Міністерство освіти України

Національний технічний університет "ХПІ"

кафедра “ННІКНІТ"

**Звіт**

**Лабораторна робота 2**

з дисципліни "Програмна технологія .NET"

Виконав: студент групи КН-921Г

Верещак А. Г.

Перевірив:

Івашко А.В.

Харків 2023

Лабораторна робота №3. **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).

# 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. (**Book**: Автор, Назва, Видавництво, Рік, Кількість сторінок. Створити масив об'єктів. Вивести: перелік книг заданого автора; перелік книжок, випущених заданим видавництвом; список книг, що випущені після заданого року.) - Варіант 19
2. Виконати завдання:
   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. Створи тимасив чи колекцію об’єктів усіх рівнів спадкування. Виконати відповідні функції над поліморфним списком із завдання власного варіанта.
   5. 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. (Визначити клас "Квадратна матриця" - **SqrMatrix**. Клас повинен містити кілька конструкторів, зокрема конструктор копіювання. Перевантажити оператори додавання, віднімання, множення та присвоювання для даного класу та розрахунок норми цієї матриці. Створити масив об'єктів класу Matrix і передати його в функцію, яка змінює матриці шляхом зведення їх в квадрат.)
   6. Виконати наступне: 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. 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

**Об'єктно-орієнтоване програмування (ООП)** - це парадигма програмування, яка базується на концепціях об'єктів та класів. Мова програмування C# - це мова, яка підтримує об'єктно-орієнтовану парадигму, і вона була розроблена з урахуванням цих концепцій. Основні принципи та відносини об'єктно-орієнтованого програмування в мові C# включають наступне:

1. Класи і об'єкти:

- Клас - це шаблон, описуючий структуру та поведінку об'єктів. У мові C# класи оголошуються за допомогою ключового слова `class`.

- Об'єкт - це конкретний екземпляр класу, який має свої власні стани (поля) та методи (функції).

2. Інкапсуляція:

- Інкапсуляція в C# відображається через модифікатори доступу, такі як `public`, `private`, `protected`, і т.д., які визначають, які частини класу можуть бути доступні ззовні.

- Інкапсуляція дозволяє обмежити доступ до деяких частин класу та захищати дані від ненавмисних змін.

3. Наслідування:

- Наслідування дозволяє створювати новий клас на основі існуючого класу, успадковуючи його властивості та методи.

- У C# класи можуть успадковувати інші класи за допомогою ключового слова `:`, наприклад: `class ChildClass : ParentClass`.

4. Поліморфізм:

- Поліморфізм дозволяє об'єктам різних класів використовувати однакові методи або властивості, що спрощує роботу з різнорідними об'єктами.

- У C# поліморфізм реалізується за допомогою віртуальних методів, інтерфейсів та абстрактних класів.

5. Композиція та агрегація:

- Композиція та агрегація вказують на те, як об'єкти можуть бути складені з інших об'єктів.

- Композиція позначає "частину-ціле" відношення, де один об'єкт складається з інших, наприклад, клас `Car` складається з об'єктів класів `Engine`, `Wheels`, тощо.

- Агрегація вказує на те, що один об'єкт може містити інший об'єкт, але той об'єкт може існувати незалежно від першого.

6. Інтерфейси:

- Інтерфейси визначають набір методів та властивостей, які клас повинен реалізовувати.

- Класи можуть реалізовувати багато інтерфейсів, що сприяє поліморфізму та структуризації коду.

Об'єктно-орієнтоване програмування в C# допомагає створювати більш структурований, модульний та легко збережений код, що спрощує розробку, підтримку та розширення програм. Правильне використання цих принципів і відносин допомагає покращити якість програм та зменшити їхню складність.

# Хід роботи

# Завдання 1:

Створюємо новий проект Task\_1.

