**Міністерство освіти і науки України**

**Національний авіаційний університет**



Курсова робота №1

з дисципліни «Об'єктно-орієнтоване програмування»

«Журнал успішності: електронний журнал успішності студентів»

Виконав студент:

групи Б-121-23-1-ПІ

Макоїд Костянтин

Перевірила викладач:

Київ 2024

**Зміст**

[Опис завдання 3](#__RefHeading___Toc1476_3394856270)

[Опис бізнес-частини проекту 6](#__RefHeading___Toc1479_3394856270)

[Опис інтерфейсної частини проекту 7](#__RefHeading___Toc1481_3394856270)

[Хід роботи 9](#__RefHeading___Toc1483_3394856270)

[ESPL (Solution) 9](#__RefHeading___Toc2485_3901895441)

[ESPL (Console App) 9](#__RefHeading___Toc22466_3901895441)

[Program.cs 9](#__RefHeading___Toc22468_3901895441)

[GradeMenu.cs 10](#__RefHeading___Toc22470_3901895441)

[GroupMenu.cs 13](#__RefHeading___Toc22472_3901895441)

[MainMenu.cs 17](#__RefHeading___Toc22474_3901895441)

[SearchMenu.cs 19](#__RefHeading___Toc22476_3901895441)

[StudentMenu.cs 23](#__RefHeading___Toc22478_3901895441)

[SubjectMenu.cs 26](#__RefHeading___Toc22480_3901895441)

[Espl\_Core (Class library) 28](#__RefHeading___Toc22482_3901895441)

[Models/IPerson.cs 28](#__RefHeading___Toc1488_3394856270)

[Models/Student.cs 28](#__RefHeading___Toc1490_3394856270)

[Models/Group.cs 29](#__RefHeading___Toc1492_3394856270)

[Services/GradeBook.cs 30](#__RefHeading___Toc1494_3394856270)

[Exceptions/DuplicateGroupException.cs 39](#__RefHeading___Toc1496_3394856270)

[Exceptions/DuplicateStudentException.cs 39](#__RefHeading___Toc1498_3394856270)

[Exceptions/GroupNotFoundException.cs 40](#__RefHeading___Toc1500_3394856270)

[Exceptions/StudentNotFoundException.cs 40](#__RefHeading___Toc1502_3394856270)

[Exceptions/InvalidStudentIdFormatException.cs 40](#__RefHeading___Toc1496_3394856270_%25D0%)

[Exceptions/DuplicateSubjectException.cs 41](#__RefHeading___Toc1504_3394856270)

[Exceptions/SubjectNotFoundException.cs 41](#__RefHeading___Toc1506_3394856270)

[Ілюстрації 42](#__RefHeading___Toc1508_3394856270)

# Опис завдання

Варіант 12

Журнал успішності: електронний журнал успішності студентів  
(ESPL - electronic student progress log)

Завдання:

1. В якості типу застосування обрати консольне застосування з командним рядком.

2. Спроектувати та реалізувати систему класів, в основу якої покладено логічну структуру даних, наведену у варіанті, для накопичення та обробки даних домену відповідно варіанту курсової роботи.

3. Для зберігання даних при виконанні програми в випадку написання коду використати:

• Масиви

• або колекції\*

• узагальнені колекції \*\*.

4. Описати елементарні операції з даними, використовуючи інтерфейси та абстрактні класи (стандартні, наприклад, IComparer, та власні).

5. Перед виходом із застосування накопичені дані зберегти у файл, на початку роботи з застосуванням дані завантажити із файлу (якщо дані існують).\*

• Роботу з файлами реалізувати через серіалізацію в C# \*\*\*

6. Реалізувати обов’язковий контроль коректності введених даних.

7. Реалізувати перевірку вийняткових ситуацій; в тому числі – при роботі з даними. При необхідності створити власні класи виключень\* (наприклад, виключення для ситуації перевищення ліміту кількості книжок на абонементі в бібліотеці).

8. Розділити бізнес-логіку та інтерфейс користувача шляхом виділення бізнес-логіки в окремий проект – dll–бібліотеку. Бібліотеку підключити в проект з реалізацією інтерфейсу користувача.\*\*

9. При написанні коду застосувати правила «хорошого» стилю програмування. А саме:

створювати мнемонічні ідентифікатори, чітко форматувати код, застосувати при необхідності коментарі. Не використовувати public членів-даних у класах. За необхідності доступу до них використати властивості, індексатори чи окремі методи- аксесори. Опис класів наводити в окремих файлах. За неохайне оформлення коду можливе зниження оцінки за курсову роботу!

Функціональні вимоги до програмного забезпечення

1. Управління групою студентів

1.1. Можливість додавати студента

1.2. Можливість видаляти студента

1.3. Можливість змінити дані про студента

1.4. Можливість перегляду списку всіх студентів

1.5. Можливість перегляду даних про успішність вказаного студента (по окремій

дисципліні чи підсумкові оцінки з дисциплін)

2. Управління групами

2.1. Можливість додавати групу

2.2. Можливість видаляти групу

2.3. Можливість змінювати дані групи

2.4. Можливість перегляду списку даних групи

2.5. Можливість перегляду даних певної групи

3. Управління навчальним процесом (успішністю студентів)

3.1. Можливість додавання предмета вивчення до даних про успішність студентів

3.2. Можливість видалення предмета вивчення

3.3. Можливість додавання/змінення оцінок студента з окремого предмету

3.4. Можливість отримання оцінок студентів/певного студенту (загалом/за предметом

вивчення)

3.5. Можливість підрахування середнього балу успішності окремого студента/студентів

групи

4. Пошук

4.1. Можливість пошуку студента за його даними (прізвище, ім’я)

4.2. Можливість пошуку студентів певної групи

4.3. Можливість пошуку студентів за певним середнім балом

4.4. Можливість пошуку успішних/неуспішних студентів (загалом/за предметом

вивчення)

# Опис бізнес-частини проекту

Бізнес-частина проекту реалізована в класі GradeBook, який містить основну логіку управління даними. Основні компоненти:

