САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №1

по курсу «ООП»

Тема: Система управления курсами и преподавателями

Выполнила:

Федорова М. В.

К3239

Проверил:

Санкт-Петербург

2025 г.

1. Общее описание проекта

Данный проект представляет собой консольное приложение для управления учебными курсами в университете.

Пользователь может просматривать курсы, преподавателей, студентов, назначать преподавателей и управлять зачислением.

Основные функции:

- Просмотр всех курсов, преподавателей и студентов
- Добавление и удаление курсов
- Назначение преподавателей на курсы
- Зачисление и отчисление студентов на курсы
- Поиск курсов по преподавателю
- Поддержка двух типов курсов: онлайн и офлайн
- Просмотр детальной информации о курсах

Программа использует загрузку данных из JSON-файлов и предоставляет удобный интерфейс взаимодействия.

2. Структура проекта и описание кода

2.1. Program.cs – Точка входа в приложение

Основной файл приложения, который инициализирует систему управления курсами и обрабатывает пользовательский ввод.

Инициализация конфигурации:

```
var configuration = new ConfigurationBuilder()
    .SetBasePath(Directory.GetCurrentDirectory())
    .AddJsonFile("data/appsettings.json", optional: false, reloadOnChange: true)
    .AddJsonFile("data/courses.json", optional: false, reloadOnChange: true)
    .AddJsonFile("data/teachers.json", optional: false, reloadOnChange: true)
    .AddJsonFile("data/students.json", optional: false, reloadOnChange: true)
    .Build();
```

Создается объект конфигурации, который загружает настройки из JSON файлов. `appsettings.json` содержит название системы, `courses.json` - данные о курсах, `teachers.json` - данные о преподавателях, `students.json` - данные о студентах.

Загрузка преподавателей

```
var teachers = new List<Teacher>();
var teachersSection = configuration.GetSection("Teachers");
foreach (var teacherSection in teachersSection.GetChildren())
{
    var id = int.Parse(teacherSection["Id"] ?? "0");
    var name = teacherSection["Name"] ?? "";
    var email = teacherSection["Email"] ?? "";
    var department = teacherSection["Department"] ?? "";
    var specialization = teacherSection["Specialization"] ?? "";
    teachers.Add(new Teacher(id, name, email, department, specialization));
}
```

Код читает секцию "Teachers" из конфигурации и создает объекты `Teacher` для каждого преподавателя, парся значения ID, имени, email, кафедры и специализации.

Загрузка студентов

```
var students = new List<Student>();
var studentsSection = configuration.GetSection("Students");
foreach (var studentSection in studentsSection.GetChildren())
{
    var id = int.Parse(studentSection["Id"] ?? "0");
    var name = studentSection["Name"] ?? "";
    var email = studentSection["Email"] ?? "";
    var year = int.Parse(studentSection["Year"] ?? "0");
    var major = studentSection["Major"] ?? "";
    students.Add(new Student(id, name, email, year, major));
}
```

Создаются объекты `Student` из JSON данных со значениями ID, имени, email, года обучения и специальности.

Загрузка курсов

```
var courses = new List<Course>();
var coursesSection = configuration.GetSection("Courses");
foreach (var courseSection in coursesSection.GetChildren())
{
    var id = int.Parse(courseSection["Id"] ?? "0");
    var name = courseSection["Name"] ?? "";
    var description = courseSection["Description"] ?? "";
    var type = courseSection["Type"] ?? "";
    var teacherId = int.Parse(courseSection["TeacherId"] ?? "0");
    var maxStudents = int.Parse(courseSection["MaxStudents"] ?? "0");

    Course course = type.ToLower() switch
    {
        "online" => new OnlineCourse(id, name, description, teacherId, maxStudents, courseSection["Platform"] ?? ""),
        "offline" => new OfflineCourse(id, name, description, teacherId, maxStudents, courseSection["Room"] ?? "", courseSection["Schedule"] ?? ""),
        _ => throw new ArgumentException($"Unknown course type: {type}")
    };

    // Загружаем студентов курса
    var courseStudentsSection = courseSection.GetSection("Students");
    foreach (var studentId in courseStudentsSection.GetChildren())
    {
        if (int.TryParse(studentId.Value, out var studentIdInt))
        {
            course.EnrollStudent(studentIdInt);
        }
    }

    courses.Add(course);
}
```

1. Создаются объекты курсов в зависимости от типа ("online" или "offline")

2. Для онлайн-курсов создается `OnlineCourse` с указанием платформы

3. Для офлайн-курсов создается `OfflineCourse` с указанием аудитории и расписания

4. Загружаются и зачисляются студенты, указанные в JSON файле.