Для вирішення поставленого завдання пишемо наступний код:

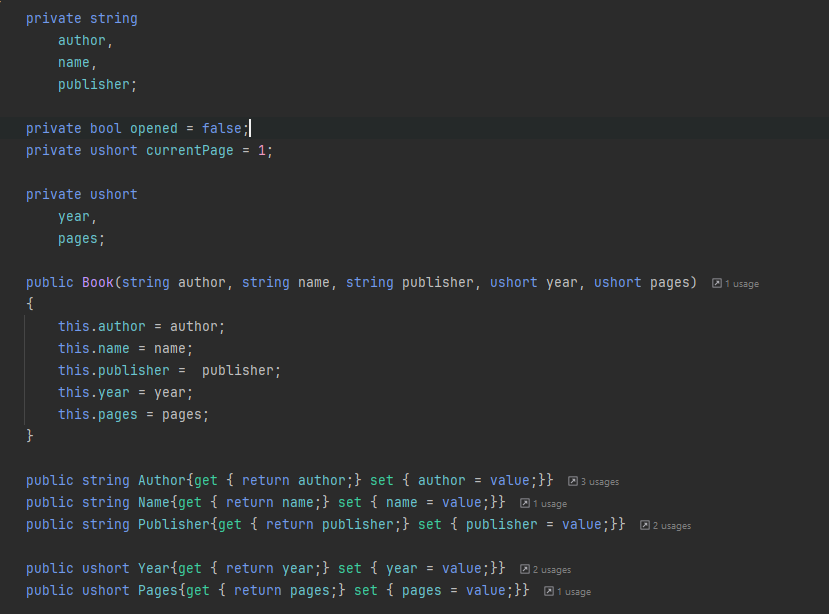


Рисунок 4.1 — Фрагмент класу Book

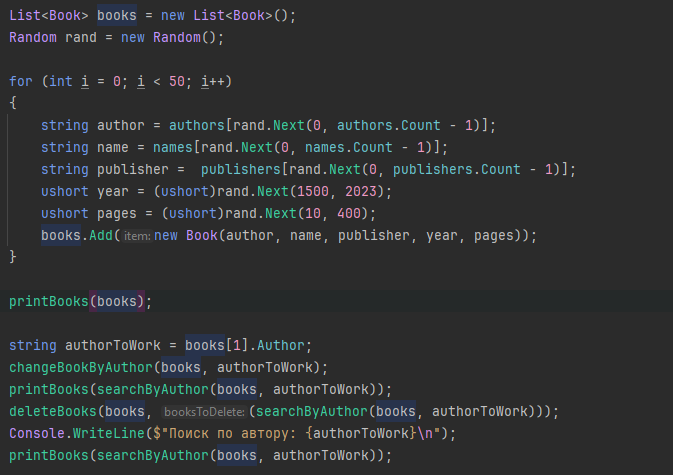


Рисунок 4.2 — Код програми Task\_1

# 

Рисунок 4.3 — Робота програми Task\_1

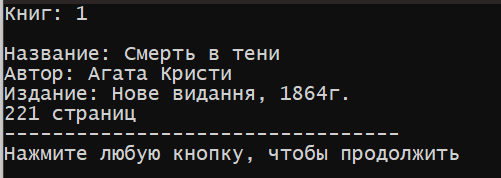


Рисунок 4.4 — Результат змінни видання у книжки Агати Крісті

# Завдання 2:

Створюємо новий проект Task\_2.

Для вирішення поставленого завдання розширимо кількість класів:

# 

Рисунок 4.5 — Список классів Task\_2

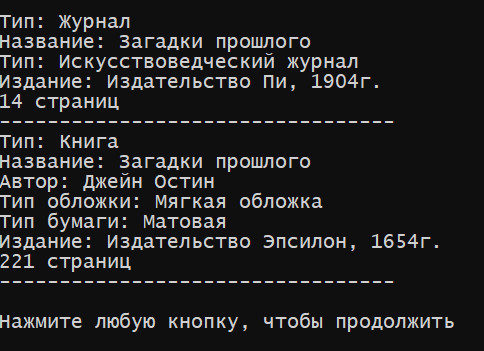


Рисунок 4.6 — Робота програми Task\_2

# Завдання 3:

Створюємо новий проект Task\_3.

