Черкаський національний університет імені Богдана Хмельницького

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ АВТОМАТИЗОВАНИХ СИСТЕМ

КУРСОВА РОБОТА

з дисципліни "Об'єктно-орієнтоване програмування"

НА ТЕМУ «Програма супроводу роботи прокатного відділу мультимедійних матеріалів»

Студента 2 курсу, групи КС-202
спеціальності «121 - Інженерія програмного забезпечення»
Царенка Романа Миколайовича

Керівник доцент, к.т.н.
Мисник Б.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна шкала: _____

Кількість балів: _____ Оцінка: ECTS _____

Члени комісії

_____	_____
(підпис)	**(прізвище та ініціали)**
_____	_____
(підпис)	**(прізвище та ініціали)**
_____	_____
(підпис)	**(прізвище та ініціали)**

м. Черкаси – 2022 рік

Зміст

Вступ

В нинішньому світі, час це гроші. Саме тому технології розвивалися і для заробітку, щоб зменшити витрати часу на виконанні ручної, рутинної непотрібної роботи. Представимо магазин, облік в якому ведеться вручну через зошит, це не вигідно ні покупцю ні продавцеві. Тому **метою курсової роботи**, є розробка «*програми супроводу роботи прокатного відділу мультимедійних матеріалів*», з використання основних принципів ООП (об'єктно-орієнтованого програмування), якими є:

- Інкапсуляція;
- Наслідування;
- Поліморфізм;

Об'єктно-орієнтоване програмування (ООП) – є одною з парадигм програмування, яка розглядає програму як множину «об'єктів», що взаємодіють між собою[1].

Інкапсуляція – це властивість, що дозволяє об'єднати дані, методи і приховати деталі реалізації від користувача. Для прикладу, це може бути об'єднання полів і методів в класі, з метою закрити прямий доступ до полів і відкрити його для методів, які цими полями керують[2] [3].

Наслідування – це властивість що дає змогу описати новий клас на основі вже існуючого який частково або повністю запозичує функціонал і який також можна доповнити. Клас від якого іде спадкування називається базовим або батьківським. Корисним від наслідування є те, що можна повторно використовувати вже написаний код, що дає змогу скоротити його[4] [5].

Поліморфізм – це властивість програми обирати різні реалізації під час виклику операцій з однією і тією ж назвою[6] [7].

Проаналізувавши означення переходимо до завдання курсової роботи.

Завдання: написати програму супроводу роботи прокатного відділу мультимедійних матеріалів. Тобто, це ведення обліку (списків) дисків які орендовані, продаються, продані.

Також це наявність:

- Назви файлу який записано на диск;
- Тип файлу (відео, програма, гра);
- Опис продукту;
- Категорія продукту (жанр);
- Ціна;

Під час виконання завдання буде використано три основних принципи ООП.

Для того щоб виконати мету курсової роботи, необхідно скласти план виконання завдання:

1. Вибрати оптимальний алгоритм створення програмного продукту;
2. Обрати потрібні структури даних;
3. Сформувати загальні правила та алгоритм реалізації з використанням принципів ООП;
4. Побудувати блок-схему загального алгоритму;
5. Схематично зобразити інтерфейс;
6. Побудувати діаграму класів програми;
7. Реалізувати програмний продукт;
8. Зробити висновки по роботі.

Розділ 1. Огляд алгоритмів роботи програми та інструментарію для реалізації програми

Для виконання завдання потрібно обрати інструментарій для написання програми, мову яка допоможе відобразити принципи ООП, та те, що допоможе зробити інтерфейс, а отже:

- Інтегроване середовище розробки Visual Studio 2022;
- Програма буде написана на мові «C#», оскільки ми обрали відповідне середовище у першому пункті, а також ця мова дозволить відобразити принципи ООП;
- Інтерфейс програми буде розроблений з використанням WPF. Мова розмітки – XAML.

1.1. Розгляд та опис алгоритму програми

Опишемо які функції матиме програма:

- Додавання товару;
- Перегляд списку наявних товарів в магазині;
- Зміна статусу товарів;
- Фільтрація за типом та категорією/жанром/типом файлу;

Попередньо в програмі будуть наявні 3 списки.

Перший список має інформацію про назву файлу записаного на диску, тип файлу (програма, гра, відео), опис файлу, та 3 кнопки, перша кнопка буде відображати категорію/жанр/типом файлу, та ціну цього диску. Дві інші кнопки будуть міняти статус диску, на проданий, чи зданий в оренду.

Другий список має ті ж самі поля, але він показує список дисків які знаходяться в оренді, перша кнопка знову таки показує додаткову інформацію, друга змінює статус диску на продається, третя змінює на статус продано.

Третій список просто буде містити продані диски, з усіма вище зазначеними полями.

При додаванні програми будуть доступні всі наявні поля, тобто це назва,

опис, ціна, тип, категорія, статус буде задано автоматично, що диск продається.

1.2. Вибір структур даних

Проаналізувавши попередньо написану інформацію, ми розуміємо, що для виконання поставленого завдання найкраще підходить List<T> , та enum.

В List<T>, ми будемо зберігати об’єкти, ще одним аргументом для використання слугує те, що дані будуть змінюватися динамічно, а саме йдеться про диски. А для зберігання станів диску, типів файлу записаних на цих дисках, та категорій/жанри (програм, ігор, фільмів).

А тепер розберемо що є List<T> та enum:

List<T> – це найпростіший строго типізований список однотипних об’єктів, до яких можна отримати доступ за допомогою індексу[8].

Enum – це набір логічно пов’язаних констант[9].

1.3. Вибір графічної бібліотеки

Для розробки графічного інтерфейсу було обрано WPF (Windows Presentation Foundation) – це фреймворк який входить до складу .NET, а також використовує мову розмітки XAML, що є доволі зручно в поєднанні з C#[10].

1.4. Висновок до першого розділу

Підведемо підсумки першого розділу, отже в цьому розділі було розглянуто алгоритм за яким буде працювати програма, також було обрано інструментарій (Visual Studio 2022), мову (C#), фреймворк з мовою розмітки для реалізації інтерфейсу (WPF та XAML), та структури даних які буде зручно використовувати для виконання поставленого завдання (List<T>, enum).

Розділ 2. Проектування програми супроводу роботи прокатного відділу мультимедійних матеріалів

2.1. Функції для реалізації програмного продукту

В цьому підрозділі розглянемо як буде використовуватися і працювати програма.

