**Виконав: Мельник Б. В.; КН-922в**

**Лабораторна робота №4**

**Тема:** Використання принципів та відносин об’єктно-орієнтованого програмування у мові C# .

**Мета роботи:** набути навички в написанні програм із застосуванням принципів та відносин об’єктно-орієнтованого програмування (ООП).

**Індивідуальні завдання**

**Варіант №12**

**Завдання 1**

Розробити клас згідно до свого варіанта. Включити до класу методи set (…), get (…), show (…) та ін., використовуючи принцип інкапсуляції. Окрім гетерів, сетерів конструкторів та інших стандартних методів розробити не менше двох додаткових методів, які реалізують власний функціонал об’єкту відповідної предметної галузі. Написати програму, яка створює список, масив чи колекцію, об’єктів на основі відповідного класу. Під час створення використати випадковий підхід для генерації числової та текстової інформації об’єктів списку. Передбачити можливість додання, редагування та видалення об’єктів зі списку за певними критеріями. В програмі реалізувати функціонал за своїм варіантом.

**12.** Word: Слово, Номери сторінок, де слово зустрічається (від 1 до 10), Число сторінок. Створити масив об'єктів. Вивести:

* слова, що зустрічаються більш ніж на сторінках N;
* слова за абеткою;
* для заданого слова номери сторінок, де воно зустрічається.

**Текст програми:**

using System; // Підключення базової бібліотеки для введення-виведення та інших функцій.

using System.Collections.Generic; // Підключення бібліотеки для роботи зі списками та іншими колекціями.

using System.Linq; // Підключення бібліотеки для LINQ-запитів.

public class Word // Оголошення класу "Word", що описує слово та його властивості.

{

public string Text { get; set; } // Властивість для зберігання тексту слова.

public List<int> PageNumbers { get; set; } // Властивість для зберігання номерів сторінок.

public int TotalPages { get; set; } // Властивість для зберігання загальної кількості сторінок.

public Word(string text, List<int> pageNumbers, int totalPages) // Конструктор для ініціалізації властивостей.

{

Text = text; // Ініціалізація тексту слова.

PageNumbers = pageNumbers; // Ініціалізація номерів сторінок.

TotalPages = totalPages; // Ініціалізація загальної кількості сторінок.

}

public void Show() // Метод для виведення інформації про слово.

{

Console.WriteLine($"Слово: {Text}, Номери сторінок: {string.Join(", ", PageNumbers)}, Число сторінок: {TotalPages}"); // Виведення у форматі: слово, номери сторінок, кількість сторінок.

}

public bool AppearsOnMoreThanNPages(int n) // Метод, що перевіряє, чи слово зустрічається більше ніж на n сторінках.

{

return PageNumbers.Count > n; // Повертає true, якщо кількість сторінок більше n.

}

public static void ShowWordsAlphabetically(List<Word> words) // Статичний метод для виведення слів за абеткою.

{

var sortedWords = words.OrderBy(w => w.Text).ToList(); // Сортування слів за текстом.

foreach (var word in sortedWords) // Перебір відсортованих слів.

{

word.Show(); // Виклик методу Show для кожного слова.

}

}

public static void ShowPagesForWord(List<Word> words, string searchWord) // Статичний метод для пошуку слова та виведення його сторінок.

{

var word = words.FirstOrDefault(w => w.Text.Equals(searchWord, StringComparison.OrdinalIgnoreCase)); // Пошук слова без урахування регістру.

if (word != null) // Якщо слово знайдено.

{

Console.WriteLine($"Слово '{searchWord}' зустрічається на сторінках: {string.Join(", ", word.PageNumbers)}"); // Виведення номерів сторінок.

}

else // Якщо слово не знайдено.

{

Console.WriteLine($"Слово '{searchWord}' не знайдено."); // Повідомлення про відсутність слова.

}

}

}

public class Task\_1 // Головний клас для виконання задачі.

{

public static void Main() // Головний метод, який виконується при запуску програми.

{

Console.InputEncoding = System.Text.Encoding.UTF8; // Встановлення кодування для введення UTF-8.

Console.OutputEncoding = System.Text.Encoding.UTF8; // Встановлення кодування для виведення UTF-8.

Random random = new Random(); // Створення об'єкта для генерації випадкових чисел.

List<Word> words = new List<Word>(); // Створення списку слів.

// Українські слова

string[] ukrainianWords = { "Книга", "Слово", "Мова", "Україна", "Програмування", "Комп'ютер", "Інтернет", "Освіта", "Наука", "Технології" }; // Масив українських слів.

// Генерація випадкових даних

for (int i = 0; i < ukrainianWords.Length; i++) // Цикл по всіх українських словах.

{

string text = ukrainianWords[i]; // Отримання тексту слова.

List<int> pageNumbers = Enumerable.Range(1, 10).OrderBy(x => random.Next()).Take(random.Next(1, 11)).ToList(); // Генерація випадкових номерів сторінок.

int totalPages = random.Next(1, 11); // Генерація випадкової кількості сторінок.

words.Add(new Word(text, pageNumbers, totalPages)); // Додавання слова до списку.

}

// Виведення всіх слів

Console.WriteLine("Всі слова:"); // Заголовок.

foreach (var word in words) // Цикл по всіх словах.

{

word.Show(); // Виведення кожного слова.

}

// Виведення слів, що зустрічаються більш ніж на N сторінках

int N = 3; // Задане число N.

Console.WriteLine($"\nСлова, що зустрічаються більш ніж на {N} сторінках:"); // Заголовок.

foreach (var word in words.Where(w => w.AppearsOnMoreThanNPages(N))) // Перебір слів, що зустрічаються більше ніж на N сторінках.

{

word.Show(); // Виведення таких слів.

}

// Виведення слів за абеткою

Console.WriteLine("\nСлова за абеткою:"); // Заголовок.

Word.ShowWordsAlphabetically(words); // Виклик статичного методу для виведення слів за абеткою.

// Виведення сторінок для заданого слова

string searchWord = "Програмування"; // Задане слово для пошуку.

Console.WriteLine($"\nНомери сторінок для слова '{searchWord}':"); // Заголовок.

Word.ShowPagesForWord(words, searchWord); // Виклик статичного методу для виведення сторінок для заданого слова.

// Додавання нового слова

string newText = "Новинка"; // Текст нового слова.

List<int> newPageNumbers = new List<int> { 1, 2, 3 }; // Номери сторінок для нового слова.

int newTotalPages = 3; // Кількість сторінок для нового слова.

words.Add(new Word(newText, newPageNumbers, newTotalPages)); // Додавання нового слова до списку.

Console.WriteLine("\nДодано нове слово:"); // Повідомлення про додавання нового слова.

words.Last().Show(); // Виведення нового слова.

// Редагування слова

string editWord = "Книга"; // Слово для редагування.

var wordToEdit = words.FirstOrDefault(w => w.Text == editWord); // Пошук слова для редагування.

if (wordToEdit != null) // Якщо слово знайдено.

{

wordToEdit.PageNumbers = new List<int> { 4, 5, 6 }; // Зміна номерів сторінок.

wordToEdit.TotalPages = 3; // Зміна кількості сторінок.

Console.WriteLine($"\nВідредаговано слово '{editWord}':"); // Повідомлення про редагування.

wordToEdit.Show(); // Виведення відредагованого слова.