Для вирішення поставленого завдання розширимо кількість класів:

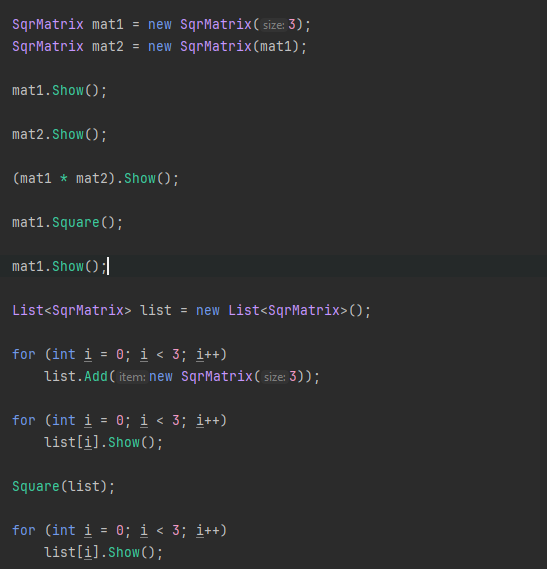


Рисунок 4.7 — Фрагмент коду Task\_3

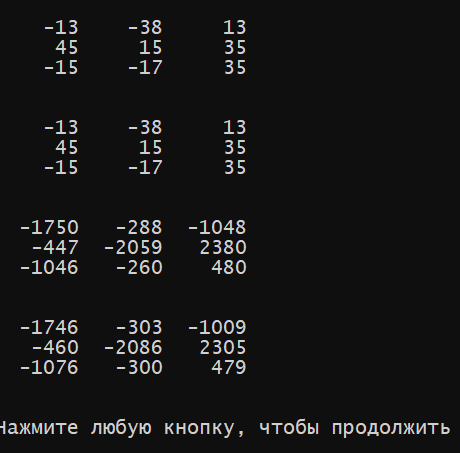
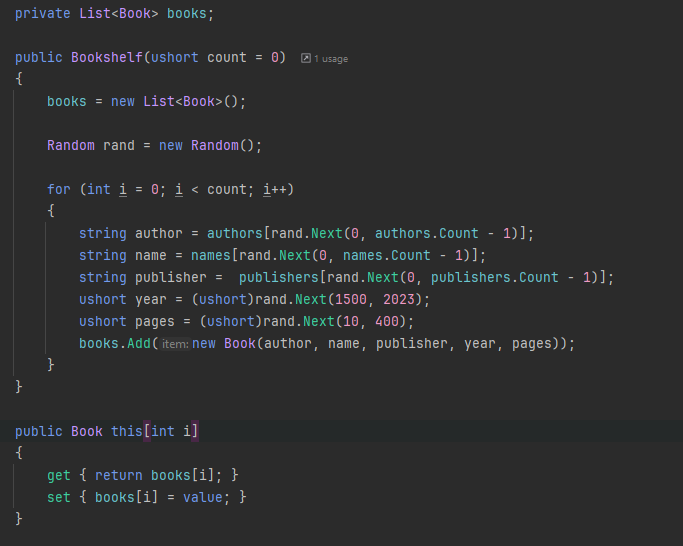


Рисунок 4.8 — Робота коду Task\_3

# Завдання 4:

Створюємо новий проект Task\_4.

