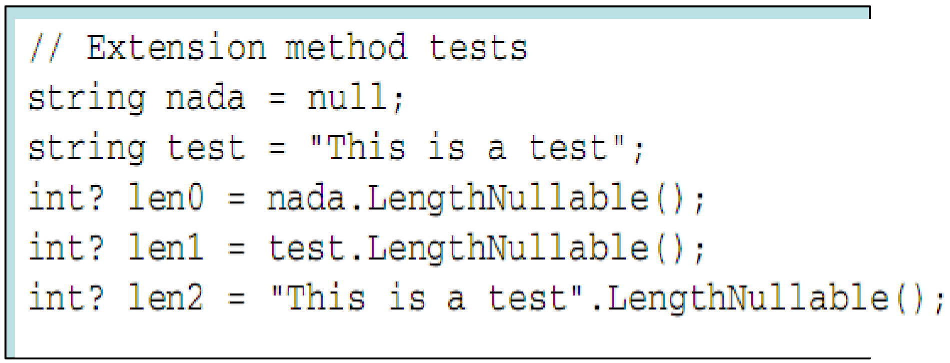
Language Extensions

- Extension Methods
  - Extension methods let you add custom methods to previously defined types.



C# 3.0

```
static class ExtensionMethods
{
    public static Int32? LengthNullable(this string test)
    {
        if (test != null)
        {
            return test.Length;
        }
        else
        {
            return null;
        }
    }
}
```



```
// Extension method tests
string nada = null;
string test = "This is a test";
int? len0 = nada.LengthNullable();
int? len1 = test.LengthNullable();
int? len2 = "This is a test".LengthNullable();
```

Language Extensions

- Lambda Expressions
  - Provide developers with a convenient way to write functions that can be passed as arguments for subsequent evaluation.
  - The basic syntax for lambda expressions
    - argument-list = > expression-or-statement-block
  - Example
    - s => s.ToUpper(); ⟺

```
string  Func(string s)
{
            return s.ToUpper();
}
```

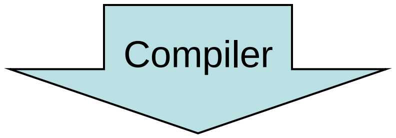
Language Extensions

- Standard Query Operators
  - The standard query operators provide query capabilities including filtering, projection, aggregation, sorting and more.
  - Some operators
    - Select
    - Where
    - SelectMany
    - Sum / Min / Max / Average
    - OrderBy
    - GroupBy
    - Count

Language Extensions

- Query Expressions

```
var noStock = from p in productList
              where p.UnitsInStock == 0
              orderby p.Category, p.ProductID
              select new { p.ProductID, p.Category, p.ProductName };

foreach (var i in noStock)
    result += i.ProductID.ToString() + "\t" + i.Category.Substring(0, 6) + "\t" +
        i.ProductName + "\r\n";
```



```
var noStock = productList
    .Where(p => p.UnitsInStock == 0)
    .OrderBy(p => p.Category)
    .ThenBy(p => p.ProductID)
    .Select(p => new { p.ProductID, p.Category, p.ProductName });
```

Content

Standard Query Operators

- Restriction operators
- Projection operators
- Partitioning operators
- Join operators
- Ordering operators
- Grouping operators
- Set operators
- Element operators
- Quantifiers
- Aggregate operators

Standard Query Operators

- Restriction operators
  - Where
    - x = products.Where(p => p.UnitPrice >= 10);

Standard Query Operators

- Projection operators
  - Select
    - productNames = products.Select(p => p.Name);
  - SelectMany
    - orders =

```
        customers.
        Where(c => c.Country == "Denmark").
        SelectMany(c => c.Orders);
```

Standard Query Operators

- Partitioning operators
  - Take
    - MostExpensive10 =

      products.OrderByDescending(p => p.UnitPrice).Take(10);
  - Skip
    - AllButMostExpensive10 =

      products.OrderByDescending(p => p.UnitPrice).Skip(10);

Standard Query Operators

- Join operators
  - Join
    - var custOrders =

```
        customers.
        Join(orders, c => c.CustomerID, o => o.CustomerID,
                (c, o) => new { c.Name, o.OrderDate, o.Total }
        );
```

Standard Query Operators

- Ordering operators
  - OrderBy / ThenBy
    - orderedProducts1 =

```
          products.
          OrderBy(p => p.Category).
          ThenByDescending(p => p.UnitPrice).
          ThenBy(p => p.Name);
```

Standard Query Operators

- Grouping operators
  - GroupBy
    - productsByCategory =
      products.GroupBy(p => p.Category);

