Автор: Абдуллін Олексій

Група: КІТ-119а

Дата: 17.12.2021

Лабораторна робота№ 6

Тема: Інтегровані запити (Language Integrated Query, LINQ)

Задачі:

1. Для доступу до колекції об'єктів (відбір, фільтрація, угруповання, розрахунок) використовувати LINQ.

Опис класів

MyCollection – власний клас контейнера для реалізації колекції об'єктів;

MyCollectionEnum – клас, який реалізує інтерфейс IEnumerator;

Student – клас, який відображує студента;

IPrinter – інтерфейс для виводу у консоль;

Menu – клас для роботи меню;

Текст програми

MyCollectionEnum

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

namespace Abdullin04
{
    public class MyCollectionEnum : IEnumerator
    {
        public List<Student> _stud;

        int position = -1;
        public MyCollectionEnum(List<Student> stud)
        {
            _stud = stud;
        }

        public bool MoveNext()
```

```
        {
            position++;
            return (position < _stud.Count);
        }

        public void Reset()
        {
            position = -1;
        }

        object IEnumerator.Current
        {
            get
            {
                return Current;
            }
        }

        public Student Current
        {
            get
            {
                try
                {
                    return _stud[position];
                }
                catch (IndexOutOfRangeException)
                {
                    throw new InvalidOperationException();
                }
            }
        }
    }
}
```

MyCollection

```
using System;
using System.Runtime.Serialization;
using System.Text;
using Abdullin03;

namespace Abdullin04
{
    [DataContract]
    public class Student
    {
        [DataMember]
        public string FirstName { get; set; }
        [DataMember]
        public string SurName { get; set; }
        [DataMember]
        public string GroupIndex { get; set; }
        [DataMember]
        public string Faculty { get; set; }
        [DataMember]
        public int Specialization { get; set; }
        [DataMember]
        public int AcademicPerformance { get; set; }
        [DataMember]
        public DateTime DateOfBirth { get; set; }
        [DataMember]
        public DateTime DateOfEnter { get; set; }
        [IgnoreDataMember]
        public IPrinter Printer { get; set; }
```

```
        public Student() : this("Oleksii", "Abdullin", "a", "CIT", 123, 86, new
DateTime(2002, 5, 31), new DateTime(2019, 8, 12))
        {

        }
        public Student(string FirstName, string SurName, string GroupIndex, string
Faculty,
            int Specialization, int AcademicPerformance, DateTime DateOfBirth, DateTime
DateOfEnter)
        {
            this.FirstName = FirstName;
            this.SurName = SurName;
            this.GroupIndex = GroupIndex;
            this.Faculty = Faculty;
            this.Specialization = Specialization;
            this.AcademicPerformance = AcademicPerformance;
            this.DateOfBirth = DateOfBirth;
            this.DateOfEnter = DateOfEnter;
            this.Printer = new ConsolePrinter();
        }
        public void Print()
        {
            Printer.Print(ToString());
        }
        public StringBuilder CountAge()
        {
            StringBuilder res = new StringBuilder();
            var diff = (DateTime.Now - DateOfBirth).TotalDays;
            var years = Math.Truncate(diff / 365);
            var days = Math.Truncate(diff % 365);
            return res.Append("Years: ").Append(years).Append("\nDays: ").Append(days);
        }
        public StringBuilder GetGroup()
        {
            StringBuilder res = new StringBuilder();
            return res.Append("\nGroup name: ").Append(Faculty).Append("-
").Append(Specialization).Append(GroupIndex).Append("Year of enter:
").Append(DateOfEnter.Year);
        }
        public StringBuilder CountCourse()
        {
            StringBuilder res = new StringBuilder();
            int course = 0;
            int semester = 0;
            int nowYear = DateTime.Now.Year;
            int nowMonth = DateTime.Now.Month;
            int year = DateOfEnter.Year;
            int month = DateOfEnter.Month;
            course = nowYear - year + 1;
            if (8 > nowMonth)
            {
                course--;
                semester = course * 2;
            }
            else
            {
                semester = course * 2 - 1;
            }
            if (1 > course)
            {
                res.Append("Error! This person can not be student");
                return res;
            }
            else if (course > 6)
            {
```

