

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ



МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

*Факультет Информационных технологий
Кафедра Информатики и информационных технологий*

направление подготовки

09.03.02 «Информационные системы и технологии»

ЛАБОРАТОРНАЯ РАБОТА № 3

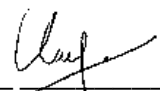
Дисциплина: Объектно-ориентированное программирование

Тема: Пространства имён, модификаторы доступа и принципы SOLID

Выполнил: студент группы 231-338

Шаура Илья Максимович
(Фамилия И.О.)

Дата, подпись 10.09.2024
(Дата)


(Подпись)

Проверил: _____
(Фамилия И.О., степень, звание)

(Оценка)

Дата, подпись _____
(Дата)

(Подпись)

Замечания:

Москва

2024

Цель: получить практические навыки в создании программ, содержащих пространства имён различных типов, и изучить влияние модификаторов доступа. Изучить принципы SOLID

Обоснование:

Пространство имён позволяет логически разделить объекты (классы, структуры, интерфейсы и т.д.) на блоки, в каждом из которых гарантирована уникальность наименований.

Модификаторы доступа позволяют реализовать принцип ООП «Инкапсуляция» путём сокрытия элементов кода в определённых масштабах. Инкапсуляция нужна, чтобы обеспечить целостность объекта и дать возможность пользоваться им, не вдаваясь в подробности его реализации.

Принципы SOLID используют для открытия возможностей развития, масштабирования и тестирования кода.

Задание.

1. Создание класса "Студент" с приватными полями и публичными свойствами для хранения информации о студенте (имя, фамилия, возраст, средний балл).

2. Создание класса "Университет" с приватным полем для хранения списка студентов и публичными методами для добавления, удаления и студентов.

3. Создание пространства имен "DataAccess", добавьте туда класс "StudentsRepository" для реализации доступа к данным (сохранение информации о студентах в файл и чтение информации о студентах из файла).

4. Убедитесь, что выполняются следующие условия:

- В работе соблюдаются принципы SOLID:
 - Принцип единственной ответственности (Single Responsibility Principle).
 - Принцип инверсии зависимостей (Dependency Inversion Principle).

- Соблюдается целостность данных: классы не должны позволять делать то, что недопустимо с логической точки зрения.

Например: валидация полей студента (возраст не может быть отрицательным), проверка данных при добавлении студента (что студент не null) и т.д.

Выполнение

Для соблюдения принципов SOLID для описанных в задании классов были созданы соответствующие интерфейсы, которые эти классы реализовывали. Это позволяет инвертировать зависимость абстракции от реализации, что открывает возможность масштабирования и быстрого рефакторинга кода.

University.cs:

```
namespace Shaura.OOP.Lab3;

public interface IPeopleUnity<T> where T : IPerson
{
    public void Add(T person);
    public void Remove(T person);
    public List<T> Search(string searchQuery);
    public List<T> Get();
}

public class University : IPeopleUnity<Student>
{
    private readonly List<Student> _students = [];

    public void Add(Student person)
    {
        _students.Add(person);
    }

    public void Remove(Student person)
    {
        _students.Remove(person);
    }

    public List<Student> Search(string searchQuery)
    {
        return _students.FindAll(student =>
student.Name.Contains(searchQuery) ||
student.Surname.Contains(searchQuery));
    }

    public List<Student> Get()
    {
        return _students;
    }
}
```

Student.cs:

```
namespace Shaura.OOP.Lab3;

public interface IPerson
{
    public string Name { get; set; }
    public string Surname { get; set; }
    public int Age { get; set; }
}

public class Student : IPerson
{
    private string _name = "EMPTY_NAME";
    private string _surname = "EMPTY_SURNAME";
    private int _age = 16;
    private double _avgScore = 2d;

    public string Name
    {
        get => _name;
        set => _name = value;
    }

    public string Surname
    {
        get => _surname;
        set => _surname = value;
    }

    public int Age
    {
        get => _age;
        set
        {
            if (value < 1) return;
            _age = value;
        }
    }

    public double AvgScore
    {
        get => _avgScore;
        set
        {
            if (value < 1d) return;
            _avgScore = value;
        }
    }

    public Student(string name, string surname, int age, double avgScore)
    {
        Name = name;
        Surname = surname;
        Age = age;
        AvgScore = avgScore;
    }
}
```

StudentsRepository.cs:

```
using System.Text.Json;

namespace Shaura.OOP.Lab3.DataAccess;

public static class StudentsRepository
{
    public static void SaveToFile(string path, List<Student> students)
    {
        var jsonString = JsonSerializer.Serialize(students);
        File.WriteAllText(path, jsonString);
    }

    public static List<Student>? ReadFromFile(string path)
    {
        var jsonString = File.ReadAllText(path);
        var students =
        JsonSerializer.Deserialize<List<Student>>(jsonString);
        return students;
    }
}
```

Program.cs:

```
using Shaura.OOP.Lab3;
using Shaura.OOP.Lab3.DataAccess;

var uni = new University();
var student1 = new Student("John", "Doe", 18, 4.25);
IPerson person2 = new Student("Joe", "Dope", 20, 3.5); // абстракция
(IPerson) не ограничивает реализацию (Student)

uni.Add(student1);
uni.Add((Student)person2); // реализация (University) ограничивает
абстракцию (IPerson)

var list = uni.Get();

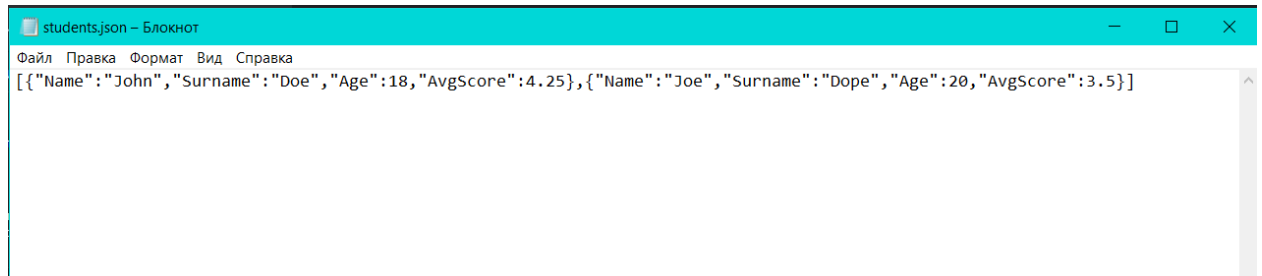
StudentsRepository.SaveToFile("./students.json", list);
var jsonList = StudentsRepository.ReadFromFile("./students.json");

if (jsonList != null)
    foreach (var item in jsonList)
        Console.WriteLine($"{item.Surname} {item.Name}, {item.Age} : ср.
        балл = {item.AvgScore}");
```

Консольный вывод:

```
Doe John, 18 : ср. балл = 4,25  
Dope Joe, 20 : ср. балл = 3,5
```

Созданный файл с сохранением данных студентов:



The screenshot shows a Notepad window with the title bar 'students.json - Блокнот'. The menu bar includes 'Файл', 'Правка', 'Формат', 'Вид', and 'Справка'. The text area contains a JSON array representing two students: `[{"Name": "John", "Surname": "Doe", "Age": 18, "AvgScore": 4.25}, {"Name": "Joe", "Surname": "Dope", "Age": 20, "AvgScore": 3.5}]`. A vertical scrollbar is visible on the right side of the text area.