Основной цикл программы

```
while (true)
{
    Console.WriteLine();
    Console.WriteLine("1) Show all courses");
    Console.WriteLine("2) Show all teachers");
    Console.WriteLine("3) Show all students");
    Console.WriteLine("4) Show courses by teacher");
    Console.WriteLine("5) Add new course");
    Console.WriteLine("6) Remove course");
    Console.WriteLine("7) Enroll student in course");
    Console.WriteLine("8) Unenroll student from course");
    Console.WriteLine("9) Show course details");
    Console.WriteLine("0) Exit");
    Console.Write("Choose: ");
```

Бесконечный цикл, который отображает меню пользователя и обрабатывает выбор через `switch` конструкцию. Каждая опция вызывает соответствующий метод для выполнения операции.

2.2. Класс Course (course.cs)

Абстрактный базовый класс для всех типов курсов, определяющий общий интерфейс и функциональность.

Поля и свойства класса

```
public int Id { get; }
public string Name { get; }
public string Description { get; }
public int TeacherId { get; }
public int MaxStudents { get; }
public List<int> Students { get; }
```

- `Id` - уникальный идентификатор курса (только чтение)

- `Name` - название курса (только чтение)

- `Description` - описание курса (только чтение)

- `TeacherId` - ID преподавателя, ведущего курс (только чтение)

- `MaxStudents` - максимальное количество студентов (только чтение)

- `Students` - список ID зачисленных студентов

Конструктор

```
protected Course(int id, string name, string description, int teacherId, int maxStudents)
{
    Id = id;
    Name = name;
    Description = description;
    TeacherId = teacherId;
    MaxStudents = maxStudents;
    Students = new List<int>();
}
```

Защищенный конструктор инициализирует базовые свойства курса и создает пустой список студентов. Используется производными классами.

** Методы управления студентами**

```
public bool CanEnrollStudent(int studentId)
{
    return !Students.Contains(studentId) && Students.Count < MaxStudents;
}

public bool EnrollStudent(int studentId)
{
    if (!CanEnrollStudent(studentId))
        return false;

    Students.Add(studentId);
    return true;
}

public bool UnenrollStudent(int studentId)
{
    return Students.Remove(studentId);
}
```

- `CanEnrollStudent` - проверяет возможность зачисления (студент не зачислен и есть свободные места)

- `EnrollStudent` - зачисляет студента, если это возможно

- `UnenrollStudent` - отчисляет студента с курса

Абстрактные методы

```
public abstract string GetCourseDetails();
public abstract string GetLocationInfo();
```

Абстрактные методы, которые должны быть реализованы в производных классах для получения детальной информации о курсе и его расположении.

2.3. Класс OnlineCourse (onlinecourse.cs)

Класс для представления онлайн-курсов, наследующийся от `Course`.

Свойства и конструктор

```
public string Platform { get; }

public OnlineCourse(int id, string name, string description, int teacherId, int maxStudents, string platform)
    : base(id, name, description, teacherId, maxStudents)
{
    Platform = platform;
}
```

Добавляет свойство `Platform` для указания платформы проведения онлайн-курса (Zoom, Teams и т.д.). Вызывает конструктор базового класса.

Переопределение методов

```
public override string GetCourseDetails()
{
    return $"Online Course: {Name}\nDescription: {Description}\nPlatform: {Platform}\nMax Students: {MaxStudents}\nCurrent Students: {Students.Count}";
}

public override string GetLocationInfo()
{
    return $"Online platform: {Platform}";
}
```

Реализует абстрактные методы базового класса, предоставляя специфичную для онлайн-курсов информацию о платформе проведения.

2.4. Класс OfflineCourse (offlinecourse.cs)

Класс для представления офлайн-курсов, наследующийся от `Course`.