```
                res.Append("This student have been graduated.");
                return res;
            }
            res.Append("Course: ").Append(course).Append("\nSemester:
").Append(semester);
            return res;
        }

        public override string ToString()
        {
            return "Fristname: " + FirstName + "\nSurName: " + SurName +
                "\nDate of birth: " + DateOfBirth.Day + "." + DateOfBirth.Month + "." +
DateOfBirth.Year +
                "\nDate of enter: " + DateOfEnter.Day + "." + DateOfEnter.Month + "." +
DateOfEnter.Year +
                "\nIndex of group: " + GroupIndex + "\nFaculty: " + Faculty +
                "\nSpecialization: " + Specialization + "\nAcademic Performance: " +
AcademicPerformance + "\n";
        }
        public override bool Equals(object obj)
        {
            if (obj == null)
            {
                return false;
            }
            Student s = obj as Student;
            if (s == null)
            {
                return false;
            }
            return s.FirstName == this.FirstName &&
                    s.SurName == this.SurName &&
                    s.GroupIndex == this.GroupIndex &&
                    s.Faculty == this.Faculty &&
                    s.Specialization == this.Specialization &&
                    s.AcademicPerformance == this.AcademicPerformance &&
                    s.DateOfBirth == this.DateOfBirth &&
                    s.DateOfEnter == this.DateOfEnter;
        }
    }
}
```

Menu

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Runtime.Serialization.Json;
using System.Text;
using System.Text.RegularExpressions;
using System.Xml;
using Abdullin03;

namespace Abdullin04
{
    public class Menu
    {
        delegate void Calculate(MyCollection students);
        public void MenuStudents()
        {
            Student student;
            var MyCollection = new MyCollection();
            MyCollection.Add(new Student());

            int i;
```

```
            int option;
            bool inMenu = true;
            string path = "lab06.json";
            var serializer = new DataContractJsonSerializer(typeof(List<Student>));
            while (inMenu)
            {
                Console.WriteLine("Menu options:");
                Console.WriteLine("1. Add");
                Console.WriteLine("2. Remove");
                Console.WriteLine("3. Show students");
                Console.WriteLine("4. Ser");
                Console.WriteLine("5. Deser");
                Console.WriteLine("6. Calculate");
                Console.WriteLine("0. Exit");
                Console.Write("Enter your option: ");

                if (!int.TryParse(Console.ReadLine(), out option))
                {
                    Console.WriteLine("\nError! Invalid datatype.\n");
                    option = -1;
                }

                switch (option)
                {
                    case 1:
                        MyCollection.Add(InsertInfo());
                        break;
                    case 2:
                        Regex regex_faculty = new Regex(@"^[A-Z]{1,3}$",
RegexOptions.IgnoreCase);
                        Regex regex_group = new Regex(@"^[A-Z]{1,3}-[0-9]{2,3}[a-
z]{1,2}$", RegexOptions.IgnoreCase);

                        int id;
                        bool result = false;
                        string faculty;
                        int specialization;
                        string group;
                        int optionRemove;
                        Console.WriteLine("\nMenu Remove options:");
                        Console.WriteLine("1. Remove by id");
                        Console.WriteLine("2. Remove by faculty");
                        Console.WriteLine("3. Remove by specialization");
                        Console.WriteLine("4. Remove by group");
                        Console.Write("Enter your option: ");
                        if (!int.TryParse(Console.ReadLine(), out optionRemove))
                        {
                            Console.Write("\nError! Invalid datatype. \n");
                            break;
                        }
                        switch (optionRemove)
                        {
                            case 1:
                                Console.Write("\nEnter student id: ");
                                if (!int.TryParse(Console.ReadLine(), out id))
                                {
                                    Console.Write("\nError! Invalid datatype. \n");
                                    break;
                                }
                                result = MyCollection.RemoveById(id);
                                break;
                            case 2:
                                Console.Write("\nEnter faculty: ");
                                faculty = Console.ReadLine();
                                if (!regex_faculty.IsMatch(faculty))
```

```
                                {
                                    Console.Write("\nError! Incorrect faculty name \n");
                                    break;
                                }
                                result = MyCollection.RemoveByFaculty(faculty);
                                break;
                            case 3:
                                Console.Write("\nEnter specialization: ");
                                if (!int.TryParse(Console.ReadLine(), out
specialization))
                                {
