Chapter 6
LINQ

Mrs. Swati Satpute
Fergusson College

Objective

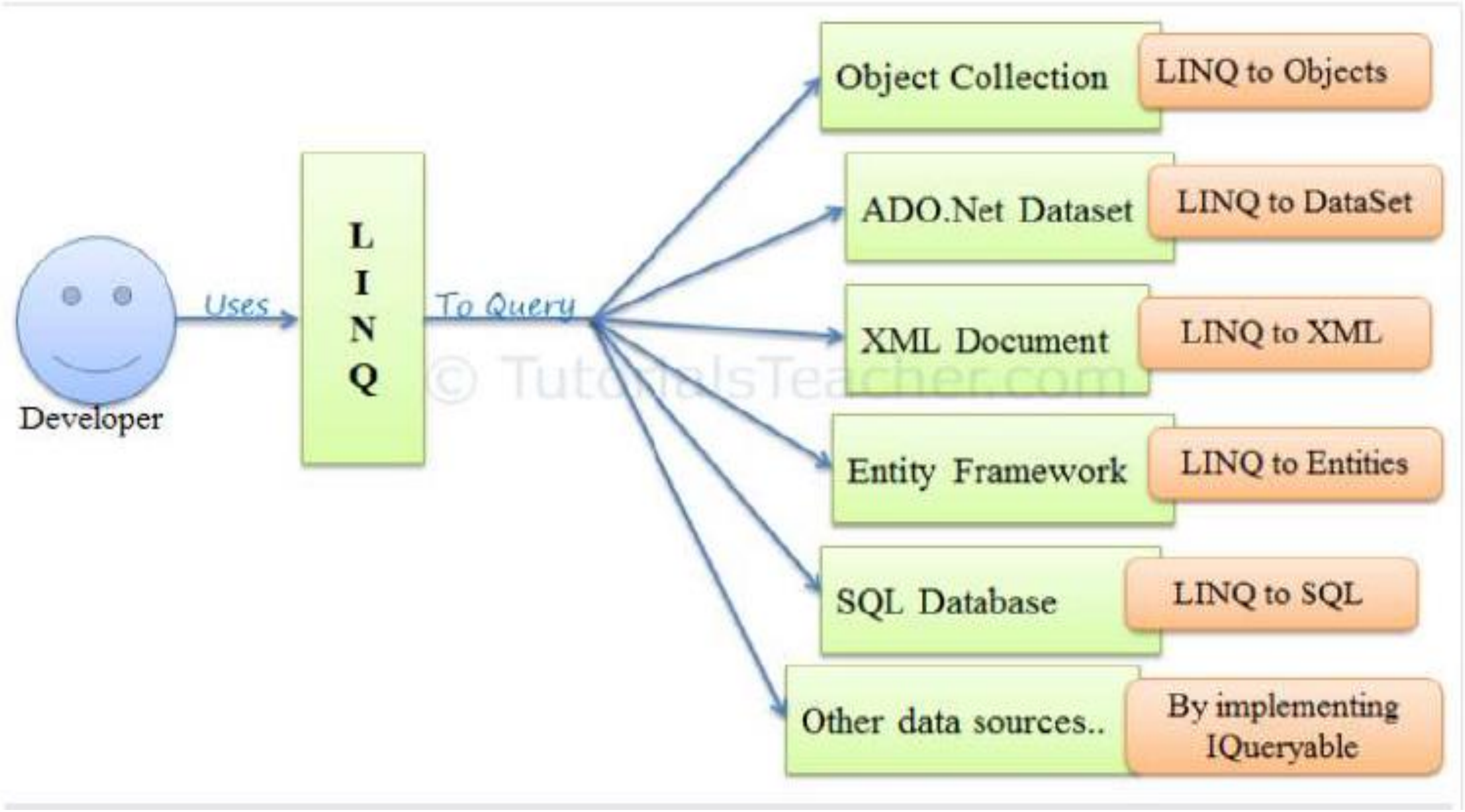
- RDBMS
- Entity Framework
- LINQ to SQL
- What is LINQ ?
- LINQ Architecture
- LINQ to Objects
- LINQ to Objects –Querying Collections

LINQ

- Language Integrated Query

- LINQ is a **query syntax** built in C# and VB.NET used to save and retrieve data from **different types of data sources like an Object Collection, SQL server database, XML, web service etc.**

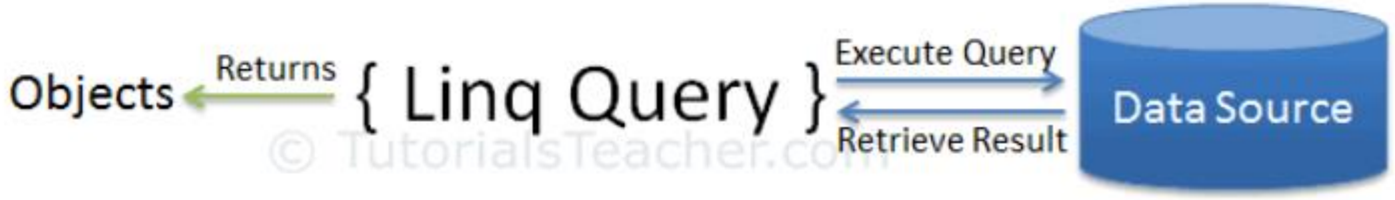
LINQ usage



Developer
Uses
L I N Q
To Query
Object Collection
LINQ to Objects
ADO.Net Dataset
LINQ to DataSet
XML Document
LINQ to XML
Entity Framework
LINQ to Entities
SQL Database
LINQ to SQL
Other data sources..
By implementing IQueryable
© TutorialsTeacher.com

LINQ usage

- LINQ queries return **results as objects**.
- It **enables to use object-oriented approach** on the result set and not to worry about transforming different formats of results into objects.



Simple LINQ Query

```
public static void SimpleLINQQuery()
{

    string[] words = { "hello", "wonderful", "LINQ", "beautiful", "world" };

    //Get only short words
    var shortWords = from word in words where word.Length <= 5 select word;

    //Print each word out
    foreach (var shword in shortWords)
    {
        Console.WriteLine(shword);
    }

    Console.ReadLine();
}
```