Для початку, як було описано вище, в програмі будуть наявні 4 сторінки, 3 з яких списки, і 4-та це сторінка додавання товару. На ці сторінки будуть спрямовувати кнопки які будуть знаходитися на боковій панелі. Назви кнопок будуть характеризувати куди перейде користувач (Add a product, Product list, Rented products, Product sold).

Тепер окремо розглянемо сторінки:

- «Add a product»

Сторінка матиме поля для написання назви, опису та ціни («Name», «Description», «Price»), та також 2 поля зі списком, в одному з яких будуть знаходитися вибір типу файлу («Type Of File»), а 2 список буде залежати від обраного в першому списку типу і він буде показувати категорію/жанр/тип («Game category», «Video genre», «Type of program»).

- «Product list»

Ця сторінка матиме 3 стовпчики («Name», «File Type», «Description»), та 4 кнопки («Info», «Rent», «Sell», «Delete»). Перша кнопка виводить додаткову інформацію про ціну та категорію/жанр/тип, друга та третя міняє статус диску, четверта видаляє диск із списку.

- «Rented products»

Ця сторінка має все теж саме, але показує тільки ті товари, які знаходяться в оренді, та кнопка «Rent», змінюється на «Unrent», щоб змінити статус диску на «On sale», і знову диск перемісти до першого списку.

- «Product sold»

Це фінальний список, і відображає продані продукти (диски), і має т кнопки

«Info» та «Delete».

Для того щоб зберігати дані, будуть написані «DTO», які будуть займатися конвертацією даних з моделей в dto, і звісно ж навпаки. Це все потрібно, щоб зміни внесені користувачем не були втрачені, і також щоб була можливість серіалізувати дані, тобто зберегти їх в «.json».

2.2. Загальна блок-схема програми

Проаналізувавши які дії будуть відбуватися в середині програми, які зміні потрібні, як теоретично користувач буде використовувати програму можна побудувати загальну блок-схему виконання програми.

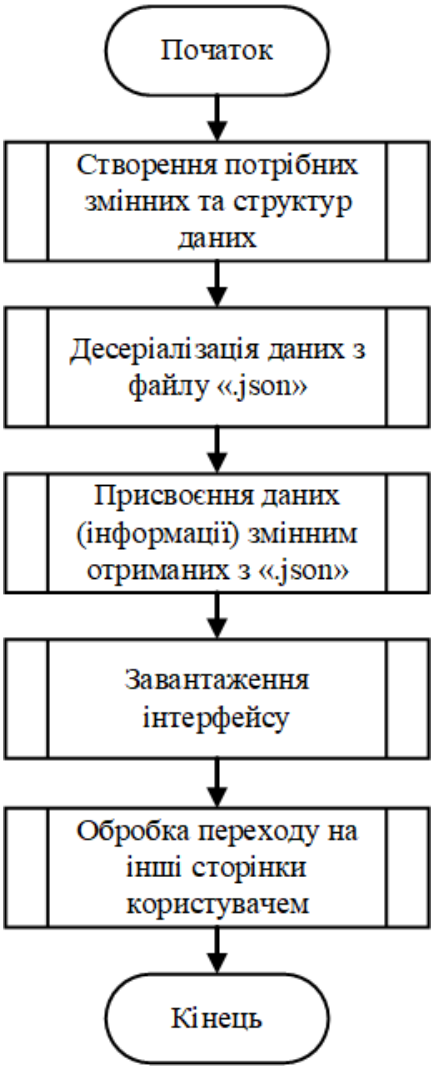

Рис. 2.2.1. Загальна блок-схема програми

Також у вигляді блок-схеми потрібно подати яким саме чином буде відбуватися додавання нового товару (диску), зміна його статусу, та його видалення.

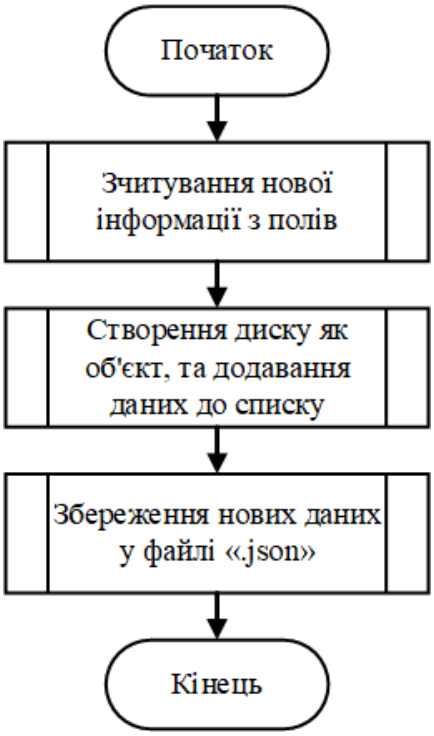

Рис. 2.2.2
Додавання нового товару (диску(ів))

Рис. 2.2.3.
Зміна статусу товару (диску(ів))

Рис. 2.2.4.
Видалення товару (диску(ів))

2.3. Схема майбутнього інтерфейсу програми

Наступним кроком є побудова схеми інтерфейсу, за цією схемою ми і виконаємо майбутній інтерфейс, опис цього інтерфейсу був в попередніх розділах, тому зайвого тексту тут не буде, тільки схеми.

