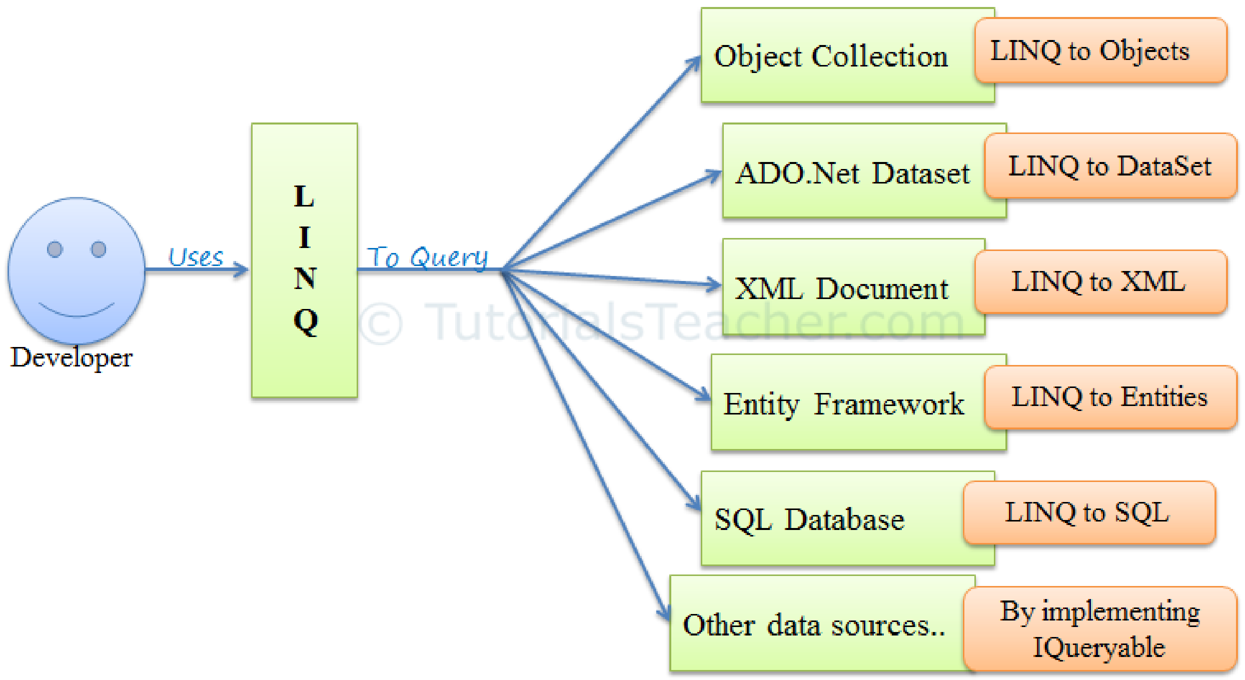
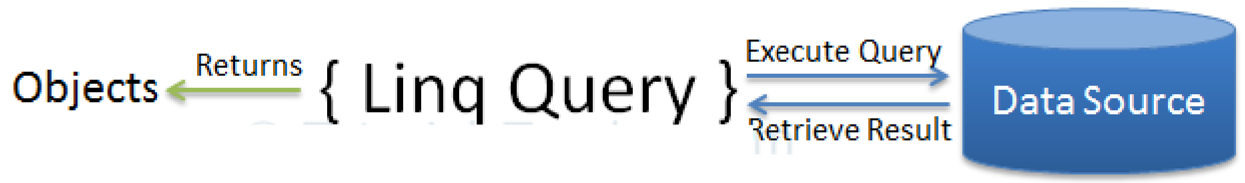
Introduction to LINQ

LINQ (Language Integrated Query) is uniform query syntax in C# to retrieve data from different sources and formats. It is integrated in C#, thereby eliminating the mismatch between programming languages and databases, as well as providing a single querying interface for different types of data sources.

For example, SQL is a Structured Query Language used to save and retrieve data from a database. In the same way, LINQ is a structured query syntax built in C# to retrieve data from different types of data sources such as collections, ADO.Net DataSet, XML Docs, web service and MS SQL Server and other databases.



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



The following example demonstrates a simple LINQ query that gets all strings from an array which contains 'a'.

