LAB # 10

LINQ (Language Integrated Query) Basics

Statement Purpose:

LINQ to Objects is a component of .NET Framework that provides a run-time infrastructure for managing objects.

Activity Outcomes:

This lab teaches you the following topics:

- LINQ Basics
- LINQ to Objects

Instructor Note:

You need to have knowledge of retrieving Information from arrays of primitive types or objects. Select GUI Controls in Windows Form Application or use Web or console applications. Querying .net collections using LINQ to objects (Data Structures and Generic Collections).

1) Stage J (Journey)

Introduction

Developers using Data Structures like arrays, Lists or other collections of primitive data types or objects for implementing many of the features of LINQ to Objects.

2) Stage a1 (apply)

Lab Activities:

Activity 1:

The following example shows the complete query operation. The complete operation includes creating a data source, defining the query expression, and executing the query in a foreach statement.

```
class IntroToLINQ
{
    static void Main()
    {
        // The Three Parts of a LINQ Query:
        //  1. Data source.
        int[] numbers = new int[7] { 0, 1, 2, 3, 4, 5, 6 };

        // 2. Query creation.
        // numQuery is an IEnumerable<int>
        var numQuery =
            from num in numbers
            where (num % 2) == 0
            select num;

        // 3. Query execution.
        foreach (int num in numQuery)
        {
            Console.Write("{0,1} ", num);
        }
    }
}
```

Query variable itself only stores the query commands. The actual execution of the query is deferred until you iterate over the query variable in a foreach statement. This concept is referred to as deferred execution and is demonstrated in the following example:

```
// Query execution.
foreach (int num in numQuery)
{
    Console.Write("{0,1} ", num);
}
```

Queries that perform aggregation functions over a range of source elements must first iterate over those elements. Examples of such queries are Count, Max, Average, and First. These execute without an explicit foreach statement because the query itself must use foreach in order to return a result. Note also that these types of queries return a single value, not an IEnumerable collection. The following query returns a count of the even numbers in the source array:

```
var evenNumQuery =
    from num in numbers
    where (num % 2) == 0
    select num;

int evenNumCount = evenNumQuery.Count();
```

To force immediate execution of any query and cache its results, you can call the `ToList<TSource>` or `ToArray<TSource>` methods.

```
List<int> numQuery2 =
       (from num in numbers
        where (num % 2) == 0
        select num).ToList();

// or like this:
// numQuery3 is still an int[]

var numQuery3 =
       (from num in numbers
        where (num % 2) == 0
        select num).ToArray();
```

You can also force execution by putting the foreach loop immediately after the query expression. However, by calling `ToList` or `ToArray` you also cache all the data in a single collection object.

Activity 2:

This example shows how to perform a simple query over a list of Student objects. Each Student object contains some basic information about the student, and a list that represents the student's scores on four examinations.

The following query returns the students who received a score of 90 or greater on their first exam.