1. Управління групами:
   * Додавання нових груп
   * Видалення груп
   * Редагування назв груп
   * Отримання списку всіх груп
   * Отримання інформації про конкретну групу
2. Управління студентами:
   * Додавання студентів до груп
   * Видалення студентів з груп
   * Редагування інформації про студентів
   * Отримання списку всіх студентів
   * Пошук студента за ідентифікатором
3. Управління предметами:
   * Додавання нових предметів
   * Видалення предметів
   * Отримання списку всіх предметів
4. Управління оцінками:
   * Додавання оцінок студентам за предметами
   * Редагування оцінок
   * Розрахунок середнього балу студента
   * Розрахунок середнього балу групи
5. Функції пошуку:
   * Пошук студентів за іменем та прізвищем
   * Пошук студентів за групою
   * Пошук студентів за середнім балом
   * Пошук успішних студентів
6. Збереження та завантаження даних:
   * Збереження всіх даних у JSON файл
   * Завантаження даних з JSON файлу при запуску програми
7. Обробка винятків:
   * Власні класи винятків для обробки специфічних помилок (наприклад, DuplicateGroupException, StudentNotFoundException)

Основні класи бізнес-логіки:

* Student: представляє студента з його особистими даними та оцінками
* Group: представляє групу студентів
* GradeBook: основний клас, що керує всіма даними та операціями

# Опис інтерфейсної частини проекту

Інтерфейсна частина проекту реалізована як консольний додаток з текстовим меню. Основні компоненти:

1. Головне меню:
   * Управління групами
   * Управління студентами
   * Управління предметами
   * Управління оцінками
   * Пошук
   * Зберегти та вийти
2. Підменю для кожного розділу:
   * Управління групами: додавання, видалення, редагування, перегляд
   * Управління студентами: додавання, видалення, редагування, перегляд
   * Управління предметами: додавання, видалення, перегляд
   * Управління оцінками: додавання, редагування, перегляд, розрахунок середніх балів
   * Пошук: за іменем, за групою, за середнім балом, пошук успішних студентів
3. Введення даних:
   * Користувач вводить дані через консоль (наприклад, назви груп, інформацію про студентів, оцінки)
   * Реалізовано базову валідацію введених даних (наприклад, перевірка на числові значення для оцінок)
4. Виведення результатів:
   * Результати операцій виводяться в консоль
   * Списки (груп, студентів, предметів) виводяться у форматованому вигляді
5. Обробка помилок:
   * Виведення повідомлень про помилки в консоль (наприклад, при спробі додати дублікат групи)
   * Перехоплення винятків для запобігання аварійного завершення програми
6. Локалізація:
   * Використання української мови для всіх повідомлень та введення даних
   * Налаштування кодування консолі для коректного відображення кирилиці
7. Збереження даних:
   * Автоматичне збереження даних при виході з програми
   * Завантаження збережених даних при запуску програми

Ключові особливості інтерфейсу:

* Простий текстовий інтерфейс, зрозумілий для користувача
* Інтуїтивна навігація між різними функціями програми
* Чітке відображення результатів операцій та помилок
* Можливість повернення до попереднього меню або головного меню
* Консистентний формат введення та виведення даних

# Хід роботи

## ESPL (Solution)

## ESPL (Console App)

### Program.cs

using System;

using System.Text;

using System.Globalization;

using GradeBookSystem.ConsoleMenu;

using GradeBookSystem.Models;

using GradeBookSystem.Services;

namespace GradeBookSystem

{

class Program

{

static GradeBook gradeBook;

static Program()

{

try

{

gradeBook = new GradeBook();

}

catch (Exception ex)

{

Console.WriteLine($"Помилка при ініціалізації GradeBook: {ex.Message}");

if (ex.InnerException != null)

{

Console.WriteLine($"Inner exception: {ex.InnerException.Message}");

}

Environment.Exit(1);

}

}

static void Main(string[] args)

{

Console.OutputEncoding = Encoding.UTF8;

Console.InputEncoding = Encoding.UTF8;

CultureInfo.CurrentCulture = new CultureInfo("uk-UA");

CultureInfo.CurrentUICulture = new CultureInfo("uk-UA");

var mainMenu = new MainMenu(gradeBook);

mainMenu.Show();

}

}

}

### GradeMenu.cs

using System;

using GradeBookSystem.Models;

using GradeBookSystem.Services;

namespace GradeBookSystem.ConsoleMenu

{

public class GradeMenu

{

private readonly GradeBook \_gradeBook;

private readonly string[] \_menuItems = {

"Додати оцінку",

"Змінити оцінку",

"Переглянути оцінки студента",

"Підрахувати середній бал студента",

"Підрахувати середній бал групи",

"Повернутися до головного меню"

};

public GradeMenu(GradeBook gradeBook)

{

\_gradeBook = gradeBook;

}

public void Show()

{

while (true)

{

Console.Clear();

Console.WriteLine("\nМеню управління оцінками:");

for (int i = 0; i < \_menuItems.Length; i++)

{

Console.WriteLine($"{i + 1}. {\_menuItems[i]}");

}

Console.Write("\nОберіть опцію: ");

if (int.TryParse(Console.ReadLine(), out int choice) && choice >= 1 && choice <= \_menuItems.Length)

{

Console.Clear();

if (ExecuteMenuItem(choice - 1))

return;

Console.WriteLine("\nНатисніть будь-яку клавішу, щоб продовжити...");

Console.ReadKey(true);

}

else

{

Console.WriteLine("Невірний вибір. Спробуйте ще раз.");

}

}

}

private bool ExecuteMenuItem(int index)

{

switch (index)

{

case 0:

AddGrade();

break;

case 1:

EditGrade();

break;

case 2:

ViewStudentGrades();

break;

case 3:

CalculateStudentAverage();

break;

case 4:

CalculateGroupAverage();

break;

case 5:

return true;

}

return false;

}

private void AddGrade()

{

try

{

Console.Write("Введіть ID студента: ");

string studentId = Console.ReadLine();

Console.Write("Введіть назву предмета: ");

string subject = Console.ReadLine();

Console.Write("Введіть оцінку: ");

if (int.TryParse(Console.ReadLine(), out int grade))

{

\_gradeBook.AddGrade(studentId, subject, grade);

Console.WriteLine("Оцінку успішно додано.");

}

else

{

Console.WriteLine("Некоректне введення оцінки.");

}

}

catch (Exception ex)

