

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ЛІСОТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
Кафедра інформаційних технологій

КУРСОВА РОБОТА
з об'єктно-орієнтованого програмування
на тему: «*Система контролю товарів*»

Студента 2-го курсу ICT-21 групи
спеціальності _____

Спаса Олега

(прізвище та ініціали)

Керівник: доц. каф. IT

к.т.н. Яцишин С.І.

(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. Львів – 2023 рік

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ЛІСОТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

Кафедра інформаційних технологій

Дисципліна: «Об'єктно-орієнтоване програмування»

ЗАВДАННЯ

На курсову роботу

студенту 2-го курсу групи ІСТ-21

Спаса Олега

(прізвище, ім'я, по батькові)

1. Тема роботи, номер варіанту: Система контролю товарів, варіант 26
2. Дата здачі студентом завершеної роботи 01 червня 2023 року
3. Вхідні дані до роботи: літературні джерела, технічна документація щодо розробки програм, ДСТУ з оформлення документації.
4. Зміст пояснювальної записки:
Вступ. Специфікація роботи. Програмна документація. Висновки. Додатки.
5. Перелік графічного матеріалу: діаграма класів додатку; схема алгоритму реалізації одного з розроблених методів (за узгодженням з керівником курсової роботи).

**Календарний план
виконання курсової роботи**

№ з/п	Назва етапу роботи	Строк виконання	Відмітка про виконання
1	2	3	4
1	З'ясування загальної постановки завдання; розробка чернетки пояснювальної записки першого розділу пояснювальної записки.		

1	2	3	4
2	Програмна реалізація, налагодження та тестування додатка; розробка чернетки пояснювальної записки (другий розділ, висновки, список використаних джерел, додатки).		
3	Остаточне налагодження та тестування додатка; розробка чернетки пояснювальної записки (вступ, другий розділ, висновки, список використаних джерел, додатки).		
4	Розробка остаточного варіанта пояснювальної записки (вступ, другий розділ, висновки, список використаних джерел, додатки).		

Дата видачі завдання 20 жовтня 2022 року

Керівник роботи _____
(підпис)

доц. каф. ІТ, к.т.н. Яцишин С.І.
(посада, П. І. Б.)

Студент _____
(підпис)

Спас О.І.
(П. І. Б.)

ЗАВДАННЯ

На курсову роботу
студенту 2-го курсу групи ІСТ-21

Спаса Олега

Варіант 26

1. Розробити клас ТОВАР НА СКЛАДІ, який містить назву товару, дату виготовлення, свідоцтво якості, вартість.
2. Визначити конструктори, деструктор та методи встановлення і виведення значень полів даних.
3. Перевантажити операції: + ("плюс") та - ("мінус") для зміни вартості товару, операцію присвоєння об'єктів =, потокові операції введення » та виведення « об'єктів.
4. Визначити похідний клас ТОВАР У МАГАЗИНІ з додатковими полями даних: націнка, термін придатності. Визначити конструктори, деструктор, методи роботи з полями даних.
5. У межах ієрархії класів побудувати поліморфічний кластер на основі віртуального методу виведення ціни товару на складі та в магазині. Продемонструвати механізм пізнього зв'язування.
6. Розробити клас МАГАЗИН, що містить масив об'єктів класу ТОВАР У МАГАЗИНІ. Визначити прибуток магазину від продажу товарів.
7. Для роботи з масивом об'єктів побудувати та використати клас-ітератор.
8. Для роботи з масивом об'єктів побудувати та використати клас-ітератор.
9. Реалізувати пункти 1-7 завдання мовою C++.
10. Реалізувати п. 1-7 завдання відповідними засобами мови C# з використанням WinForms.
11. Вимоги до оформлення курсової роботи дивитись у методичних вказівках до виконання курсового проектування з ООП (диск Р)

Зміст

Реферат	5
Вступ	6
1 Специфікація роботи	7
1.1. Основи об'єктно орієнтованого програмування	7
1.2. Створення класів: поля, методи	9
1.3. Перевантаження операторів	9
1.4. Використання контейнерів	11
1.5. Робота з ітераторами	11
2 Програмна документація	13
2.1 Проектування структури програми	13
2.2 Проектування інтерфейсу користувача	13
2.2.1 Консольний інтерфейс	16
2.2.2 Графічний інтерфейс з використанням Windows Forms	17
2.3 Розроблення консольної версії програми на мові C++	18
2.4 Розроблення програми з графічним інтерфейсом на C#	20
2.5 Приклад та аналіз результатів виконання програми з консольним інтерфейсом	22
2.6 Приклад та аналіз результатів виконання програми з графічним інтерфейсом на Windows Forms	24
Висновки	26
Список використаних джерел	28
Додатки	30
Додаток А. Діаграми класів	29
Додаток Б. Код консольної реалізації програми на мові C++	30
Додаток В. Код реалізації програми з графічним інтерфейсом з використанням Windows Forms на мові C#	35

РЕФЕРАТ

Курсова робота складається з 40 сторінок пояснювальної записки, 8 рисунків, і 7 джерел. У роботі розглянуто теоретичний та практичний матеріал, який спрямований на набуття досвіду роботи з технологіями об'єктно-орієнтованого програмування (ООП). Під час виконання роботи були реалізовані дві версії програми в середовищі Visual Studio 2019. Перша версія є консольною і написана на мові програмування C++, а друга версія має графічний інтерфейс і реалізована на мові C# з використанням Windows Forms.

Клас `ProductOnStock` представляє продукти на складі. Він містить поля, такі як назва продукту (`productName`), дата виробництва (`manufacturingDate`), сертифікат якості (`qualityCertificate`) та вартість (`cost`). Клас надає методи для встановлення та отримання значень цих полів, а також метод `getPrice()`, який повертає вартість продукту. Клас також реалізує перевантаження операторів `+=`, `-=`, `=` для зміни вартості продукту та присвоєння значень об'єкта.

Клас `ProductInStore` успадковує властивості класу `ProductOnStock` і представляє продукти, які знаходяться в магазині. Додатково до полів батьківського класу, він містить поля `markup` (націнка на продукт) і `expirationDate` (строк придатності). Клас надає методи для встановлення та отримання значень цих полів, а також перевизначає метод `getPrice()`, який обчислює ціну продукту з урахуванням націнки.

Клас `Store` представляє магазин і містить колекцію об'єктів `ProductInStore`. Він має методи для додавання продукту до магазину (`addProduct()`), отримання ітератора для перебору продуктів (`getProducts()`) і обчислення загального прибутку магазину (`getTotalProfit()`).

Головна мета проекту - забезпечити систему управління продуктами в магазині, що дозволяє додавати нові продукти, встановлювати націнку та строк придатності, а також обчислювати загальний прибуток магазину. Проект також забезпечує збереження інформації про продукти у файлі за допомогою операцій введення-виведення (`operator>>` і `operator<<`).

Ключові слова: клас, контейнери, ітератори, конструктор, деструктор, C#, C++, Visual Studio, пізнє зв'язування, перевантаження операторів.