}

// Видалення слова

string deleteWord = "Слово"; // Слово для видалення.

var wordToDelete = words.FirstOrDefault(w => w.Text == deleteWord); // Пошук слова для видалення.

if (wordToDelete != null) // Якщо слово знайдено.

{

words.Remove(wordToDelete); // Видалення слова зі списку.

Console.WriteLine($"\nВидалено слово '{deleteWord}'."); // Повідомлення про видалення.

}

// Виведення всіх слів після редагування та видалення

Console.WriteLine("\nВсі слова після редагування та видалення:"); // Заголовок.

foreach (var word in words) // Перебір всіх слів після змін.

{

word.Show(); // Виведення кожного слова.

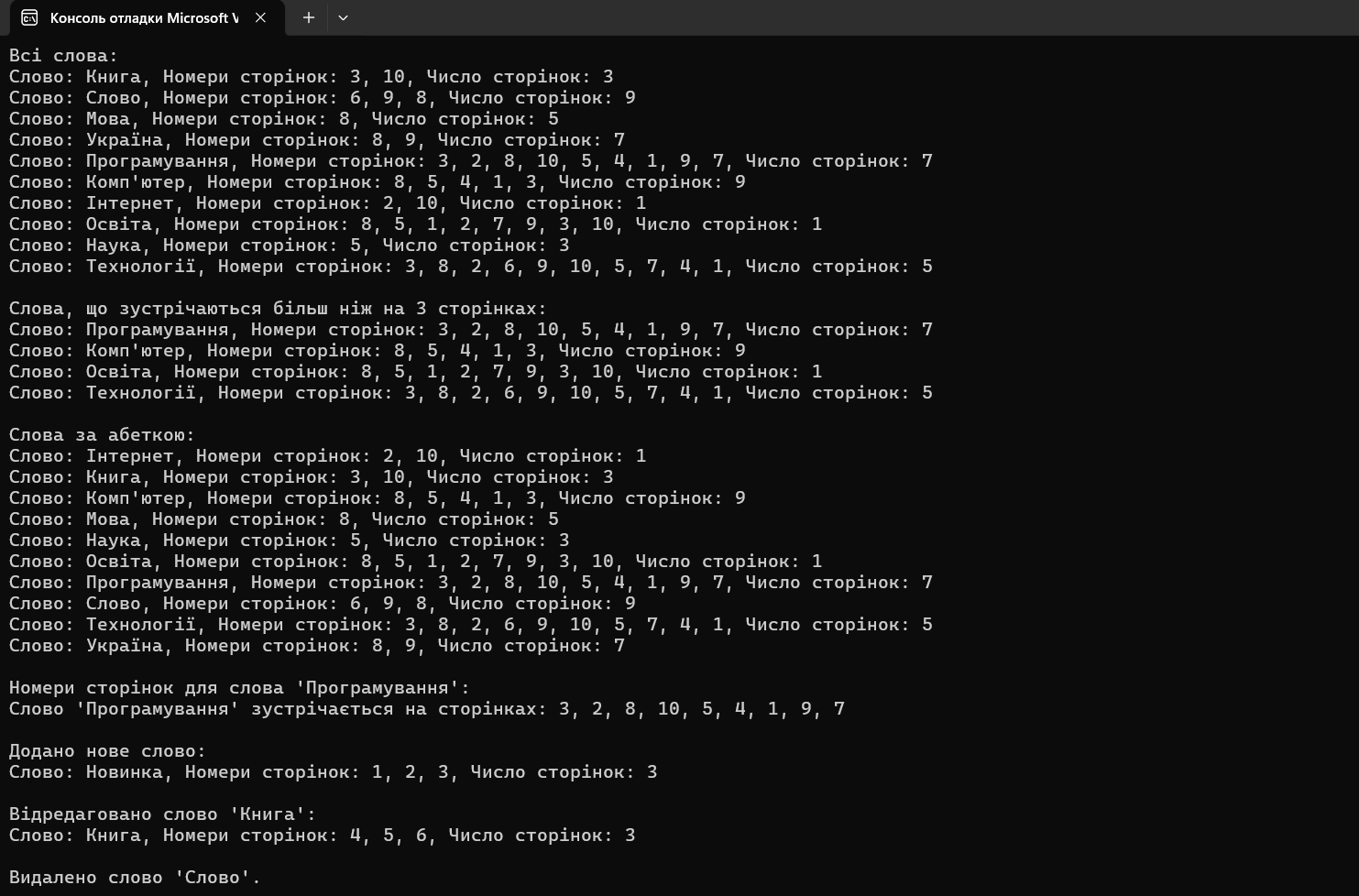
}

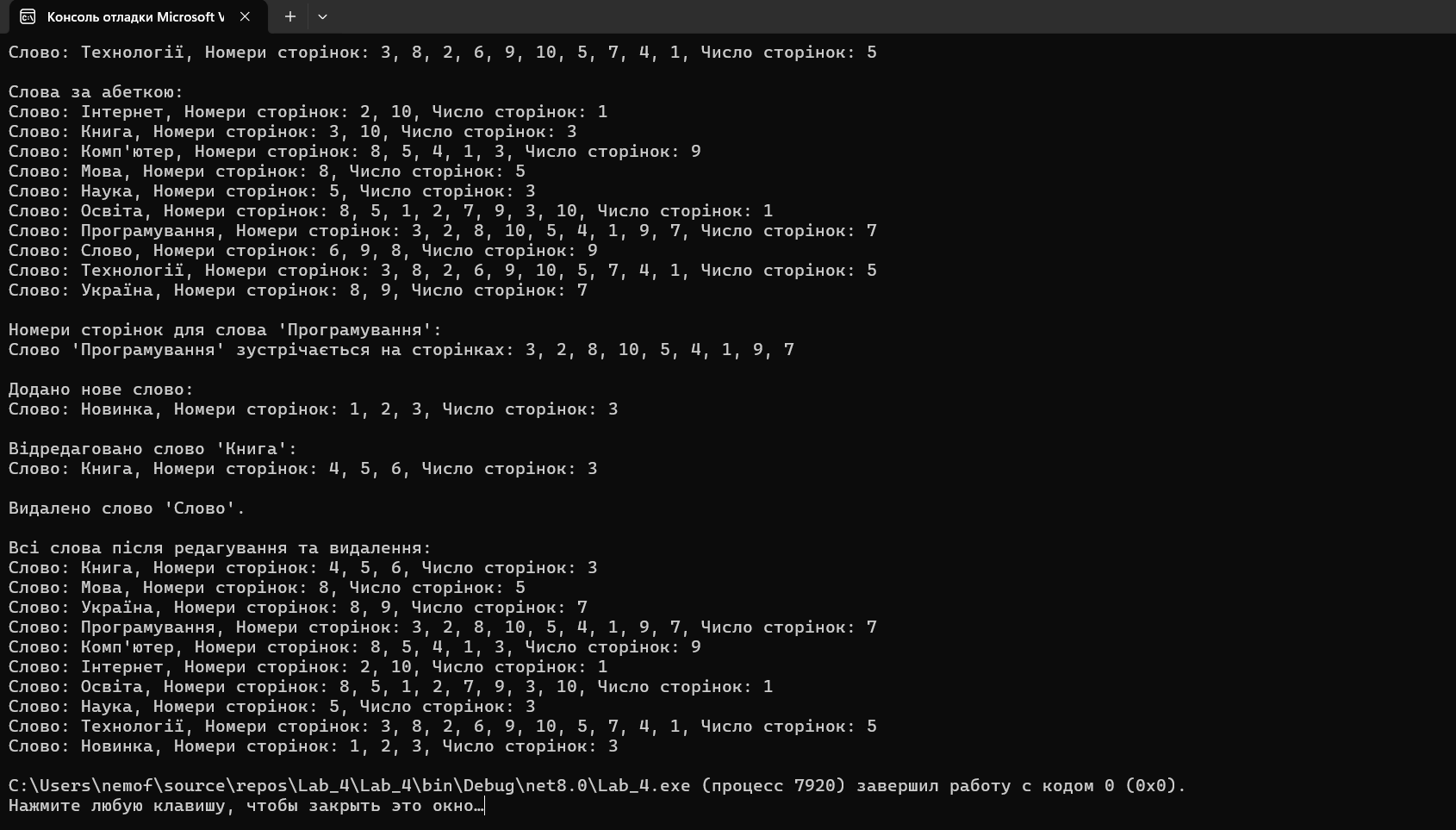
Console.ReadKey(); // Очікування натискання клавіші для завершення програми.