{

Console.WriteLine($"Помилка: {ex.Message}");

}

}

private void EditGrade()

{

try

{

Console.Write("Введіть ID студента: ");

string studentId = Console.ReadLine();

Console.Write("Введіть назву предмета: ");

string subject = Console.ReadLine();

Console.Write("Введіть індекс оцінки для зміни: ");

if (int.TryParse(Console.ReadLine(), out int index))

{

Console.Write("Введіть нову оцінку: ");

if (int.TryParse(Console.ReadLine(), out int newGrade))

{

\_gradeBook.EditGrade(studentId, subject, index, newGrade);

Console.WriteLine("Оцінку успішно змінено.");

}

else

{

Console.WriteLine("Некоректне введення оцінки.");

}

}

else

{

Console.WriteLine("Некоректне введення індексу.");

}

}

catch (Exception ex)

{

Console.WriteLine($"Помилка: {ex.Message}");

}

}

private void ViewStudentGrades()

{

Console.Write("Введіть ID студента: ");

string studentId = Console.ReadLine();

try

{

var student = \_gradeBook.GetStudentById(studentId);

Console.WriteLine($"Оцінки студента {student.FirstName} {student.LastName}:");

foreach (var subject in student.Grades.Keys)

{

Console.WriteLine($"{subject}: {string.Join(", ", student.Grades[subject])}");

}

}

catch (Exception ex)

{

Console.WriteLine($"Помилка: {ex.Message}");

}

}

private void CalculateStudentAverage()

{

try

{

Console.Write("Введіть ID студента: ");

string studentId = Console.ReadLine();

double average = \_gradeBook.CalculateStudentAverage(studentId);

Console.WriteLine($"Середній бал студента: {average:F2}");

}

catch (Exception ex)

{

Console.WriteLine($"Помилка: {ex.Message}");

}

}

private void CalculateGroupAverage()

{

try

{

Console.Write("Введіть назву групи: ");

string groupName = Console.ReadLine();

double average = \_gradeBook.CalculateGroupAverage(groupName);

Console.WriteLine($"Середній бал групи: {average:F2}");

}

catch (Exception ex)

{

Console.WriteLine($"Помилка: {ex.Message}");

}

}

}

}

### GroupMenu.cs

using System;

using GradeBookSystem.Models;

using GradeBookSystem.Services;

using GradeBookSystem.Exceptions;

namespace GradeBookSystem.ConsoleMenu

{

public class GroupMenu

{

private readonly GradeBook \_gradeBook;

private readonly int \_menuItemCount = 6;

private int \_selectedIndex = 0;

private readonly string[] \_menuItems = {

"Додати групу",

"Видалити групу",

"Змінити дані групи",

"Переглянути список груп",

"Переглянути дані певної групи",

"Повернутися до головного меню"

};

public GroupMenu(GradeBook gradeBook)

{

\_gradeBook = gradeBook;

}

public void Show()

{

while (true)

{

Console.Clear();

DrawMenu();

ConsoleKeyInfo keyInfo = Console.ReadKey(true);

switch (keyInfo.Key)

{

case ConsoleKey.UpArrow:

\_selectedIndex = (\_selectedIndex - 1 + \_menuItemCount) % \_menuItemCount;

break;

case ConsoleKey.DownArrow:

\_selectedIndex = (\_selectedIndex + 1) % \_menuItemCount;

break;

case ConsoleKey.Enter:

Console.Clear();

try

{

ExecuteMenuItem();

if (\_selectedIndex == \_menuItemCount - 1) // Return to main menu option

{

return;

}

}

catch (Exception ex)

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine($"\nПомилка: {ex.Message}");

Console.ResetColor();

}

Console.WriteLine("\nНатисніть будь-яку клавішу, щоб повернутися до меню груп...");

Console.ReadKey(true);

break;

}

}

}

private void DrawMenu()

{

Console.WriteLine("Меню управління групами:");

for (int i = 0; i < \_menuItems.Length; i++)

{

if (i == \_selectedIndex)

{

Console.BackgroundColor = ConsoleColor.Gray;

Console.ForegroundColor = ConsoleColor.Black;

}

Console.WriteLine($"{i + 1}. {\_menuItems[i]}");

Console.ResetColor();

}

}

private void ExecuteMenuItem()

{

switch (\_selectedIndex)

{

case 0:

AddGroup();

break;

case 1:

RemoveGroup();

break;

case 2:

EditGroup();

break;

case 3:

ViewAllGroups();

break;

case 4:

ViewGroupDetails();

break;

}

}

private void AddGroup()

{

try

{

Console.Write("Введіть назву групи: ");

string groupName = Console.ReadLine();

if (string.IsNullOrWhiteSpace(groupName))

{

throw new ArgumentException("Назва групи не може бути порожньою.");

}

\_gradeBook.AddGroup(groupName);

Console.WriteLine("Групу успішно додано.");

}

catch (ArgumentException ex)

{

throw new ArgumentException($"Помилка при додаванні групи: {ex.Message}");

}

}

private void RemoveGroup()

{

try

{

Console.Write("Введіть назву групи: ");

string groupName = Console.ReadLine();

if (string.IsNullOrWhiteSpace(groupName))

{

throw new ArgumentException("Назва групи не може бути порожньою.");

}

\_gradeBook.RemoveGroup(groupName);

Console.WriteLine("Групу успішно видалено.");

}

catch (GroupNotFoundException ex)

{

throw new GroupNotFoundException($"Помилка при видаленні групи: {ex.Message}");

}

}

private void EditGroup()

{

try

{

Console.Write("Введіть поточну назву групи: ");

string currentName = Console.ReadLine();

Console.Write("Введіть нову назву групи: ");

string newName = Console.ReadLine();

if (string.IsNullOrWhiteSpace(currentName) || string.IsNullOrWhiteSpace(newName))

{

throw new ArgumentException("Назва групи не може бути порожньою.");

}

\_gradeBook.EditGroup(currentName, newName);

Console.WriteLine("Дані групи успішно змінено.");

}

catch (GroupNotFoundException ex)

{

throw new GroupNotFoundException($"Помилка при редагуванні групи: {ex.Message}");

}

catch (ArgumentException ex)

{

throw new ArgumentException($"Помилка при редагуванні групи: {ex.Message}");