Вступ

Об'єктно-орієнтоване програмування (ООП) є одним з еволюційних кроків у розвитку програмування, яке дозволяє програмістам стати не просто виконавцями, але й архітекторами, створюючи структуровані програми з елегантною архітектурою. ООП надає нам ряд переваг. Воно сприяє більшій повторному використанню коду, оскільки класи можна інстанціювати у багатьох місцях програми. Воно також допомагає уникнути конфліктів та забезпечує легшу підтримку і розширення програм. Крім того, ООП сприяє вищій абстракції та модульності, дозволяючи нам розбити складні завдання на менші, більш керовані частини.

Ціль даної курсової роботи полягає в демонстрації основ ООП на прикладі розробки класів "ТОВАР НА СКЛАДІ", "ТОВАР У МАГАЗИНІ" і "МАГАЗИН".

Цей проект демонструє важливі концепції ООП, такі як спадкування, перевантаження операторів, поліморфізм та контейнери. Інтерфейс програмних застосунків реалізований як консольний застосунок на мові програмування C++ та графічний застосунок з використанням мови програмування C# та фреймворку Windows Forms.

1. Специфікація роботи

Для розробки програмного забезпечення відповідно до поставленого завдання необхідно знати основи:

Класи і об'єкти: Розуміння принципів класів, об'єктів, атрибутів та методів.

Конструктори і деструктори: Вміння створювати конструктори та деструктори для ініціалізації об'єктів і звільнення ресурсів.

Перевантаження операторів: Здатність виконувати різні дії з об'єктами, перегружаючи оператори, такі як +, -, =, >> та <<.

Наслідування: Використання похідного класу для успадкування властивостей та методів від базового класу.

Поліморфізм: Використання віртуальних методів та пізнього зв'язування для забезпечення різних реалізацій методів у похідних класах.

Масиви: Використання масивів для зберігання та маніпулювання об'єктами.

Ітератори: Використання класу-ітератора для послідовного доступу до елементів масиву об'єктів.

1.1. Основи об'єктно орієнтованого програмування

Основи ООП - це основи об'єктно-орієнтованого програмування. Це одна з парадигм програмування, яка розглядає програму як множину «об'єктів», що взаємодіють між собою. Основу ООП складають чотири основні концепції: інкапсуляція, успадкування, поліморфізм та абстракція .

Інкапсуляція - це процес приховування деталей реалізації об'єкта від зовнішнього світу. Це означає, що внутрішній стан об'єкта захищений від зовнішнього доступу і може бути змінений тільки через його методи. Інкапсуляція дозволяє забезпечити безпеку даних об'єкта, оскільки внутрішній стан об'єкта може бути змінений тільки через його методи. Це також дозволяє змінювати реалізацію об'єкта без впливу на зовнішній код, що використовує цей об'єкт.

Модифікатори доступу у мовах програмування C++ та C# дуже схожі. У обох мовах є такі модифікатори доступу: public, private та protected.

Public - це модифікатор доступу, який дозволяє доступ до члена класу з будь-якого місця у програмі. Це означає, що будь-який код може звертатися до **public**-члена класу.

Private - це модифікатор доступу, який обмежує доступ до члена класу тільки у межах цього класу. Це означає, що код поза класом не може звертатися до **private**-члена класу.

Protected - це модифікатор доступу, який дозволяє доступ до члена класу у межах цього класу та його нащадкам. Це означає, що код у класах-нащадках може звертатися до **protected**-члена базового класу.

Успадкування - це механізм, що дозволяє створювати новий клас на основі існуючого класу, успадковуючи його властивості та методи. Це дозволяє уникати дублювання коду та полегшує розробку програм. Успадкування дозволяє уникати дублювання коду, оскільки новий клас може успадкувати властивості та методи базового класу. Це також полегшує розробку програм, оскільки можна використовувати готові рішення з базових класів.

Поліморфізм - це властивість об'єктів різних класів мати однаковий інтерфейс. Це означає, що об'єкти різних класів можуть бути використані у загальному контексті без необхідності знати про їх конкретний тип. Поліморфізм дозволяє використовувати об'єкти різних класів у загальному контексті без необхідності знати про їх конкретний тип. Це полегшує розробку програм, оскільки не потрібно писати окремий код для кожного типу об'єкта.

Абстракція - це процес виділення найбільш важливих характеристик об'єкта, абстрагуючись від його деталей. Це дозволяє спростувати моделювання складних систем. Абстракція дозволяє спростувати моделювання складних систем, виділяючи найбільш важливі характеристики об'єкта і абстрагуючись від його деталей. Це допомагає зосередитись на тому, що робить об'єкт, а не на тому, як він це робить.

1.2. Створення класів: поля, методи

Щоб створити клас в C++, ви можете використовувати ключове слово `class`, після якого йде ім'я класу та його тіло, що містить поля та методи. Доступ до них регулюється специфікаторами доступу `private`, `public` та `protected`.

Ось приклад оголошення класу в C++:

```
class MyClass {  
public:  
    int myField;  
    void myMethod() {  
        // код методу  
    }  
};
```

В C#, клас також описується ключовим словом `class`, після якого йде ім'я класу та його тіло, що містить поля та методи. Доступ до них регулюється за допомогою специфікаторів доступу `public`, `private`, `protected`, `internal` та `protected internal`.

Ось приклад оголошення класу в C#:

```
class MyClass {  
  
    public int myField;  
  
    public void myMethod() {  
  
        // код методу  
    }  
  
}
```

1.3. Перевантаження операторів

Перевантаження операторів дозволяє вам змінювати поведінку стандартних операторів (таких як `+`, `-`, `*`, `/` тощо) для власних типів даних. В C++, перевантаження

операторів реалізується за допомогою операторних функцій, які оголошуються з ключовим словом `operator`¹. Операторні функції можуть бути реалізовані всередині класу або поза ним². Ось приклад перевантаження оператора `+` для класу `MyClass`:

```
class MyClass {  
  
public:  
  
    int myField;  
  
    MyClass(int field) : myField(field) {}  
  
    MyClass operator+(const MyClass &other) {  
  
        return MyClass(myField + other.myField);  
  
    }  
  
};
```

В `C#`, перевантаження операторів також реалізується за допомогою операторних функцій, які оголошуються з ключовим словом `operator`. Операторні функції в `C#` повинні бути статичними та публічними. Ось приклад перевантаження оператора `+` для класу `MyClass`:

```
class MyClass {  
  
    public int myField;  
  
    public MyClass(int field) { myField = field; }  
  
    public static MyClass operator+(MyClass a, MyClass b) {  
  
        return new MyClass(a.myField + b.myField);  
  
    }  
  
}
```

1.4. Контейнери

Контейнери - це класи або структури даних, що дозволяють зберігати колекції об'єктів. Вони застосовуються для зберігання об'єктів у вигляді організованої структури на основі конкретних правил збереження і доступу до елементів.

