

Supplement 2

Language Integrated Query (LINQ)

Language Integrated Query (LINQ)

Objectives

After completing this unit you will be able to:

- **Describe Language Integrated Query (LINQ).**
- **Use query keywords in C# to write queries against data sources.**
- **Describe the basic elements of LINQ query syntax, including:**
 - Selection
 - Projection
 - Filtering
 - Ordering
 - Grouping
 - Aggregates
- **Explain the use of deferred execution of queries.**

Language-Integrated Query (LINQ)

- The C# language has support for Language-Integrated Query (LINQ), whereby you can use query keywords in C# to write queries against data sources.
- Here is a simple complete example.
 - See SimpleQuery.

```
using System;
using System.Linq;

...

static void Main(string[] args)
{
    int[] primes = { 2, 3, 5, 7, 11, 13, 17, 19 };
    ShowArray(primes);
    var query =
        from num in primes
        where num < 10
        select num;
    foreach (int x in query)
    {
        Console.Write("{0} ", x);
    }
    Console.WriteLine();
}

static void ShowArray(int[] arr)
{
    foreach (int x in arr)
    {
        Console.Write("{0} ", x);
    }
    Console.WriteLine();
}
```

LINQ Example

- **The query on the preceding page selects all numbers less than 10 from an array of primes.**

```
var query =  
    from num in primes  
    where num < 10  
    select num;
```

- The keywords **from**, **where**, and **select** define the query using an intuitive SQL-like notation.
- More complicated queries can be created using additional keywords, such as **group**, **orderby**, and **join**.
- **The query is not executed until you iterate over the query using a *foreach* loop.**

```
foreach (int x in query)  
{  
    Console.WriteLine("{0} ", x);  
}
```

- **Note use of the *var* keyword to simplify writing the query.**

Using IEnumerable<T>

- Let's redo the example without using *var*.
 - See **EnumerableQuery**.

```
using System;
using System.Collections.Generic;
using System.Linq;

...

static void Main(string[] args)
{
    int[] primes = { 2, 3, 5, 7, 11, 13, 17, 19 };
    ShowArray(primes);
    IEnumerable<int> query =
        from num in primes
        where num < 10
        select num;
    ShowArray(query.ToArray<int>());
}
static void ShowArray(int[] arr)
{
    foreach (int x in arr)
    {
        Console.Write("{0} ", x);
    }
    Console.WriteLine();
}
```

- We can reuse our **ShowArray()** method by obtaining an array from the **IEnumerable<int>**.
- Both examples produce this output:

```
2 3 5 7 11 13 17 19
2 3 5 7
```

Basic LINQ Query Operators

- **The table summarizes the most important LINQ query operators, which have corresponding C# keywords.**

from in	specifies a data source
select	selects a sequence from the data source
where	filters the selected data according to some criterion
orderby ascending descending	sorts the selected data
group by	groups the selected data based on a specified key
join on equals into	performs various kinds of joins

Obtaining a Data Source

- **The first thing you need to do in a LINQ query is obtain a data source, which you do with the *from ... in* keywords.**

- In this course, the data source is a .NET collection class:

```
List<Book> books;  
...  
var query = from bk in books  
            select bk;
```

- The data source could be an entity class for accessing a table in a SQL Server database:

```
BookClassesDataContext dc =  
    new BookClassesDataContext();  
var query = from bk in dc.Books  
            select bk;
```

- It could also be an XML document:

```
root = XElement.Load(@"\OIC\Data\Books.xml");  
...  
var query = from bk in root.Elements("Book")  
            select bk;
```

- **In these examples, *bk* is the *range variable*.**

- The range variable serves as a reference to each selected element.

```
foreach (var bk in query)  
    ...
```

LINQ Query Example

- The example program *QueryLinq* illustrates a number of basic query operations.

```
private static List<Book> books;

static void Main(string[] args)
{
    books = Book.InitBooks();
    all();
    ...
}

private static void ShowBook(Book bk)
{
    Console.WriteLine(bk.ToString());
}

private static void all()
{
    var query = from bk in books
                select bk;
    foreach (var bk in query)
        ShowBook(bk);
}
```