Standard Query Operators

- Set operators
  - Distinct
    - productCategories =
      products.Select(p => p.Category).Distinct();
  - Union
  - Intersect
  - Except

Standard Query Operators

- Element operators
  - First
    - string phone = "206-555-1212";
      Customer c = customers.First(c => c.Phone == phone);
  - Last
  - ElementAt
    - Product thirdMostExpensive =
          products.OrderByDescending(p => p.UnitPrice).ElementAt(2);

Standard Query Operators

- Quantifiers
  - Any
    - bool b =

      products.Any(p => p.UnitPrice >= 100 && p.UnitsInStock == 0);
  - All
  - Contains

Standard Query Operators

- Aggregate operators
  - Count
    - int count = customers.Count(c => c.City == "London");
  - Sum
  - Min
  - Max
  - Average

Content

- Introduction
- Language Extensions
- Standard Query Operators
- **LINQ providers**

LINQ providers

- LINQ Providers
  - LINQ to Objects
  - LINQ to XML
  - LINQ to SQL
  - LINQ to DataSets
  - …

LINQ providers

- LINQ to Object
  - LINQ to Objects allows .NET developers to write "queries" over collections of objects.
  - Example:

```
int[] nums = new int[] {0,4,2,6,3,8,3,1};

int result = nums.Sum();

Console.WriteLine(result);



-------------------------------------------------------

Output: 27
```

LINQ providers

- LINQ to XML
- LINQ to XML is a new way to construct, write and read XML data in the .NET languages.
- This new API simplifies working with XML data without having to resort to using additional language syntax like XPath or XSLT.
- Key classes:
    - XDocument
    - XElement
    - XAttribute

LINQ providers

- LINQ to XML
  - Example

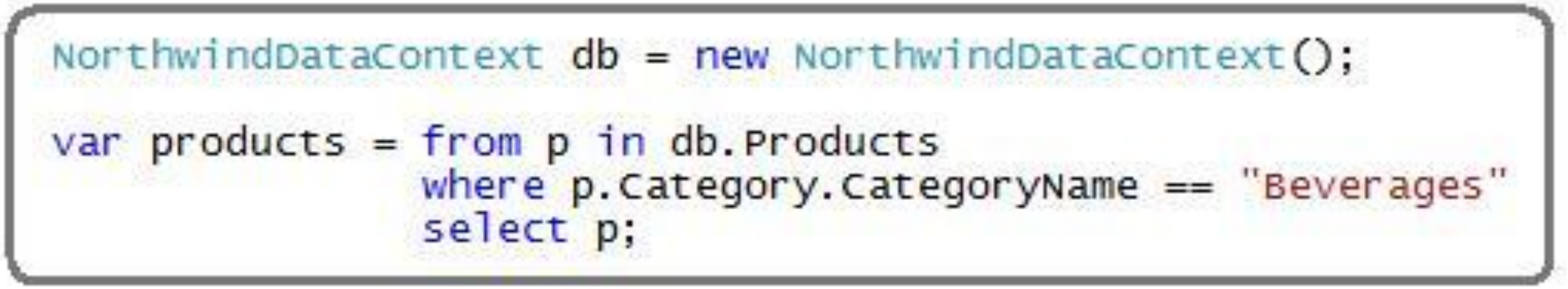
```
XDocument loaded = XDocument.Load(@"C:\contacts.xml");
var q = from c in loaded.Descendants("contact")
        where (int)c.Attribute("contactId") < 4
        select (string)c.Element("firstName") + “ “ + (string)c.Element("lastName");

foreach (string name in q)
    Console.WriteLine("Customer name = {0}", name);
------------------------------------------------------------
Output: Customer name = Barney Gottshall
        Customer name = Armando Valdes
```

LINQ providers

- LINQ to SQL
  - is a component of .NET Framework 3.5 that provides a run-time infrastructure for managing relational data as objects.
  - allows .NET developers to write “queries” in their .NET language of choice to retrieve and manipulate data from a SQL Server database.
  - LINQ to SQL supports rapid development of applications that query Microsoft SQL Server databases using objects that map directly to SQL Server schemas.