}

}

**Результат роботи програми:**





**Алгоритм побудови коду**

1. **Створення класу Word:**
   * Включено властивості Text (текст слова), PageNumbers (номери сторінок), TotalPages (загальна кількість сторінок).
   * Реалізовано методи get, set, show для виведення інформації.
   * Додаткові методи:
     + AppearsOnMoreThanNPages(int n) – перевіряє, чи слово зустрічається більше, ніж на заданій кількості сторінок.
     + ShowWordsAlphabetically(List<Word>) – виводить всі слова в алфавітному порядку.
2. **Генерація випадкових даних:**
   * Для кожного слова випадково генерується кількість сторінок і їх номери.
3. **Операції зі списком об'єктів:**
   * Додавання нового об'єкта.
   * Редагування існуючого об'єкта (оновлення сторінок і кількості сторінок).
   * Видалення об'єкта зі списку.
4. **Додаткові функції:**
   * Виведення всіх слів, які зустрічаються більше ніж на N сторінках.
   * Пошук слова і виведення номерів сторінок для нього.

**Функціональність**

Програма дозволяє:

* Створювати список об'єктів на основі класу Word з випадковими даними.
* Виводити інформацію про кожен об'єкт.
* Сортувати об'єкти за абеткою.
* Здійснювати пошук і виводити сторінки, на яких зустрічається слово.
* Додавати нові об'єкти до списку.
* Редагувати та видаляти об'єкти зі списку за заданими критеріями.

**Висновок**

Програма успішно реалізує роботу з класом за допомогою принципу інкапсуляції, додаючи функції роботи зі списком об'єктів: додавання, редагування, видалення. Реалізовані методи демонструють, як клас можна використовувати для роботи з текстовими даними та їх обробкою.

**Завдання 2**

**1.** Для класу з минулого завдання, ретельно занурившись у предметну область власного варіанта, створити абстрактний клас чи інтерфейс, який має стати предком, зв’язати ці класи відносинами спадкування. Наприклад для класу Book (Книга) можна створити клас Reader (Засіб читання).

**2.** Окрім вашого класу додати до абстрактного класу чи інтерфейсу, створеному у попередньому пункті, ще два класи-спадкоємця першого рівня. Наприклад до класу Book (Книга) можна додати класи Newspaper (Газета) Magazine (Журнал) з відповідними полями та методами.

**3.** Для одного з доданих у попередньому завданні класів створити не менше двох спадкоємців другого рівня спадкування. Наприклад до класу Magazine (Журнал) можна додати класи PMagazine (Паперовий журнал), EMagazine (Електронний журнал), IMagazine (Інтернет журнал). У кожного нащадка має бути не менш ніж на два поля та метода більше ніж у предка.

**4.** Створити масив чи колекцію об’єктів усіх рівнів спадкування. Виконати відповідні функції над поліморфним списком із завдання власного варіанта.**Текст програми:**

using System; // Підключення базової бібліотеки для роботи з консоллю та іншими системними функціями

using System.Collections.Generic; // Підключення бібліотеки для роботи з колекціями (списками)

using System.Linq; // Підключення бібліотеки для зручних запитів до колекцій

// Завдання 1: Створення абстрактного класу TextItem

public abstract class TextItem

{

public string Text { get; set; } // Властивість для збереження тексту

public List<int> PageNumbers { get; set; } // Властивість для збереження списку номерів сторінок

public int TotalPages { get; set; } // Властивість для збереження загальної кількості сторінок

// Конструктор класу TextItem

public TextItem(string text, List<int> pageNumbers, int totalPages)

{

Text = text; // Ініціалізація тексту

PageNumbers = pageNumbers; // Ініціалізація номерів сторінок

TotalPages = totalPages; // Ініціалізація загальної кількості сторінок

}

// Абстрактний метод Show, який мають реалізувати спадкоємці

public abstract void Show();

}

// Завдання 2: Додавання класів-спадкоємців першого рівня

public class WordItem : TextItem

{

// Конструктор класу WordItem

public WordItem(string text, List<int> pageNumbers, int totalPages)

: base(text, pageNumbers, totalPages) // Виклик конструктора базового класу

{

}

// Перевизначений метод Show для відображення інформації про слово

public override void Show()

{

Console.WriteLine($"Слово: {Text}, Номери сторінок: {string.Join(", ", PageNumbers)}, Число сторінок: {TotalPages}");

}

// Метод для перевірки, чи зустрічається слово більш ніж на n сторінках

public bool AppearsOnMoreThanNPages(int n)

{

return PageNumbers.Count > n;

}

}

public class SentenceItem : TextItem

{

// Конструктор класу SentenceItem

public SentenceItem(string text, List<int> pageNumbers, int totalPages)

: base(text, pageNumbers, totalPages) // Виклик конструктора базового класу

{

}

// Перевизначений метод Show для відображення інформації про речення

public override void Show()

{

Console.WriteLine($"Речення: {Text}, Номери сторінок: {string.Join(", ", PageNumbers)}, Число сторінок: {TotalPages}");

}

}

public class ParagraphItem : TextItem

{

// Конструктор класу ParagraphItem

public ParagraphItem(string text, List<int> pageNumbers, int totalPages)

: base(text, pageNumbers, totalPages) // Виклик конструктора базового класу

{

}

// Перевизначений метод Show для відображення інформації про параграф

public override void Show()

{

Console.WriteLine($"Параграф: {Text}, Номери сторінок: {string.Join(", ", PageNumbers)}, Число сторінок: {TotalPages}");

}

}

// Завдання 3: Додавання класів-спадкоємців другого рівня

public class TitleSentenceItem : SentenceItem

{

public string Title { get; set; } // Властивість для збереження заголовку

// Конструктор класу TitleSentenceItem

public TitleSentenceItem(string text, List<int> pageNumbers, int totalPages, string title)

: base(text, pageNumbers, totalPages) // Виклик конструктора базового класу SentenceItem

{

Title = title; // Ініціалізація заголовку

}

// Перевизначений метод Show для відображення заголовку та речення

public override void Show()

{

Console.WriteLine($"Заголовок: {Title}, Речення: {Text}, Номери сторінок: {string.Join(", ", PageNumbers)}, Число сторінок: {TotalPages}");

}

}

public class BodySentenceItem : SentenceItem

{

public string Body { get; set; } // Властивість для збереження тіла тексту

// Конструктор класу BodySentenceItem

public BodySentenceItem(string text, List<int> pageNumbers, int totalPages, string body)