                                    Console.Write("\nError! Invalid datatype. \n");
                                    break;
                                }
                                result =
MyCollection.RemoveBySpecialization(specialization);
                                break;
                            case 4:
                                Console.Write("\nEnter group: ");
                                group = Console.ReadLine();
                                if (!regex_group.IsMatch(group))
                                {
                                    Console.Write("\nError! Incorrect group name \n");
                                    break;
                                }
                                result = MyCollection.RemoveByGroup(group);
                                break;
                            default:
                                Console.WriteLine("\nIncorrect option! Try again. \n");
                                break;
                        }
                        if (result)
                        {
                            Console.Write("\nStudent was deleted succsessfully.\n");
                        }
                        break;

                    case 3:
                        int optionOutput;
                        Console.WriteLine("\nMenu Output options:");
                        Console.WriteLine("1. Show all students");
                        Console.WriteLine("2. Show course and semester of student");
                        Console.WriteLine("3. Show group of student");
                        Console.WriteLine("4. Show age of student");
                        Console.Write("Enter your option: ");
                        if (!int.TryParse(Console.ReadLine(), out optionOutput))
                        {
                            Console.Write("\nError! Invalid datatype. \n");
                            break;
                        }
                        switch (optionOutput)
                        {
                            case 1:
                                i = 0;
                                foreach (var stud in MyCollection)
                                {
                                    Console.WriteLine("\nStudent ID: " + i);
                                    stud.Print();
                                    i++;
                                }
                                break;
                            case 2:
                                Console.Write("Enter the student id: ");
                                if (!int.TryParse(Console.ReadLine(), out id))
                                {
```

```
                                    Console.Write("\nError! Invalid datatype. \n");
                                    break;
                                }
                                student = MyCollection.GetStudentById(id);
                                if (student != null)
                                {
                                    Console.WriteLine(student.CountCourse());
                                }
                                else
                                {
                                    Console.WriteLine("\nError! Invalid student id.");
                                }
                                break;
                            case 3:
                                Console.Write("Enter the student id: ");
                                if (!int.TryParse(Console.ReadLine(), out id))
                                {
                                    Console.Write("\nError! Invalid datatype. \n");
                                    break;
                                }
                                student = MyCollection.GetStudentById(id);
                                if (student != null)
                                {
                                    Console.WriteLine(student.GetGroup());
                                }
                                else
                                {
                                    Console.WriteLine("\nError! Invalid student id.");
                                }
                                break;
                            case 4:
                                Console.Write("Enter the student id: ");
                                if (!int.TryParse(Console.ReadLine(), out id))
                                {
                                    Console.Write("\nError! Invalid datatype. \n");
                                    break;
                                }
                                student = MyCollection.GetStudentById(id);
                                if (student != null)
                                {
                                    Console.WriteLine(student.CountAge());
                                }
                                else
                                {
                                    Console.WriteLine("\nError! Invalid student id.");
                                }
                                break;
                            default:
                                Console.WriteLine("\nIncorrect option. Try again.\n");
                                break;
                        }
                        break;
                    case 4:
                        using (var file = new FileStream(path, FileMode.Create))
                        {
                            using (var jsonw =