Для демонстрації взаємозв’язків композиції та агрегації додаємо класи Bookshelf та Covering

Рисунок 4.9 — Фрагмент коду класса Bookshelf

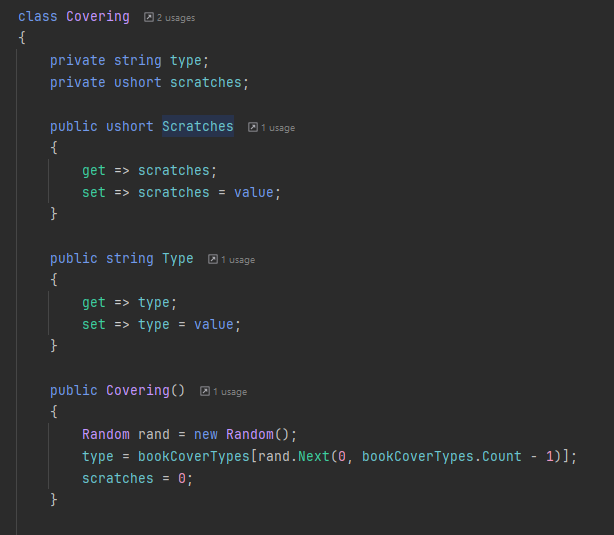


Рисунок 4.10 —Фрагмент коду класса Covering

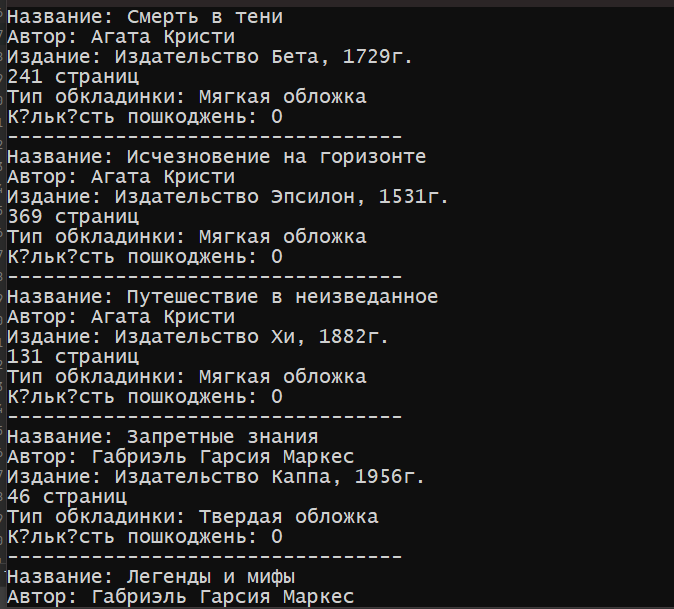


Рисунок 4.11 — Робота коду Task\_4

# Висновок

У даній лабораторній роботі ми вивчили важливі принципи та відносини об'єктно-орієнтованого програмування (ООП) у мові програмування C#. Ми розглянули основні концепції, які лежать в основі ООП, такі як класи і об'єкти, інкапсуляція, наслідування, поліморфізм, композиція та агрегація, а також інтерфейси.

Використання принципів ООП у мові C# дозволяє нам створювати більш структуровані, модульні та легко збережені програми. Ці принципи допомагають нам розділити функціональність програми на окремі класи та об'єкти, що сприяє покращенню їхньої читабельності та підтримуваності. Інкапсуляція дозволяє захищати дані від несанкціонованого доступу, а наслідування дозволяє використовувати та розширювати існуючий код.

Поліморфізм робить наш код більш гнучким і дозволяє нам працювати з об'єктами різних класів, використовуючи їхні спільні методи та властивості. Композиція та агрегація дозволяють нам моделювати складні структури об'єктів, що є невід'ємною частиною багатьох програм.

Інтерфейси роблять наш код більш узагальненим і структурованим, дозволяючи визначити, які методи повинні бути реалізовані в класах. Використання принципів та відносин ООП сприяє покращенню якості програм, зменшенню складності коду та полегшує їхню розробку та підтримку.

У результаті цієї лабораторної роботи ми здобули важливі знання та практичні навички використання об'єктно-орієнтованого програмування у мові C#, що допоможе нам створювати більш ефективні та модульні програми в майбутньому.