Advantages

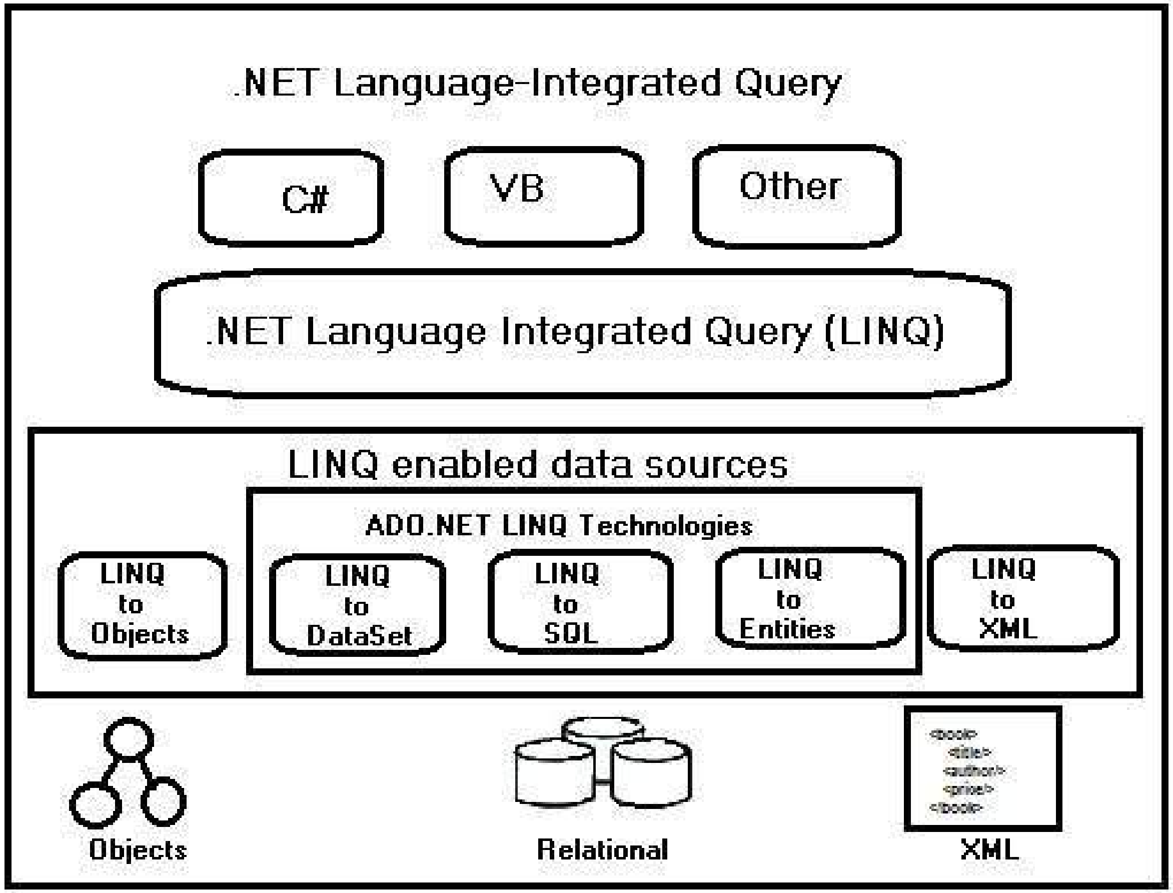
- **Familiar language:** Developers don’t have to learn a new query language for each type of data source or data format.
- **Syntax highlighting** that proves helpful to find out mistakes during design time.
- Easy debugging
- Extensible that means it is **possible to query new data source types.**
- Facility of joining several data sources in a single query
- **Easy transformation** (like transforming SQL data to XML data.)
- **Shaping data:** You can retrieve data in different shapes.

Advantages

- **Compile time safety of queries:** It provides type checking of objects at compile time

LINQ Architecture in .NET

- LINQ has a 3-layered architecture in which
  - the **uppermost layer** consists of the **language extensions**
  - the **bottom laye**r consists of **data sources** that are typically objects implementing IEnumerable <T> or IQueryable <T> generic interfaces.

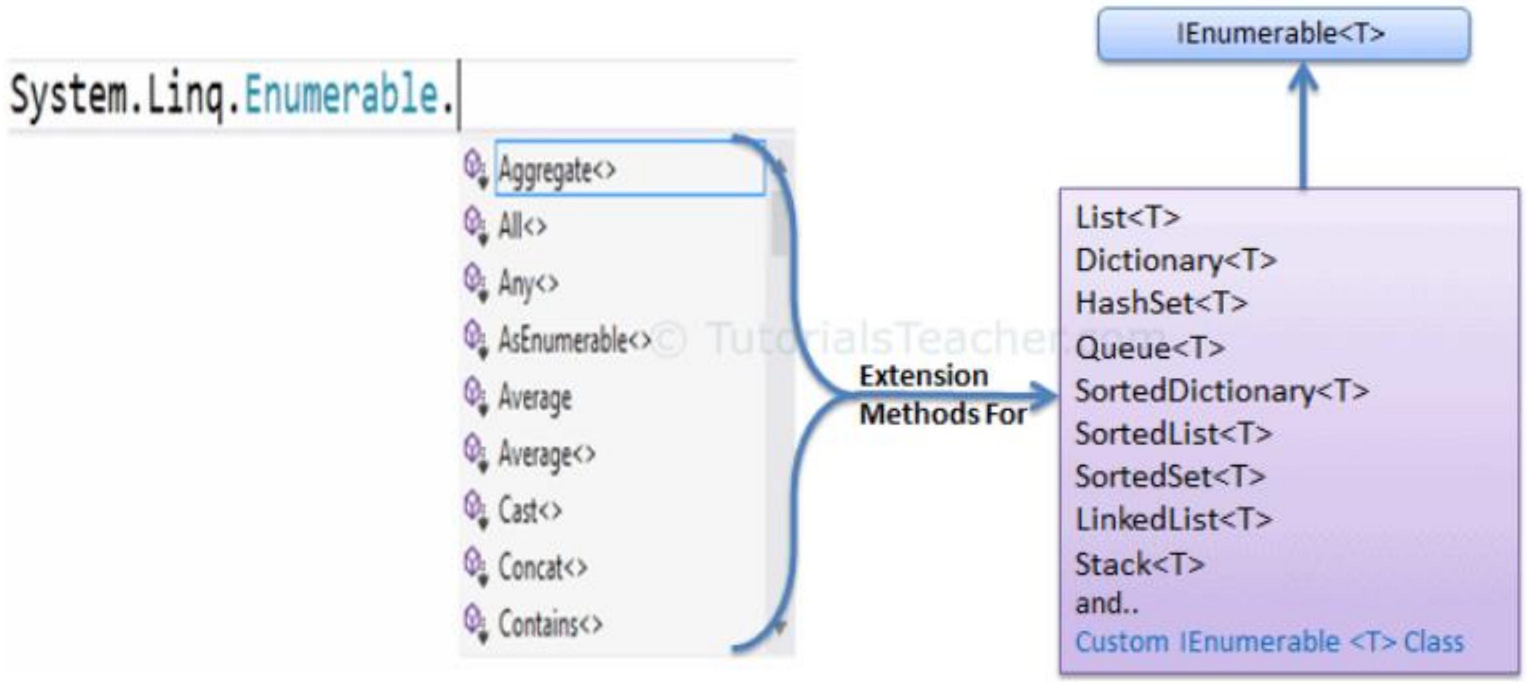

.NET Language-Integrated Query
C#
VB
Other
.NET Language Integrated Query (LINQ)
LINQ enabled data sources
ADO.NET LINQ Technologies
LINQ to Objects
LINQ to DataSet
LINQ to SQL
LINQ to Entities
LINQ to XML
Objects
Relational
XML