Example: LINQ Query to Array

```
// Data source
string[] names = {"Bill", "Steve", "James", "Mohan" };
// LINQ Query
var myLinqQuery = from name in names
                  where name.Contains('a')
                  select name;
// Query execution
foreach(string name in myLinqQuery)
        Console.Write(name + " ");
```

Example: LINQ Query to List

```
// string collection
List<string> stringList = new List<string>() {
        "C# Tutorials",
        "VB.NET Tutorials",
        "Learn C++",
        "MVC Tutorials" ,
        "Java"
};
var result = from s in stringList
                where s.Contains("Tutorials")
                select s;
foreach(string value in result)
        Console.Write(value + " ");
```

LINQ Method

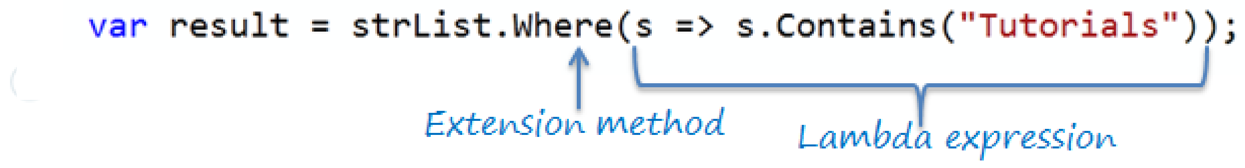
The following is a sample LINQ method syntax query that returns a collection of strings, which contains a word "Tutorials". **We use lambda expression for this purpose.**

Example: LINQ Method Syntax in C#

```
// string collection
List<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"));
```

The following figure illustrates the structure of LINQ method syntax.



Use of lambda Expression to LINQ

Example 1

```
using System;
using System.Collections.Generic;
using System.Linq;
public class demo {
        public static void Main() {
                List<int> list = new List<int>() { 1, 2, 3, 4, 5, 6 };
                List<int> evenNumbers = list.FindAll(x => (x % 2) == 0);

                foreach (var num in evenNumbers) {
                        Console.Write("{0} ", num);
                }
                Console.WriteLine();
                Console.Read();
     }
}
```

Output:

```
2 4 6
```

Example 2

```
using System;
using System.Collections.Generic;
using System.Linq;
class Dog {
	public string Name { get; set; }
	public int Age { get; set; }
}
class demo{
	static void Main() {
		List<Dog> dogs = new List<Dog>() {
			new Dog { Name = "Rex", Age = 4 },
			new Dog { Name = "Sean", Age = 0 },
			new Dog { Name = "Stacy", Age = 3 }
		};
		var names = dogs.Select(x => x.Name);
		foreach (var name in names) {
			Console.WriteLine(name);
		}
		Console.Read();
	}
}
```

Output:

```
Rex
Sean
Stacy
```

Sorting using a lambda expression

```
var sortedDogs = dogs.OrderByDescending(x => x.Age);
foreach (var dog in sortedDogs) {
	Console.WriteLine("Dog {0} is {1} years old.", dog.Name, dog.Age );
}
```

Output:

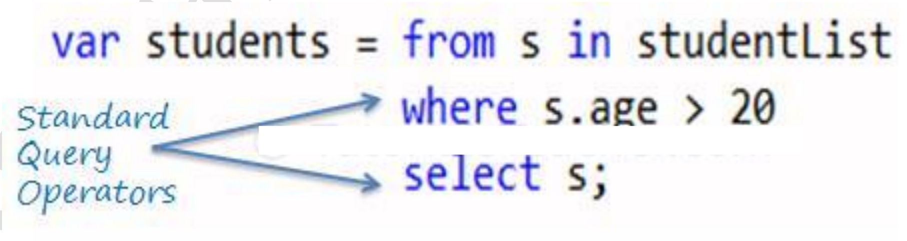
Dog Rex is 4 years old.

Dog Stacy is 3 years old.

Dog Sean is 0 years old.

LINQ Operators

Classification	LINQ 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	Equals,SequenceEqual
Generation	DefaultEmpty, Empty, Range, Repeat
Conversion	AsEnumerable, AsQueryable, Cast, ToArray, ToDictionary, ToList



```
var students = from s in studentList
               where s.age > 20
               select s;
```

Some examples of LINQ Query

Where Condition – Example 1

using System;

using System.Collections.Generic;

using System.Linq;

namespace BCA {

```
        class LinqTest {
        static void Main(string[] args) {
                List<string> names = new List<string>() { "Ram","Shyam","Hari","Gita"};
                //single condition
                //var result = names.Where(s => s.Contains("Ram"));

                //Multiple Condition
                var result = names.Where(s=>s.Contains("Ram") || s.Contains("Gita"));
                foreach (string val in result) {
                        Console.WriteLine(val); }Console.ReadLine();
                }
        }
}
```

Output:

```
Ram
Gita
```

Where Condition – Example 2 (with object list)

```
class Student {
        public int sid { get;set;}
        public string name { get; set; }
        public string address { get; set; }
        public Student(int sid, string name, string address) {
                this.sid = sid;
                this.name = name;
                this.address = address;
        }
}
class LinqTest {
        static void Main(string[] args) {
                List<Student> mylist = new List<Student>(){
                        new Student(1,"Ram","Btm"),
                        new Student(2, "Hari", "Ktm"),
                new Student(3,"Shyam","Btm"),
                new Student(4, "Gita", "Ktm")
            };

            //var result = mylist.Where(s=>s.address.Contains("Btm"));
            var result = mylist.Where(s => s.address.Equals("Btm") && s.sid.Equals(1));
            Console.WriteLine("Sid\tName\tAddress");

            foreach (var res in result) {
                Console.WriteLine(res.sid+"\t"+res.name+"\t"+res.address);
            }
            Console.ReadLine();
        }
}
```