LINQ providers

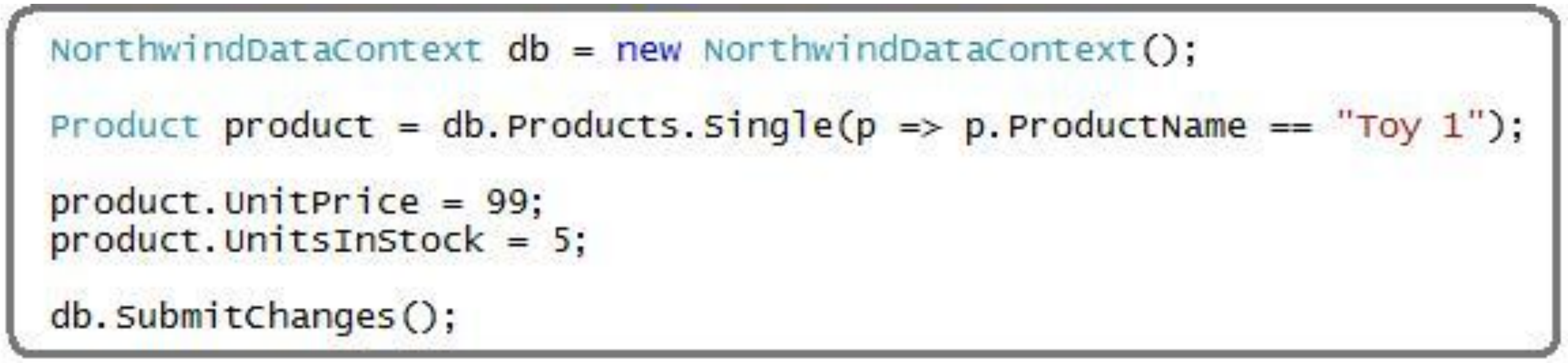
- LINQ to SQL
  - Example



```
NorthwindDataContext db = new NorthwindDataContext();

var products = from p in db.Products
               where p.Category.CategoryName == "Beverages"
               select p;
```

LINQ providers

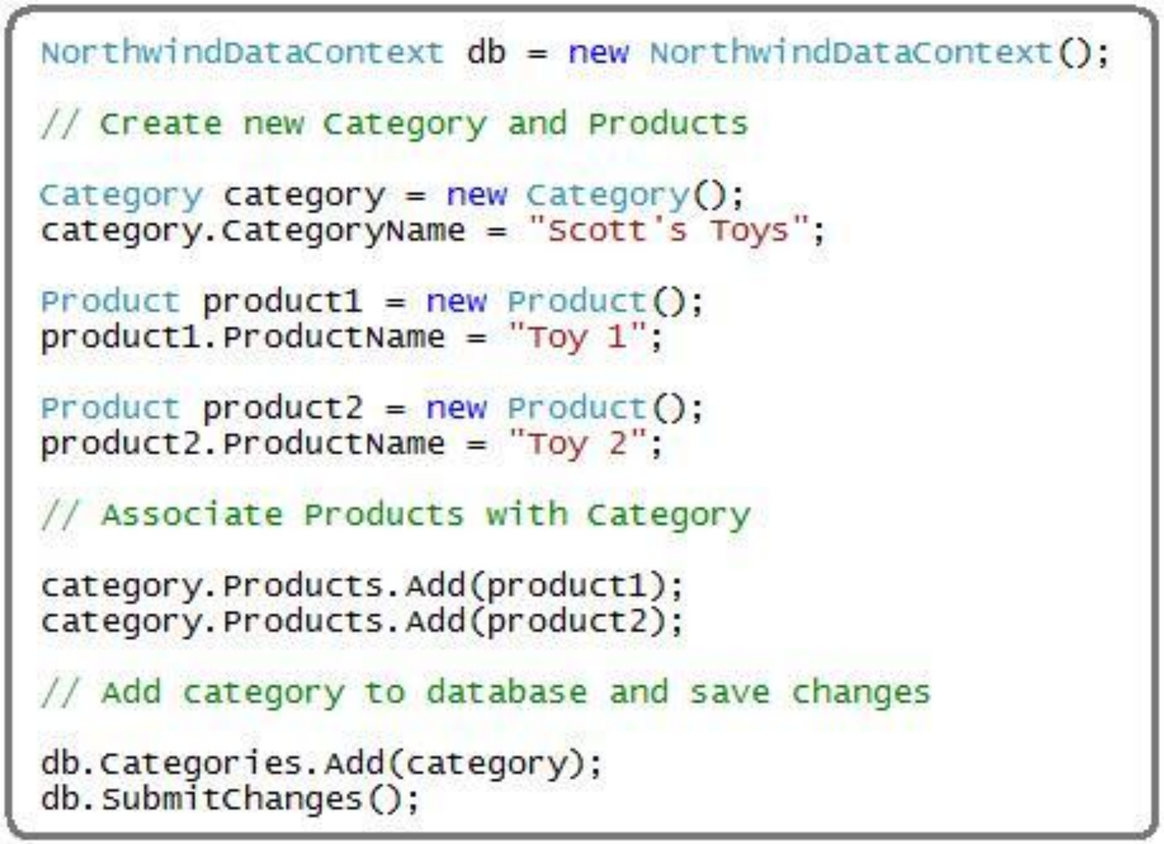
- LINQ to SQL
  - Example:



```
NorthwindDataContext db = new NorthwindDataContext();

Product product = db.Products.Single(p => p.ProductName == "Toy 1");

product.UnitPrice = 99;
product.UnitsInStock = 5;

db.SubmitChanges();
```