LINQ API

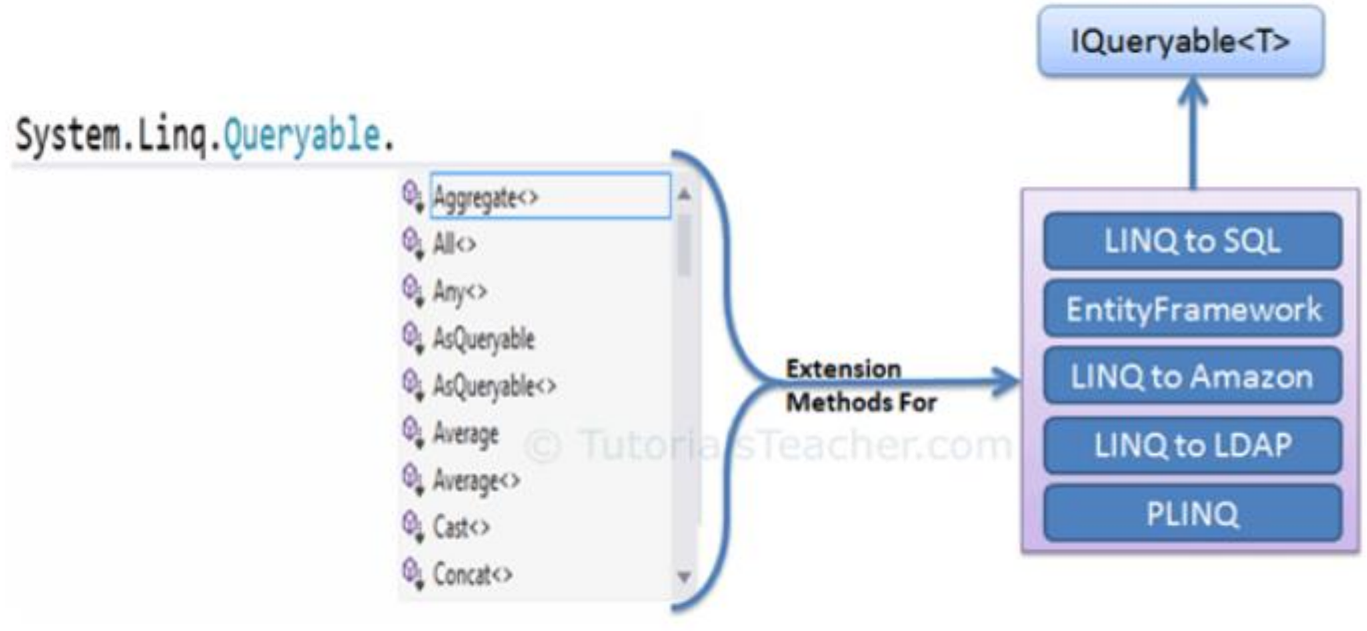
- We can write LINQ queries for the classes that implement IEnumerable<T> or IQueryable<T> interface.
- Namespace: System.Linq
- LINQ queries uses extension methods for classes that implement **IEnumerable** or **IQueryable** interface.

- The **Enumerable** and **Queryable** are two static classes that contain extension methods to write LINQ queries.

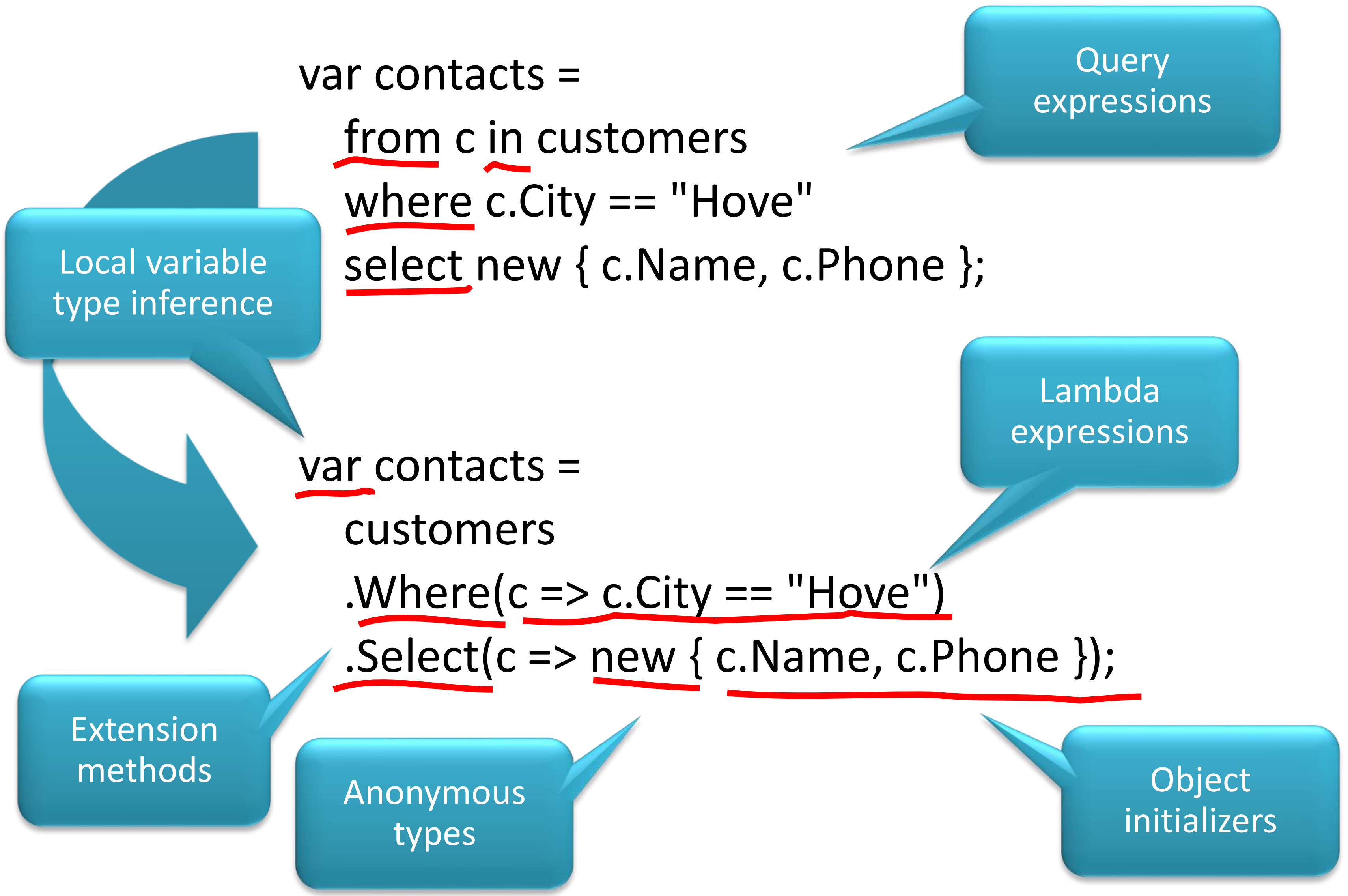
Classes implementing IEnumerable



Classes implementing IQueryable



Language Innovations



```
var contacts =
    from c in customers
    where c.City == "Hove"
    select new { c.Name, c.Phone };
```

```
var contacts =
    customers
    .Where(c => c.City == "Hove")
    .Select(c => new { c.Name, c.Phone });
```

LINQ Query Syntax

1. **Query Syntax** or Query Expression Syntax
2. **Method Syntax** or Method Extension Syntax or Fluent