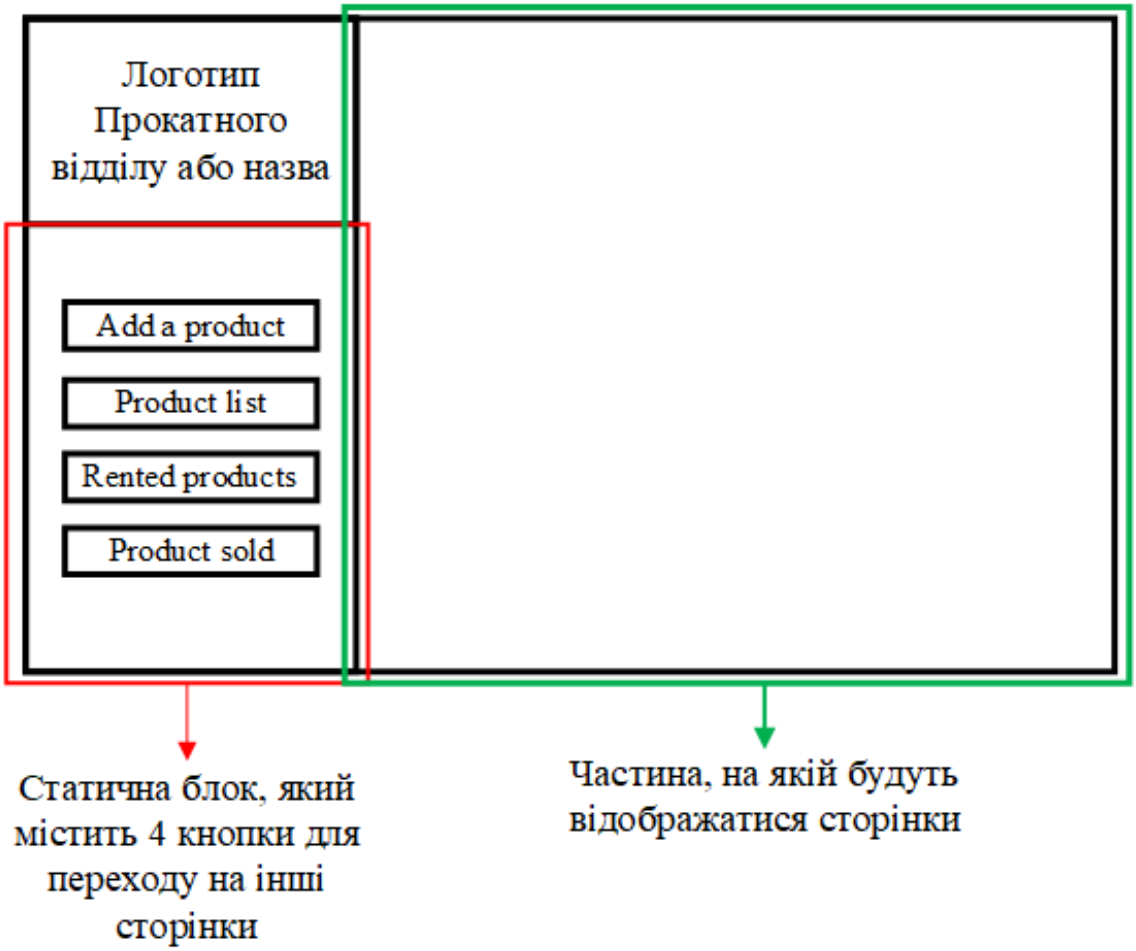

Рис. 2.3.1. Схема загального інтерфейсу

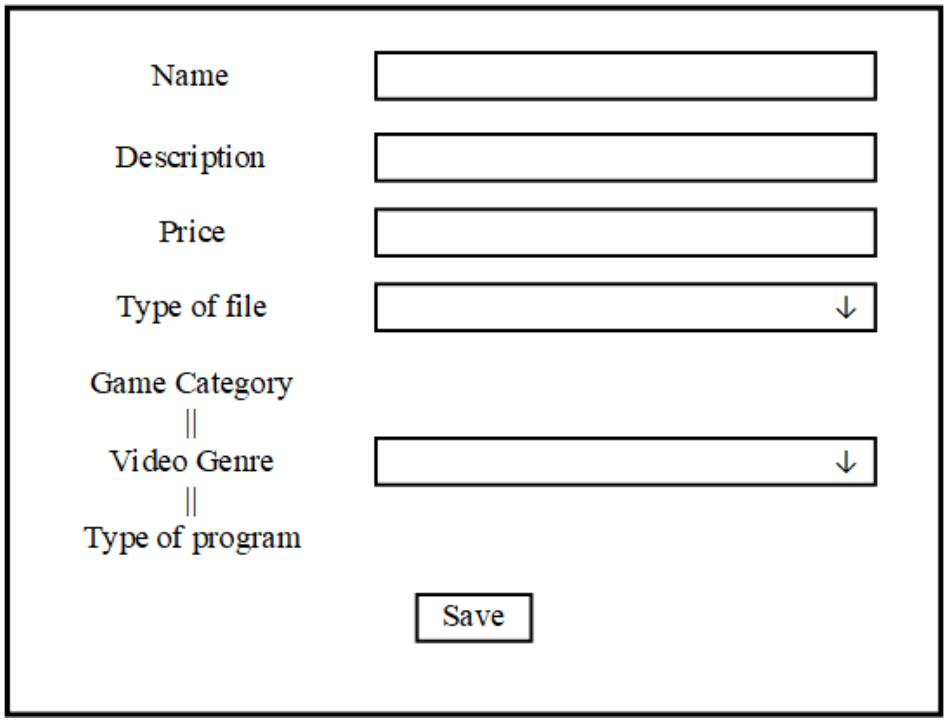

Рис. 2.3.2. Схема інтерфейсу сторінки «Add a product»

Це вигляд для сторінок «Product list» та «Rented products», синім позначено кнопки, зеленим текстові блоки з інформацією про товар (диск)