Свойства и конструктор

```
public string Room { get; }
public string Schedule { get; }

public OfflineCourse(int id, string name, string description, int teacherId, int maxStudents, string room, string s
    : base(id, name, description, teacherId, maxStudents)
{
    Room = room;
    Schedule = schedule;
}
```

Добавляет свойства `Room` (аудитория) и `Schedule` (расписание). Вызывает конструктор базового класса.

Переопределение методов

```
public override string GetCourseDetails()
{
    return $"Offline Course: {Name}\nDescription: {Description}\nRoom: {Room}\nSchedule: {Schedule}\nMax Students: {MaxStudents}\nCurrent Stud
}

public override string GetLocationInfo()
{
    return $"Room: {Room}, Schedule: {Schedule}";
}
```

Реализует абстрактные методы, предоставляя информацию об аудитории и расписании офлайн-курса.

2.5. Класс Teacher (teacher.cs)

Модель преподавателя, хранящая информацию о преподавателях университета.

Свойства и конструктор

```
public int Id { get; }
public string Name { get; }
public string Email { get; }
public string Department { get; }
public string Specialization { get; }

public Teacher(int id, string name, string email, string department, string specialization)
{
    Id = id;
    Name = name;
    Email = email;
    Department = department;
    Specialization = specialization;
}
```

Инициализирует все свойства преподавателя переданными значениями. Все свойства доступны только для чтения после создания объекта.

2.6. Класс Student (student.cs)

Модель студента, хранящая информацию о студентах университета.

Свойства и конструктор

```
public int Id { get; }
public string Name { get; }
public string Email { get; }
public int Year { get; }
public string Major { get; }

public Student(int id, string name, string email, int year, string major)
{
    Id = id;
    Name = name;
    Email = email;
    Year = year;
    Major = major;
}
```

Инициализирует все свойства студента, включая ID, имя, email, год обучения и специальность.

2.7. Класс CourseManager (coursemanager.cs)

Основной класс для управления курсами, преподавателями и студентами.

Поля класса

```
private readonly List<Course> _courses;
private readonly List<Teacher> _teachers;
private readonly List<Student> _students;
```

- `_courses` - список всех курсов в системе

- `_teachers` - список всех преподавателей

- `_students` - список всех студентов

Конструкторы

```
public CourseManager()
{
    _courses = new List<Course>();
    _teachers = new List<Teacher>();
    _students = new List<Student>();
}

public CourseManager(List<Course> courses, List<Teacher> teachers, List<Student> students)
{
    _courses = courses ?? new List<Course>();
    _teachers = teachers ?? new List<Teacher>();
    _students = students ?? new List<Student>();
}
```

- Первый конструктор создает пустой менеджер

- Второй инициализирует менеджер с переданными данными

Метод добавления курса

```
public bool AddCourse(Course course)
{
    if (course == null || _courses.Any(c => c.Id == course.Id))
        return false;

    _courses.Add(course);
    return true;
}

public bool RemoveCourse(int courseId)
{
    var course = _courses.FirstOrDefault(c => c.Id == courseId);
    if (course == null)
        return false;

    _courses.Remove(course);
    return true;
}
```

1: Проверяет корректность курса и отсутствие дубликатов по ID, затем добавляет курс в список.

2: Находит курс по ID и удаляет его из списка, возвращает false если курс не найден.

Метод зачисления студента

```
public bool EnrollStudentInCourse(int courseId, int studentId)
{
    var course = GetCourse(courseId);
    var student = GetStudent(studentId);

    if (course == null || student == null)
        return false;

    return course.EnrollStudent(studentId);
}
```

1. Проверяет существование курса и студента

2. Если оба существуют, вызывает метод зачисления курса

3. Возвращает false если курс или студент не найдены

** Метод поиска курсов по преподавателю**

```
public IEnumerable<Course> GetCoursesByTeacher(int teacherId)
{
    return _courses.Where(c => c.TeacherId == teacherId);
}
```

Использует LINQ для фильтрации курсов по ID преподавателя и возвращает коллекцию соответствующих курсов.

2.8. Класс CourseFactory (coursefactory.cs)

Фабрика для создания различных типов курсов (паттерн Factory Method).