Output:

```
Sid     Name        Address
1       Ram         Btm
```

Joining multiple lists – (join, concat, union)

```
class LinqTest {
        static void Main(string[] args) {
            List<string> names = new List<string>() { "Ram","Shyam","Hari"};
            List<string> address = new List<string>() {"Btm","Ktm","Btm" };

            /*using join
            var result = names.Join(address,
                str1 => str1,
                str2 => str2,
                (str1, str2) => str1);*/
            /*using concat
            var result = names.Concat(address);*/
            //using union
            var result = names.Union(address);
            foreach (var res in result) {
                Console.WriteLine(res);
            }
            Console.ReadLine();
        }
}
```

Using aggregate functions – Example 1

```
class LinqTest {
    static void Main(string[] args) {
        List<int> marks = new List<int>() { 10,30,50,20,5};
        int max = marks.Max();
        int min = marks.Min();
        int sum = marks.Sum();
        int total = marks.Count();

        Console.WriteLine("Maximum marks="+max);
        Console.WriteLine("Minimum marks=" + min);
        Console.WriteLine("Sum of marks=" + sum);
        Console.WriteLine("Total Count=" + total);
        Console.ReadLine();
    }
}
```

Using aggregate functions – Example 2 (Object List)

```
class Student {
    public int sid { get; set; }
    public string name { get; set; }
    public string address { get; set; }

    public Student(int sid, string name, string address) {
        this.sid = sid;
        this.name = name;
        this.address = address;
```

```
          }
}
class LinqTest { static void Main(string[] args) {
          List<Student> mylist = new List<Student>(){
                    new Student(1,"Ram","Btm"),
                    new Student(2, "Hari", "Ktm"),
                    new Student(3,"Shyam","Btm"),
                    new Student(4, "Gita", "Ktm") };

                    int maxId = mylist.Max(s=>s.sid);
                    int count = mylist.Count();

                    Console.WriteLine("Max Id="+maxId);
                    Console.WriteLine("Total Students="+count); Console.ReadLine();
          }
}
```

Output

```
Max Id=4
Total Students=4
```

Using Order By

```
class Student {
          public int sid { get; set; }
          public string name { get; set; }
          public string address { get; set; }

          public Student(int sid, string name, string address) {
                    this.sid = sid;
                    this.name = name;
                    this.address = address;
          }
}
class LinqTest { static void Main(string[] args) {
          List<Student> mylist = new List<Student>(){
                    new Student(1,"Ram","Btm"),
                    new Student(2, "Hari", "Ktm"),
                    new Student(3,"Shyam","Btm"),
                    new Student(4, "Gita", "Ktm")
          };

          //var result = mylist.OrderBy(s=>s.name);
          //var result = mylist.OrderByDescending(s => s.name);
          //select name and address of student order by name in ascending where address is Ktm
          var result = mylist.Where(s=>s.address.Equals("Ktm")). OrderBy(s=>s.name);
          Console.WriteLine("Name\tAddress");
          foreach (var res in result) {
                    Console.WriteLine(res.name + "\t" + res.address);
          }
          Console.ReadLine();
      }
}
```

Output:

```
Name  Address
Gita  Ktm
Hari  Ktm
```

Using Group By

```
class Student {
        public int sid { get; set; }
        public string name { get; set; }
        public string address { get; set; }

        public Student(int sid, string name, string address) {
        this.sid = sid;
        this.name = name;
        this.address = address;
        }
}
class LinqTest { static void Main(string[] args) {
        List<Student> mylist = new List<Student>(){
                new Student(1,"Ram","Btm"),
                new Student(2, "Hari", "Ktm"),
                new Student(3,"Shyam","Btm"),
                new Student(4, "Gita", "Ktm")
        };

        //select records group by address
        var groupResult = mylist.GroupBy(s=>s.address);
        foreach (var result in groupResult) {
                Console.WriteLine("Group Key: " + result.Key);

                //Each group has key
                Console.WriteLine("Sid\tName\tAddress");
                foreach (var res in result) {
                        Console.WriteLine(res.sid+"\t"+res.name+"\t" + res.address);
                }
        }
        Console.ReadLine();
        }
}
```

Output:

```
Group Key: Btm
Sid     Name  Address
1       Ram   Btm
3       Shyam Btm
Group Key: Ktm
Sid     Name  Address
2       Hari  Ktm
4       Gita  Ktm
```