}

}

private void ViewAllGroups()

{

var groups = \_gradeBook.GetAllGroups();

if (!groups.Any())

{

Console.WriteLine("Список груп порожній.");

return;

}

foreach (var group in groups)

{

Console.WriteLine(group.GroupName);

}

}

private void ViewGroupDetails()

{

try

{

Console.Write("Введіть назву групи: ");

string groupName = Console.ReadLine();

if (string.IsNullOrWhiteSpace(groupName))

{

throw new ArgumentException("Назва групи не може бути порожньою.");

}

var group = \_gradeBook.GetGroupByName(groupName);

Console.WriteLine($"Група: {group.GroupName}");

Console.WriteLine("Студенти:");

if (!group.Students.Any())

{

Console.WriteLine("У групі немає студентів.");

return;

}

foreach (var student in group.Students)

{

Console.WriteLine($"{student.FirstName} {student.LastName} (ID: {student.StudentId})");

}

}

catch (GroupNotFoundException ex)

{

throw new GroupNotFoundException($"Помилка при перегляді даних групи: {ex.Message}");

}

}

}

}

### MainMenu.cs

using System;

using GradeBookSystem.Models;

using GradeBookSystem.Services;

namespace GradeBookSystem.ConsoleMenu

{

public class MainMenu

{

private readonly GradeBook \_gradeBook;

private readonly int \_menuItemCount = 6;

private int \_selectedIndex = 0;

private readonly string[] \_menuItems = {

"Управління групами",

"Управління студентами",

"Управління предметами",

"Управління оцінками",

"Пошук",

"Зберегти та вийти"

};

public MainMenu(GradeBook gradeBook)

{

\_gradeBook = gradeBook;

}

public void Show()

{

while (true)

{

Console.Clear();

DrawMenu();

ConsoleKeyInfo keyInfo = Console.ReadKey(true);

switch (keyInfo.Key)

{

case ConsoleKey.UpArrow:

\_selectedIndex = (\_selectedIndex - 1 + \_menuItemCount) % \_menuItemCount;

break;

case ConsoleKey.DownArrow:

\_selectedIndex = (\_selectedIndex + 1) % \_menuItemCount;

break;

case ConsoleKey.Enter:

Console.Clear();

ExecuteMenuItem();

if (\_selectedIndex == \_menuItemCount - 1) // Exit option

{

\_gradeBook.SaveData();

Console.CursorVisible = true;

return;

}

Console.WriteLine("\nНатисніть будь-яку клавішу, щоб повернутися до головного меню...");

Console.ReadKey(true);

break;

}

}

}

private void DrawMenu()

{

Console.WriteLine("Головне меню:");

for (int i = 0; i < \_menuItems.Length; i++)

{

if (i == \_selectedIndex)

{

Console.BackgroundColor = ConsoleColor.Yellow;

Console.ForegroundColor = ConsoleColor.Blue;

}

Console.WriteLine($"{i + 1}. {\_menuItems[i]}");

Console.ResetColor();

}

}

private void ExecuteMenuItem()

{

switch (\_selectedIndex)

{

case 0:

var groupMenu = new GroupMenu(\_gradeBook);

groupMenu.Show();

break;

case 1:

var studentMenu = new StudentMenu(\_gradeBook);

studentMenu.Show();

break;

case 2:

var subjectMenu = new SubjectMenu(\_gradeBook);

subjectMenu.Show();

break;

case 3:

var gradeMenu = new GradeMenu(\_gradeBook);

gradeMenu.Show();

break;

case 4:

var searchMenu = new SearchMenu(\_gradeBook);

searchMenu.Show();

break;

case 5:

Console.WriteLine("Дані збережено. Дякуємо за використання програми!");

break;

}

}

}

}

### SearchMenu.cs

using System;

using GradeBookSystem.Models;

using GradeBookSystem.Services;

using GradeBookSystem.Exceptions;

namespace GradeBookSystem.ConsoleMenu

{

public class SearchMenu

{

private readonly GradeBook \_gradeBook;

private readonly string[] \_menuItems = {

"Пошук студента за прізвищем та ім'ям",

"Пошук студентів певної групи",

"Пошук студентів за середнім балом",

"Пошук успішних/неуспішних студентів",

"Повернутися до головного меню"

};

public SearchMenu(GradeBook gradeBook)

{

\_gradeBook = gradeBook;

}

public void Show()

{

while (true)

{

Console.Clear();

Console.WriteLine("\nМеню пошуку:");

for (int i = 0; i < \_menuItems.Length; i++)

{

Console.WriteLine($"{i + 1}. {\_menuItems[i]}");

}

Console.Write("\nОберіть опцію: ");

if (int.TryParse(Console.ReadLine(), out int choice) && choice >= 1 && choice <= \_menuItems.Length)

{

Console.Clear();

try

{

if (ExecuteMenuItem(choice - 1))

return;

}

catch (Exception ex)

{

Console.WriteLine($"Помилка: {ex.Message}");

}

Console.WriteLine("\nНатисніть будь-яку клавішу, щоб продовжити...");

Console.ReadKey(true);

}

else

{

Console.WriteLine("Невірний вибір. Спробуйте ще раз.");

}

}

}

private bool ExecuteMenuItem(int index)

{

switch (index)

{

case 0:

SearchStudentByName();

break;

case 1:

SearchStudentsByGroup();

break;

case 2:

SearchStudentsByAverageGrade();

break;

case 3:

SearchSuccessfulStudents();

break;

case 4:

return true;

}

return false;

}

private void SearchStudentByName()

{

Console.Write("Введіть прізвище студента: ");

string lastName = Console.ReadLine();

Console.Write("Введіть ім'я студента: ");

string firstName = Console.ReadLine();

try

{

var students = \_gradeBook.SearchStudentsByName(lastName, firstName);

if (students.Count == 0)

{

throw new StudentNotFoundException($"Студентів з прізвищем {lastName} та ім'ям {firstName} не знайдено.");

}

Console.WriteLine("Знайдені студенти:");

foreach (var student in students)

{

Console.WriteLine($"{student.FirstName} {student.LastName} (ID: {student.StudentId})");

}

}

catch (StudentNotFoundException ex)

{

Console.WriteLine(ex.Message);