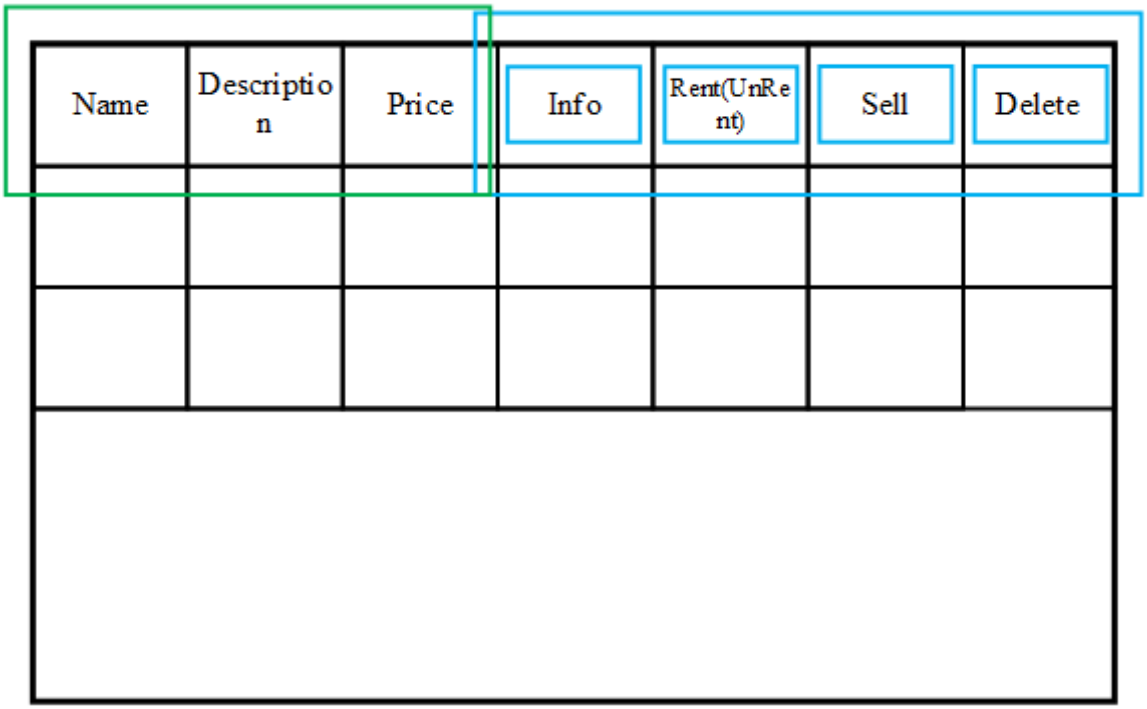

Рис. 2.3.3. Схема інтерфейсу сторінок «Product list» та «Rented products»

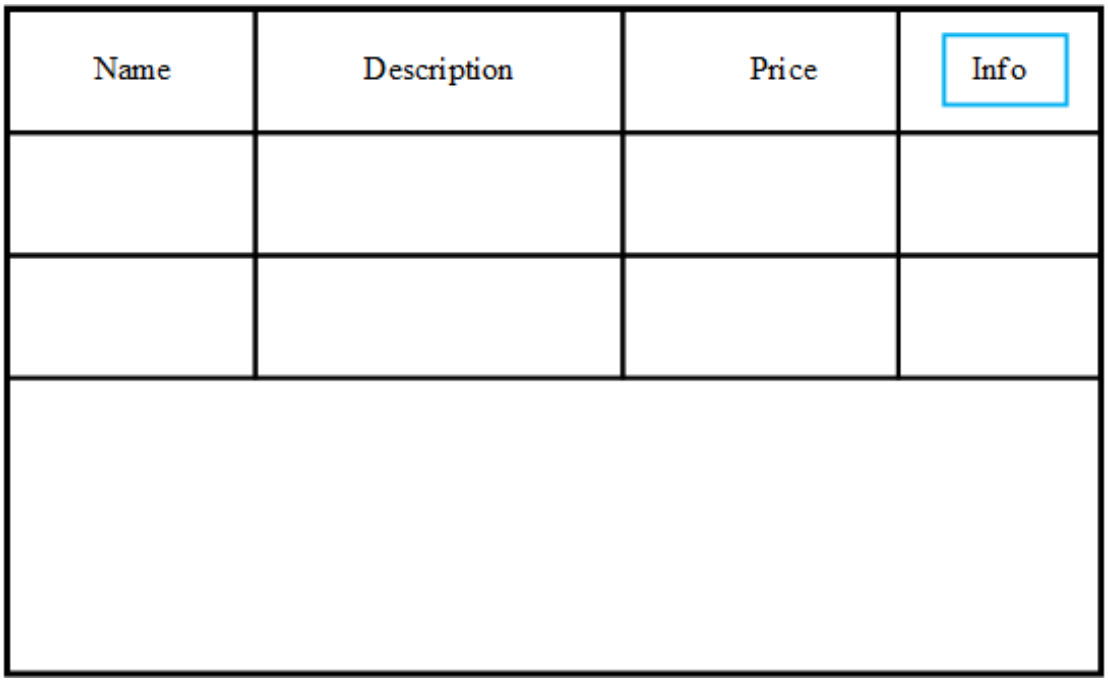

Рис. 2.3.4. Схема інтерфейсу сторінки «Product sold»

Проектування інтерфейсу завершено. Тому переходимо до наступного розділу.

2.4. Створення діаграми класів

Для виконання поставленого завдання, а саме реалізації програми, потрібно зрозуміти які класи нам потрібні, та як вони будуть пов’язані.

Для початку нам потрібний батьківський абстрактний клас, в якому будуть спільні поля, абстрактні методи, які згодом успадкують його нащадки. В нашому випадку, батьківським файлом буде клас «File».

Наступним кроком буде створення трьох класів, які для реалізації потребують наслідування, оскільки це будуть нащадки, які успадкують поля та методи батьківського, а саме це будуть класи: «GameFile», «ProgramFile», «VideoFile».

Тепер зобразимо схематично відношення класів, для кращого розуміння (Рисунок 2.4.1).

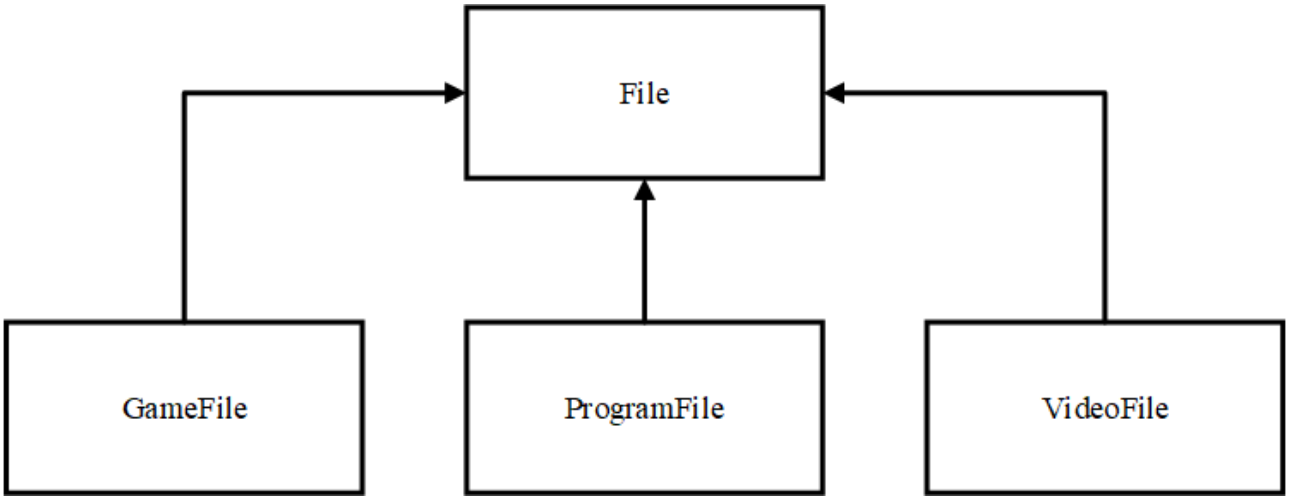

Рис. 2.4.1. Діаграма взаємодії класів

2.5. Висновок до другого розділу

Отже, в другому розділі було розглянуто які функції потрібні, а саме списки, яких три види, бокова панель з навігаційними кнопками, які будуть переключати сторінки, на яких знаходяться ці списки. Кнопки для видалення непотрібних дисків із списку, зміни статусу диску, для показу додаткової інформації. Також окремо була потрібна сторінка з додаванням товару. Було побудовано загальні блок-схеми роботи програми.