: base(text, pageNumbers, totalPages) // Виклик конструктора базового класу SentenceItem

{

Body = body; // Ініціалізація тіла тексту

}

// Перевизначений метод Show для відображення тіла та речення

public override void Show()

{

Console.WriteLine($"Тіло: {Body}, Речення: {Text}, Номери сторінок: {string.Join(", ", PageNumbers)}, Число сторінок: {TotalPages}");

}

}

// Завдання 4: Створення масиву об'єктів та виконання функцій над поліморфним списком

public class Task\_2

{

public static void Main()

{

Console.InputEncoding = System.Text.Encoding.UTF8; // Встановлення кодування для вводу

Console.OutputEncoding = System.Text.Encoding.UTF8; // Встановлення кодування для виводу

Random random = new Random(); // Створення об'єкта для генерації випадкових чисел

List<TextItem> textItems = new List<TextItem>(); // Створення списку текстових елементів

// Українські слова

string[] ukrainianWords = { "Книга", "Слово", "Мова", "Україна", "Програмування", "Комп'ютер", "Інтернет", "Освіта", "Наука", "Технології" };

// Генерація випадкових даних для класів першого рівня

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

{

string text = ukrainianWords[i]; // Вибір слова

List<int> pageNumbers = Enumerable.Range(1, 10).OrderBy(x => random.Next()).Take(random.Next(1, 11)).ToList(); // Генерація випадкових номерів сторінок

int totalPages = random.Next(1, 11); // Випадкова кількість сторінок

textItems.Add(new WordItem(text, pageNumbers, totalPages)); // Додавання слова в список

}

// Генерація випадкових даних для класів другого рівня

textItems.Add(new TitleSentenceItem("Заголовок 1", new List<int> { 1, 2, 3 }, 3, "Заголовок")); // Додавання заголовка

textItems.Add(new BodySentenceItem("Тіло 1", new List<int> { 4, 5, 6 }, 3, "Тіло")); // Додавання тіла тексту

// Виведення всіх текстових елементів

Console.WriteLine("Всі текстові елементи:");

foreach (var item in textItems) // Перебір кожного елемента в списку

{

item.Show(); // Виклик методу Show для кожного елемента

}

// Виведення слів, що зустрічаються більш ніж на N сторінках

int N = 3; // Встановлення порогу для кількості сторінок

Console.WriteLine($"\nСлова, що зустрічаються більш ніж на {N} сторінках:");

foreach (var item in textItems.OfType<WordItem>().Where(w => w.AppearsOnMoreThanNPages(N))) // Фільтрація слів, що зустрічаються більше ніж на N сторінках

{

item.Show(); // Виведення цих слів

}

// Виведення слів за абеткою

Console.WriteLine("\nСлова за абеткою:");

var sortedWords = textItems.OfType<WordItem>().OrderBy(w => w.Text).ToList(); // Сортування слів за алфавітом

foreach (var word in sortedWords) // Перебір кожного слова

{

word.Show(); // Виведення слова

}

// Виведення сторінок для заданого слова

string searchWord = "Програмування"; // Слово для пошуку

Console.WriteLine($"\nНомери сторінок для слова '{searchWord}':");

var wordToShow = textItems.OfType<WordItem>().FirstOrDefault(w => w.Text.Equals(searchWord, StringComparison.OrdinalIgnoreCase)); // Пошук слова у списку

if (wordToShow != null) // Якщо слово знайдено

{

wordToShow.Show(); // Виведення інформації про слово

}

else

{

Console.WriteLine($"Слово '{searchWord}' не знайдено."); // Якщо слово не знайдено