В C++, контейнери реалізовані у вигляді шаблонних класів у Стандартній бібліотеці шаблонів (STL). Це означає, що ви можете використовувати контейнери для зберігання об'єктів будь-якого типу. Деякими з найпоширеніших контейнерів у C++ є `vector`, `list`, `deque`, `set`, `map`.

В C#, контейнери реалізовані у вигляді класів у просторі імен `System.Collections` та `System.Collections.Generic`. Деякими з найпоширеніших контейнерів у C# є `List<T>`, `LinkedList<T>`, `Dictionary<TKey,TValue>`, `HashSet<T>` тощо.

Використання контейнерів дозволяє більш продуктивно створювати програмний код. Використання стандартних, добре перевірених класів контейнерів дозволяє програмісту створювати більш надійний код, і уникати типових помилок.

1.5. Ітератори

Ітератори - це об'єкти, що дозволяють переміщатися по елементах контейнера та отримувати доступ до них. Вони дуже схожі на вказівники та працюють з контейнерами у C++ та C#.

В C++, ітератори реалізовані у вигляді класів у Стандартній бібліотеці шаблонів (STL). Ітератори можуть бути різних типів, залежно від контейнера та операцій, які можна з ними виконувати. Наприклад, ітератори `vector` є ітераторами випадкового доступу, що дозволяє переміщатися по елементах контейнера у будь-якому напрямку та отримувати доступ до них за константний час.

В С#, ітератори реалізовані у вигляді класу `IEnumerator`, що дозволяє переміщатися по елементах контейнера та отримувати доступ до них. Ітератори у С# також можуть бути різних типів, залежно від контейнера та операцій, які можна з ними виконувати.

Ітератори дозволяють більш продуктивно працювати з контейнерами. Вони дозволяють писати більш читабельний код та уникати типових помилок при роботі з масивами.

2. Програмна документація

2.1. Проектування структури програми

Програма включає різні класи для представлення продуктів, магазину та його функціональності. Класи мають конструктори, методи та властивості для збереження та обробки інформації про продукти, розрахунку прибутку та інші операції.

Крім того, тут використовується принципи об'єктно-орієнтованого програмування, такі як наслідування, поліморфізм та інкапсуляція, для створення гнучкої та розширюваної архітектури програми.

Розробка програми для моделювання магазину дозволить практикувати роботу з класами, об'єктами, методами та даними. Можна створити реалістичну модель магазину, додати продукти, розрахувати прибуток та виконати різні операції, які пов'язані з управлінням продуктами та магазином в цілому.

Розглянемо структуру кожного класу окремо.

C++:

Клас Store

Цей клас представляє магазин і містить вектор об'єктів ProductInStore.

Приватні поля:

- `vector<ProductInStore> products`: вектор, який містить продукти в магазині.

Приватний вкладений клас StoreIterator:

- Цей клас реалізує ітератор для перебору продуктів у магазині.

Публічні методи:

- `Store()`: конструктор за замовчуванням.

- `~Store()`: деструктор за замовчуванням.
- `StoreIterator begin()`: повертає ітератор, що вказує на початок магазину.
- `StoreIterator end()`: повертає ітератор, що вказує на кінець магазину.
- `void addProduct(const ProductInStore& product)`: додає продукт до магазину.
- `double getTotalProfit()`: обчислює загальний прибуток магазину.

Клас ProductOnStock

Цей клас є базовим класом для продуктів на складі і містить основні властивості та методи для продукту.

- Захищені поля:

1. `string productName`: назва продукту.
2. `string manufacturingDate`: дата виробництва.
3. `string qualityCertificate`: сертифікат якості.
4. `double cost`: вартість продукту.

- Захищені методи:

1. `void validateDate(string date)`: перевіряє правильність формату дати.

- Публічні методи:

1. `ProductOnStock(string name, string date, string certificate, double cost)`: конструктор, що ініціалізує поля продукту.
2. `~ProductOnStock()`: деструктор за замовчуванням.
3. `void setProductName(string name)`: встановлює назву продукту.
4. `void setManufacturingDate(string date)`: встановлює дату виробництва продукту.
5. `void setQualityCertificate(string certificate)`: встановлює сертифікат якості продукту.
6. `void setCost(double cost)`: встановлює вартість продукту.

7. `string getProductName()`: повертає назву продукту.
8. `string getManufacturingDate()`: повертає дату виробництва продукту.
9. `string getQualityCertificate()`: повертає сертифікат якості продукту.
10. `double getCost() const`: повертає вартість продукту.
11. `double getPrice()`: повертає ціну продукту.
12. `ProductOnStock& operator+=(double value)`: перевантажений оператор "+=", що додає вартість до продукту.
13. `ProductOnStock& operator-=(double value)`: перевантажений оператор "-=", що віднімає вартість від продукту.
14. `ProductOnStock& operator=(const ProductOnStock& other)`: перевантажений оператор присвоєння.
15. `friend istream& operator>>(istream& in, ProductOnStock& product)`: перевантажений оператор вводу з потоку.
16. `friend ostream& operator<<(ostream& out, const ProductOnStock& product)`: перевантажений оператор виводу в потік.

Клас ProductInStore

Цей клас наслідує клас `ProductOnStock` і представляє продукт у магазині з додатковими властивостями.

- Захищені поля:

1. `double markup`: націнка на продукт у магазині.
2. `string expirationDate`: термін придатності продукту.

- Публічні методи:

1. `ProductInStore(string name, string date, string certificate, double cost, double markup, string expirationDate)`: конструктор, що ініціалізує поля продукту у магазині.
2. `~ProductInStore()`: деструктор за замовчуванням.

3. `void setMarkup(double markup)`: встановлює націнку на продукт у магазині.
4. `void setExpirationDate(string date)`: встановлює термін придатності продукту.
5. `double getMarkup() const`: повертає націнку на продукт у магазині.
6. `string getExpirationDate()`: повертає термін придатності продукту.
7. `double getPrice()`: повертає ціну продукту у магазині (включаючи націнку).

Головна функція `main`:

1. У головній функції створюється об'єкт класу `Store` з назвою `store`.
2. Запитується користувача про кількість продуктів.
3. У циклі запитується ввід даних для кожного продукту.
4. При спробі створити об'єкт `ProductInStore` може статися помилка, яка виводиться на екран.
5. Після цього об'єкт додається до магазину за допомогою методу `addProduct()`.
6. Відкривається вихідний файл з заданим ім'ям.
7. Проходиться по кожному продукту в магазині і записуються дані про нього у вихідний файл.
8. Вихідний файл закривається.
9. Обчислюється загальний прибуток магазину за допомогою методу `getTotalProfit()` і виводиться на екран.
10. Функція повертає 0 для завершення програми.

C#:

Клас `ProductOnStock`:

1. Клас `ProductOnStock` містить поля для назви продукту (`productName`), дати виробництва (`manufacturingDate`), сертифікату якості (`qualityCertificate`) та вартості (`cost`) продукту.
2. Клас має конструктор, який приймає значення для усіх полів, а також методи для встановлення і отримання значень полів.
3. Метод `GetPrice()` повертає вартість продукту.