JsonReaderWriterFactory.CreateJsonWriter(file, Encoding.GetEncoding("utf-8")))
                            {
                                serializer.WriteObject(jsonw,
MyCollection.GetStudents());
                                jsonw.Flush();
                            }
                        }
                        break;
                    case 5:
```

```
                        List<Student> obj = Activator.CreateInstance<List<Student>>();
                        using (FileStream file = new FileStream(path, FileMode.Open))
                        {
                            using (XmlDictionaryReader jsonr =
JsonReaderWriterFactory.CreateJsonReader(file,
                                    Encoding.GetEncoding("utf-8"),
XmlDictionaryReaderQuotas.Max, null))
                            {
                                obj = serializer.ReadObject(jsonr) as List<Student>;
                            }
                        }
                        MyCollection.Clear();
                        foreach (var stud in obj)
                        {
                            stud.Printer = new ConsolePrinter();
                            MyCollection.Add(stud);
                        }
                        break;
                    case 6:
                        Calculate calculate = AvgAge;
                        calculate += AvgPerformance;
                        calculate(MyCollection);
                        break;
                    case 0:
                        inMenu = false;
                        break;
                    default:
                        if (option == -1)
                        {
                            break;
                        }
                        Console.WriteLine("\nIncorrect option. Try again.\n");
                        break;

                }
            }
        }
        public void AvgAge(MyCollection students)
        {
            List<Student> studentList = new List<Student>();
            foreach (var item in students)
            {
                studentList.Add(item);
            }
            var avg = studentList.Average(student =>
FindAge(student.CountAge().ToString()));
            Console.WriteLine("\nAvarange age: " + avg.ToString());
        }
        public int FindAge(string str)
        {
            int age = 0;
            int start = str.IndexOf(": ");
            if (int.TryParse(str.Substring(start + 2, 2), out age))
            {
                return age;
            }
            return 0;
        }
        public void AvgPerformance(MyCollection students)
        {
            List<Student> studentList = new List<Student>();
            foreach (var item in students)
            {
                studentList.Add(item);
            }
```

```
    var avg = studentList.Average(student => student.AcademicPerformance);
    Console.WriteLine("Avarange performance: " + avg.ToString() + "\n");
}

public Student InsertInfo()
{
    Regex regex_string = new Regex(@"^[a-z]+$", RegexOptions.IgnoreCase);

    string firstname;
    string surname;
    string groupIndex;
    string faculty;
    int specialization;
    int academicPerformance;
    DateTime dateOfBirth;
    DateTime dateOfEnter;

    Console.Write("Enter firstname of student: ");
    firstname = Console.ReadLine();
    if (!regex_string.IsMatch(firstname))
    {
        Console.WriteLine("\nError! Invalid datatype.\n");
        return null;
    }

    Console.Write("Enter surname of student: ");
    surname = Console.ReadLine();
    if (!regex_string.IsMatch(surname))
    {
        Console.WriteLine("\nError! Invalid datatype.\n");
        return null;
    }

    Console.Write("Enter index of group: ");
    groupIndex = Console.ReadLine();
    if (!regex_string.IsMatch(groupIndex))
    {
        Console.WriteLine("\nError! Invalid datatype.\n");
        return null;
    }

    Console.Write("Enter faculty of student: ");
    faculty = Console.ReadLine();
    if (!regex_string.IsMatch(faculty))
    {
        Console.WriteLine("\nError! Invalid datatype.\n");
        return null;
    }

    Console.Write("Enter specialization of student: ");
    if (!int.TryParse(Console.ReadLine(), out specialization))
    {
        Console.WriteLine("\nError! Invalid datatype.\n");
        return null;
    }

    Console.Write("Enter academic performance of student: ");
    if (!int.TryParse(Console.ReadLine(), out academicPerformance))
    {
        Console.WriteLine("\nError! Invalid datatype.\n");
        return null;
    }

    if (academicPerformance > 100 || academicPerformance < 0)
    {
```

```
                Console.WriteLine("\nError! Invalid value\n");
                return null;
            }

            Console.Write("Enter date of birth of student (e.g. 01/01/2001 or 1.1.2001):
");
            if (!DateTime.TryParse(Console.ReadLine(), out dateOfBirth))
            {
                Console.WriteLine("\nError! Invalid datatype.\n");
                return null;
            }

            Console.Write("Enter date of enter to university (e.g. 01/01/2001 or
1.1.2001): ");
            if (!DateTime.TryParse(Console.ReadLine(), out dateOfEnter))
            {
                Console.WriteLine("\nError! Invalid datatype.\n");
                return null;
            }

            Student s = new Student(firstname, surname, groupIndex, faculty,
specialization,
                academicPerformance, dateOfBirth, dateOfEnter);
            return s;
        }
    }
}
```