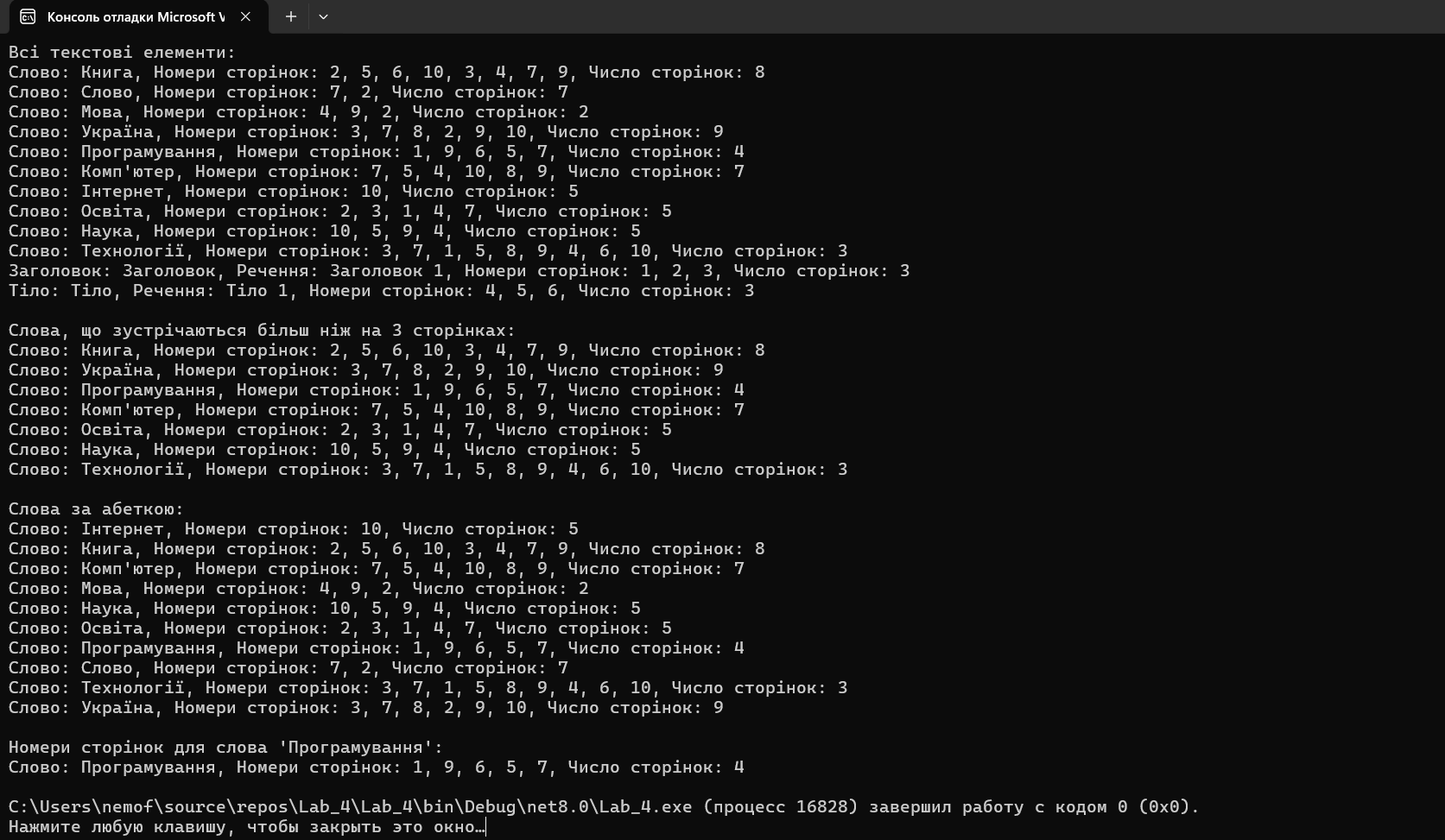
}

Console.ReadKey(); // Очікування натискання клавіші для завершення програми

}

}

**Результат роботи програми:**



**Алгоритм побудови коду:**

1. **Створення абстрактного класу TextItem**:
   * Він є батьківським класом для різних типів текстових елементів.
   * Має властивості Text, PageNumbers, і TotalPages.
   * Абстрактний метод Show() для виведення інформації.
2. **Створення класів першого рівня спадкування**:
   * WordItem, SentenceItem, і ParagraphItem є спадкоємцями TextItem та реалізують метод Show().
   * У класі WordItem додано метод AppearsOnMoreThanNPages(), що перевіряє кількість сторінок, на яких зустрічається слово.
3. **Створення класів другого рівня спадкування**:
   * TitleSentenceItem та BodySentenceItem успадковують клас SentenceItem і додають нові поля (Title і Body відповідно) та методи для відображення цих нових властивостей.
4. **Робота з поліморфним списком**:
   * Створено колекцію об'єктів різних рівнів спадкування.
   * Виконано операції над списком, такі як виведення текстових елементів, фільтрація слів за кількістю сторінок, сортування за алфавітом та пошук конкретного слова.

**Функціональність:**

* Програма дозволяє створювати текстові елементи, такі як слова, речення, і параграфи.
* Використовуючи поліморфізм, різні типи текстових елементів можуть оброблятися єдиним методом.
* Реалізовано функції для сортування, пошуку та фільтрації текстових елементів.

**Висновок:**

У даному завданні було створено систему класів із багаторівневим спадкуванням, що ілюструє роботу з абстрактними класами та поліморфізмом. Код ефективно демонструє можливість обробки різних типів даних у єдиній колекції та виконує дії, характерні для кожного класу.

**Завдання 3**

Розробити клас та перевантажити оператори згідно до свого варіанта. В головній функції програми протестувати роботу створених класів на прикладі використання окремих об’єктів та масивів чи колекцій цих об’єктів. Під час створення об’єктів застосувати випадковий підхід чи зчитування інформації з файлів.

**12.** Скласти описи класу, що забезпечує подання матриці заданого розміру n х m та будь-якого мінору в ній. Пам'ять для матриці виділяти динамічно. Написати кілька конструкторів, зокрема конструктор копіювання. Перевантажити оператори додавання, віднімання, множення та присвоювання для даного класу. Створити масив об'єктів даного класу і передати його в функцію, яка змінює для матриці її мінор шляхом множення на константу.

**Текст програми:**

using System; // Підключення простору імен для використання основних класів і методів

public class Matrix // Оголошення класу Matrix для роботи з матрицями

{

private double[,] data; // Двовимірний масив для збереження елементів матриці

private int rows; // Кількість рядків у матриці

private int cols; // Кількість стовпців у матриці

// Конструктор для створення матриці заданого розміру

public Matrix(int rows, int cols)

{

this.rows = rows; // Ініціалізація кількості рядків

this.cols = cols; // Ініціалізація кількості стовпців

data = new double[rows, cols]; // Створення матриці з вказаними розмірами

}

// Конструктор копіювання

public Matrix(Matrix other)

{

rows = other.rows; // Копіювання кількості рядків

cols = other.cols; // Копіювання кількості стовпців

data = new double[rows, cols]; // Створення нової матриці

Array.Copy(other.data, data, other.data.Length); // Копіювання даних з іншої матриці

}

// Метод для копіювання даних з іншої матриці

public void CopyFrom(Matrix other)

{

if (this.rows != other.rows || this.cols != other.cols) // Перевірка розмірів матриць

throw new ArgumentException("Матриці повинні мати однакові розміри для копіювання.");

Array.Copy(other.data, this.data, other.data.Length); // Копіювання даних

}

// Перевантаження оператора додавання

public static Matrix operator +(Matrix a, Matrix b)

{

if (a.rows != b.rows || a.cols != b.cols) // Перевірка розмірів матриць

throw new ArgumentException("Матриці повинні мати однакові розміри для додавання.");

Matrix result = new Matrix(a.rows, a.cols); // Створення нової матриці для результату

for (int i = 0; i < a.rows; i++) // Проходження по рядках

{

for (int j = 0; j < a.cols; j++) // Проходження по стовпцях

{

result.data[i, j] = a.data[i, j] + b.data[i, j]; // Додавання відповідних елементів

}

}

return result; // Повернення результату

}

// Перевантаження оператора віднімання

public static Matrix operator -(Matrix a, Matrix b)

{

if (a.rows != b.rows || a.cols != b.cols) // Перевірка розмірів матриць

throw new ArgumentException("Матриці повинні мати однакові розміри для віднімання.");

Matrix result = new Matrix(a.rows, a.cols); // Створення нової матриці для результату

for (int i = 0; i < a.rows; i++) // Проходження по рядках

{

for (int j = 0; j < a.cols; j++) // Проходження по стовпцях

{

result.data[i, j] = a.data[i, j] - b.data[i, j]; // Віднімання відповідних елементів

}

}

return result; // Повернення результату

}

// Перевантаження оператора множення

public static Matrix operator \*(Matrix a, Matrix b)

{

if (a.cols != b.rows) // Перевірка на можливість множення (кількість стовпців у першій має дорівнювати кількості рядків у другій)

throw new ArgumentException("Кількість стовпців у першій матриці повинна дорівнювати кількості рядків у другій матриці.");

Matrix result = new Matrix(a.rows, b.cols); // Створення нової матриці для результату

for (int i = 0; i < a.rows; i++) // Проходження по рядках

{

for (int j = 0; j < b.cols; j++) // Проходження по стовпцях

{

for (int k = 0; k < a.cols; k++) // Виконання множення та підсумовування

{

result.data[i, j] += a.data[i, k] \* b.data[k, j];

}

}

}

return result; // Повернення результату

}

// Метод для зміни мінору матриці шляхом множення на константу

public void MultiplyMinor(int row, int col, double constant)

{

if (row < 0 || row >= rows || col < 0 || col >= cols) // Перевірка правильності індексів

throw new ArgumentException("Неправильна позиція мінору.");

data[row, col] \*= constant; // Множення елемента на константу

}

// Метод для отримання значення елемента матриці

public double GetValue(int row, int col)

{

if (row < 0 || row >= rows || col < 0 || col >= cols) // Перевірка правильності індексів

throw new ArgumentException("Неправильна позиція.");

return data[row, col]; // Повернення значення елемента

}

// Метод для встановлення значення елемента матриці

public void SetValue(int row, int col, double value)

{

if (row < 0 || row >= rows || col < 0 || col >= cols) // Перевірка правильності індексів

throw new ArgumentException("Неправильна позиція.");

data[row, col] = value; // Встановлення значення елемента

}

// Метод для виведення матриці

public void Print()

{

for (int i = 0; i < rows; i++) // Проходження по рядках

{

for (int j = 0; j < cols; j++) // Проходження по стовпцях

{

Console.Write(data[i, j] + " "); // Виведення елемента з пробілом

}

Console.WriteLine(); // Перехід на новий рядок після кожного рядка матриці

}

}

}

class Task\_3

{

static void Main(string[] args)

{

Console.InputEncoding = System.Text.Encoding.UTF8; // Встановлення кодування для вводу

