Laboratórios de Desenvolvimento de Software - LEI

Language Integrated Queries (LINQ)

Ricardo Santos **Nuno Macedo** Ângelo Costa
nfm@estgf.ipp.pt

November 6, 2013

Language Integrated Queries (LINQ)

- *LINQ* was designed to provide a common programming model for querying data;
- Able to project, filter and sort data from:
  - Objects;
  - XML (LINQ to XML);
  - Databases (LINQ to SQL, LINQ to Entities);
  - Anything for which a LINQ provider is defined.

Preliminaries

- Built over other .NET features:
  - Delegates and lambda expressions;
  - Extension methods;
  - Type inference and anonymous types.

Delegates

- *Delegates* are addresses to methods (compare with *C* pointers);
- Allow methods to take other methods as parameters;
- `Func<T>` is a predefined delegate that represents functions;
- E.g., `Func<string,bool>` is a function from strings to booleans:

```
Func<string,bool> f = delegate(string x) {
    return x == "hello";
}
```

Lambda expressions

- *Lambda expressions* are used to define anonymous delegates;
- Syntax is simpler than defined delegates;

```
Func<string,bool> f = x => x == "hello";
```

Extension methods

- Classes can be extended with methods that were not defined;

```
public static class Enumerable {
    public static IEnumerable<T> Where<T>(this IEnumerable<T>
        source, Func<T, bool> predicate) {
        foreach (T item in source) {
           if (predicate(item))
               yield return item;
        }
    }
}
```

- Can be used as:

```
IEnumerable<string> s = ... ;
Func<string,bool> p = ... ;
s.Where(p);
```

- The original class is not modified, it is simply rewritten as

```
Enumerable.Where(s,p);
```

Anonymous types

- Objects can have anonymous types defined by the keyword `var`;

```
var named_person = { Name = "name", Age = 20 };
```

- The concrete nameless classes that are inferred at compile-time;
- Avoids creating new classes when performing projections;

```
Person p = ... ;
var named_person = { p.Name = "name", p.Age = 20 };
```

Pre-LINQ queries

- Performing queries may be a very verbose task;
- The previous features somehow simplify the process.

Pre-LINQ queries

```
private void OldStyleQuery() {
  Customer[] customers = BuildCustomers();
  List<SearchResult> results = new List<SearchResults>();
  SearchForProduct matcher = new SearchForProduct() { "Milk" };
  foreach (Customer c in custmores) {
    if (c.FirstName.Length >= 5) {
      Order[] orders = Array.FindAll(c.Orders, matcher.ProductMatch);
      if (orders.Length > 0) {
        SearchResult cr = new SearchResult();
        cr.Customer = c.FirstName + " " + c.LastName;
        foreach (Order o in orders) {
          cr.Quantity += o.Quantity;
          cr.Count++;  }
        results.Add(cr);
      }
    }
  }
  results.Sort(CompareSearchResults);
}
```

Pre-LINQ queries

- Using lambda expressions:

```
private void OldStyleQuery() {
  Customer[] customers = BuildCustomers();
  List<SearchResult> results = new List<SearchResults>();
  foreach (Customer c in custmores) {
    if (c.FirstName.Length >= 5) {
      Order[] orders = Array.FindAll(c.Orders, order=>order.Product ==
      if (orders.Length > 0) {
        SearchResult cr = new SearchResult();
        cr.Customer = c.FirstName + " " + c.LastName;
        foreach (Order o in orders) {
          cr.Quantity += o.Quantity;
          cr.Count++;  }
        results.Add(cr);
      }
    }
  }
  results.Sort((r1,r2)=>string.Compare(r1.Name,r2.Name));
}
```

Pre-LINQ queries

- Using extension methods:

```
private void OldStyleQuery() {
  Customer[] customers = BuildCustomers();
  List<SearchResult> results = new List<SearchResults>();
  foreach (Customer c in custmores) {
    if (c.FirstName.Length >= 5) {
      Order[] orders = c.Orders.Where(order=>order.Product == "Milk");
      if (orders.Length > 0) {
        SearchResult cr = new SearchResult();
        cr.Customer = c.FirstName + " " + c.LastName;
        foreach (Order o in orders) {
          cr.Quantity += o.Quantity;
          cr.Count++;  }
        results.Add(cr);
      }
    }
  }
  results.Sort((r1,r2)=>string.Compare(r1.Name,r2.Name));
}
```

LINQ

- Importing the `System.Linq` namespace provides extension methods for `IEnumerable<T>` objects;
- `Where`, `Select`, `OrderBy`, ...
- Combined in statements that define the LINQ query syntax.
- Rely on lambda expressions;
- The result is an anonymous type containing the selected properties;

LINQ

- Using extension methods:

```
c.Orders.Where(order=>order.Product == "Milk");
```

- ... becomes:

```
from o in c.Orders
where order.Product == "Milk"
select o;
```

Operations

- *From*: defines the source collection and the iteration variable;
- *Select*: projects information;
- *Where*: filters results;
- *GroupBy*: groups results;
- *OrderBy*: sorts results;

LINQ example

```
private void LinqQuery() {
  var customers = BuildCustomers();
  var results = from c in customers
                from o in c.Orders
                orderby c.FirstName
                where o.Product = "Milk" &&
                      c.FirstName.Length >= 5
                select new { c.FirstName, c.LastName,
                             o.Product, o.Quantity };
}
```

LINQ example

```
private void LinqQuery() {
  var customers = BuildCustomers();
  var results = from c in customers
                from o in c.Orders
                orderby c.FirstName
                where o.Product = "Milk" &&
                      c.FirstName.Length >= 5
                group o by c into avg
                select new { avg.Key.FirstName, avg.Key.LastName,
                             avg = avg.Average(o => o.Quantity) };
}
```

LINQ to SQL

- LINQ can be mapped to SQL:
- Yet another way to connect .NET with databases;
- Allows the representation of the database as object diagrams;
- However, the mapping is 1:1 (not customizable);
- Superseded by the Entities Framework.

Bibliography


N. Randolph, D. Gardner, M. Minutillo and C. Anderson
Professional Visual Studio 2010.
Wrox, 2010.


Nagel, C., Evjen, B., Glynn, J., Watson, K. and Skinner, M.
Professional C# 4 and .NET 4.
Wrox, 2010.