Міністерство освіти і науки України
Національний університет «Львівська політехніка»

ЗВІТ

Про виконання лабораторної роботи № 2

На тему: "*Linq, List and Dictionary***"**
з дисципліни «Моделювання та аналіз ПЗ»

Лектор:
доцент каф. ПЗ
Сердюк П.В.

Виконав:
студ. гр. ПЗ-22
Місяйло О.О.

Прийняв:
Викл. каф.ПЗ
Микуляк А.В.

«___» _____ 2023 р.

∑ = ____ _____

Львів – 2023

Тема: Linq, List and Dictionary.
Мета: Навчитися працювати з масивами та структурами List, Dictionary. Засвоїти технологію LINQ.

Теоретичні відомості

LINQ (англ. Language Integrated Query - запити, інтегровані в мову) - компонент Microsoft .NET Framework, який додає нативні можливості виконання запитів даних до мов, що входять у .NET. Хоча порти існують для PHP (PHPLinq), JavaScript (linq.js), TypeScript (linq.ts), і ActionScript (ActionLinq), - жоден з них не є абсолютно еквівалентним LINQ в C# (де LINQ - не просто додаткова бібліотека, а частина мови).

LINQ розширює можливості мови, додаючи до неї вирази запитів, що є схожими на твердження SQL та можуть бути використані для зручного отримання та обробки даних масивів, XML документів, реляційних баз даних та сторонніх джерел. LINQ також визначає набір імен методів (що називаються стандартними операторами запитів, або стандартними операторами послідовностей), а також правила перекладу, що має використовувати компілятор для перекладу текучих виразів у звичайні, використовуючи їх назву, лямбда-вирази та анонімні типи.

Найбільш використовувані запити:

Select
Даний оператор виконує проєкцію на колекцію, щоб отримати необхідні члени елементів. Користувач передає у якості параметру довільну функцію у вигляді лямбда-виразу, що проєктує члени даного об'єкту. Функція передається до оператора у вигляді делегату.

Where
Оператор Where дозволяє визначити набір предикатів, що виконуються над кожним елементом колекції та відкидає елементи, що не підпадають під задану умову з результуючого набору. Предикат передається до оператора у вигляді делегату.

Завдання

Для демонстрації роботи розробити Unit-тести для моделі ПЗ відповідно до варіанту проекту. Unit-тести повинні створювати списки, словники або інші структури об’єктів, та робити із ними різні операції використовуючи LINQ запити.
Як альтернатива Unit-тестам можна використати консольне застосування (для максимальної кількості балів таких запитів повинно бути більше).

Повинні бути реалізовані такі запити:

1. Селекція частини інформації (метод Select)
2. Вибірка певної інформації (Where)
3. Операції як з списком List, так і з словником Dictionary
4. Власні методи розширювання
5. Використання анонімних класів та ініціалізаторів
6. Сортування за певним критерієм використовуючи шаблон IComparer
7. Конвертування списків в масив
8. Сортування масиву/списку за ім’ям чи за кількістю елементів

Тести повинні працювати виключно з моделлю об’єктів. Модель об'єктів повинна включати в себе різні типи даних (чисельні, стрічкові, логічні, множинні і т.д.), для кожного типу даних повинен бути мінімум 1 тест.

Хід виконання

1. На початку виконання я написав два зв’язних між собою класи – MusicalInstrument та InstrumenType (перший містить поле типу InstrumentType).

```
public class InstrumentType
{
    public InstrumentType(string t, string d)
    {
        Type = t;
        Description = d;
    }
    public InstrumentType(string t)
    {
        Type = t;
        Description = null;
    }
    public string Type { get; set; }
    public string Description { get; set; }


}
```

Рисунок 1. InstrumentType class

```
public class MusicalInstrument
{
    public MusicalInstrument(string n, bool st, int p, string t)
    {
        Name = n;
        isElectric = st;
        Price = p;
        type = new InstrumentType(t);
    }
    public MusicalInstrument(string n, bool st, int p, string t, string d)
    {
        Name = n;
        isElectric = st;
        Price = p;
        type = new InstrumentType(t, d);
    }
    public MusicalInstrument(string n, bool st, int p, InstrumentType t)
    {
        Name = n;
        isElectric = st;
        Price = p;
        type = t;
    }
    public string Name { get; set; }
    public readonly bool isElectric;
    public readonly int Price;
    public readonly InstrumentType type;
}
```