}

catch (Exception ex)

{

Console.WriteLine($"Сталася непередбачена помилка: {ex.Message}");

}

}

private void SearchStudentsByGroup()

{

Console.Write("Введіть назву групи: ");

string groupName = Console.ReadLine();

try

{

var students = \_gradeBook.SearchStudentsByGroup(groupName);

if (students.Count == 0)

{

throw new StudentNotFoundException($"Студентів у групі {groupName} не знайдено.");

}

Console.WriteLine($"Студенти групи {groupName}:");

foreach (var student in students)

{

Console.WriteLine($"{student.FirstName} {student.LastName} (ID: {student.StudentId})");

}

}

catch (StudentNotFoundException ex)

{

Console.WriteLine(ex.Message);

}

catch (Exception ex)

{

Console.WriteLine($"Сталася непередбачена помилка: {ex.Message}");

}

}

private void SearchStudentsByAverageGrade()

{

try

{

Console.Write("Введіть мінімальний середній бал: ");

if (!double.TryParse(Console.ReadLine(), out double minAverage))

{

throw new ArgumentException("Некоректне введення. Будь ласка, введіть число.");

}

Console.Write("Введіть максимальний середній бал: ");

if (!double.TryParse(Console.ReadLine(), out double maxAverage))

{

throw new ArgumentException("Некоректне введення. Будь ласка, введіть число.");

}

if (minAverage > maxAverage)

{

throw new ArgumentException("Мінімальний середній бал не може бути більшим за максимальний.");

}

var students = \_gradeBook.SearchStudentsByAverageGrade(minAverage, maxAverage);

if (students.Count == 0)

{

throw new StudentNotFoundException($"Студентів з середнім балом від {minAverage} до {maxAverage} не знайдено.");

}

Console.WriteLine($"Студенти з середнім балом від {minAverage} до {maxAverage}:");

foreach (var student in students)

{

double avgGrade = student.GetAverageGrade();

Console.WriteLine($"{student.FirstName} {student.LastName} (ID: {student.StudentId}), Середній бал: {avgGrade:F2}");

}

}

catch (ArgumentException ex)

{

Console.WriteLine(ex.Message);

}

catch (StudentNotFoundException ex)

{

Console.WriteLine(ex.Message);

}

catch (Exception ex)

{

Console.WriteLine($"Сталася непередбачена помилка: {ex.Message}");

}

}

private void SearchSuccessfulStudents()

{

try

{

Console.Write("Введіть мінімальний середній бал для успішних студентів: ");

if (!double.TryParse(Console.ReadLine(), out double minAverage))

{

throw new ArgumentException("Некоректне введення. Будь ласка, введіть число.");

}

var students = \_gradeBook.SearchSuccessfulStudents(minAverage);

if (students.Count == 0)

{

throw new StudentNotFoundException($"Студентів з середнім балом вище {minAverage} не знайдено.");

}

Console.WriteLine($"Успішні студенти (середній бал вище {minAverage}):");

foreach (var student in students)

{

double avgGrade = student.GetAverageGrade();

Console.WriteLine($"{student.FirstName} {student.LastName} (ID: {student.StudentId}), Середній бал: {avgGrade:F2}");

}

}

catch (ArgumentException ex)

{

Console.WriteLine(ex.Message);

}

catch (StudentNotFoundException ex)

{

Console.WriteLine(ex.Message);

}

catch (Exception ex)

{

Console.WriteLine($"Сталася непередбачена помилка: {ex.Message}");

}

}

}

}

### StudentMenu.cs

using System;

using GradeBookSystem.Models;

using GradeBookSystem.Exceptions;

using GradeBookSystem.Services;

namespace GradeBookSystem.ConsoleMenu

{

public class StudentMenu

{

private readonly GradeBook \_gradeBook;

private readonly string[] \_menuItems = {

"Додати студента",

"Видалити студента",

"Змінити дані студента",

"Переглянути список всіх студентів",

"Переглянути дані про успішність студента",

"Повернутися до головного меню"

};

public StudentMenu(GradeBook gradeBook)

{

\_gradeBook = gradeBook;

}

public void Show()

{

while (true)

{

Console.Clear();

Console.WriteLine("\nМеню управління студентами:");

for (int i = 0; i < \_menuItems.Length; i++)

{

Console.WriteLine($"{i + 1}. {\_menuItems[i]}");

}

Console.Write("\nОберіть опцію: ");

if (int.TryParse(Console.ReadLine(), out int choice) && choice >= 1 && choice <= \_menuItems.Length)

{

Console.Clear();

if (ExecuteMenuItem(choice - 1))

return;

Console.WriteLine("\nНатисніть будь-яку клавішу, щоб продовжити...");

Console.ReadKey(true);

}

else

{

Console.WriteLine("Невірний вибір. Спробуйте ще раз.");

}

}

}

private bool ExecuteMenuItem(int index)

{

switch (index)

{

case 0:

AddStudent();

break;

case 1:

RemoveStudent();

break;

case 2:

EditStudent();

break;

case 3:

ViewAllStudents();

break;

case 4:

ViewStudentGrades();

break;

case 5:

return true;

}

return false;

}

private void AddStudent()

{

try

{

Console.Write("Введіть назву групи: ");

string groupName = Console.ReadLine();

Console.Write("Введіть ім'я студента: ");

string firstName = Console.ReadLine();

Console.Write("Введіть прізвище студента: ");

string lastName = Console.ReadLine();

Console.Write("Введіть ID студента(у форматі STU12345): ");

string studentId = Console.ReadLine();

var student = new Student

{

FirstName = firstName,

LastName = lastName,

StudentId = studentId

};

\_gradeBook.AddStudent(groupName, student);

Console.WriteLine("Студента успішно додано.");

}

catch (InvalidStudentIdFormatException ex)

{

Console.WriteLine($"Помилка: {ex.Message}");

}

catch (Exception ex)

{

Console.WriteLine($"Помилка: {ex.Message}");

}

}

private void RemoveStudent()

{

Console.Write("Введіть назву групи: ");

string groupName = Console.ReadLine();

Console.Write("Введіть ID студента: ");

string studentId = Console.ReadLine();

try

{

\_gradeBook.RemoveStudent(groupName, studentId);