4. Оператори $+=$, $-=$ використовуються для зміни вартості продукту.

Клас `ProductInStore`:

1. Клас `ProductInStore` успадковує від класу `ProductOnStock`.
2. Він містить додаткові поля для націнки (`markup`) і дати закінчення терміну придатності (`expirationDate`).
3. Конструктор класу `ProductInStore` приймає значення для усіх полів, викликає конструктор базового класу `ProductOnStock` та встановлює значення поля `expirationDate`.
4. Метод `GetPrice()` повертає ціну продукту, яка обчислюється з урахуванням вартості і націнки.
5. Перевизначений оператор `~` слугує для фіналізації об'єкту, але не має додаткової логіки.

Клас `Store`:

1. Клас `Store` містить приватне поле `products`, яке є списком об'єктів класу `ProductInStore`. Список `products` зберігає всі продукти, які є в магазині.
2. Клас `Store` реалізує інтерфейс `IEnumerable<ProductInStore>`, що дозволяє перебирати елементи списку `products`.
3. Клас має конструктор за замовчуванням, який ініціалізує список `products`.
4. Метод `AddProduct()` додає об'єкт типу `ProductInStore` до списку `products`.
5. Метод `GetTotalProfit()` обчислює загальний прибуток магазину шляхом просумування цін продуктів зі списку `products`.

Клас `StoreIterator`:

1. Клас `StoreIterator` реалізує інтерфейс `IEnumerator<ProductInStore>`, що дозволяє ітерувати елементи списку `products`.
2. Клас містить приватне поле `products`, яке є масивом об'єктів `ProductInStore`.

3. Методи MoveNext(), Reset() та Dispose() реалізовані для виконання інтерфейсу IEnumerable<ProductInStore>.
4. Властивість Current повертає поточний елемент списку products.

Головна форма Form1:

1. Клас Store містить приватне поле products, яке є списком об'єктів класу ProductInStore. Список products зберігає всі продукти, які є в магазині.
2. Клас Store реалізує інтерфейс IEnumerable<ProductInStore>, що дозволяє перебирати елементи списку products.
3. Клас має конструктор за замовчуванням, який ініціалізує список products.
4. Метод AddProduct() додає об'єкт типу ProductInStore до списку products.
5. Метод GetTotalProfit() обчислює загальний прибуток магазину шляхом просумування цін продуктів зі списку products.

2.2. Проектування інтерфейсу користувача

2.2.1. Консольний інтерфейс

Задача проектування інтерфейсу користувача відіграє важливу роль у створенні зручних та ефективних програм та систем. Інтерфейс користувача є мостом між користувачем і програмним забезпеченням, що дозволяє здійснювати взаємодію, передавати інформацію та керувати функціональністю.

Проектування ефективного інтерфейсу користувача вимагає ретельного аналізу потреб користувачів, їхніх очікувань та вміння взаємодіяти з програмою. Це охоплює вивчення контексту використання, профілю користувачів, їхніх завдань та цілей. При цьому важливо забезпечити простоту використання, зрозумілість, консистентність та ефективність інтерфейсу.

Проектування інтерфейсу користувача в даному випадку передбачає використання консольного інтерфейсу.

Перша версія програми має консольний інтерфейс, який дозволяє користувачеві вводити дані про продукти безпосередньо в консолі. Користувачу потрібно ввести кількість продуктів, а також назву, дату виготовлення, сертифікат якості, ціну, націнку та дату придатності для кожного продукту.

Під час введення дати виготовлення та дати придатності, програма перевіряє, чи дата виготовлення не перевищує дату придатності. Якщо дата виготовлення більша за дату придатності, консоль виводить помилку і повідомляє користувачеві про це.

Можна продовжувати вводити дані про інші продукти, а програма буде обробляти та зберігати ці дані для подальшого використання. Після введення всіх даних, програма обчислює загальний прибуток магазину на основі цих даних.

Такий консольний інтерфейс дозволяє зручно та швидко вводити дані, простими командами у консолі. Користувач може бачити результати та помилки безпосередньо в консолі, що полегшує взаємодію з програмою.

2.2.2. Графічний інтерфейс з використанням Windows Forms

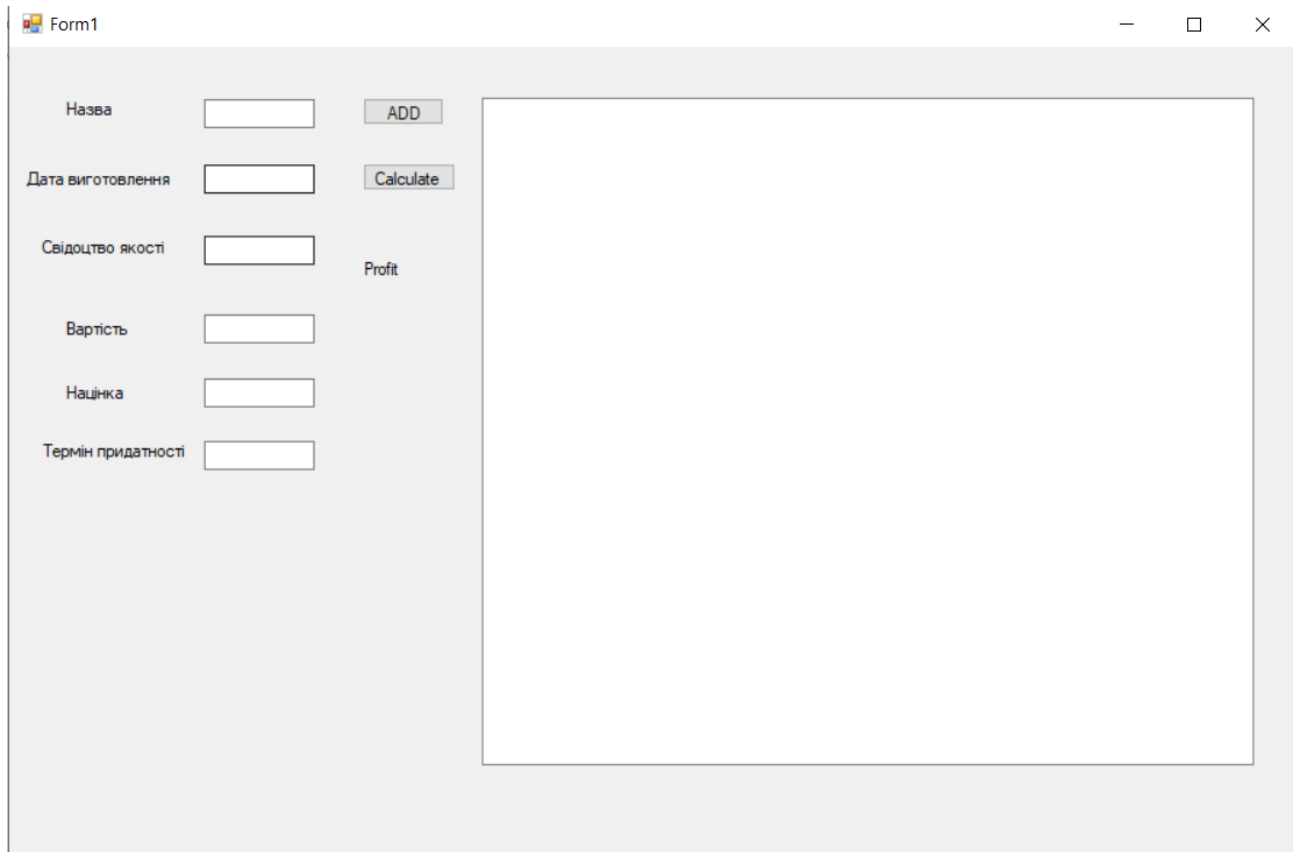
Друга версія програми включає графічний інтерфейс (ГІ), розроблений з використанням Windows Forms. Цей інтерфейс надає зручний спосіб взаємодії з користувачем та включає різноманітні елементи управління для введення та відображення даних.