** Методы создания курсов**

```
public Course CreateOnlineCourse(int id, string name, string description, int teacherId, int maxStudents, string platform)
{
    return new OnlineCourse(id, name, description, teacherId, maxStudents, platform);
}

public Course CreateOfflineCourse(int id, string name, string description, int teacherId, int maxStudents, string room, string schedule)
{
    return new OfflineCourse(id, name, description, teacherId, maxStudents, room, schedule);
}
```

Инкапсулирует создание объектов курсов разных типов, возвращая их через общий интерфейс `Course`.

2.9. Класс CourseBuilder (coursefactory.cs)

Реализация паттерна Builder для пошагового создания курсов с различными конфигурациями.

Поля класса

```
private int _id;
private string _name = string.Empty;
private string _description = string.Empty;
private int _teacherId;
private int _maxStudents;
private string _platform = string.Empty;
private string _room = string.Empty;
private string _schedule = string.Empty;
private CourseType _type;
```

Приватные поля для хранения параметров курса до его создания.

** Методы настройки **

```
public CourseBuilder SetId(int id)
{
    _id = id;
    return this;
}

public CourseBuilder SetName(string name)
{
    _name = name;
    return this;
}
```

Каждый метод устанавливает соответствующий параметр и возвращает `this` для цепочки вызовов (fluent interface).

** Метод построения **

```
public Course Build()
{
    return _type switch
    {
        CourseType.Online => new OnlineCourse(_id, _name, _description, _teacherId, _maxStudents, _platform),
        CourseType.Offline => new OfflineCourse(_id, _name, _description, _teacherId, _maxStudents, _room, _schedule),
        _ => throw new ArgumentException("Invalid course type")
    };
}
```

Создает соответствующий объект курса в зависимости от установленного типа, используя накопленные параметры.

** Пример использования **

```
var course = new CourseBuilder()
    .SetId(1)
    .SetName("Programming")
    .SetDescription("Learn programming")
    .SetTeacherId(1)
    .SetMaxStudents(30)
    .SetType(CourseType.Online)
    .SetPlatform("Zoom")
    .Build();
```

2.10. Класс CourseManagementSystem (coursemanagementsystem.cs)

Реализация паттерна Singleton для обеспечения единственного экземпляра системы управления курсами.

** Поля, свойство и конструктор **

```
private static CourseManagementSystem? _instance;
private static readonly object _lock = new object();

private CourseManagementSystem()
{
    CourseManager = new CourseManager();
    CourseFactory = new CourseFactory();
}

public static CourseManagementSystem Instance
{
    get
    {
        if (_instance == null)
        {
            lock (_lock)
            {
                _instance ??= new CourseManagementSystem();
            }
        }
        return _instance;
    }
}
```

- Использует паттерн double-checked locking для потокобезопасности

- Гарантирует создание только одного экземпляра системы

- Возвращает существующий экземпляр при повторных обращениях

- Приватный конструктор инициализирует компоненты системы и предотвращает создание экземпляров извне.

3. Принципы ООП в проекте

Наследование

* Базовый абстрактный класс `Course` определяет общую структуру и поведение для всех курсов

* Классы `OnlineCourse` и `OfflineCourse` наследуются от `Course` и расширяют функциональность:

* `OnlineCourse` добавляет свойство платформы

* `OfflineCourse` добавляет свойства аудитории и расписания

* Наследование позволяет использовать полиморфизм и переиспользовать общий код

Полиморфизм

* Методы `GetCourseDetails()` и `GetLocationInfo()` переопределяются в производных классах

* Оба типа курсов могут обрабатываться через общий интерфейс `Course`

* Использование абстрактных методов обеспечивает различную реализацию для каждого типа курса

Пример полиморфизма

```
Course onlineCourse = new OnlineCourse(...);
Course offlineCourse = new OfflineCourse(...);

// Одинаковый вызов, разная реализация
string details1 = onlineCourse.GetCourseDetails();   // Вернет информацию о платформе
string details2 = offlineCourse.GetCourseDetails();  // Вернет информацию об аудитории
```

Абстракция

* Абстрактный класс `Course` скрывает детали реализации, предоставляя общий интерфейс