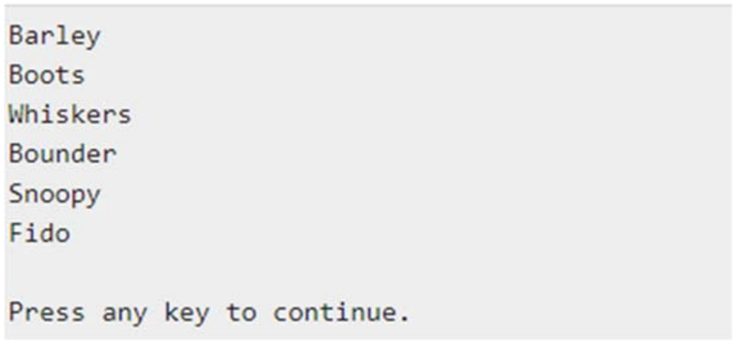
```
public class StudentClass
{
    protected enum GradeLevel { FirstYear = 1, SecondYear, ThirdYear, FourthYear };
    protected class Student
    {
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public int ID { get; set; }
        public GradeLevel Year;
        public List<int> ExamScores;
    }

    protected static List<Student> students = new List<Student>
        {
            new Student {FirstName = "Terry", LastName = "Adams", ID = 120,
                Year = GradeLevel.SecondYear,
                ExamScores = new List<int>{ 99, 82, 81, 79}},
            new Student {FirstName = "Fadi", LastName = "Fakhouri", ID = 116,
                Year = GradeLevel.ThirdYear,
                ExamScores = new List<int>{ 99, 86, 90, 94}},
            new Student {FirstName = "Hanying", LastName = "Feng", ID = 117,
                Year = GradeLevel.FirstYear,
                ExamScores = new List<int>{ 93, 92, 80, 87}},
            new Student {FirstName = "Cesar", LastName = "Garcia", ID = 114,
                Year = GradeLevel.FourthYear,
                ExamScores = new List<int>{ 97, 89, 85, 82}},
            new Student {FirstName = "Debra", LastName = "Garcia", ID = 115,
                Year = GradeLevel.ThirdYear,
                ExamScores = new List<int>{ 35, 72, 91, 70}},
            new Student {FirstName = "Hugo", LastName = "Garcia", ID = 118,
                Year = GradeLevel.SecondYear,
                ExamScores = new List<int>{ 92, 90, 83, 78}},
            new Student {FirstName = "Sven", LastName = "Mortensen", ID = 113,
                Year = GradeLevel.FirstYear,
                ExamScores = new List<int>{ 88, 94, 65, 91}},
            new Student {FirstName = "Claire", LastName = "O'Donnell", ID = 112,
                Year = GradeLevel.FourthYear,
                ExamScores = new List<int>{ 75, 84, 91, 39}},
            new Student {FirstName = "Svetlana", LastName = "Omelchenko", ID = 111,
                Year = GradeLevel.SecondYear,
                ExamScores = new List<int>{ 97, 92, 81, 60}},
            new Student {FirstName = "Lance", LastName = "Tucker", ID = 119,
                Year = GradeLevel.ThirdYear,
                ExamScores = new List<int>{ 68, 79, 88, 92}},
            new Student {FirstName = "Michael", LastName = "Tucker", ID = 122,
                Year = GradeLevel.FirstYear,
                ExamScores = new List<int>{ 94, 92, 91, 91}},
            new Student {FirstName = "Eugene", LastName = "Zabokritski", ID = 121,
                Year = GradeLevel.FourthYear,
                ExamScores = new List<int>{ 96, 85, 91, 60}}
        };

    //Helper method, used in GroupByRange.
    protected static int GetPercentile(Student s)
    {
        double avg = s.ExamScores.Average();
        return avg > 0 ? (int)avg / 10 : 0;
    }



    public void QueryHighScores(int exam, int score)
    {
        var highScores = from student in students
                         where student.ExamScores[exam] > score
                         select new { Name = student.FirstName, Score =
student.ExamScores[exam] };

        foreach (var item in highScores)
        {
            Console.WriteLine("{0,-15}{1}", item.Name, item.Score);
        }
    }
}

public class Program
{
    public static void Main()
    {
        StudentClass sc = new StudentClass();
        sc.QueryHighScores(1, 90);

        // Keep the console window open in debug mode.
        Console.WriteLine("Press any key to exit");
        Console.ReadKey();
    }
}
```

Activity 3:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Operators
{
    class Pet
    {
        public string Name { get; set; }
        public int Age { get; set; }
    }

    class Concat
    {
        static void Main(string[] args)
        {

            Pet[] cats = GetCats();
            Pet[] dogs = GetDogs();
```

```
     IEnumerable<string> query = cats.Select(cat ⇒ cat.Name).Concat(dogs.Select(dog => dog.Name));


       foreach (var e in query)
            {
                Console.WriteLine("Name = {0} ", e);
            }

            Console.WriteLine("\nPress any key to continue.");
            Console.ReadKey();
        }

        static Pet[] GetCats()
        {
            Pet[] cats = { new Pet { Name = "Barley", Age = 8 },
                       new Pet { Name = "Boots", Age = 4 },
                       new Pet { Name = "Whiskers", Age = 1 } };
            return cats;
        }

        static Pet[] GetDogs()
        {
            Pet[] dogs = { new Pet { Name = "Bounder", Age = 3 },
                       new Pet { Name = "Snoopy", Age = 14 },
                       new Pet { Name = "Fido", Age = 9 } };

            return dogs;
        }
    }
}
```

Output:



```
Barley
Boots
Whiskers
Bounder
Snoopy
Fido

Press any key to continue.
```

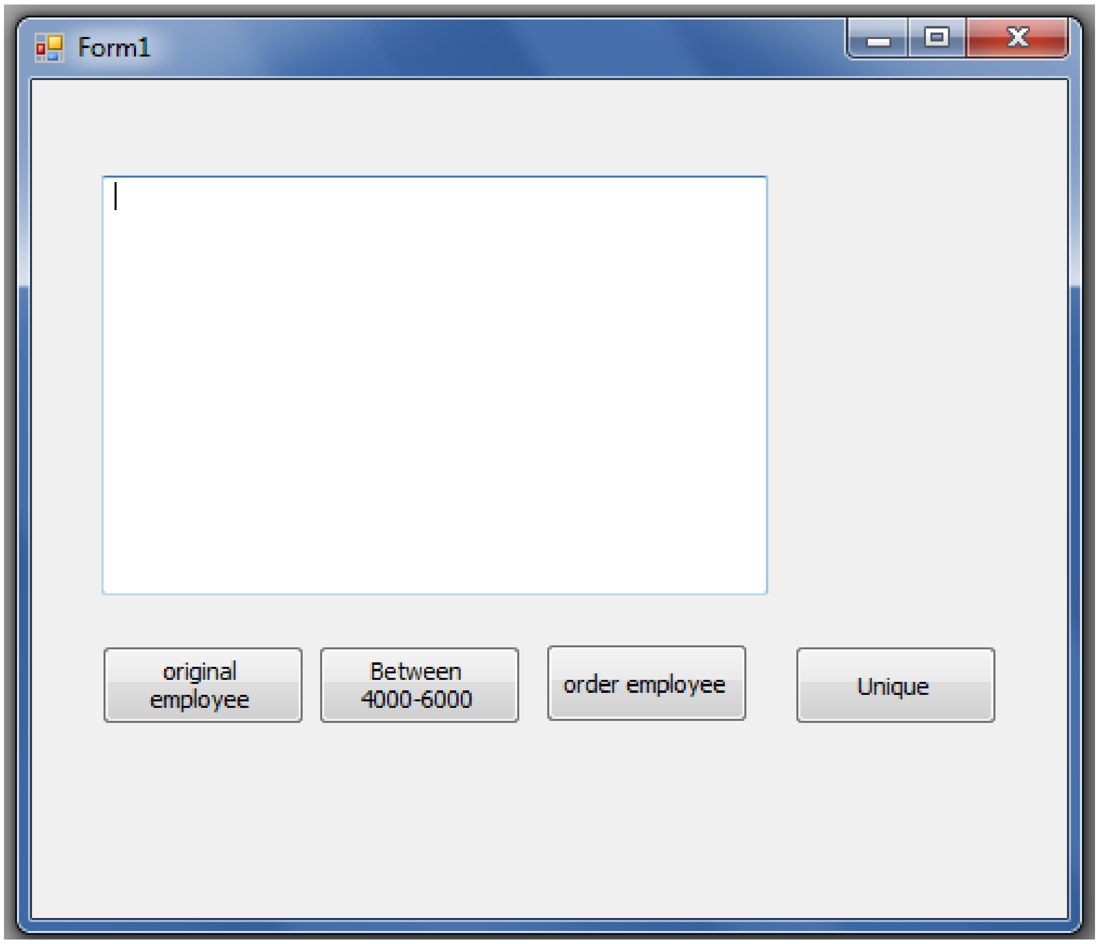
Activity 4:

Consider following GUI. Create a Windows Form Application with this GUI and perform following steps for using LINQ to objects

1.Create a class Employee having (FirstName, LastName, Salary) as data members. Create a parametrized constructor for initializing values of objects.

2. In main create 10 objects of Employee by using its parametrized constructor.

3. Perform LINQ operations on each button click event for the required result. And show values in the ListView.



Links: https://freeasphosting.net/linq-tutorial-filtering-operators.html