На формі ГІ відображено 6 текстових полів (TextBox), у яких користувач може ввести дані про продукт, такі як назву, дату виготовлення, сертифікат якості, вартість, націнку та дату закінчення терміну придатності. Кожне текстове поле надає можливість введення відповідного типу даних.

Також на формі розміщено дві кнопки. Перша кнопка, після натискання, збирає введені дані з текстових полів і створює об'єкт продукту. Цей об'єкт додається до списку продуктів у магазині. Доданий продукт також відображається у списку, що розміщений на формі.

Друга кнопка, після натискання, обчислює загальний прибуток магазину на основі даних про всі продукти у списку. Отриманий прибуток виводиться на екран або на спеціально розміщену етикетку.

Цей графічний інтерфейс надає зручний та інтуїтивно зрозумілий спосіб взаємодії з програмою. Користувач може легко вводити дані про продукти, додавати їх до магазину та отримувати інформацію про загальний прибуток. На рисунку нижче показано спроектований графічний інтерфейс програми.



2.3 Розроблення консольної версії програми на мові C++

Програма дозволяє користувачу вводити дані про продукти та обчислювати загальний прибуток магазину на основі цих даних.

Клас Store включає в себе вектор об'єктів ProductInStore, який зберігає дані про продукти у магазині. Крім того, в класі є внутрішній клас StoreIterator, який реалізує ітератор для перебору продуктів у магазині.

Клас ProductOnStock представляє базовий клас для продуктів на складі. Він містить реалізацію основних методів для роботи з продуктами, таких як встановлення

та отримання назви, дати виготовлення, сертифікату якості, вартості та інших атрибутів.

Клас `ProductInStore` є похідним класом від `ProductOnStock` і додає до нього додаткові атрибути, такі як націнка та дата закінчення терміну придатності. Крім того, він перевизначає метод `getPrice()`, який обчислює ціну продукту з урахуванням націнки.

У функції `main()` виконується основний потік виконання програми. Користувачу пропонується ввести дані про кількість продуктів, а потім вводити деталі кожного продукту. Введені дані додаються до магазину за допомогою методу `addProduct()`. Після завершення вводу, дані про продукти записуються у текстовий файл.

На завершення, розраховується загальний прибуток магазину з використанням методу `getTotalProfit()` та виводиться на екран.

Цей консольний інтерфейс дозволяє користувачу взаємодіяти з програмою шляхом введення даних з клавіатури та отримання результатів на екрані.

2.3. Розроблення програми з графічним інтерфейсом на C#

Запрограмуємо графічний інтерфейс програми на мові C# за допомогою Windows Forms. Зображений нижче рисунок показує спроектований графічний інтерфейс програми, створений за допомогою Visual Studio 2019. Кожен елемент керування графічним інтерфейсом може генерувати відповідні події в залежності від дій користувача, виконаних над ним. Ці згенеровані події можуть бути пов'язані з відповідними методами для їх обробки, які реалізовані у відповідних класах програми.

Основний функціонал коду полягає в наступному:

Головна форма програми має назву `Form1` і наслідує клас `Form` з Windows Forms.

В конструкторі форми ініціалізується об'єкт `store`, який представляє магазин.

У методі `addProductButton_Click` обробляється подія натискання кнопки "Add Product". Зчитуються дані про продукт з текстових полів форми, створюється об'єкт `ProductInStore` на основі зчитаних даних, додається до магазину (`store.AddProduct(product)`) та відображається в `productsListBox` на формі.

У методі `calculateProfitButton_Click` обробляється подія натискання кнопки "Calculate Profit". Викликається метод `GetTotalProfit()` магазину для обчислення загального прибутку. Результат виводиться на етикетку `profitLabel` на формі.

Клас `ProductOnStock` представляє продукт на складі. Він має властивості для зберігання назви, дати виготовлення, сертифікату якості та вартості. Також містить метод `GetPrice()`, який повертає вартість продукту.

Клас `ProductInStore` успадковує від `ProductOnStock` і додає властивості для націнки та дати закінчення терміну придатності. Також перевизначає метод `GetPrice()`, щоб врахувати націнку.

Клас `Store` представляє магазин і має список `products` для зберігання продуктів. Він має метод `AddProduct()`, який додає продукт до магазину, метод `GetTotalProfit()`, який обчислює загальний прибуток магазину, а також реалізує інтерфейс `IEnumerable<ProductInStore>`, щоб дозволити ітерацію по продуктах у магазині.

Клас `StoreIterator` реалізує інтерфейс `IEnumerator<ProductInStore>` і використовується для ітерації по продуктах у магазині. Він зберігає масив продуктів і має методи для переміщення по масиву.

Цей код демонструє використання графічного інтерфейсу з Windows Forms для взаємодії з користувачем, додавання продуктів до магазину та обчислення загального прибутку.

2.4. Приклад та аналіз результатів виконання програми з консольним інтерфейсом

Консольна реалізація на мові C++.

```
Select Microsoft Visual Studio Debug Console

Enter the number of products: 2
Enter product name: Product1
Enter manufacturing date: 01.01.2022
Enter quality certificate: Cert1
Enter cost: 100
Enter markup: 0.2
Enter expiration date: 01.01.2023

Enter product name: Product2
Enter manufacturing date: 01.01.2023
Enter quality certificate: Cert2
Enter cost: 150
Enter markup: 0.2
Enter expiration date: 01.01.2024

Total profit: 300

C:\Users\spaso\Desktop\Курсова\Course_OOP_C++\Debug\Course_OOP_C++.exe (process 19980) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

У програмі також записує файл у форматі txt.