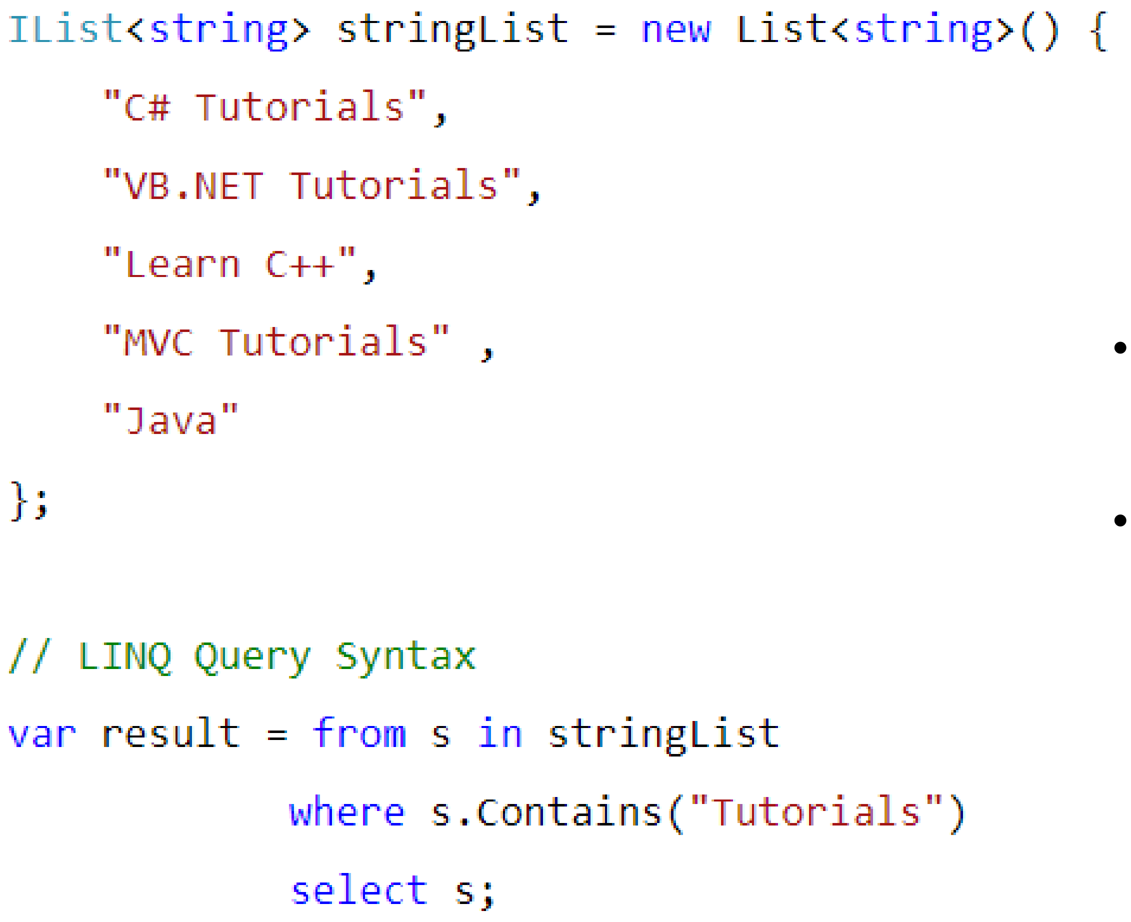
Query Syntax

from *<range variable>* in *<IEnumerable<T> or IQueryable<T> Collection>*

<Standard Query Operators> *<lambda expression>*

<select or groupBy operator> *<result formation>*

Query Syntax



```
IList<string> stringList = new List<string>() {
    "C# Tutorials",
    "VB.NET Tutorials",
    "Learn C++",
    "MVC Tutorials" ,
    "Java"
};

// LINQ Query Syntax
var result = from s in stringList
            where s.Contains("Tutorials")
            select s;
```

- Where and select are standard Query operators.
- The Select clause is used to shape the data.

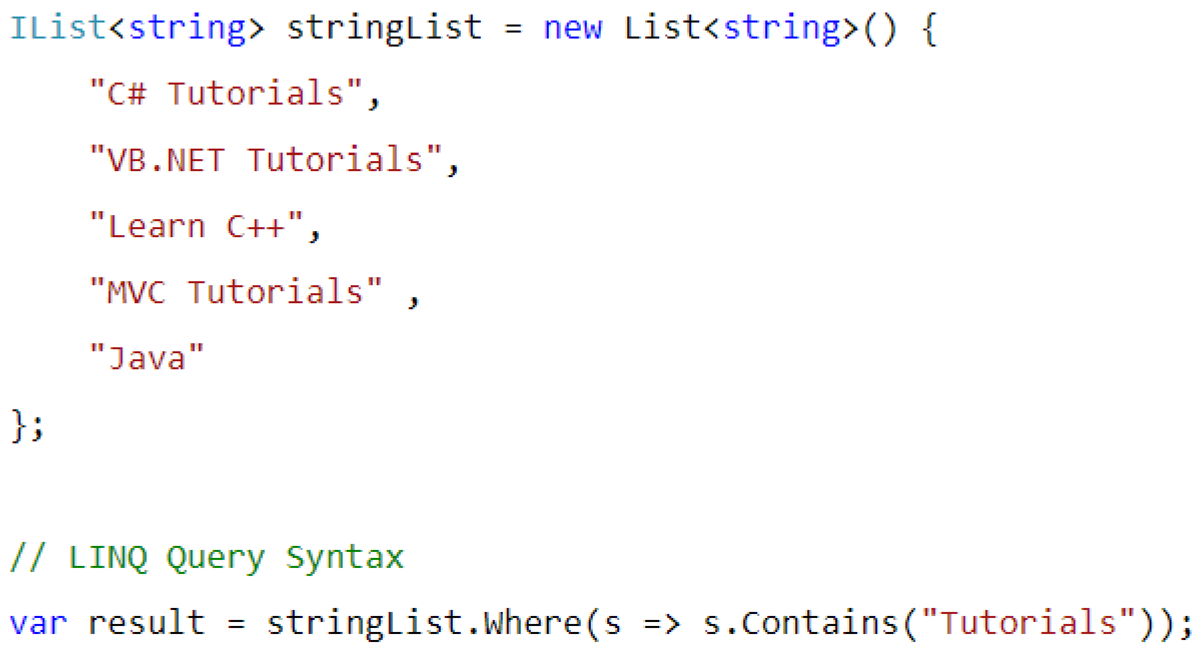
LINQ query syntax
- Starts with From clause
- always ends with a Select or Group clause.

Method Syntax

- Method syntax (also known as fluent syntax) **uses extension methods** included in the Enumerable or Queryable static class.

- The **compiler converts** query syntax into method syntax at **compile time**.