Окремо було схематично побудовано та розглянуто інтерфейс.

Описано відношення класів, як саме вони будуть взаємодіяти між собою, як наслідок було побудовано діаграму класів.

Розділ 3. Реалізація програми супроводу роботи прокатного відділу мультимедійних матеріалів

3.1. Використання ООП в програмі

В попередніх розділах були описані 3 основні принципи ООП, а саме «Наслідування», «Поліморфізм», «Інкапсуляція», тепер прийшов час реалізувати ці принципи в коді.

1. Інкапсуляція:

Для полів будуть створені властивості, для геттерів вони будуть публічними, а сеттери будуть захищеними, що не дозволить змінювати поля зовні.

```
public Guid Id { get; protected set; }
public string Name { get; protected set; }
public string Description { get; protected set; }
public double Price { get; protected set; }
public TypesOfFiles FileType { get; protected set; }
```

2. Наслідування:

В розділі 2.4. написано, що ми маємо абстрактний клас «File», від якого саме і будуть унаслідуватися нащадки, як відомо їх 3: «GameFile», «ProgramFile», «VideoFile».

```
public abstract class File : IEquatable<File>
public class GameFile : File
public class ProgramFile : File
public class VideoFile : File
```

3. Поліморфізм:

В даному проекті реалізований доволі простий поліморфізм. В базовому класі «File» наявний абстрактний метод «GetInfo», який перевизначений в нащадках класу «File»:

```
public abstract string GetInfo();

public override string GetInfo()
        {
```

```
            return $"Game price: {Price}\nGame category:
{GameCategory}" ;

        }


public override string GetInfo()
        {
            return $"Program price: {Price}\nProgram type:
{TypeOfProgram}";
        }



public override string GetInfo()
        {
            return $"Video price: {Price}\nVideo genre:
{TypeOfVideo}";
        }
```

3.2. Реалізація базових класів

Для зручності сприйняття тексту, код буде винесено в окремі додатки.

Найпростішим і першим кроком буде створення перерахувань, тобто enum-мів, які будуть відповідати за позначення типу файлу «TypesOfFiles», статусу оренди/продажу «RentalStatus», типу (жанру) відео «TypesOfVideos», жанру гри «GameCategories», та категорії програми «TypesOfPrograms», код знаходиться в додатку А

Наступним кроком буде реалізація абстрактного класу «File» та інтерфейсу «IEquatable<File>», це зроблено для того, щоб під час пошуку для видалення об’єкту використовувався ідентифікатор. Від батьківського класу унаслідуються наступні нащадки: «GameFile», «ProgramFile», «VideoFile», також вони унаслідують п’ять полів (властивостей), та один метод, код в додатку Б:

1. Ідентифікатор «Id», тип даних «Guid» → поле за яким буде перевірятися унікальність об’єкту (запису);

2. «Name», тип даних «string» → поле з назвою файлу записаного на диску;

3. «Description» тип даних «string» → поле з описом файлу записаного на диску;

4. «Price» тип даних «double» → поле з ціною диску;

5. «FileType» тип даних «TypesOfFiles» → поле з типом файлу записаного на диску;

6. «GetInfo», тип даних «string» → абстрактний метод для виведення додаткової інформації.

Далі вже реалізовуємо нащадків, код в додотку В:

- **«GameFile»**

1. Цей нащадок успадковує батьківські поля (властивості);

2. Реалізовуємо в ньому додаткове поле (властивість)

«GameCategory» тип даних «GameCategories» → поле з категорією гри записаної на диску;

3. Перевизначаємо в ньому метод «GetInfo»;

- **«ProgramFile»**

1. Цей нащадок успадковує батьківські поля (властивості);

2. Реалізовуємо в ньому додаткове поле (властивість)

«TypeOfProgram» тип даних «TypesOfPrograms» → поле з типом гри записаної на диску;

3. Перевизначаємо в ньому метод «GetInfo»;

- **«VideoFile»**

1. Цей нащадок успадковує батьківські поля (властивості);

2. Реалізовуємо в ньому додаткове поле (властивість)

«TypeOfVideo», тип даних «TypesOfVideos» → поле з типом гри записаної на диску;

3. Перевизначаємо в ньому метод «GetInfo»;

Оскільки в нас прокат мультимедійних матеріалів, то файл повинен зберігатися на диску, оскільки він і продається, тому реалізуємо додатковий клас «Disk», який в свою чергу буде мати (додаток Г):

1. Буде зберігати цей, файл, а якщо точніше інформацію про нього в окремому полі;

2. Реалізуємо в ньому поле (властивість) яка буде характеризувати статус диску

«RentalStatus» тип даних «RentalStatus» → поле з статусом диску;

3. Реалізуємо в ньому поле (властивість) яка буде характеризувати тип диску, який буде отримано від типу файлу.

«DiskType», тип даних «TypesOfFiles» → поле з типом диску;

4. І на останок в цьому класі буде реалізовано метод зміни статусу оренди диску.

І на останок потрібно реалізувати клас «DiskShop», в якому буде зберігатися список дисків «List<Disk> DiskList», методи які будуть виконувати операції по додаванню, видаленню, присвоєнню статусу дискам, розділенню файлів (дисків) по різних списках, тобто які продаються/знаходяться в оренді/продані, код доступний в додатку Д.

3.3. Реалізація користувальницького інтерфейсу

Використовуючи схему в попередніх розділах, реалізуємо інтерфейс. Рисунок 3.3.1 буде відображати рисунок 2.3.1, а саме це дві частини, боковий статичний блок з кнопками, та зона, на якій будуть відображатися сторінки із списками.