```
Course_OOP.txt: Блокнот
Файл Редагування Формат Вигляд Довідка
Product1 01.01.2022 Cert1 100
Product2 01.01.2023 Cert2 150

Рд 4, ствп 1    100%    Windows (CRLF)    UTF-8
```

2.5. Приклад та аналіз результатів виконання програми з графічним інтерфейсом на Windows Forms

Реалізація програми з графічним інтерфейсом на мові C# з використанням Windows Forms.

У віконному варіанті ми успішно додали перший продукт, який відобразився у listbox.

Form1

Назва: Product1 ADD

Дата виготовлення: 01.01.2022 Calculate

Свідоцтво якості: Cert1 Profit

Вартість: 100

Націнка: 0.2

Термін придатності: 01.01.2023

Product1 01.01.2022 Cert1 100 0.2 01.01.2023

Наступним кроком ми добавили наступний продукт який відобразився також у list box.

Form1

Назва: Product2 [ADD]

Дата виготовлення: 01.01.2023 [Calculate]

Свідectво якості: Cert2

Вартість: 150

Націнка: 0,2

Термін придатності: 01.01.2024

Profit

Product1	01.01.2022	Cert1	100	0,2	01.01.2023
Product2	01.01.2023	Cert2	150	0,2	01.01.2024

Також ми успішно порахували націнку двох продуктів.

Form1

Назва: Product2 [ADD]

Дата виготовлення: 01.01.2023 [Calculate]

Свідectво якості: Cert2

Вартість: 150

Націнка: 0,2

Термін придатності: 01.01.2024

Total Profit: 300

Product1	01.01.2022	Cert1	100	0,2	01.01.2023
Product2	01.01.2023	Cert2	150	0,2	01.01.2024

Висновки

У цій курсовій роботі було розроблено програму для моделювання магазину з використанням об'єктно-орієнтованого підходу. Програма включає різні класи для представлення продуктів, магазину та його функціональності. Класи мають конструктори, методи та властивості для збереження та обробки інформації про продукти, розрахунку прибутку та інші операції.

Основними класами програми є `ProductOnStock`, `ProductInStore` та `Store`. Клас `ProductOnStock` є базовим класом для продукту і містить загальну інформацію про продукт на складі. Клас `ProductInStore` успадковує клас `ProductOnStock` і додає до нього додаткову інформацію про націнку та термін придатності. Клас `Store` представляє магазин і містить список продуктів типу `ProductInStore`.

Програма дозволяє додавати продукти до магазину, обчислювати загальний прибуток магазину та переглядати всі продукти у магазині. Використання об'єктно-орієнтованого підходу дозволяє створити гнучку та розширювану архітектуру програми.

Також було розглянуто можливості розробки графічного інтерфейсу програми на мові C# за допомогою Windows Forms. Графічний інтерфейс дозволяє створити візуально привабливий інтерфейс для програми та полегшити взаємодію користувача з програмою.

У цьому коді важливо звернути увагу на перевірку коректності введених даних користувачем. Якщо користувач введе некоректні дані (наприклад, рядок замість числа), програма може вести себе некоректно або навіть аварійно завершитися. Для надійності слід додати перевірку коректності введених даних та обробку помилок.

Загалом, ця курсова робота демонструє можливості об'єктно-орієнтованого підходу до розробки програми для моделювання магазину. Використання класів, успадкування та інтерфейсів дозволяє створити гнучку та розширювану архітектуру

програми. Також було розглянуто можливості розробки графічного інтерфейсу програми на мові C# за допомогою Windows Forms, що полегшує взаємодію користувача з програмою.

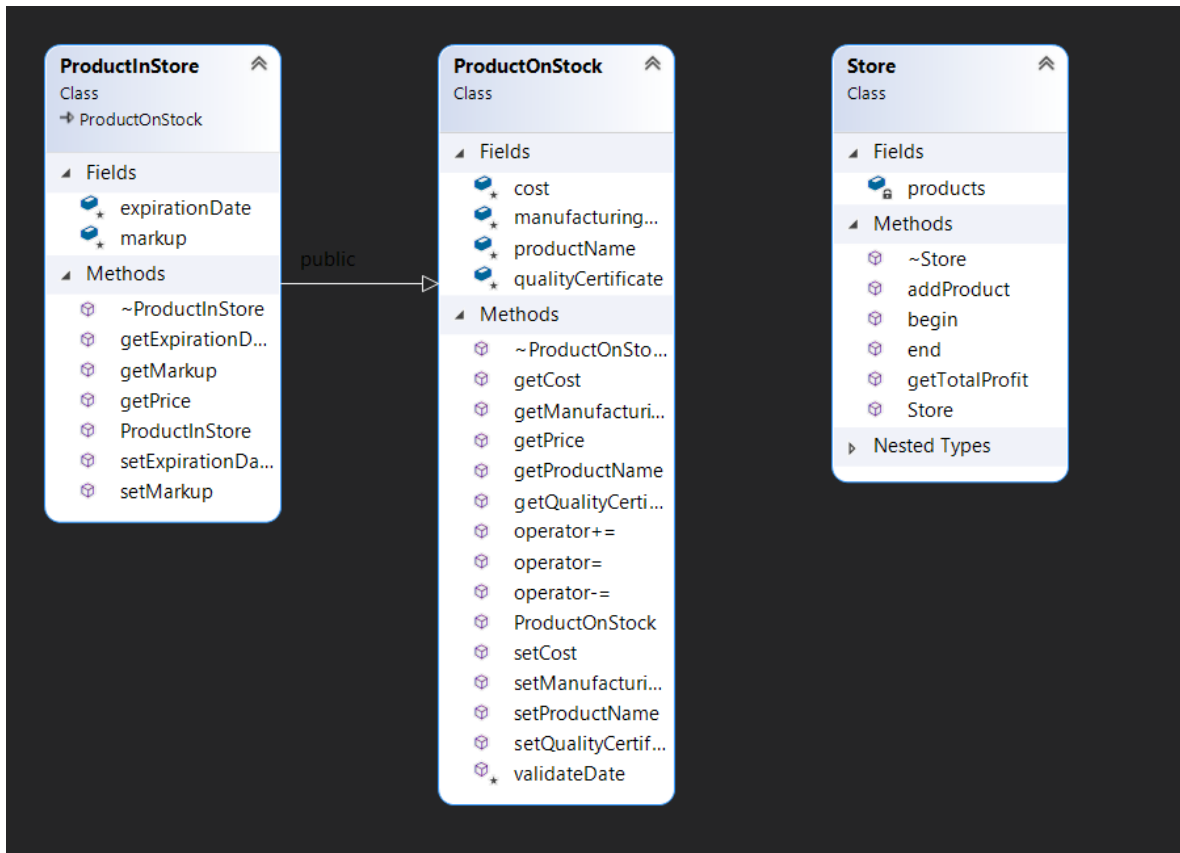
Список використаних джерел

1. Об'єктно-орієнтоване програмування — Вікіпедія (wikipedia.org)
2. C++. Класи. Частина 1. Поняття класу. Оголошення типу даних “клас”. Об'єкт класу. Інкапсуляція даних в класі | BestProg
3. C++. Перевантаження операторів в C++. Операторна функція. Ключове слово operator. Перевантаження базових арифметичних операторів | BestProg
4. Перевантаження операторів — Вікіпедія (wikipedia.org)
5. Контейнер (програмування) — Вікіпедія (wikipedia.org)
6. Введення в ітератори в C++ / Уроки по C++ / aCode
7. Електронний ресурс по C#: <https://www.tutorialspoint.com/csharp/>

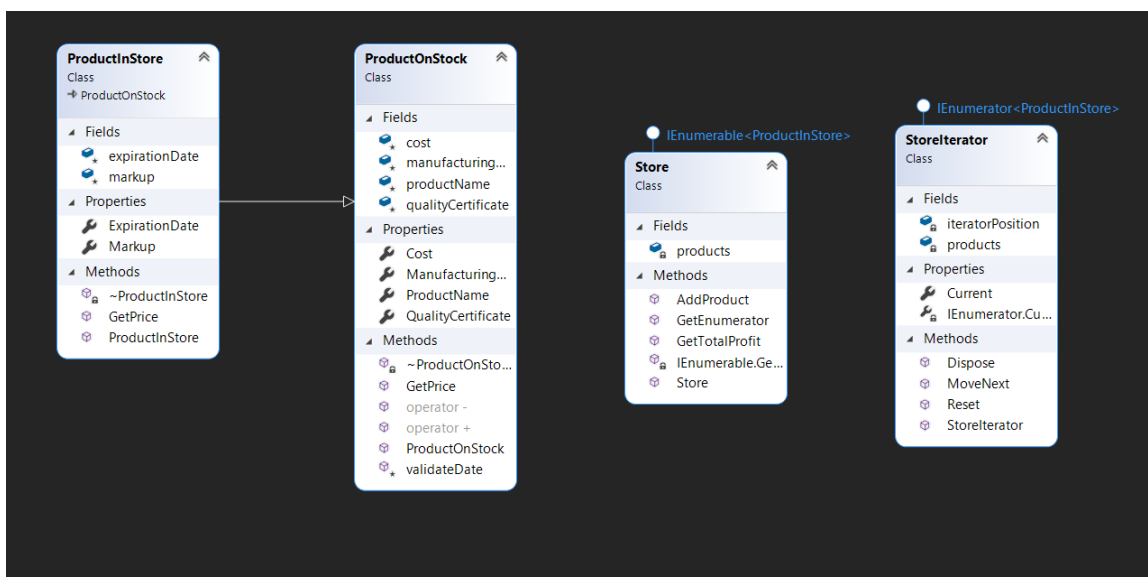
Додатки

Додаток А. Діаграми класів

Діаграми класів консольної програми реалізованої на мові C++.



Діаграми класів програмного додатку з графічним інтерфейсом, які реалізовані на C#.



Додаток Б. Код консольної реалізації програми на мові C++

Розв'язання задачі, в якій реалізовані розглянуті концепції, є багатофайловий проект, що містить файли Course_OOP_C++.cpp, Store.h, Store.cpp, ProductInStore.h, ProductInStore.cpp, ProductOnStock.h, ProductOnStock.cpp:

Course_OOP_C++.cpp