Method Syntax

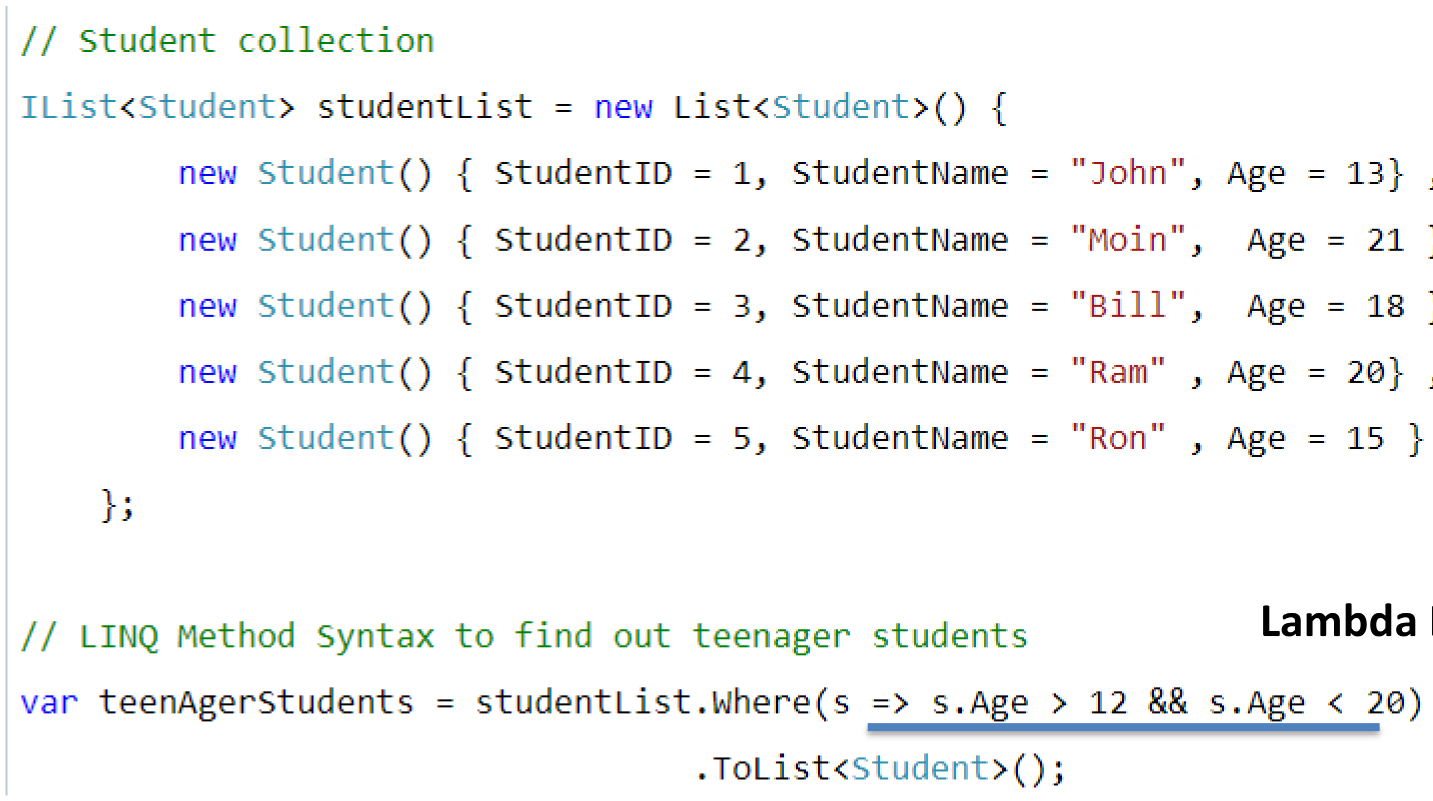


```
IList<string> stringList = new List<string>() {
    "C# Tutorials",
    "VB.NET Tutorials",
    "Learn C++",
    "MVC Tutorials" ,
    "Java"
};

// LINQ Query Syntax
var result = stringList.Where(s => s.Contains("Tutorials"));
```

Where () accepts a delegate as Func<Student, bool>, student is an input object and returns a bool value

Method Syntax example



```
// Student collection
IList<Student> studentList = new List<Student>() {
        new Student() { StudentID = 1, StudentName = "John", Age = 13} ,
        new Student() { StudentID = 2, StudentName = "Moin",  Age = 21 } ,
        new Student() { StudentID = 3, StudentName = "Bill",  Age = 18 } ,
        new Student() { StudentID = 4, StudentName = "Ram" , Age = 20} ,
        new Student() { StudentID = 5, StudentName = "Ron" , Age = 15 }
    };

// LINQ Method Syntax to find out teenager students
var teenAgerStudents = studentList.Where(s => s.Age > 12 && s.Age < 20)
                                  .ToList<Student>();
```

Lambda Expression

Syntax of LINQ

- Query

  var longwords = from w in words where w.length > 10;

- Method (Lambda Expression)

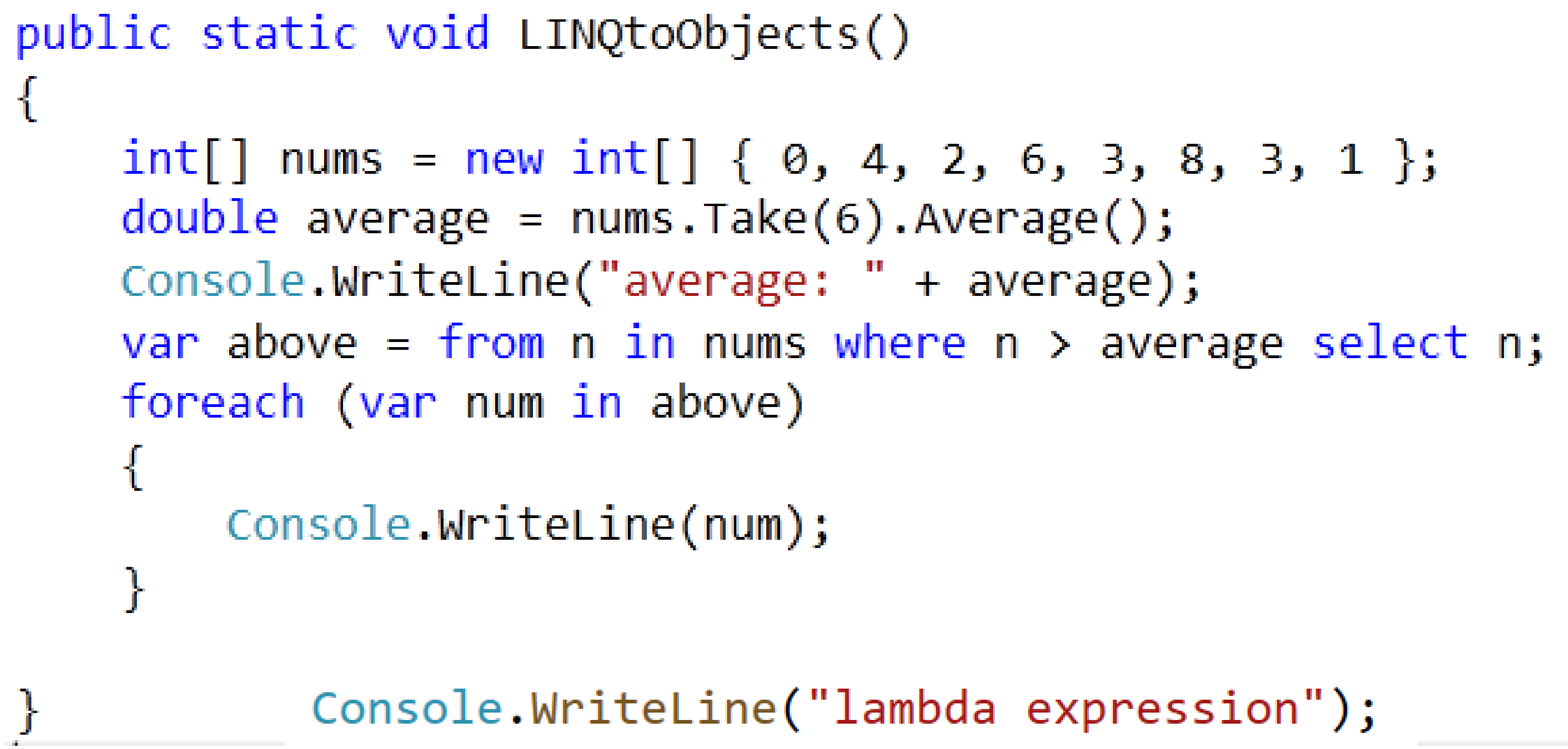
  var longWords = words.Where(w ⇒ w.length > 10);