Рис. 3.3.1. Головне меню інтерфейсу

Наступним кроком потрібно реалізувати сторінку з полями для додавання товару, схема (Рисунок 2.3.2) якої була в попередньому розділі.

Рис. 3.3.2. Сторінка додавання товару

Тепер іде реалізація сторінок із списками «Product list» (Рисунок 3.3.3), «Rented products» (Рисунок 3.3.4), «Product sold» (Рисунок 3.3.5)

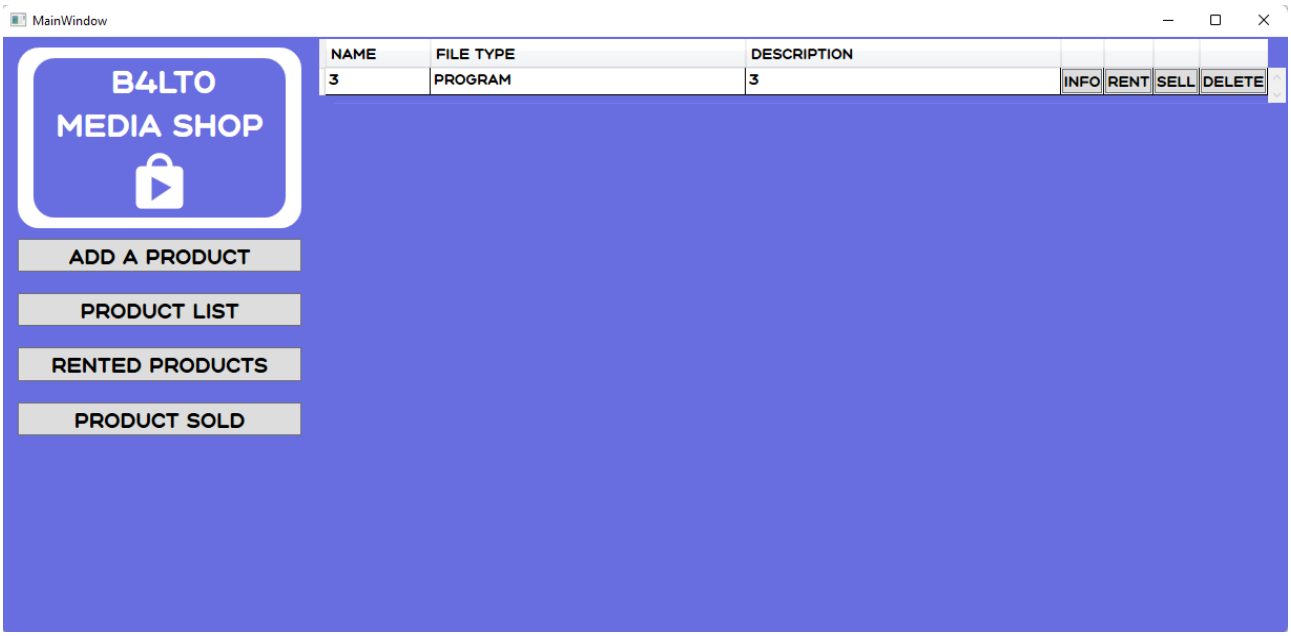

Рис. 3.3.3. Сторінка-список товарів які знаходяться в продажі

Рис. 3.3.4. Сторінка-список товарів які знаходяться в оренді

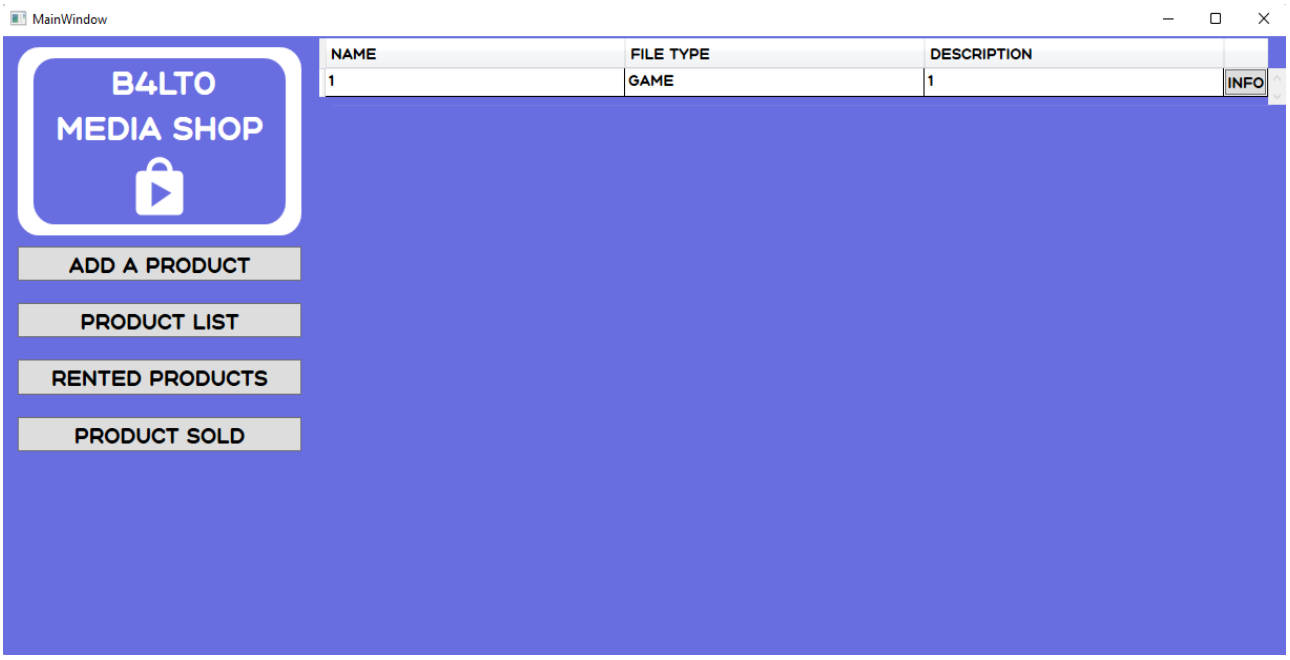

Рис. 3.3.5. Сторінка-список товарів які продано

3.4. Фінальна діаграма класів

Реалізувавши програму, можна підвести підсумки, і за допомогою інструментарію Visual Studio 2022 побудувати фінальну діаграму (рисунок 3.4.1) класів і порівняти з діаграмою на рисунку 2.4.1