Console.WriteLine("Студента успішно видалено.");

}

catch (Exception ex)

{

Console.WriteLine($"Помилка: {ex.Message}");

}

}

private void EditStudent()

{

try

{

Console.Write("Введіть назву групи: ");

string groupName = Console.ReadLine();

Console.Write("Введіть поточний ID студента: ");

string currentId = Console.ReadLine();

Console.Write("Введіть нове ім'я студента: ");

string newFirstName = Console.ReadLine();

Console.Write("Введіть нове прізвище студента: ");

string newLastName = Console.ReadLine();

Console.Write("Введіть новий ID студента: ");

string newId = Console.ReadLine();

\_gradeBook.EditStudent(groupName, currentId, newFirstName, newLastName, newId);

Console.WriteLine("Дані студента успішно змінено.");

}

catch (Exception ex)

{

Console.WriteLine($"Помилка: {ex.Message}");

}

}

private void ViewAllStudents()

{

var students = \_gradeBook.GetAllStudents();

if (students.Count == 0)

{

Console.WriteLine("Список студентів порожній.");

return;

}

foreach (var student in students)

{

Console.WriteLine($"{student.FirstName} {student.LastName} (ID: {student.StudentId})");

}

}

private void ViewStudentGrades()

{

Console.Write("Введіть ID студента: ");

string studentId = Console.ReadLine();

try

{

var student = \_gradeBook.GetStudentById(studentId);

Console.WriteLine($"Оцінки студента {student.FirstName} {student.LastName}:");

if (student.Grades.Count == 0)

{

Console.WriteLine("У студента ще немає оцінок.");

return;

}

foreach (var subject in student.Grades.Keys)

{

Console.WriteLine($"{subject}: {string.Join(", ", student.Grades[subject])}");

}

}

catch (Exception ex)

{

Console.WriteLine($"Помилка: {ex.Message}");

}

}

}

}

### SubjectMenu.cs

using System;

using GradeBookSystem.Models;

using GradeBookSystem.Services;

namespace GradeBookSystem.ConsoleMenu

{

public class SubjectMenu

{

private readonly GradeBook \_gradeBook;

private readonly string[] \_menuItems = {

"Додати предмет",

"Видалити предмет",

"Переглянути список предметів",

"Повернутися до головного меню"

};

public SubjectMenu(GradeBook gradeBook)

{

\_gradeBook = gradeBook;

}

public void Show()

{

while (true)

{

Console.Clear();

Console.WriteLine("\nМеню управління предметами:");

for (int i = 0; i < \_menuItems.Length; i++)

{

Console.WriteLine($"{i + 1}. {\_menuItems[i]}");

}

Console.Write("\nОберіть опцію: ");

if (int.TryParse(Console.ReadLine(), out int choice) && choice >= 1 && choice <= \_menuItems.Length)

{

Console.Clear();

if (ExecuteMenuItem(choice - 1))

return;

Console.WriteLine("\nНатисніть будь-яку клавішу, щоб продовжити...");

Console.ReadKey(true);

}

else

{

Console.WriteLine("Невірний вибір. Спробуйте ще раз.");

}

}

}

private bool ExecuteMenuItem(int index)

{

switch (index)

{

case 0:

AddSubject();

break;

case 1:

RemoveSubject();

break;

case 2:

ViewAllSubjects();

break;

case 3:

return true;

}

return false;

}

private void AddSubject()

{

try

{

Console.Write("Введіть назву предмета: ");

string subjectName = Console.ReadLine();

\_gradeBook.AddSubject(subjectName);

Console.WriteLine("Предмет успішно додано.");

}

catch (Exception ex)

{

Console.WriteLine($"Помилка: {ex.Message}");

}

}

private void RemoveSubject()

{

try

{

Console.Write("Введіть назву предмета: ");

string subjectName = Console.ReadLine();

\_gradeBook.RemoveSubject(subjectName);

Console.WriteLine("Предмет успішно видалено.");

}

catch (Exception ex)

{

Console.WriteLine($"Помилка: {ex.Message}");

}

}

private void ViewAllSubjects()

{

var subjects = \_gradeBook.GetAllSubjects();

if (subjects.Count == 0)

{

Console.WriteLine("Список предметів порожній.");

return;

}

foreach (var subject in subjects)

{

Console.WriteLine(subject);

}

}

}

}

## Espl\_Core (Class library)

### Models/IPerson.cs

namespace GradeBookSystem.Models

{

public interface IPerson

{

string FirstName { get; set; }

string LastName { get; set; }

}

}

### Models/Student.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text.RegularExpressions;

using GradeBookSystem.Exceptions;

namespace GradeBookSystem.Models

{

public class Student : IPerson, IComparable<Student>

{

private string firstName;

private string lastName;

private string studentId;

public string FirstName

{

get => firstName;

set

{

if (!IsValidName(value))

throw new ArgumentException("Ім'я не може містити цифри.");

firstName = value;

}

}

public string LastName

{

get => lastName;

set

{

if (!IsValidName(value))

throw new ArgumentException("Прізвище не може містити цифри.");

lastName = value;

}

}

public string StudentId

{

get => studentId;

set

{

if (!IsValidStudentId(value))

throw new InvalidStudentIdFormatException("ID студента повинен бути у форматі STU12345.");

studentId = value;

}

}

public Dictionary<string, List<int>> Grades { get; set; } = new Dictionary<string, List<int>>();

private bool IsValidName(string name)

{

return !string.IsNullOrWhiteSpace(name) && !name.Any(char.IsDigit);

}

private bool IsValidStudentId(string id)

{

return Regex.IsMatch(id, @"^STU\d{5}$");

}

public double GetAverageGrade()

{

if (Grades.Count == 0) return 0;

return Grades.SelectMany(g => g.Value).Average();

}

public int CompareTo(Student other)

{

return this.GetAverageGrade().CompareTo(other.GetAverageGrade());

}

}

}

### Models/Group.cs

using System.Collections.Generic;

namespace GradeBookSystem.Models

{

public class Group

{

public string GroupName { get; set; }

public List<Student> Students { get; set; } = new List<Student>();