Console.OutputEncoding = System.Text.Encoding.UTF8; // Встановлення кодування для виводу

// Створення масиву об'єктів матриці

Matrix[] matrices = new Matrix[2];

matrices[0] = new Matrix(2, 2); // Ініціалізація першої матриці

matrices[1] = new Matrix(2, 2); // Ініціалізація другої матриці

// Заповнення матриць випадковими значеннями

Random rand = new Random(); // Створення генератора випадкових чисел

for (int k = 0; k < matrices.Length; k++) // Проходження по кожній матриці

{

for (int i = 0; i < 2; i++) // Проходження по рядках

{

for (int j = 0; j < 2; j++) // Проходження по стовпцях

{

matrices[k].SetValue(i, j, rand.Next(1, 10)); // Встановлення випадкового значення

}

}

}

// Виведення початкових матриць

Console.WriteLine("Матриця 1:");

matrices[0].Print(); // Виведення першої матриці

Console.WriteLine("Матриця 2:");

matrices[1].Print(); // Виведення другої матриці

// Додавання матриць

Matrix sum = matrices[0] + matrices[1]; // Сума матриць

Console.WriteLine("Сума матриць:");

sum.Print(); // Виведення результату

// Віднімання матриць

Matrix diff = matrices[0] - matrices[1]; // Різниця матриць

Console.WriteLine("Різниця матриць:");

diff.Print(); // Виведення результату

// Множення матриць

Matrix product = matrices[0] \* matrices[1]; // Добуток матриць

Console.WriteLine("Добуток матриць:");

product.Print(); // Виведення результату

// Зміна мінору матриці

matrices[0].MultiplyMinor(0, 0, 2); // Множення мінору першої матриці на 2

Console.WriteLine("Матриця 1 після множення мінору на 2:");

matrices[0].Print(); // Виведення результату

Console.ReadKey(); // Очікування натискання клавіші

}

}

**Результат роботи програми:**



**Алгоритм Побудови Коду**

1. **Створення класу Matrix**:
   * Визначити приватні поля для збереження даних матриці, кількості рядків та стовпців.
   * Реалізувати конструктор для ініціалізації матриці заданого розміру.
   * Додати конструктор копіювання для створення копії матриці.
   * Реалізувати методи для копіювання даних, встановлення та отримання значення елементів матриці.
2. **Перевантаження операторів**:
   * Додати перевантаження операторів +, -, \* для виконання відповідних математичних операцій над матрицями.
   * Реалізувати перевантаження оператора = для присвоєння однієї матриці іншій.
3. **Методи для роботи з мінором**:
   * Реалізувати метод MultiplyMinor, який дозволяє множити елемент матриці (мінор) на константу.
4. **Тестування функціональності**:
   * Створити масив об'єктів класу Matrix.
   * Заповнити матриці випадковими значеннями.
   * Провести операції додавання, віднімання, множення та змінити мінор матриці.
   * Вивести результати на консоль.

**Функціональність**

* **Створення матриці**: Клас Matrix дозволяє створювати матриці заданого розміру з можливістю заповнення їх даними.
* **Операції над матрицями**: Реалізовані базові математичні операції, такі як додавання, віднімання та множення матриць.
* **Зміна мінору**: Можливість множення конкретного елемента (мінору) матриці на константу.
* **Тестування**: У головному класі протестовані всі функціональні можливості класу Matrix на прикладах.

**Висновок**

Розробка класу для роботи з матрицями дозволила реалізувати основні математичні операції, а також продемонструвати принципи об'єктно-орієнтованого програмування через використання класів, конструкторів і перевантаження операторів. Результати тестування показали коректність реалізації.

**Завдання 4**

1. Для класу першого згідно до свого варіанту створити головний чи підпорядкований клас із яким організувати відносини агрегації. Продемонструвати роботу відносин, створивши декілька об’єктів відповідних класів та позбавляючи чи додаючи властивості до цих об’єктів.

2. Для класу першого завдання згідно до свого варіанту створити головний чи підпорядкований клас із яким організувати відносини композиції. Продемонструвати роботу відносин, створивши декілька об’єктів відповідних класів та позбавляючи чи додаючи властивості до цих об’єктів

|  |  |  |  |
| --- | --- | --- | --- |
| **Агрегація** | | **Композиція** | |
| *Головний клас* | *Підпорядкований клас* | *Головний клас* | *Підпорядкований клас* |
| Факультет | Студент | Студент | Оцінки з різних дисциплін |

**Текст програми для Завдання 1:**

using System; // Підключає базову бібліотеку для введення/виведення.

using System.Collections.Generic; // Додає можливість використовувати списки (List) та інші колекції.

public class Student // Оголошення класу Student, що представляє студента.

{

public string Name { get; set; } // Властивість для зберігання імені студента.

public int ID { get; set; } // Властивість для зберігання ID студента.

public Student(string name, int id) // Конструктор для ініціалізації імені та ID.

{

Name = name; // Присвоєння переданого значення імені.

ID = id; // Присвоєння переданого значення ID.

}

public void Show() // Метод для виведення інформації про студента.

{

Console.WriteLine($"Студент: {Name}, ID: {ID}"); // Виведення імені та ID студента.

}

}

public class Faculty // Оголошення класу Faculty, що представляє факультет.

{

public string FacultyName { get; set; } // Властивість для зберігання назви факультету.

private List<Student> students; // Приватний список студентів на факультеті.

public Faculty(string facultyName) // Конструктор для ініціалізації назви факультету.

{

FacultyName = facultyName; // Присвоєння назви факультету.

students = new List<Student>(); // Створення порожнього списку студентів.

}

public void AddStudent(Student student) // Метод для додавання студента до факультету.

{

students.Add(student); // Додає студента в список.

Console.WriteLine($"Додано студента: {student.Name} на факультет {FacultyName}"); // Виведення повідомлення про додавання студента.

}

public void RemoveStudent(Student student) // Метод для видалення студента з факультету.

{

if (students.Remove(student)) // Якщо видалення студента пройшло успішно.

{

Console.WriteLine($"Видалено студента: {student.Name} з факультету {FacultyName}"); // Повідомлення про видалення студента.

}

else // Якщо студента не знайдено в списку.

{

Console.WriteLine($"Студент {student.Name} не знайдено на факультеті {FacultyName}"); // Повідомлення про помилку.

}

}

public void ChangeStudentName(int id, string newName) // Метод для зміни імені студента за ID.

{

var student = students.Find(s => s.ID == id); // Пошук студента за його ID.

if (student != null) // Якщо студент знайдений.

{

student.Name = newName; // Зміна імені студента.

Console.WriteLine($"Змінено ім'я студента з ID {id} на {newName}"); // Повідомлення про зміну імені.

}

else // Якщо студента не знайдено.

{

Console.WriteLine($"Студента з ID {id} не знайдено."); // Повідомлення про помилку.

}

}

public void ShowStudents() // Метод для виведення списку студентів на факультеті.

{

Console.WriteLine($"Студенти факультету {FacultyName}:"); // Виведення назви факультету.

foreach (var student in students) // Перебір кожного студента у списку.

{

student.Show(); // Виведення інформації про студента.

}

}

}

public class Task\_4\_1 // Оголошення класу Task\_4\_1 для основної програми.

{

public static void Main() // Головна функція програми.

{

Console.InputEncoding = System.Text.Encoding.UTF8; // Налаштування кодування для введення.

Console.OutputEncoding = System.Text.Encoding.UTF8; // Налаштування кодування для виведення.