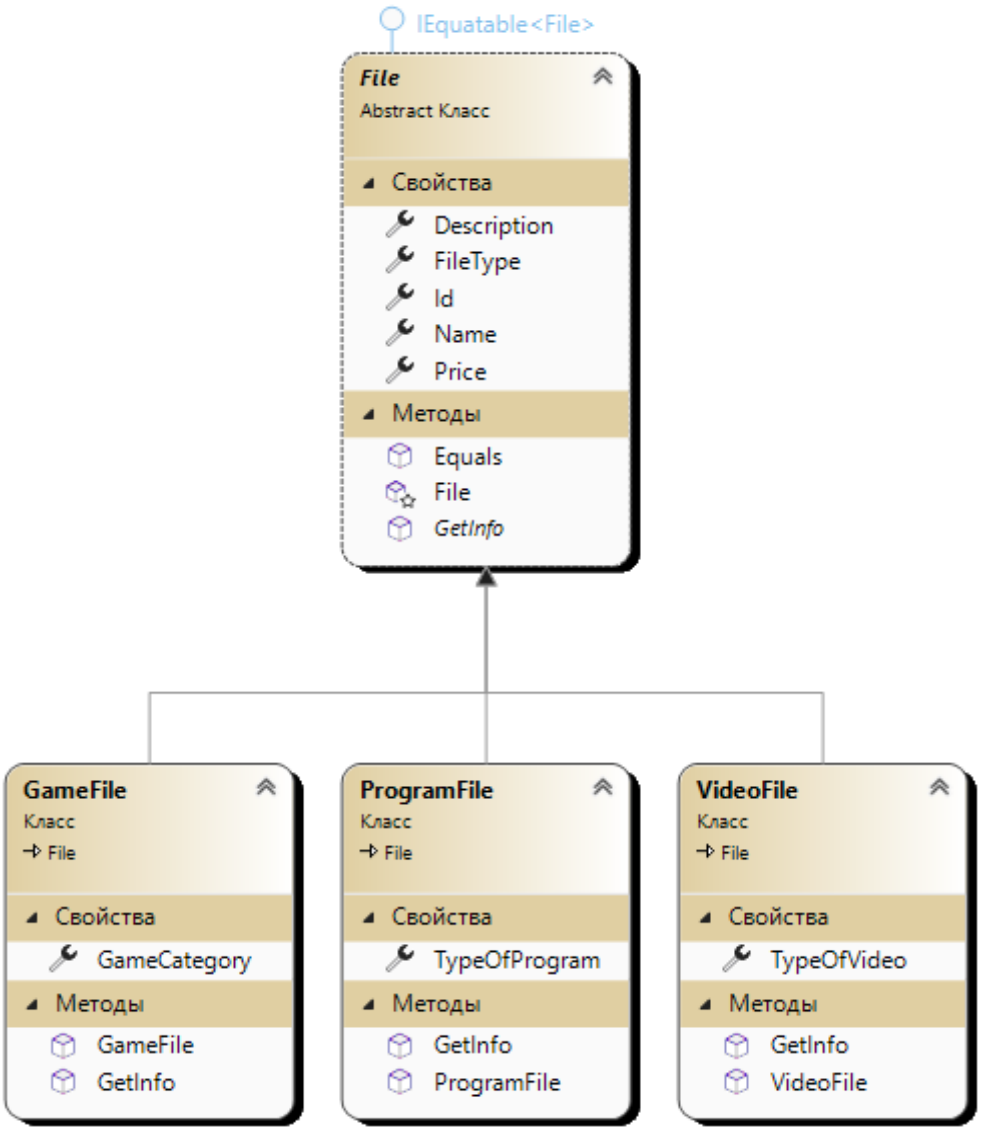

Рис. 3.4.1. Діаграма класів

3.5. Висновок до третього розділу

Підводимо підсумки третього розділу. В цьому розділі було реалізовано функції які були описані і потрібні для роботи програми супроводу роботи прокатного відділу мультимедійних матеріалів з використанням трьох основних концепцій ООП, реалізовано інтерфейс, схему якого було спроектовано в другому розділі, та присутній опис всього реалізованого.

Висновки

Для досягнення мети курсової роботи, було виконано план завдань, написаний в вступі. Обрали зручний і потрібний інструментарій. Спробували поставити себе на місце користувача, та розробити загальний алгоритм використання програми, проаналізовано який саме функціонал потрібний користувачу. Сформовано загальні алгоритми збереження, додавання, видалення товару, також сформовано алгоритм програми з використанням принципів ООП. Сформовано схематично, а згодом було реалізовано інтерфейс, який буде зручний користувачеві. Побудовано первинну діаграму класів.

Після завершення проектування програми, ми перейшли до реалізації всього вище описаного з використанням трьох принципів ООП, а саме інкапсуляція, наслідування, поліморфізму, також ці принципи були описані з прикладами їх застосування у коді.

Список інформаційних джерел

1. ООП [Електронний документ]. Режим доступу: https://goo.su/aO1hc; Перевірено: 04.06.2022.
2. Інкапсуляція[Електронний документ]. Режим доступу: https://goo.su/Aka4uP Перевірено: 04.06.2022.
3. Інкапсуляція[Електронний документ]. Режим доступу: http://devnotes.geega.net/note.php?note=oop-ua Перевірено: 04.06.2022.
4. Наслідування (успадкування) [Електронний документ]. Режим доступу: https://goo.su/DR7ku Перевірено: 04.06.2022.
5. Наслідування (успадкування) [Електронний документ]. Режим доступу: http://devnotes.geega.net/note.php?note=oop-ua Перевірено: 04.06.2022.
6. Поліморфізм [Електронний документ]. Режим доступу: https://goo.su/LVBN Перевірено: 04.06.2022.
7. Поліморфізм [Електронний документ]. Режим доступу: http://devnotes.geega.net/note.php?note=oop-ua Перевірено: 04.06.2022.
8. List<T> [Електронний документ]. Режим доступу: https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1?view=net-6.0 Перевірено: 04.06.2022.
9. Enum [Електронний документ]. Режим доступу: https://docs.microsoft.com/en-us/dotnet/api/system.enum?view=net-6.0 https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/enum Перевірено: 04.06.2022.