- Here is the output:

```
$29.99  2009  WPF in Action
$24.99  2009  Sivlerlight 3 Unleashed
$24.99  2005  Software Testing
$39.99  2005  Introduction to SQL
$35.00  2007  JavaServer Pages
$40.00  2008  Introduction to Java
$45.00  2004  PHP Programming
$35.00  2010  C# Programming
$30.00  2003  XML Fundamentals
```


Filtering

- **With a *where* clause you can filter the result data according to a Boolean expression.**
 - The query returns only the elements for which the Boolean expression is true.
- **This example selects books published before 2009 costing more than \$25.00.**

```
private static void filter()
{
    Console.WriteLine("filter");
    var query = from bk in books
                where bk.Year < 2009
                   && bk.Price > 25m
                select bk;
    foreach (var bk in query)
        ShowBook(bk);
}
```

- **Here is the output:**

```
filter
$39.99  2005  Introduction to SQL
$35.00  2007  JavaServer Pages
$40.00  2008  Introduction to Java
$45.00  2004  PHP Programming
$30.00  2003  XML Fundamentals
```

Ordering

- With an *orderby* clause you can sort the result data into ascending or descending order.
- This example sorts books published after 2004 in descending order by price.

```
private static void order()
{
    Console.WriteLine("order");
    var query = from bk in books
                where bk.Year > 2004
                orderby bk.Price descending
                orderby bk.Price descending
                select bk;
    foreach (var bk in query)
        ShowBook(bk);
}
```

- Here is the output:

```
order
$40.00  2008  Introduction to Java
$39.99  2005  Introduction to SQL
$35.00  2007  JavaServer Pages
$35.00  2010  C# Programming
$29.99  2009  WPF in Action
$24.99  2009  Sivlerlight 3 Unleashed
$24.99  2005  Software Testing
```

Aggregation

- Besides obtaining the individual items in a result set, **LINQ** also supports common aggregation operators, such as:
 - Count
 - Sum
 - Max
 - Min
- The example obtains these aggregate values for Price.

```
private static void aggregate()
{
    Console.WriteLine("aggregate");
    var query = from bk in books
                select bk.Price;
    Console.WriteLine("Count = {0}", query.Count());
    Console.WriteLine("Sum = {0}", query.Sum());
    Console.WriteLine("Max = {0}", query.Max());
    Console.WriteLine("Min = {0}", query.Min());
}
```

- Here is the output:

```
aggregate
Count = 9
Sum = 304.96
Max = 45.00
Min = 24.99
```

Obtaining Lists and Arrays

- LINQ select queries return an *IEnumerable<T>*.
- *IEnumerable<T>* has extension methods to create an array or a list from the query result.
 - **ToArray<T>** creates an array of elements of type T.
 - **ToList<T>** creates a list of type **List<T>**.
- Our example program illustrates obtaining an array of books and displaying the first and last elements.

```
private static void array()
{
    Console.WriteLine("array");
    var query = from bk in books
                select bk;
    Book[] array = query.ToArray<Book>();
    int count = array.Length;
    Console.WriteLine("{0} books", count);
    Console.WriteLine("First book:");
    ShowBook(array[0]);
    Console.WriteLine("Last book:");
    ShowBook(array[count - 1]);
}
```

- Here is the output:

```
array
9 books
First book:
$29.99  2009  WPF in Action
Last book:
$30.00  2003  XML Fundamentals
```

Deferred Execution

- The statement creating a LINQ query does not cause the query to be actually executed.
- The execution is deferred until some operation is performed that uses the results of the query, such as:
 - Iterating over the result set in a **foreach** loop.

```
foreach (var bk in query)  
    ShowBook(bk);
```

- Obtaining an aggregate such as **Count()** or **Sum()** from the result set.

```
Console.WriteLine("Count = {0}", query.Count());
```

- Converting the result set to an array or list.

```
Book[] array = query.ToArray<Book>();
```

Summary

- **With Language-Integrated Query (LINQ) you can use query keywords in C# to write queries against data sources.**
- **LINQ provides a consistent query syntax, including these features:**
 - Selection
 - Projection
 - Filtering
 - Ordering
 - Grouping
 - Aggregates
- **The execution of a query is deferred until you iterate over the elements of the result set, convert the result to an array or list, find an aggregate, and so on.**