Extension Methods

Lambda Expression

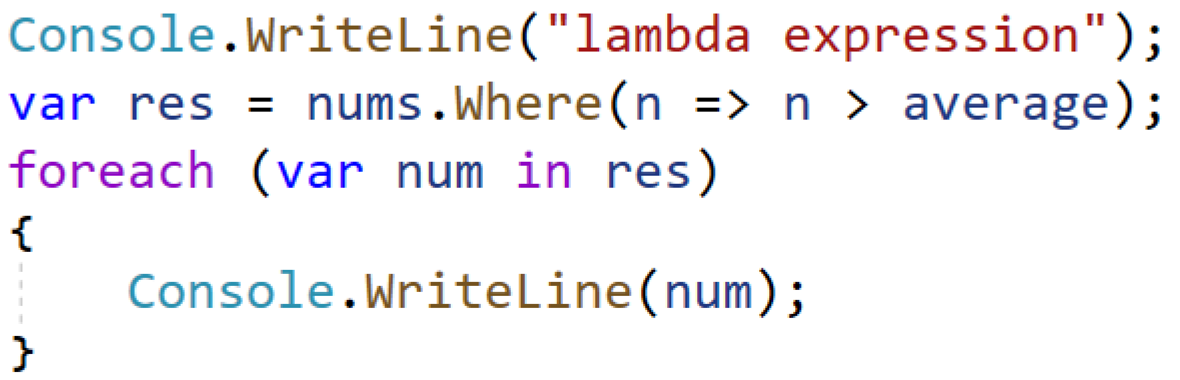
Implicitly typed variable - **var** can be used to hold the result of the LINQ query.

LINQ to Objects



```
public static void LINQtoObjects()
{
    int[] nums = new int[] { 0, 4, 2, 6, 3, 8, 3, 1 };
    double average = nums.Take(6).Average();
    Console.WriteLine("average: " + average);
    var above = from n in nums where n > average select n;
    foreach (var num in above)
    {
        Console.WriteLine(num);
    }

}
```

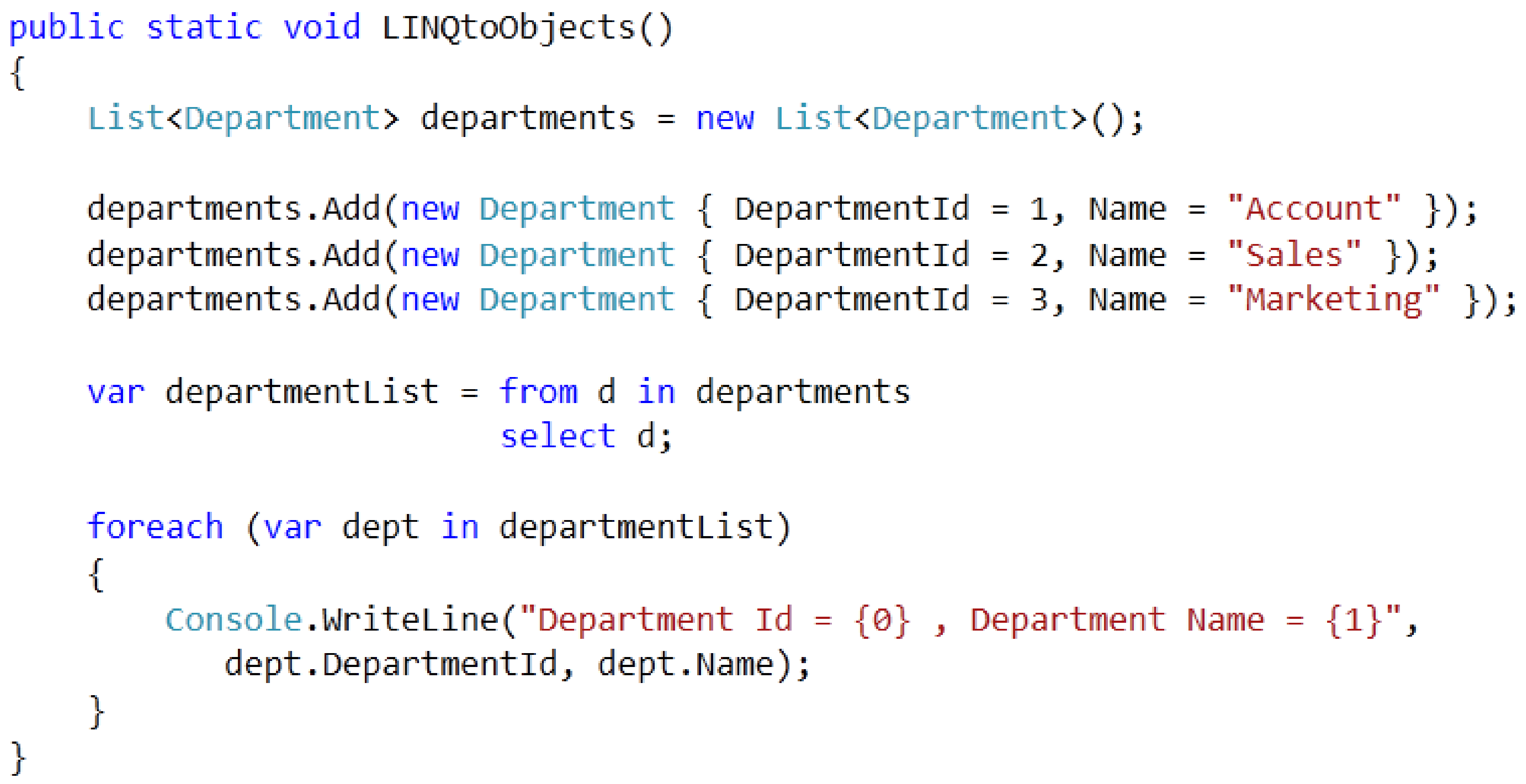


```
Console.WriteLine("lambda expression");
var res = nums.Where(n => n > average);
foreach (var num in res)
{
    Console.WriteLine(num);
}
```