Student

```
using System;
using System.Runtime.Serialization;
using System.Text;
using Abdullin03;

namespace Abdullin04
{
    [DataContract]
    public class Student
    {
        [DataMember]
        public string FirstName { get; set; }
        [DataMember]
        public string SurName { get; set; }
        [DataMember]
        public string GroupIndex { get; set; }
        [DataMember]
        public string Faculty { get; set; }
        [DataMember]
        public int Specialization { get; set; }
        [DataMember]
        public int AcademicPerformance { get; set; }
        [DataMember]
        public DateTime DateOfBirth { get; set; }
        [DataMember]
        public DateTime DateOfEnter { get; set; }
        [IgnoreDataMember]
        public IPrinter Printer { get; set; }
        public Student() : this("Oleksii", "Abdullin", "a", "CIT", 123, 86, new
DateTime(2002, 5, 31), new DateTime(2019, 8, 12))
        {

        }
        public Student(string FirstName, string SurName, string GroupIndex, string
Faculty,
```

```
            int Specialization, int AcademicPerformance, DateTime DateOfBirth, DateTime
DateOfEnter)
        {
            this.FirstName = FirstName;
            this.SurName = SurName;
            this.GroupIndex = GroupIndex;
            this.Faculty = Faculty;
            this.Specialization = Specialization;
            this.AcademicPerformance = AcademicPerformance;
            this.DateOfBirth = DateOfBirth;
            this.DateOfEnter = DateOfEnter;
            this.Printer = new ConsolePrinter();
        }
        public void Print()
        {
            Printer.Print(ToString());
        }
        public StringBuilder CountAge()
        {
            StringBuilder res = new StringBuilder();
            var diff = (DateTime.Now - DateOfBirth).TotalDays;
            var years = Math.Truncate(diff / 365);
            var days = Math.Truncate(diff % 365);
            return res.Append("Years: ").Append(years).Append("\nDays: ").Append(days);
        }
        public StringBuilder GetGroup()
        {
            StringBuilder res = new StringBuilder();
            return res.Append("\nGroup name: ").Append(Faculty).Append("-
").Append(Specialization).Append(GroupIndex).Append("Year of enter:
").Append(DateOfEnter.Year);
        }
        public StringBuilder CountCourse()
        {
            StringBuilder res = new StringBuilder();
            int course = 0;
            int semester = 0;
            int nowYear = DateTime.Now.Year;
            int nowMonth = DateTime.Now.Month;
            int year = DateOfEnter.Year;
            int month = DateOfEnter.Month;
            course = nowYear - year + 1;
            if (8 > nowMonth)
            {
                course--;
                semester = course * 2;
            }
            else
            {
                semester = course * 2 - 1;
            }
            if (1 > course)
            {
                res.Append("Error! This person can not be student");
                return res;
            }
            else if (course > 6)
            {
                res.Append("This student have been graduated.");
                return res;
            }
            res.Append("Course: ").Append(course).Append("\nSemester:
").Append(semester);
            return res;
        }
```

```
        public override string ToString()
        {
            return "Fristname: " + FirstName + "\nSurName: " + SurName +
                "\nDate of birth: " + DateOfBirth.Day + "." + DateOfBirth.Month + "." +
DateOfBirth.Year +
                "\nDate of enter: " + DateOfEnter.Day + "." + DateOfEnter.Month + "." +
DateOfEnter.Year +
                "\nIndex of group: " + GroupIndex + "\nFaculty: " + Faculty +
                "\nSpecialization: " + Specialization + "\nAcademic Performance: " +
AcademicPerformance + "\n";
        }
        public override bool Equals(object obj)
        {
            if (obj == null)
            {
                return false;
            }
            Student s = obj as Student;
            if (s == null)
            {
                return false;
            }
            return s.FirstName == this.FirstName &&
                    s.SurName == this.SurName &&
                    s.GroupIndex == this.GroupIndex &&
                    s.Faculty == this.Faculty &&
                    s.Specialization == this.Specialization &&
                    s.AcademicPerformance == this.AcademicPerformance &&
                    s.DateOfBirth == this.DateOfBirth &&
                    s.DateOfEnter == this.DateOfEnter;
        }
    }
}
```

Результати роботи програми

```
Enter your option: 3

Menu Output options:
1. Show all students
2. Show course and semester of student
3. Show group of student
4. Show age of student
Enter your option: 4
Enter the student id: 2
Years: 22
Days: 44
```

```
Menu options:
1. Add
2. Remove
3. Show students
4. Ser
5. Deser
6. Calculate
0. Exit
Enter your option: 6

Avarange age: 19,5
Avarange performance: 78,5
```

Рисунок 1 – Результати роботи програми

Висновок: У результаті виконання лабораторної роботи було проведено роботу з LINQ, а саме для доступу до колекції об'єктів (відбір, фільтрація, угруповання, розрахунок) було використано LINQ.