Автор: Аблуллін Олексій

Група: КІТ-119а

Дата: 17.12.2021

Лабораторна робота№ 7

Тема: Об'єктно орієнтована декомпозиція. Рефакторинг – реорганізація програмного коду

Задачі:

1. Оптимізувати структуру класів, згрупувавши методи обробки відповідних класах згідно з призначенням.
2. Реалізувати можливість перегляду користувачем довідника студентів, продемонструвавши різні варіанти застосування LINQ:
   - відкладене виконання запитів;
   - примусове виконання запитів;
   - використання статистичних функцій;
   - використання лямбда виражень;
   - змішаний синтаксис запиту та методу

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;

Calculation – клас, який виконує обробку даних студента;

Текст програми

MyCollection

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

namespace Abdullin07
{
    public class MyCollection : IEnumerable
    {

        private List<Student> _studentsArray = new List<Student>();

        public void Add(Student student)
        {
            if (student is null)
            {
                student = new Student();
            }
            _studentsArray.Add(student);
        }
        public bool RemoveById(int id)
        {
            if (id >= _studentsArray.Count || 0 > id)
            {
                return false;
            }
            _studentsArray.RemoveAt(id);
            return true;
        }
        public bool RemoveByFaculty(string faculty)
        {
            bool flag = false;
            var query = from studens in _studentsArray
                        where studens.Faculty == faculty
                        select studens;

            foreach (var item in query.ToArray())
            {
                flag = _studentsArray.Remove(item);
            }
            return flag;
        }
        public bool RemoveBySpecialization(int specialization)
        {
            bool flag = false;
            var query = (from studens in _studentsArray
                        where studens.Specialization == specialization
                        select studens).ToArray();
            foreach (var item in query)
            {
                flag = _studentsArray.Remove(item);
            }
            return flag;
        }
        public bool RemoveByGroup(string group)
        {
            bool flag = false;
            var query = _studentsArray.Where(_studentsArray =>
_studentsArray.GetGroup().ToString() == group)
                                      .Select(_studentsArray => _studentsArray);
```

```
            //var guery = from studens in _studentsArray
            //            where studens.GetGroup().ToString() == group
            //            select studens;
            foreach (var item in query.ToArray())
            {
                flag = _studentsArray.Remove(item);
            }
            return flag;
        }
        public void Clear()
        {
            _studentsArray.Clear();
        }
        public Student GetStudentById(int id)
        {
            int i = 0;
            if (id >= _studentsArray.Count || 0 > id)
            {
                return null;
            }
            foreach (var stud in _studentsArray)
            {
                if (id == i)
                {
                    return stud;
                }
                i++;
            }
            return null;
        }

        public Student GetStudent(Student student)
        {
            foreach (var stud in _studentsArray)
            {
                if (stud.Equals(student))
                {
                    return student;
                }
            }
            return null;
        }
        public List<Student> GetStudents()
        {
            return _studentsArray;
        }
        public int Count()
        {
            return _studentsArray.Count;
        }

        IEnumerator IEnumerable.GetEnumerator()
        {
            return (IEnumerator)GetEnumerator();
        }

        public MyCollectionEnum GetEnumerator()
        {
            return new MyCollectionEnum(_studentsArray);
        }
    }
}
```

MyCollectionEnum

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

namespace Abdullin07
{
    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();
                }
            }
        }
    }
}
```

Calculation

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

namespace Abdullin07
{
    public static class Calcualtion
    {
```

```
        public static 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 static 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 static 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");
        }
    }
}
```

Menu

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

namespace Abdullin07
{
    public static class Menu
    {
        delegate void Calculate(MyCollection students);
        public static void MenuStudents()
        {
            Student student;
            var MyCollection = new MyCollection();
            MyCollection.Add(new Student());

            int i;
            int option;
            bool inMenu = true;
            string path = "lab07.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 = Calcualtion.AvgAge;
                calculate += Calcualtion.AvgPerformance;
                calculate(MyCollection);
                break;
            case 0:
                inMenu = false;
                break;
            default:
                if (option == -1)
                {
                    break;
                }
                Console.WriteLine("\nIncorrect option. Try again.\n");
                break;
        }
    }
}
public static 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;
        }
    }
}
```

Результат роботи програми

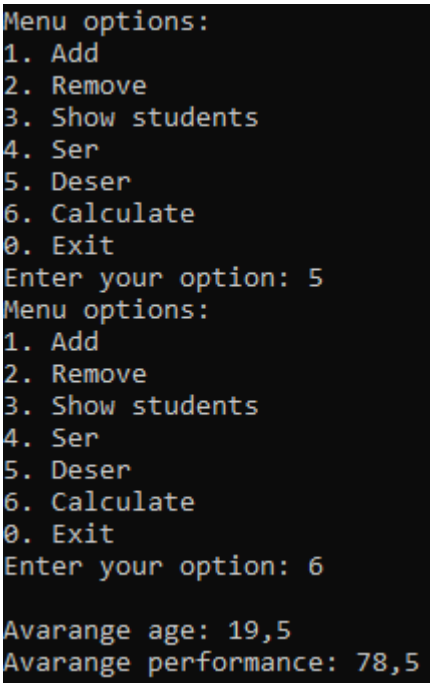

Рисунок 1 – Результати роботи програми

Висновок: У результаті виконання лабораторної роботи було оптимізовано структуру класів, згрупувавши методи обробки відповідних класах згідно з призначенням, реалізовано можливість перегляду користувачем довідника студентів, за використанням різних варіантів застосування LINQ: відкладене виконання, примусове, використання статистичних функцій та лямбда виражень, змішаний синтаксис запиту та методу.