Faculty faculty = new Faculty("Комп'ютерні науки"); // Створення об'єкта факультету.

// Додавання студентів

Student student1 = new Student("Олег", 101); // Створення першого студента.

Student student2 = new Student("Ірина", 102); // Створення другого студента.

Student student3 = new Student("Сергій", 103); // Створення третього студента.

faculty.AddStudent(student1); // Додавання першого студента на факультет.

faculty.AddStudent(student2); // Додавання другого студента на факультет.

faculty.AddStudent(student3); // Додавання третього студента на факультет.

// Показати всіх студентів

faculty.ShowStudents(); // Виведення списку студентів на факультеті.

// Змінити ім'я студента

faculty.ChangeStudentName(102, "Ірина Петрівна"); // Зміна імені другого студента.

// Видалення студента

faculty.RemoveStudent(student2); // Видалення другого студента.

// Показати всіх студентів після видалення

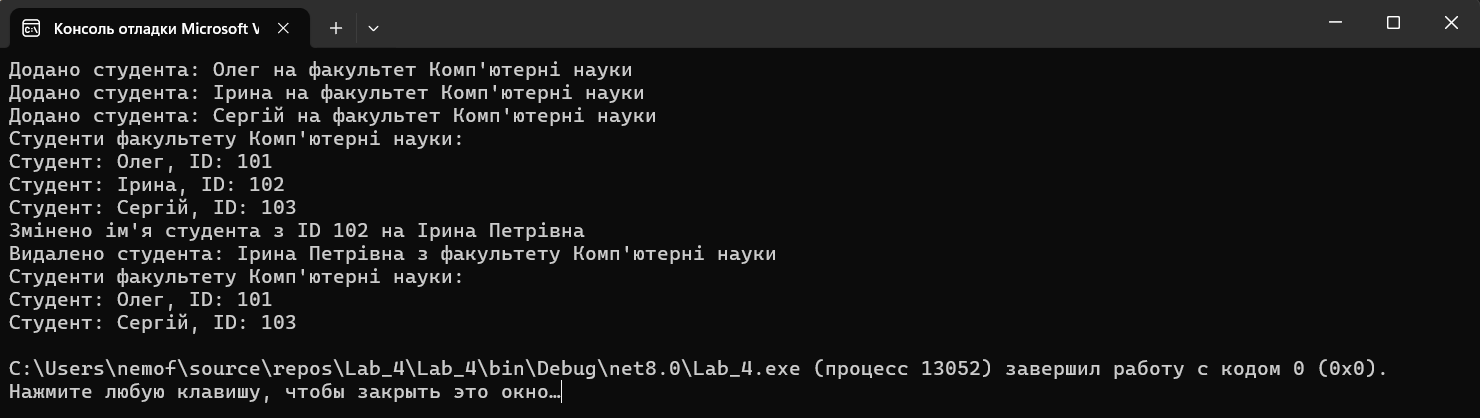
faculty.ShowStudents(); // Виведення списку студентів після видалення.

Console.ReadKey(); // Очікування натискання клавіші для завершення програми.

}

}

**Результат роботи програми:**



**Аглоритм побудови коду**

1. **Створення класу Student:**
   * Оголосити клас Student, який міститиме властивості Name та ID для зберігання інформації про студента.
   * Реалізувати конструктор для ініціалізації цих властивостей.
   * Додати метод Show() для виведення інформації про студента.
2. **Створення класу Faculty:**
   * Оголосити клас Faculty, що міститиме властивість FacultyName та приватний список студентів students.
   * Реалізувати конструктор для ініціалізації назви факультету та створення порожнього списку студентів.
   * Додати методи:
     + AddStudent(Student student): для додавання студента до списку.
     + RemoveStudent(Student student): для видалення студента зі списку.
     + ChangeStudentName(int id, string newName): для зміни імені студента за ID.
     + ShowStudents(): для виведення списку студентів факультету.
3. **Створення основної програми в класі Task\_4\_1:**
   * Налаштувати кодування для введення/виведення.
   * Створити об'єкт факультету.
   * Додати декілька студентів до факультету.
   * Показати список студентів.
   * Змінити ім'я одного зі студентів.
   * Видалити студента з факультету.
   * Показати оновлений список студентів.

**Функціональність**

* Клас Student представляє студента з основними властивостями (ім'я та ID), а також можливістю виведення інформації про нього.
* Клас Faculty агрегує об'єкти класу Student, дозволяючи додавати, видаляти та змінювати інформацію про студентів.
* Програма демонструє роботу з цими класами, включаючи додавання та видалення студентів, зміну їх імен та виведення інформації на екран.

**Висновок**

У даному завданні було продемонстровано організацію відносин агрегації між класами Faculty та Student. Створення та управління об'єктами цих класів дозволяє реалізувати основні функціональні можливості, такі як додавання, видалення та модифікація інформації про студентів. Це забезпечує ефективне управління даними у рамках факультету. Код є гнучким і розширюваним, що дозволяє в подальшому доповнювати функціональність за необхідності.

**Текст програми для Завдання 2:**

using System; // Підключає простір імен для базових функцій, як Console.

using System.Collections.Generic; // Підключає простір імен для роботи зі списками (List).

using System.Linq; // Дозволяє використовувати LINQ-запити для колекцій, наприклад FirstOrDefault.

public class Grade // Оголошуємо клас Grade для збереження інформації про предмет і оцінку.

{

public string Subject { get; set; } // Властивість для збереження назви предмету.

public int Score { get; set; } // Властивість для збереження оцінки.

public Grade(string subject, int score) // Конструктор класу Grade.

{

Subject = subject; // Ініціалізуємо властивість Subject значенням параметра.

Score = score; // Ініціалізуємо властивість Score значенням параметра.

}

}

public class Students // Оголошуємо клас Students для збереження інформації про студента і його оцінки.

{

public string Name { get; set; } // Властивість для збереження імені студента.

private List<Grade> grades; // Приватний список для зберігання оцінок студента.

public Students(string name) // Конструктор класу Students.

{

Name = name; // Ініціалізуємо властивість Name значенням параметра.

grades = new List<Grade>(); // Ініціалізуємо порожній список оцінок.

}

public void AddGrade(string subject, int score) // Метод для додавання оцінки.

{

grades.Add(new Grade(subject, score)); // Створюємо новий об'єкт Grade і додаємо його до списку.

Console.WriteLine($"Додано оцінку для студента {Name}: {subject} - {score}"); // Виводимо повідомлення про додавання оцінки.

}

public void RemoveGrade(string subject) // Метод для видалення оцінки за назвою предмету.

{

var gradeToRemove = grades.FirstOrDefault(g => g.Subject.Equals(subject, StringComparison.OrdinalIgnoreCase)); // Шукаємо предмет у списку.

if (gradeToRemove != null) // Якщо предмет знайдено.

{

grades.Remove(gradeToRemove); // Видаляємо оцінку.

Console.WriteLine($"Видалено оцінку {subject} для студента {Name}"); // Повідомляємо про видалення.

}

else

{

Console.WriteLine($"Оцінка {subject} не знайдена для студента {Name}"); // Повідомляємо, що предмет не знайдено.

}

}

public void ShowGrades() // Метод для виведення всіх оцінок студента.

{

Console.WriteLine($"Оцінки студента {Name}:"); // Заголовок для оцінок.

foreach (var grade in grades) // Проходимо по кожній оцінці у списку.