* Интерфейсы `ICourseManager` и `ICourseFactory` определяют контракты без реализации

* Пользователь работает с высокоуровневыми абстракциями, не зная деталей реализации

Инкапсуляция

* Приватные поля скрыты от внешнего доступа (`_courses`, `_teachers`, `_students`)

* Публичные методы предоставляют контролируемый интерфейс для взаимодействия

* Свойства с геттерами только для чтения защищают данные от несанкционированного изменения

* Singleton паттерн инкапсулирует логику создания единственного экземпляра

4. Паттерны проектирования

4.1 Singleton Pattern

Реализован в классе `CourseManagementSystem` для обеспечения единственного экземпляра системы управления курсами в приложении. Полезно для управления общим состоянием и ресурсами.

4.2 Builder Pattern

Реализован в классе `CourseBuilder` для пошагового создания объектов курсов с различными конфигурациями. Позволяет гибко настраивать параметры курса перед его созданием.

4.3 Factory Method Pattern

Реализован в классе `CourseFactory` для создания объектов различных типов курсов через единый интерфейс. Инкапсулирует логику создания и позволяет легко добавлять новые типы курсов.

5. Unit тестирование

Проект включает 36 unit тестов, написанных с использованием фреймворка xUnit, которые покрывают основную бизнес-логику:

** Тесты для классов данных **

StudentTests (2 теста) - создание и форматирование студентов

TeacherTests (2 теста) - создание и форматирование преподавателей

** Тесты для курсов **

CourseTests (12 тестов) - создание курсов, зачисление/отчисление студентов, проверка деталей курсов, проверка локации

** Тесты для менеджера **

CourseManagerTests (10 тестов) - управление курсами, студентами, валидация существования студентов и курсов

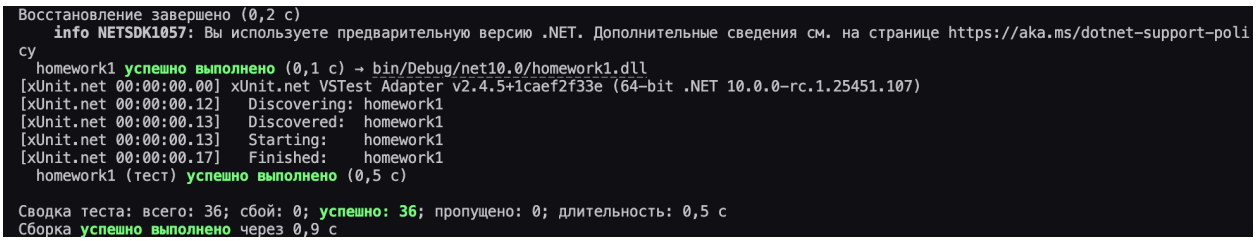
Тесты для паттернов

CourseBuilderTests (3 теста) - проверка работы паттерна Builder для создания онлайн и офлайн курсов

CourseManagementSystemSingletonTests (4 теста) - проверка паттерна Singleton

CourseFactoryTests (2 теста) - проверка фабрики курсов

** Запуск тестов **



Для запуска всех тестов используется команда: dotnet test

Для запуска с подробным выводом: dotnet test --verbosity detailed

6. Обработка ошибок и валидация

- Валидация существования студентов: При попытке зачислить студента система проверяет, существует ли студент в базе данных

- Валидация существования курсов: При операциях с курсами проверяется их наличие

- Проверка лимитов: Система предотвращает зачисление студентов сверх лимита курса

- Проверка дубликатов: Предотвращается зачисление одного студента дважды на один курс

- Информативные сообщения: При ошибках пользователь получает понятные сообщения о причинах

- Обработка некорректных данных: Проверка пользовательского ввода на всех этапах

7. Вывод

Система демонстрирует комплексное использование принципов ООП:

- Наследование для создания иерархии типов курсов

- Полиморфизм для единообразной работы с разными типами

- Абстракция для скрытия деталей реализации

- Инкапсуляция для защиты данных и контролируемого доступа

Проект использует современные паттерны проектирования (Singleton, Builder, Factory Method) и полностью покрыт unit тестами, что обеспечивает надежность и поддерживаемость кода.