Рисунок 2. MusicalInstrument class

2. Також я написав клас для фіктивного об’єкту, який генерує списки типів обох класів та містить в собі усі функції, що в подальшому я буду тестувати.

```
public class Mock
{
    public readonly List<MusicalInstrument> Insts;
    public readonly List<InstrumentType> Types;

    public Mock()
    {

        Types = new List<InstrumentType>()
        {
            new InstrumentType("String"),
            new InstrumentType("Percussion"),
            new InstrumentType("Keyboard"),
            new InstrumentType("Brass")
        };
        Insts = new List<MusicalInstrument>()
        {
            new MusicalInstrument("Guitar", false, 200, Types[0]),
            new MusicalInstrument("EGuitar", true, 500, Types[0]),
            new MusicalInstrument("Violin", false, 300, Types[0]),
            new MusicalInstrument("EViolin", true, 600, Types[0]),
            new MusicalInstrument("Bango", false, 200, Types[0]),

            new MusicalInstrument("Drum", false, 300, Types[1]),
            new MusicalInstrument("BassDrum", false, 400, Types[1]),
            new MusicalInstrument("EDrum", true, 400, Types[1]),
            new MusicalInstrument("Tambourine", false, 100, Types[1]),
            new MusicalInstrument("Kalimba", false, 150, Types[1]),

            new MusicalInstrument("Piano", false, 400, Types[2]),
            new MusicalInstrument("EPiano", true, 800, Types[2]),
            new MusicalInstrument("Synthesizer", true, 500, Types[2]),
            new MusicalInstrument("Accordion", true, 700, Types[2]),

            new MusicalInstrument("Trumpet", false, 600, Types[3]),
            new MusicalInstrument("Tuba", false, 550, Types[3]),
            new MusicalInstrument("Trombone", false, 650, Types[3]),
            new MusicalInstrument("Saxophone", false, 750, Types[3])
        };
    }
```

Рисунок 3. Mock-object клас.з генерацією списків при ініціалізації

3. Далі я написав функцію для демонстрації даних у консолі.

```
public void demonstrateLists()
{
    foreach (var x in Types)
    {Console.WriteLine(x.Type);}
    Console.WriteLine("");
    int i = 0;
    foreach (var x in Insts)
    {
        Console.WriteLine(String.Format("Object {0, 2}: {1,11} | {2, 5} | {3,
3} | {4, 10}"
            , ++i
            , x.Name
            , x.isElectric
            , x.Price
            , x.type.Type));
    }
}
```

Рисунок 4. Функція виводу списків на екран

```
/Users/olegmisialo/Desktop/lab02/lab02/bin/Debug/net7.0/lab02
String
Percussion
Keyboard
Brass

Object  1:      Guitar | False | 200 |     String
Object  2:     EGuitar |  True | 500 |     String
Object  3:      Violin | False | 300 |     String
Object  4:     EViolin |  True | 600 |     String
Object  5:       Bango | False | 200 |     String
Object  6:        Drum | False | 300 | Percussion
Object  7:    BassDrum | False | 400 | Percussion
Object  8:       EDrum |  True | 400 | Percussion
Object  9:  Tambourine | False | 100 | Percussion
Object 10:     Kalimba | False | 150 | Percussion
Object 11:       Piano | False | 400 |   Keyboard
Object 12:      EPiano |  True | 800 |   Keyboard
Object 13: Synthesizer |  True | 500 |   Keyboard
Object 14:   Accordion |  True | 700 |   Keyboard
Object 15:     Trumpet | False | 600 |      Brass
Object 16:        Tuba | False | 550 |      Brass
Object 17:    Trombone | False | 650 |      Brass
Object 18:   Saxophone | False | 750 |      Brass
```

Рисунок 5. Результат виконання функції

4. Далі я створив тест для перебору функцій перетворення списків у інші структури. Також я написав клас що імплементує IComparator та визначає метод Compare для класу MusicalInstrument.

```
[Fact]
public void Structures_test()
{
    Mock mock = new Mock();
    var dict = mock.Insts.ToDictionary(x => x.Name, x => x.type.Type);
    dict.Should().Contain(x=>x.Key=="EGuitar"&&x.Value=="String");

    var queue = new Queue<MusicalInstrument>(mock.Insts);
    queue.Peek().type.Type.Should().Be("String");

    var hashSet = new HashSet<MusicalInstrument>(mock.Insts);
    hashSet.Add(mock.Insts[3]).Should().Be(false);

    SortedSet<MusicalInstrument> sortList = new
SortedSet<MusicalInstrument>(mock.Insts, new InstrumentComparator());

    sortList.Select(x => x.Name).Should().ContainInOrder("EGuitar",
"Saxophone");
}
```

Рисунок 6. Data Structures

```
public class InstrumentComparator: IComparer<MusicalInstrument>
{
    public int Compare(MusicalInstrument? i1, MusicalInstrument? i2)
    {
        int length1 = i1.Name.Length;
        int length2 = i2.Name.Length;
        return length1 - length2;
    }
}
```

Рисунок 7. Compare implementing

5. Наступним кроком є написання функцій та тестів для перевірки коректності їх роботи.

```
public IEnumerable<MusicalInstrument> GetElectricInstruments()
{
    return Insts.Where(n => n.isElectric);
}
```

Рисунок 8

```
[Fact]
public void IEnum_ElectricGuitars_test()
{
    //Arrange
    Mock mock = new Mock();
    //Act
    IEnumerable<MusicalInstrument> list = mock.GetElectricInstruments();
    //Assert
    list.Should().Contain(x => x.isElectric).And.NotContain(x =>
!x.isElectric);
}
```