10. WPF [Електронний документ]. Режим доступу: https://docs.microsoft.com/en-us/visualstudio/designers/getting-started-with-wpf?view=vs-2022
Перевірено: 04.06.2022.

Додатки

Додаток А

```
public enum TypesOfFiles
    {
        Game,
        Video,
        Program
    }
public enum GameCategories
    {
        Sandbox,
        RealTimeStrategy,
        Shooters,
        MOBA,
        RolePlaying,
        SimulationAndSports,
        PuzzlersAndPartyGames,
        ActionAdventure,
        SurvivalAndHorror,
        Platformer
    }
public enum TypesOfVideos
    {
        FeatureFilms,
        Documentary,
        Serial,
        Cartoons
    }
public enum TypesOfPrograms
    {
        System,
        Application,
        Utility
    }
public enum RentalStatus
    {
        Sold,
        Rented,
        OnSale
    }
```

```
public abstract class File : IEquatable<File>
    {
        public Guid Id { get; protected set; }
        public string Name { get; protected set; }
        public string Description { get; protected set; }
        public double Price { get; protected set; }
        public TypesOfFiles FileType { get; protected set; }
        protected File(Guid id, string name, string description,
double price, TypesOfFiles typesOfFiles)
        {
            Name = name;
            Description = description;
            Price = price;
            FileType = typesOfFiles;
            Id = id;
        }

        public abstract string GetInfo();

        public bool Equals(File? other)
        {
            if (other == null)
                return false;

            return Id.Equals(other.Id);
        }
    }
```

```
public class GameFile : File
    {
        public GameFile(Guid id, string name, string description,
double price, TypesOfFiles typesOfFiles, GameCategories
gameCategory) : base(id, name, description, price, typesOfFiles)
        {
            GameCategory = gameCategory;
        }
        public GameCategories GameCategory { get; private set; }
        public override string GetInfo()
        {
            return $"Game price: {Price}\nGame category:
{GameCategory}" ;

        }
    }
public class ProgramFile : File
    {
        public ProgramFile(Guid id, string name, string description,
double price, TypesOfFiles typesOfFiles, TypesOfPrograms
typeOfProgram) : base(id, name, description, price, typesOfFiles)
        {
            TypeOfProgram = typeOfProgram;
        }
        public TypesOfPrograms TypeOfProgram  { get; private set; }
        public override string GetInfo()
        {
            return $"Program price: {Price}\nProgram type:
{TypeOfProgram}";
        }
    }
public class VideoFile : File
    {
        public VideoFile(Guid id,string name, string description,
double price, TypesOfFiles typesOfFiles, TypesOfVideos typeOfVideo)
: base(id, name, description, price, typesOfFiles)
        {
            TypeOfVideo = typeOfVideo;
        }
        public TypesOfVideos TypeOfVideo { get; private set; }
        public override string GetInfo()
        {
            return $"Video price: {Price}\nVideo genre:
{TypeOfVideo}";
        }
    }
```

```
public class Disk : IEquatable<Disk>
    {
        public File File { get; private set;}
        public RentalStatus RentalStatus { get; protected set; }
        public TypesOfFiles DiskType => File.FileType;

        public Disk(RentalStatus rentalStatus, File file)
        {
            RentalStatus = rentalStatus;
            File = file;
        }
        public void ChangeRentalStatus(RentalStatus rentalStatus)
        {
            RentalStatus = rentalStatus;
        }

        public bool Equals(Disk? other)
        {
            if (other == null)
                return false;

            return File.Id.Equals(other.File.Id);
        }
    }
```

```
public class DiskShop
    {
        public List<Disk> DiskList { get; private set; }

        public DiskShop(List<Disk> disks)
        {
            DiskList = disks;
        }

        public void AddDisk(Disk disk)
        {
            DiskList.Add(disk);
        }

        public void RemoveDisk(Disk disk)
        {
            DiskList.Remove(disk);
        }

        public void RentDisk(Disk disk)
        {
            if (disk.RentalStatus == RentalStatus.OnSale)
            {
                disk.ChangeRentalStatus(RentalStatus.Rented);
            }
        }

        public void RentRefund(Disk disk)
        {
            if (disk.RentalStatus == RentalStatus.Rented)
            {
                disk.ChangeRentalStatus(RentalStatus.OnSale);
            }
        }

        public List<File> GetSoldFiles()
        {
            List<File> files = new List<File>();

            foreach (var disk in DiskList)
            {
                if (disk.RentalStatus == RentalStatus.Sold)
files.Add(disk.File);
            }

            return files;
        }

        public List<File> GetOnSaleFiles()
        {
```

```
            List<File> files = new List<File>();

            foreach (var disk in DiskList)
            {
                if (disk.RentalStatus == RentalStatus.OnSale)
files.Add(disk.File);
            }

            return files;
        }

        public List<File> GetRentedFiles()
        {
            List<File> files = new List<File>();

            foreach (var disk in DiskList)
            {
                if (disk.RentalStatus == RentalStatus.Rented)
files.Add(disk.File);
            }

            return files;
        }

        public void SellDisk(Disk disk)
        {
            if (disk.RentalStatus == RentalStatus.OnSale ||
disk.RentalStatus == RentalStatus.Rented)
            {
                disk.ChangeRentalStatus(RentalStatus.Sold);
            }
        }

        public void SellDiskByFile(File file)
        {
            foreach (var item in DiskList)
            {
                if (item.File.Equals(file))
                {
                   SellDisk(item);
                    return;
                }
            }
        }

        internal void UnRentDiskByFile(File file)
        {
            foreach (var item in DiskList)
            {
                if (item.File.Equals(file))
                {
```

```
                RentRefund(item);
                return;
            }

        }
    }

    public void RemoveDiskByFile(File selectedItem)
    {
        foreach (var item in DiskList)
        {
            if (item.File.Equals(selectedItem))
            {
                RemoveDisk(item);
                return;
            }
        }
    }

    public void RentDiskByFile(File file)
    {
        foreach (var item in DiskList)
        {
            if (item.File.Equals(file))
            {
                RentDisk(item);
                return;
            }
        }
    }

    public List<File> GetFiles()
    {
        List<File> files = new List<File>();

        foreach (var disk in DiskList)
        {
            files.Add(disk.File);
        }
        return files;
    }

}
```