}

}

### Services/GradeBook.cs

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using GradeBookSystem.Models;

using GradeBookSystem.Exceptions;

using Newtonsoft.Json;

namespace GradeBookSystem.Services

{

public class GradeBook

{

private List<Group> groups = new List<Group>();

private List<string> subjects = new List<string>();

private const string JsonDataFile = "gradebook.json";

private const string TextDataFile = "gradebook.txt";

public GradeBook()

{

LoadData();

}

public void AddGroup(string groupName)

{

if (groups.Any(g => g.GroupName == groupName))

throw new DuplicateGroupException($"Група {groupName} вже існує.");

groups.Add(new Group { GroupName = groupName });

}

public void RemoveGroup(string groupName)

{

var group = FindGroup(groupName);

groups.Remove(group);

}

public void EditGroup(string currentName, string newName)

{

var group = FindGroup(currentName);

group.GroupName = newName;

}

public List<Group> GetAllGroups()

{

return groups;

}

public Group GetGroupByName(string groupName)

{

return FindGroup(groupName);

}

public void AddStudent(string groupName, Student student)

{

var group = FindGroup(groupName);

if (group.Students.Any(s => s.StudentId == student.StudentId))

throw new DuplicateStudentException($"Студент з ID {student.StudentId} вже існує в групі {groupName}.");

try

{

// Перевірка імені, прізвища та ID відбувається при присвоєнні значень властивостям

student.FirstName = student.FirstName;

student.LastName = student.LastName;

student.StudentId = student.StudentId;

}

catch (ArgumentException ex)

{

throw new ArgumentException($"Помилка при додаванні студента: {ex.Message}");

}

catch (InvalidStudentIdFormatException ex)

{

throw new ArgumentException($"Помилка при додаванні студента: {ex.Message}");

}

group.Students.Add(student);

}

public void RemoveStudent(string groupName, string studentId)

{

var group = FindGroup(groupName);

var student = FindStudent(group, studentId);

group.Students.Remove(student);

}

public void EditStudent(string groupName, string currentId, string newFirstName, string newLastName, string newId)

{

var group = FindGroup(groupName);

var student = FindStudent(group, currentId);

student.FirstName = newFirstName;

student.LastName = newLastName;

student.StudentId = newId;

}

public List<Student> GetAllStudents()

{

return groups.SelectMany(g => g.Students).ToList();

}

public Student GetStudentById(string studentId)

{

return groups.SelectMany(g => g.Students).FirstOrDefault(s => s.StudentId == studentId)

?? throw new StudentNotFoundException($"Студента з ID {studentId} не знайдено.");

}

public void AddSubject(string subjectName)

{

if (subjects.Contains(subjectName))

throw new DuplicateSubjectException($"Предмет {subjectName} вже існує.");

subjects.Add(subjectName);

}

public void RemoveSubject(string subjectName)

{

if (!subjects.Remove(subjectName))

throw new SubjectNotFoundException($"Предмет {subjectName} не знайдено.");

foreach (var student in GetAllStudents())

{

student.Grades.Remove(subjectName);

}

}

public List<string> GetAllSubjects()

{

return subjects;

}

public void AddGrade(string studentId, string subject, int grade)

{

var student = GetStudentById(studentId);

if (!subjects.Contains(subject))

throw new SubjectNotFoundException($"Предмет {subject} не знайдено.");

if (!student.Grades.ContainsKey(subject))

student.Grades[subject] = new List<int>();

student.Grades[subject].Add(grade);

}

public void EditGrade(string studentId, string subject, int index, int newGrade)

{

var student = GetStudentById(studentId);

if (!student.Grades.ContainsKey(subject))

throw new SubjectNotFoundException($"Предмет {subject} не знайдено для цього студента.");

if (index < 0 || index >= student.Grades[subject].Count)

throw new IndexOutOfRangeException("Невірний індекс оцінки.");

student.Grades[subject][index] = newGrade;

}

public double CalculateStudentAverage(string studentId)

{

var student = GetStudentById(studentId);

return student.GetAverageGrade();

}

public double CalculateGroupAverage(string groupName)

{

var group = FindGroup(groupName);

return group.Students.Average(s => s.GetAverageGrade());

}

public List<Student> SearchStudentsByName(string lastName, string firstName)

{

return GetAllStudents().Where(s => s.LastName == lastName && s.FirstName == firstName).ToList();

}

public List<Student> SearchStudentsByGroup(string groupName)

{

var group = FindGroup(groupName);

return group.Students;

}

public List<Student> SearchStudentsByAverageGrade(double minAverage, double maxAverage)

{

return GetAllStudents().Where(s => s.GetAverageGrade() >= minAverage && s.GetAverageGrade() <= maxAverage).ToList();

}

public List<Student> SearchSuccessfulStudents(double minAverage)

{

return GetAllStudents().Where(s => s.GetAverageGrade() >= minAverage).ToList();

}

public void SaveData()

{

SaveJsonData();

SaveTextData();

}

private void SaveJsonData()

{

var data = new { Groups = groups, Subjects = subjects };

File.WriteAllText(JsonDataFile, JsonConvert.SerializeObject(data, Formatting.Indented));

}

private void SaveTextData()

{

using (StreamWriter writer = new StreamWriter(TextDataFile))

{

writer.WriteLine("Subjects:");

foreach (var subject in subjects)

{

writer.WriteLine(subject);

}

writer.WriteLine("\nGroups and Students:");

foreach (var group in groups)

{

writer.WriteLine($"Group: {group.GroupName}");

foreach (var student in group.Students)

{

writer.WriteLine($" Student: {student.FirstName} {student.LastName}, ID: {student.StudentId}");

foreach (var grade in student.Grades)

{

writer.WriteLine($" Subject: {grade.Key}, Grades: {string.Join(", ", grade.Value)}");

}

}

}

}

}

private void LoadData()

{

if (File.Exists(JsonDataFile))

{

LoadJsonData();

}

else if (File.Exists(TextDataFile))

{

LoadTextData();

}

}

private void LoadJsonData()

{

var data = JsonConvert.DeserializeObject<dynamic>(File.ReadAllText(JsonDataFile));

groups = JsonConvert.DeserializeObject<List<Group>>(data.Groups.ToString());

subjects = JsonConvert.DeserializeObject<List<string>>(data.Subjects.ToString());

}

private void LoadTextData()