Рисунок 9 Вибірка електричних інструментів зі списку у типу IEnumerable

```
public Dictionary<string, int> GetKeyboardInstrumentsWithDiscount()
{
    return Insts.Where(x => x.type.Type == "Keyboard")
        .Select(x => new MusicalInstrument(x.Name, x.isElectric, x.Price / 2,
x.type))
        .ToDictionary(x => x.Name, x => x.Price);
}
```

Рисунок 10

```
[Fact]
public void Dict_KeyboardInstruments_Discount_test()
{
    //Arrange
    Mock mock = new Mock();
    Dictionary<string, int> expected = new Dictionary<string, int>();
    expected.Add("Piano", 200);
    expected.Add("EPiano", 400);
    expected.Add("Synthesizer", 250);
    expected.Add("Accordion", 350);

    //Act
    Dictionary<string, int> result =
mock.GetKeyboardInstrumentsWithDiscount();

    //Assert
    foreach (var x in expected)
    {
        result.Should().ContainEquivalentOf(x);
    }
    result.Should().HaveCount(expected.Count);
}
```

Рисунок 11. Створення ditionary за ключем назва гітари та значенням ціна зі знижкою

```
public List<MusicalInstrument> GetInstrumentCheaperThan(int num)
{
    List<MusicalInstrument> list = Insts.Where((x) => x.Price <=
num).OrderBy(x => x.Price).ToList();
    foreach (var VARIABLE in list)
    {
        Console.WriteLine(VARIABLE.Name + " "+ VARIABLE.Price);
    }
    return list;
}
```

Рисунок 12

```
[Fact]
public void List_CheaperThan_test()
{
    //Arrange
    Mock mock = new Mock();
    string[] expected = {"Tambourine", "Kalimba", "Guitar", "Bango",
"Violin", "Drum"};

    //Act
    List<MusicalInstrument> result = mock.GetInstrumentCheaperThan(300);

    //Assert
    result.Select(x=>x.Name).Should().ContainInOrder(expected);
}
```

Рисунок 13. Отримання списку елементів, посортованих за ціною

```
[Fact]
public void Array_KeyboardInstruments_Discount_test()
{
    //Arrange
    Mock mock = new Mock();
    string[] expected = {"Piano", "EPiano", "Synthesizer", "Accordion"};
    //Act
    string[] result = mock.GetKeyboardInstrumentsWithDiscount()
        .Select(x=>x.Key)
        .ToArray();
    //Assert
    Assert.Equal(expected, result);
}
```

Рисунок 14. Тест для перетворення списку у масив

```
public static class Extensions
{
    public static MusicalInstrument[] GetSortedByNameInstruments(this
List<MusicalInstrument> insts)
    {
        return insts.OrderBy(x => x.Name).ToArray();
    }
}
```

Рисунок 15

```
public MusicalInstrument[] GetOrderedByName()
{
    return Insts.GetSortedByNameInstruments();
}
```

Рисунок 16

```
[Fact]
public void Array_ExtensionMethod_OrderedByName()
{
    //Arrange
    Mock mock = new Mock();
    string[] expected = new[] { "Bango", "EPiano","Saxophone", "Trombone" };

    //Act
    MusicalInstrument[] result = mock.GetOrderedByName();

    //Assert
    result.Select(x=>x.Name).Should().ContainInOrder(expected);
}
```

Рисунок 17 Імплементація власного методу розширювання

```
public IEnumerable<object> Enum_GetJoined()
{
    return Insts.Join(Types,
        i=>i.type.Type,
        t=>t.Type,
        (i, t) =>  new { Name = i.Name, Type = i.type.Type }
    );
}
```

Рисунок 18

```
[Fact]
public void IEnum_JoinByType()
{
    //Arrange
    Mock mock = new Mock();

    //Act
    IEnumerable<object> result = mock.Enum_GetJoined();

    //Assert
    result.Should().HaveCount(mock.Insts.Count);
    result.Should().ContainEquivalentOf(new { Name = "EGuitar", Type =
"String" });
}
```

Рисунок 19. Імплементація анонімного класу

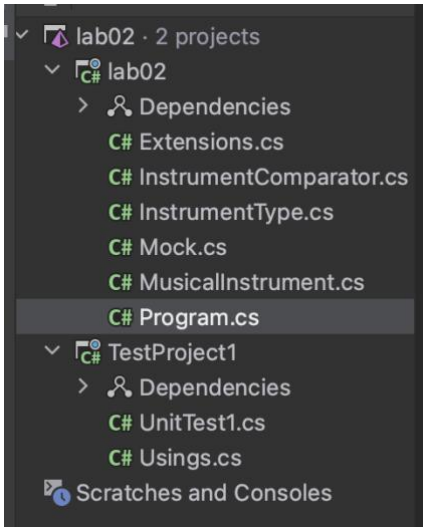

Рисунок 20. Загальний вигляд розміщення файлів роботи

Висновок

У цій лабораторній роботі було вивчено технологію LINQ мови програмування C# та продемонстровано на практиці деякі методи розширення. Також було написано власний метод розширення та клас, що імплементує інтерфейс IComparer.