```
#include <iostream>
#include <string>
#include "Store.h"
#include "ProductOnStock.h"
#include "ProductInStore.h"
#include <fstream>

int main() {
    Store store;
    string filename = "Course_OOP.txt";
    ofstream outputFile(filename);

    if (!outputFile.is_open()) {
        cout << "Failed to open the file." << endl;
        return 1;
    }

    int productCount;
    cout << "Enter the number of products: ";
    cin >> productCount;

    for (int i = 0; i < productCount; i++) {
        string name, date, certificate, expirationDate;
        double cost, markup;

        cout << "Enter product name: ";
        cin >> name;
        cout << "Enter manufacturing date: ";
        cin >> date;

        cout << "Enter quality certificate: ";
        cin >> certificate;
        cout << "Enter cost: ";
        cin >> cost;
        cout << "Enter markup: ";
        cin >> markup;
        cout << "Enter expiration date: ";
        cin >> expirationDate;
        cout << endl;

        try {
            ProductInStore product(name, date, certificate, cost, markup, expirationDate);
            store.addProduct(product);
        }
        catch (const exception& ex) {
            cout << "Error: " << ex.what() << endl;
        }
    }

    for (auto& product : store) {
        outputFile << product << endl;
    }
}
```

```

    }
    outputFile.close();

    double totalProfit = store.getTotalProfit();
    cout << "Total profit: " << totalProfit << endl;

    return 0;
}

```

Store.h

```

#pragma once
#include <iostream>
#include <vector>
#include "ProductInStore.h"
class Store {
private:
    vector<ProductInStore> products;
    class StoreIterator
    {
    public:
        using iterator_category = std::forward_iterator_tag;
        using difference_type = std::ptrdiff_t;
        using value_type = ProductInStore;
        using pointer = ProductInStore*;
        using reference = ProductInStore&;

        StoreIterator(pointer ptr) : m_ptr(ptr) {}

        StoreIterator() : m_ptr(nullptr) {}

        reference operator*() const { return *m_ptr; }
        pointer operator->() { return m_ptr; }

        StoreIterator& operator++() { m_ptr++; return *this; }

        StoreIterator operator++(int) { StoreIterator tmp = *this; ++(*this); return tmp; }

        friend bool operator==(const StoreIterator& a, const StoreIterator& b) { return a.m_ptr
== b.m_ptr; };
        friend bool operator!=(const StoreIterator& a, const StoreIterator& b) { return a.m_ptr
!= b.m_ptr; };

    private:
        pointer m_ptr;
    };
public:
    Store();
    ~Store();
    StoreIterator begin();
    StoreIterator end();
    void addProduct(const ProductInStore& product);
    double getTotalProfit();
};

```

Store.cpp

```

#include "Store.h"
Store::Store() {}

Store::~Store() {}

```

```

void Store::addProduct(const ProductInStore& product) {
    products.push_back(product);
}

Store::StoreIterator Store::begin()
{
    if (products.empty()) {
        return StoreIterator();
    }
    return StoreIterator(&products[0]);
}

Store::StoreIterator Store::end()
{
    if (products.empty()) {
        return StoreIterator();
    }
    return StoreIterator(&products[0] + products.size());
}

double Store::getTotalProfit() {
    double profit = 0;
    for (ProductInStore& product : products) {
        profit += product.getPrice();
    }
    return profit;
}

```

ProductInStore.h

```

#pragma once

#include "ProductOnStock.h"

class ProductInStore : public ProductOnStock {
protected:

    double markup;
    string expirationDate;

public:
    ProductInStore(string name, string date, string certificate,
        double cost, double markup, string expirationDate);
    ~ProductInStore();

    void setMarkup(double markup);
    void setExpirationDate(string date);

    double getMarkup() const;
    string getExpirationDate();

    virtual double getPrice() override;
};

```

ProductInStore.cpp

```

#include "ProductInStore.h"

ProductInStore::ProductInStore(string name, string date, string certificate, double cost, double
markup, string expirationDate)

```



```

        : ProductOnStock(name, date, certificate, cost),
        markup(markup) {
            setExpirationDate(expirationDate);
            if (boost::gregorian::from_string(expirationDate) < boost::gregorian::from_string(date)) {
                throw exception("Expiration Date can not be before manufacturing date");
            }
        }
    }