{

groups.Clear();

subjects.Clear();

using (StreamReader reader = new StreamReader(TextDataFile))

{

string line;

Group currentGroup = null;

Student currentStudent = null;

while ((line = reader.ReadLine()) != null)

{

if (line.StartsWith("Subjects:"))

{

while ((line = reader.ReadLine()) != null && !string.IsNullOrWhiteSpace(line))

{

subjects.Add(line.Trim());

}

}

else if (line.StartsWith("Group:"))

{

string groupName = line.Substring(7).Trim();

currentGroup = new Group { GroupName = groupName };

groups.Add(currentGroup);

}

else if (line.Trim().StartsWith("Student:"))

{

string[] studentInfo = line.Trim().Substring(9).Split(',');

string[] nameParts = studentInfo[0].Trim().Split(' ');

string studentId = studentInfo[1].Trim().Substring(4);

currentStudent = new Student

{

FirstName = nameParts[0],

LastName = nameParts[1],

StudentId = studentId

};

currentGroup.Students.Add(currentStudent);

}

else if (line.Trim().StartsWith("Subject:"))

{

string[] subjectInfo = line.Trim().Substring(9).Split(',');

string subject = subjectInfo[0].Trim();

string[] grades = subjectInfo[1].Trim().Substring(8).Split(',');

currentStudent.Grades[subject] = grades.Select(int.Parse).ToList();

}

}

}

}

private Group FindGroup(string groupName)

{

return groups.FirstOrDefault(g => g.GroupName == groupName)

?? throw new GroupNotFoundException($"Групу {groupName} не знайдено.");

}

private Student FindStudent(Group group, string studentId)

{

return group.Students.FirstOrDefault(s => s.StudentId == studentId)

?? throw new StudentNotFoundException($"Студента з ID {studentId} не знайдено в групі {group.GroupName}.");

}

}

}

### Exceptions/DuplicateGroupException.cs

using System;

namespace GradeBookSystem.Exceptions

{

public class DuplicateGroupException : Exception

{

public DuplicateGroupException(string message) : base(message) { }

}

}

### Exceptions/DuplicateStudentException.cs

using System;

namespace GradeBookSystem.Exceptions

{

public class DuplicateStudentException : Exception

{

public DuplicateStudentException(string message) : base(message) { }

}

}

### Exceptions/GroupNotFoundException.cs

using System;

namespace GradeBookSystem.Exceptions

{

public class GroupNotFoundException : Exception

{

public GroupNotFoundException(string message) : base(message) { }

}

}

### Exceptions/StudentNotFoundException.cs

using System;

namespace GradeBookSystem.Exceptions

{

public class StudentNotFoundException : Exception

{

public StudentNotFoundException(string message) : base(message) { }

}

}

### Exceptions/InvalidStudentIdFormatException.cs

using System;

namespace GradeBookSystem.Exceptions

{

public class InvalidStudentIdFormatException : Exception

{

public InvalidStudentIdFormatException(string message) : base(message) { }

}

}

### Exceptions/DuplicateSubjectException.cs

using System;

namespace GradeBookSystem.Exceptions

{

public class DuplicateSubjectException : Exception

{

public DuplicateSubjectException(string message) : base(message) { }

}

}

### Exceptions/SubjectNotFoundException.cs

using System;

namespace GradeBookSystem.Exceptions

{

public class SubjectNotFoundException : Exception

{

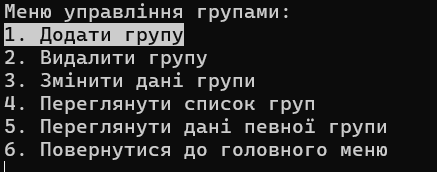
public SubjectNotFoundException(string message) : base(message) { }

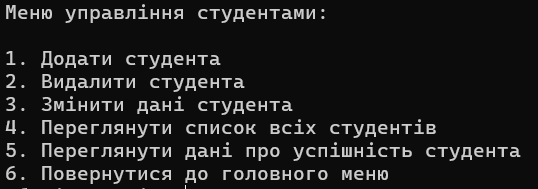
}

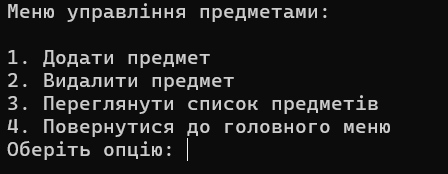
}

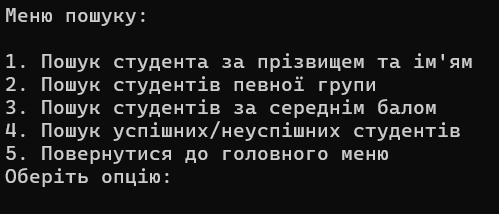
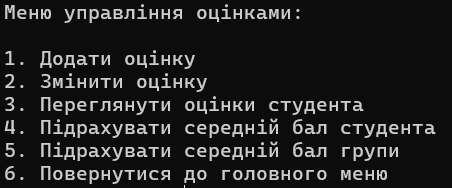
# Ілюстрації











# Висновок

У ході виконання поставленого завдання було розроблено консольний додаток "Електронний журнал успішності студентів" (ESPL), який реалізує функціональність управління даними про студентів, групи, предмети та їхню успішність. Основною метою було створення інтуїтивного та ефективного інструменту для обліку та аналізу навчальних досягнень.

Бізнес-логіка програми організована у вигляді окремого класу GradeBook, який забезпечує функції додавання, редагування та видалення даних про студентів, групи, предмети, а також операції над успішністю, включаючи розрахунок середніх балів та пошук. Усі ключові операції реалізовано з використанням інтерфейсів, абстрактних класів та узагальнених колекцій, що гарантує модульність, гнучкість і масштабованість коду. Для збереження даних використано JSON-серіалізацію, яка дозволяє легко зберігати й відновлювати стан програми.

Інтерфейс програми побудований у вигляді текстового меню, що дозволяє виконувати основні дії з управління групами, студентами, предметами та оцінками. Для забезпечення зручності користувача були реалізовані такі можливості, як перевірка коректності введених даних, обробка помилок і локалізація інтерфейсу українською мовою. Всі операції супроводжуються зрозумілими повідомленнями, а результати відображаються у форматованому вигляді.