LINQ providers

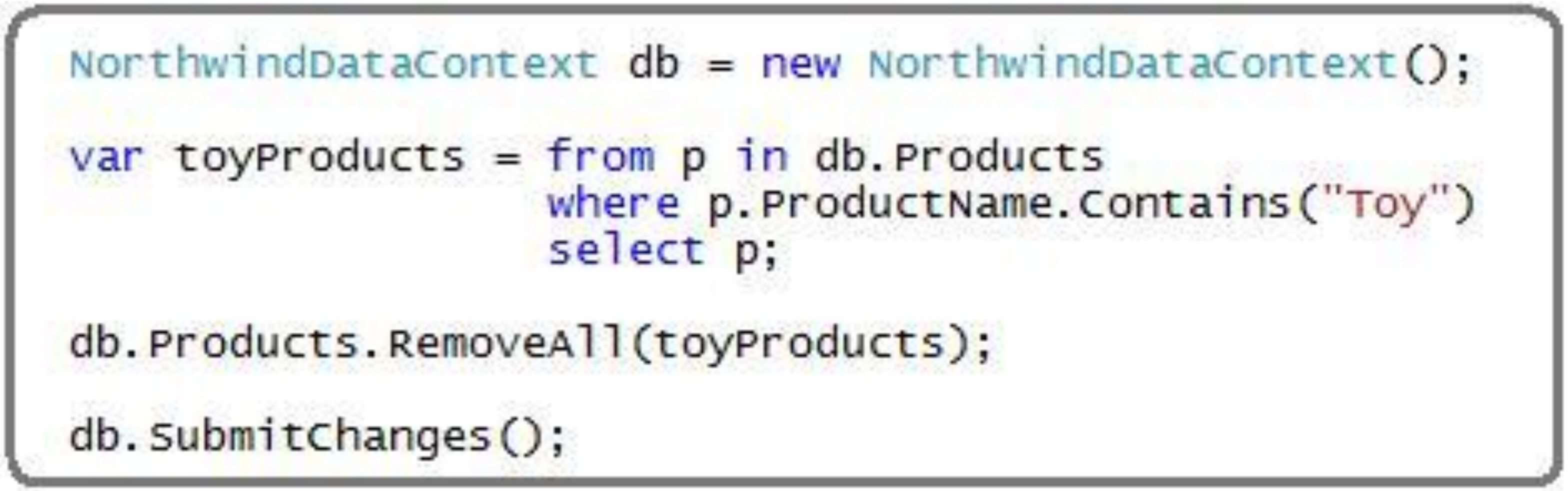
- LINQ to SQL
  - Example:



```
NorthwindDataContext db = new NorthwindDataContext();

// Create new Category and Products

Category category = new Category();
category.CategoryName = "Scott's Toys";

Product product1 = new Product();
product1.ProductName = "Toy 1";

Product product2 = new Product();
product2.ProductName = "Toy 2";

// Associate Products with Category

category.Products.Add(product1);
category.Products.Add(product2);

// Add category to database and save changes

db.Categories.Add(category);
db.SubmitChanges();
```

LINQ providers

- LINQ to SQL
  - Example:



```
NorthwindDataContext db = new NorthwindDataContext();

var toyProducts = from p in db.Products
                  where p.ProductName.Contains("Toy")
                  select p;

db.Products.RemoveAll(toyProducts);

db.SubmitChanges();
```

LINQ providers

- LINQ to DataSets
  - LINQ to DataSet allows developers to write “queries” over existing DataSet sources within applications.
  - Example

```
DataTable orders = ds.Tables["SalesOrderHeader"];
var ordersQuery = orders.ToQueryable();
var query = from o in ordersQuery
            where o.Field<bool>("OnlineOrderFlag") == true
            select new { SalesOrderID = o.Field<int>("SalesOrderID"),
                         OrderDate = o.Field<DateTime>("OrderDate") };
```