LINQ to Objects

- Query any IEnumerable<T> source
  Includes arrays, List<T>, Dictionary...
- Many useful operators available
  Sum, Max, Min, Distinct, Intersect, Union
- Expose your own data with IEnumerable<T> or IQueryable<T>
- Create operators using extension methods

Querying in Memory Collections Using LINQ to Objects



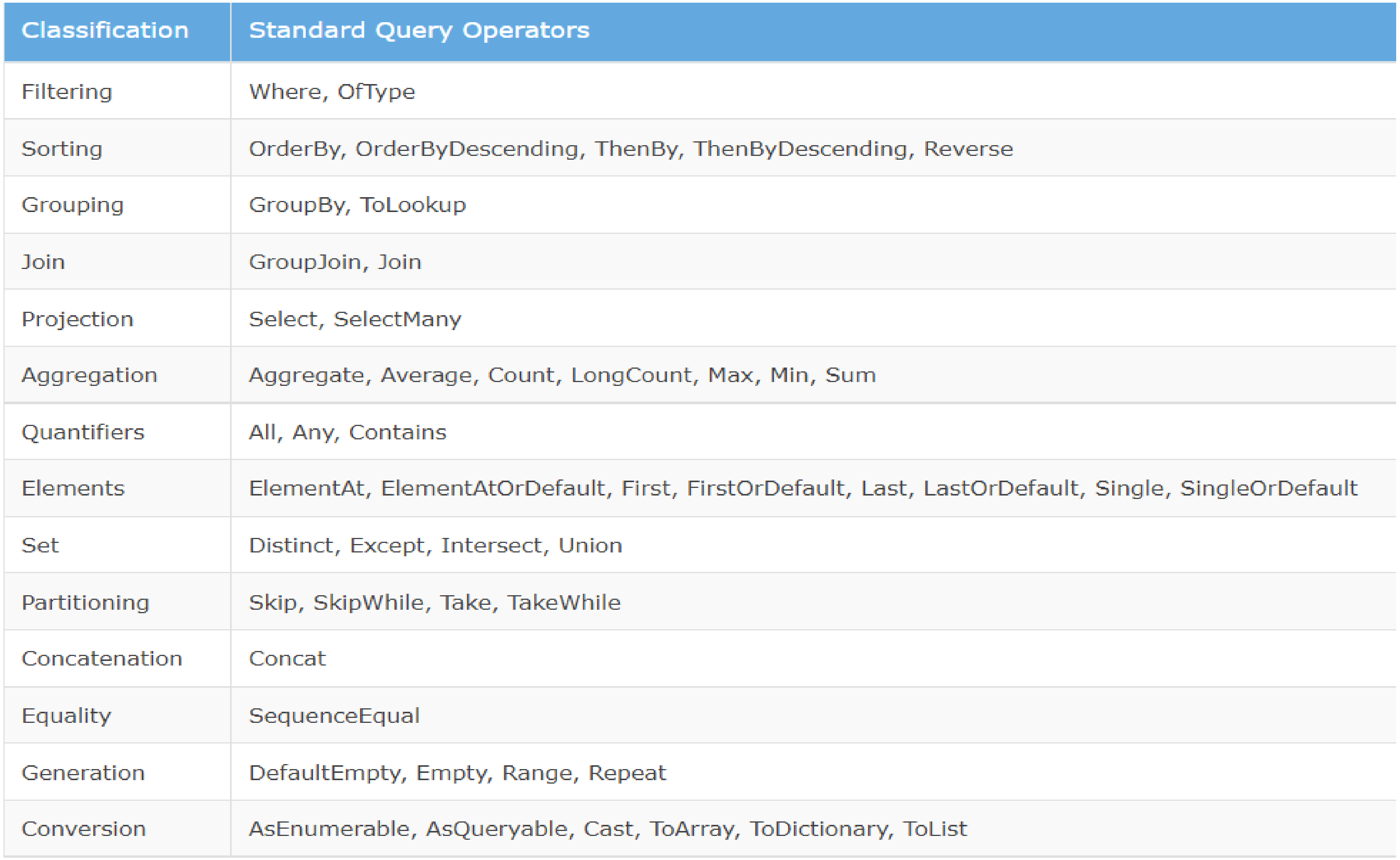
```
public static void LINQtoObjects()
{
    List<Department> departments = new List<Department>();

    departments.Add(new Department { DepartmentId = 1, Name = "Account" });
    departments.Add(new Department { DepartmentId = 2, Name = "Sales" });
    departments.Add(new Department { DepartmentId = 3, Name = "Marketing" });

    var departmentList = from d in departments
                         select d;

    foreach (var dept in departmentList)
    {
        Console.WriteLine("Department Id = {0} , Department Name = {1}",
           dept.DepartmentId, dept.Name);
    }
}
```

LINQ operators



Classification	Standard Query Operators
Filtering	Where, OfType
Sorting	OrderBy, OrderByDescending, ThenBy, ThenByDescending, Reverse
Grouping	GroupBy, ToLookup
Join	GroupJoin, Join
Projection	Select, SelectMany
Aggregation	Aggregate, Average, Count, LongCount, Max, Min, Sum
Quantifiers	All, Any, Contains
Elements	ElementAt, ElementAtOrDefault, First, FirstOrDefault, Last, LastOrDefault, Single, SingleOrDefault
Set	Distinct, Except, Intersect, Union
Partitioning	Skip, SkipWhile, Take, TakeWhile
Concatenation	Concat
Equality	SequenceEqual
Generation	DefaultEmpty, Empty, Range, Repeat
Conversion	AsEnumerable, AsQueryable, Cast, ToArray, ToDictionary, ToList

LINQ operators

Demo

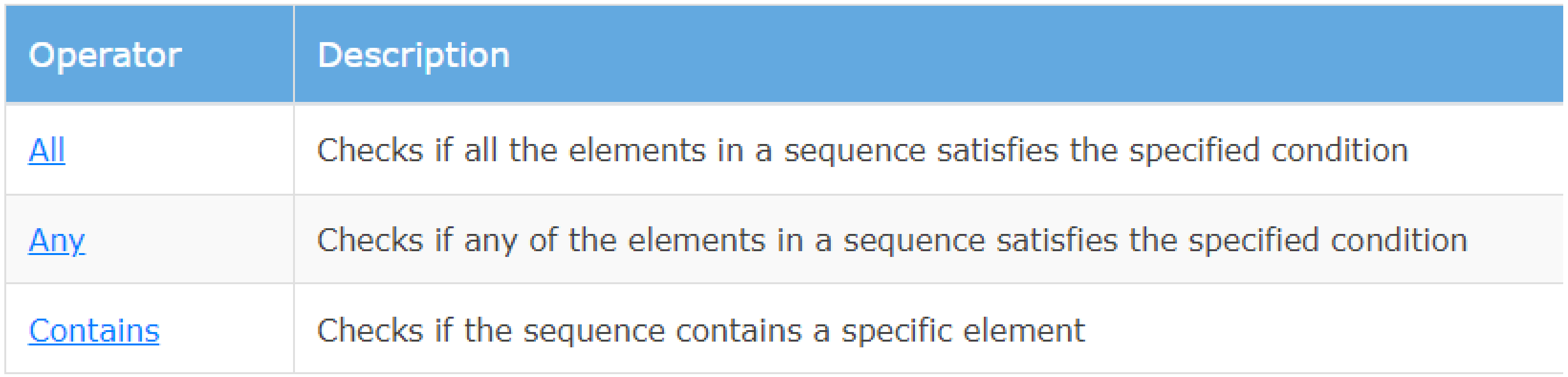
- **Where** is used for filtering the collection based on given criteria.
- The **OfType** operator filters the collection **based on the ability to cast an element in a collection to a specified type.**
- **OrderBy, ThenBy**