{

Console.WriteLine($"Дисципліна: {grade.Subject}, Оцінка: {grade.Score}"); // Виводимо предмет і оцінку.

}

}

}

public class Task\_4\_2 // Основний клас для виконання програми.

{

public static void Main() // Точка входу в програму.

{

Console.InputEncoding = System.Text.Encoding.UTF8; // Встановлюємо кодування UTF-8 для введення.

Console.OutputEncoding = System.Text.Encoding.UTF8; // Встановлюємо кодування UTF-8 для виведення.

Students studentOleg = new Students("Олег"); // Створюємо об'єкт студента Олега.

Students studentIrina = new Students("Ірина"); // Створюємо об'єкт студента Ірини.

// Додаємо оцінки для студентів.

studentOleg.AddGrade("Математика", 95);

studentOleg.AddGrade("Фізика", 89);

studentIrina.AddGrade("Хімія", 92);

studentIrina.AddGrade("Біологія", 85);

// Виводимо оцінки студентів.

studentOleg.ShowGrades();

studentIrina.ShowGrades();

// Видаляємо оцінку і знову виводимо оцінки для перевірки.

studentOleg.RemoveGrade("Фізика");

studentOleg.ShowGrades();

// Додаємо нову оцінку для Ірини.

studentIrina.AddGrade("Інформатика", 98);

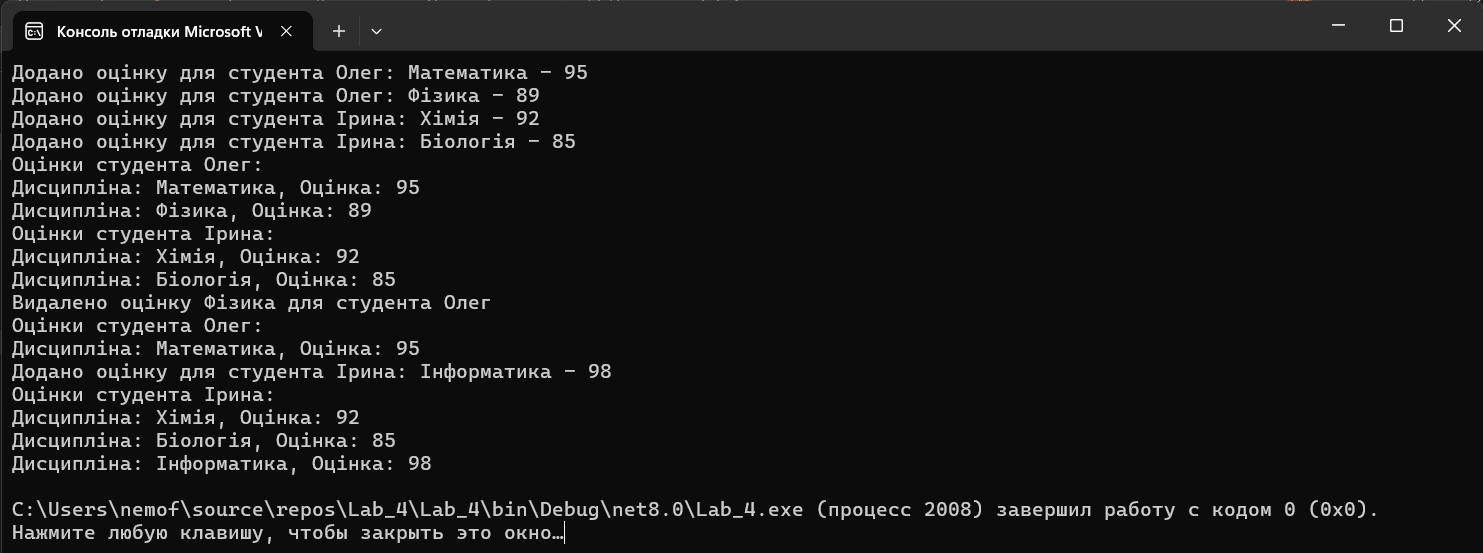
studentIrina.ShowGrades();

Console.ReadKey(); // Чекаємо натискання клавіші для завершення програми.

}

}

**Результат роботи програми:**



**Аглоритм побудови коду**

1. **Створення класів**:
   * Оголосити клас Grade, що представляє оцінку за предмет.
     + Властивості: Subject (назва предмету) і Score (оцінка).
     + Конструктор для ініціалізації цих властивостей.
   * Оголосити клас Students, що представляє студента і його оцінки.
     + Властивість: Name (ім'я студента).
     + Приватний список grades для зберігання об'єктів Grade.
     + Конструктор для ініціалізації імені студента та списку оцінок.
     + Методи:
       - AddGrade: для додавання нової оцінки.
       - RemoveGrade: для видалення оцінки за назвою предмету.
       - ShowGrades: для виведення всіх оцінок студента.
2. **Головний клас програми**:
   * Створити клас Task\_4\_2 з методом Main, що є точкою входу.
   * Налаштувати кодування для введення і виведення.
   * Створити кілька об'єктів класу Students.
   * Викликати методи для додавання, видалення та виведення оцінок.

**Функціональність**

Код реалізує наступну функціональність:

* Додає нові оцінки для студентів, зберігаючи їх у списку.
* Дозволяє видаляти оцінки за предметами.
* Виводить список оцінок для кожного студента на екран.
* Взаємодіє з користувачем через консоль, виводячи відповідні повідомлення.

**Висновок**

У результаті виконання завдання було створено два класи, Grade і Students, з відносинами композиції, оскільки об'єкти класу Grade є частиною об'єкта класу Students. Це дозволяє зберігати і управляти оцінками студентів у єдиній структурі. Код демонструє можливість додавання, видалення та виведення оцінок, що підтверджує правильність реалізації композиційних відносин. У подальшому можливе розширення функціональності, наприклад, додавання підрахунку середнього балу або можливість збереження даних в зовнішні файли.

**Загальний висновок**

Загальний висновок до лабораторної роботи №4 "Використання принципів та відносин об’єктно-орієнтованого програмування у мові C#"

У ході виконання лабораторної роботи було глибоко досліджено та практично застосовано ключові концепції об'єктно-орієнтованого програмування (ООП) у мові C#. Розглянуто такі важливі аспекти:

1. **Інкапсуляція**: Забезпечено приховування даних та реалізації об'єктів, що сприяє підвищенню надійності та зручності супроводу коду.
2. **Спадкування**: Використано для створення ієрархії класів, що дозволяє уникнути дублювання коду та забезпечує можливість повторного використання функціональності.
3. **Поліморфізм**: Дозволено обробляти об'єкти різних класів єдиним чином, що робить код більш гнучким та адаптивним.
4. **Перевантаження операторів**: Розширено функціональність класів, надавши їм можливість взаємодіяти з операторами мови програмування.
5. **Агрегація та композиція**: Використано для моделювання складних об'єктів, що складаються з інших об'єктів, забезпечуючи різні рівні залежності між ними.

У результаті виконання лабораторної роботи було набуто практичних навичок у застосуванні принципів та відносин ООП, що є фундаментальними для розробки сучасних програмних систем. Отриманий досвід дозволить ефективніше проектувати, реалізовувати та супроводжувати програмні рішення, використовуючи переваги об'єктно-орієнтованого підходу. Крім того, виконання лабораторної роботи сприяло поглибленню розуміння мови C# та її можливостей у контексті ООП.