Projection Operators: Select, SelectMany

Demo

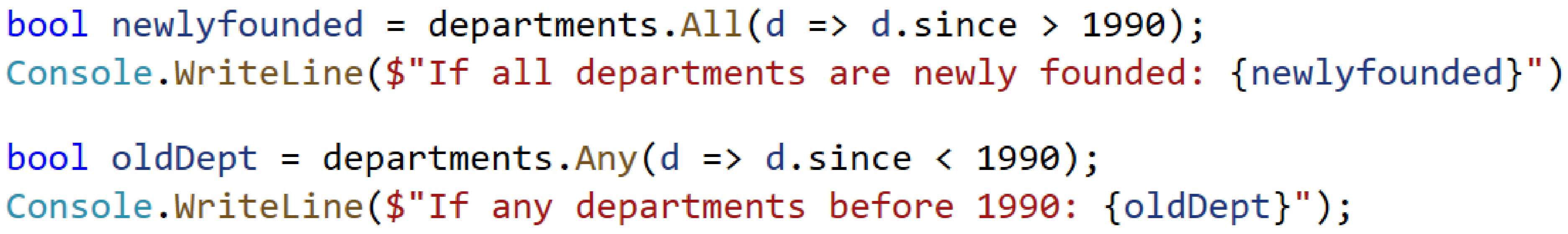
- The **Select** operator always returns an IEnumerable collection which contains elements based on a transformation function.
- It is similar to the Select clause of SQL that produces a flat result set.

Quantifier Operators



Operator	Description
All	Checks if all the elements in a sequence satisfies the specified condition
Any	Checks if any of the elements in a sequence satisfies the specified condition
Contains	Checks if the sequence contains a specific element

- Returns Boolean value as a result



```
bool newlyfounded = departments.All(d => d.since > 1990);
Console.WriteLine($"If all departments are newly founded: {newlyfounded}")

bool oldDept = departments.Any(d => d.since < 1990);
Console.WriteLine($"If any departments before 1990: {oldDept}");
```

Aggregate

- Aggregate method performs an accumulate operation

Demo - Assignment

- Query on Strings
- Query on Integers
- OfType operator
- Query on Objects collection – Solve

SQL vs LINQ

- LINQ is **IN**tegrated with C# (or VB), so eliminating mismatch between programming languages and databases.
- Provides a single querying interface for a multitude of data sources.
- LINQ is in most cases a **significantly more productive querying language** than SQL.
- LINQ is simpler, tidier, and *higher-level*.

SQL vs LINQ

- Simple query to retrieve customer, name starting with ‘A’
  - Retrieve 10 records between 21-30 (paging e.g.)

```
SELECT TOP 10 UPPER (c1.Name)
FROM Customer c1
WHERE
   c1.Name LIKE 'A%'
   AND c1.ID NOT IN
   (
      SELECT TOP 20 c2.ID
      FROM Customer c2
      WHERE c2.Name LIKE 'A%'
      ORDER BY c2.Name
   )
ORDER BY c1.Name
```

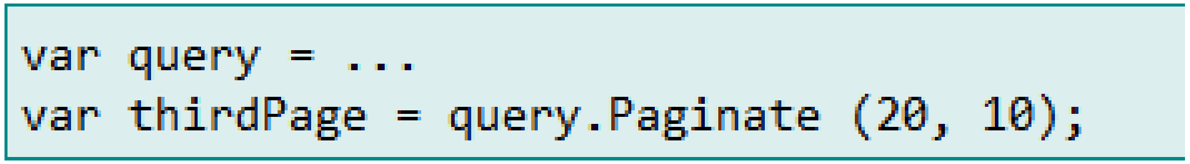
SQL vs LINQ

Simplicity in LINQ

```
var query =
   from c in db.Customers
   where c.Name.StartsWith ("A")
   orderby c.Name
   select c.Name.ToUpper();

var thirdPage = query.Skip(20).Take(10);
```

Composibility in LINQ: Code can be composed in two steps
- Query
- pagination logic



```
var query = ...
var thirdPage = query.Paginate (20, 10);
```

Associations

Use case: List all purchases of $1000 or greater made by customers who live in Washington. Purchases are itemized. Include cash sales (with no customer). This requires querying across four tables (Purchase, Customer, Address and PurchaseItem)

SQL

```
SELECT p.*
FROM Purchase p
   LEFT OUTER JOIN
      Customer c INNER JOIN Address a ON
c.AddressID = a.ID
   ON p.CustomerID = c.ID
WHERE
  (a.State = 'WA' || p.CustomerID IS NULL)
   AND p.ID in
   (
      SELECT PurchaseID FROM PurchaseItem
      GROUP BY PurchaseID HAVING SUM
(SaleAmount) > 1000
   )
```

LINQ

LINQ is can query across relationships without having to join

```
from p in db.Purchases
where p.Customer.Address.State
== "WA" || p.Customer == null
where p.PurchaseItems.Sum (pi =>
pi.SaleAmount) > 1000
select p
```

Shaping Data - LINQ

- LINQ lets you retrieve shaped or hierarchical data.
- Obviates the need for joining tables.

Use case: Retrieve a selection of customers, each with their high-value purchases.

```
from c in db.Customers
where c.Address.State == "WA"
select new
{
   c.Name,
   c.CustomerNumber,
   HighValuePurchases = c.Purchases.Where (p => p.Price > 1000)
}
```

Parameterization

Use case: Specifying State = WA

```
string state = "WA";

var query =
        from c in db.Customers
        where c.Address.State == state
```

When not to use LINQ for querying databases

- Hand-tweaked queries (especially with optimization or locking hints)
- Queries that involve selecting into temporary tables, then querying those tables
- Predicated updates and bulk inserts
- Invoking Triggers, stored procedures and functions

Assignment

- On Northwind Database Display list of all products where category name is “Beverages”

LINQ to XML

- LINQ to XML is **a LINQ-enabled**, **in-memory XML programming interface** that enables you to work with XML .
- Can Query and modify the document.
- Can save changes to file.
- Serialize and send it over internet.

- LINQ to XML is new object model, which is lighter weight and easier to work with compared to XML DOM.

LINQ to XML

Advantages:

- integration with Language-Integrated Query (LINQ).
  - Writing queries on in memory XML document to retrieve collections of elements and attributes

- The integration of LINQ in C# provides **stronger typing, compile-time checking, and improved debugger support.**

References

- Book referred “Beginning Visual C# 2010” by Wrox publication.