ProductInStore::~~ProductInStore() {}

void ProductInStore::setMarkup(double markup) {
    this->markup = markup;
}

void ProductInStore::setExpirationDate(string date) {
    validateDate(date);
    expirationDate = date;
}

double ProductInStore::getMarkup() const {
    return markup;
}

string ProductInStore::getExpirationDate() {
    return expirationDate;
}

double ProductInStore::getPrice() {
    return cost * (1 + markup);
}

```

ProductOnStock.h

```

#pragma once

#include <iostream>
#include <boost/date_time/gregorian/gregorian.hpp>
using namespace std;

class ProductOnStock {
protected:
    string productName;
    string manufacturingDate;
    string qualityCertificate;
    double cost;
    void validateDate(string date);

public:
    ProductOnStock(string name, string date, string certificate, double cost);
    ~ProductOnStock();

    void setProductName(string name);
    void setManufacturingDate(string date);
    void setQualityCertificate(string certificate);
    void setCost(double cost);

    string getProductName();
    string getManufacturingDate();
    string getQualityCertificate();
    double getCost() const;

    virtual double getPrice();

    ProductOnStock& operator+=(double value);
}

```

```

    ProductOnStock& operator-=(double value);
    ProductOnStock& operator=(const ProductOnStock& other);

    friend istream& operator>>(istream& in, ProductOnStock& product);
    friend ostream& operator<<(ostream& out, const ProductOnStock& product);
};

```

ProductOnStock.cpp

```
#include "ProductOnStock.h"
```

```

void ProductOnStock::validateDate(string date)
{
    boost::gregorian::from_string(date);
}

ProductOnStock::ProductOnStock(string name, string date, string certificate, double cost)
    : productName(name),
    qualityCertificate(certificate),
    cost(cost) {
    setManufacturingDate(date);
}

ProductOnStock::~ProductOnStock() {}

void ProductOnStock::setProductName(string name) {
    productName = name;
}

void ProductOnStock::setManufacturingDate(string date) {
    validateDate(date);
    manufacturingDate = date;
}

void ProductOnStock::setQualityCertificate(string certificate) {
    qualityCertificate = certificate;
}

void ProductOnStock::setCost(double cost) {
    this->cost = cost;
}

string ProductOnStock::getProductName() {
    return productName;
}

string ProductOnStock::getManufacturingDate() {
    return manufacturingDate;
}

string ProductOnStock::getQualityCertificate() {
    return qualityCertificate;
}

double ProductOnStock::getCost() const {
    return cost;
}

double ProductOnStock::getPrice() {
    return cost;
}

```

```

ProductOnStock& ProductOnStock::operator+=(double value) {
    cost += value;
    return *this;
}

ProductOnStock& ProductOnStock::operator-=(double value) {
    cost -= value;
    return *this;
}

ProductOnStock& ProductOnStock::operator=(const ProductOnStock& other) {
    if (this == &other) return *this;
    productName = other.productName;
    manufacturingDate = other.manufacturingDate;
    qualityCertificate = other.qualityCertificate;
    cost = other.cost;
    return *this;
}

istream& operator>>(istream& in, ProductOnStock& product) {
    in >> product.productName >> product.manufacturingDate >> product.qualityCertificate >>
    product.cost;
    return in;
}

ostream& operator<<(ostream& out, const ProductOnStock& product) {
    out << product.productName << " " << product.manufacturingDate << " " <<
    product.qualityCertificate << " " << product.cost;
    return out;
};

```

Додаток В. Код реалізації програми з графічним інтерфейсом з використанням Windows Forms на мові C#

Розв'язання задачі, в якій реалізовані розглянуті концепції, є багатофайловий проект, що містить файли ProductInStore.cs, ProductOnStock.cs, StoreIterator.cs, Store.cs, Form1.cs:

Form1.cs

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Course_OOP_Csharp
{
    public partial class Form1 : Form
    {
        private Store store;
        public Form1()

```

```

    {
        InitializeComponent();
        store = new Store();
    }

    private void addProductButton_Click(object sender, EventArgs e)
    {
        string name = productNameTextBox.Text;
        string date = manufacturingDateTextBox.Text;
        string certificate = qualityCertificateTextBox.Text;
        double cost = double.Parse(costTextBox.Text);
        double markup = double.Parse(markupTextBox.Text);
        string expirationDate = expirationDateTextBox.Text;
        try
        {
            ProductInStore product = new ProductInStore(name, date, certificate, cost,
markup, expirationDate);
            store.AddProduct(product);
            productsListBox.Items.Add(product.ProductName + "\t" + product.ManufacturingDate
+ "\t" + product.QualityCertificate + "\t" + product.Cost + "\t" + product.Markup + "\t" +
product.ExpirationDate);
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
        foreach(ProductInStore product in store)
        { Console.WriteLine(product.ProductName + "\t" + product.ManufacturingDate + "\t" +
product.QualityCertificate + "\t" + product.Cost + "\t" + product.Markup + "\t" +
product.ExpirationDate); }
    }

    private void calculateProfitButton_Click(object sender, EventArgs e)
    {
        double profit = store.GetTotalProfit();
        profitLabel.Text = $"Total Profit: {profit}";
    }
}

```

ProductInStore.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Course_OOP_Csharp
{
    public class ProductInStore : ProductOnStock
    {
        protected double markup;
        protected string expirationDate;

        public ProductInStore(string name, string date, string certificate, double cost, double
markup, string expirationDate)
            : base(name, date, certificate, cost)
        {
            if (DateTime.Parse(expirationDate) < DateTime.Parse(date))
            {
                throw new Exception("Expiration Date can not be before manufacturing date");
            }
            Markup = markup;
        }
    }
}

```

```

        ExpirationDate = expirationDate;
    }
    ~ProductInStore()
    {

    }
    public double Markup
    {
        set
        {
            markup = value;
        }
        get
        {
            return markup;
        }
    }
    public string ExpirationDate
    {
        set
        {
            validateDate(value);
            expirationDate = value;
        }
        get
        {
            return expirationDate;
        }
    }
    public override double GetPrice()
    {
        return cost * (1 + markup);
    }
}
}

```

StoreIterator.cs

```

using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Course_OOP_Csharp
{
    public class StoreIterator : IEnumerator<ProductInStore>
    {
        private ProductInStore[] products;

        private int iteratorPosition = -1;

        public StoreIterator(ProductInStore[] products)
        {
            this.products = products;
        }

        public ProductInStore Current
        {
            get
            {
                return products[iteratorPosition];
            }
        }
    }
}

```

```

    }

    object IEnumerator.Current
    {
        get
        {
            return Current;
        }
    }

    public void Dispose()
    {
    }

    public bool MoveNext()
    {
        iteratorPosition++;
        return iteratorPosition < products.Length;
    }

    public void Reset()
    {
        iteratorPosition = 0;
    }
}
}

```

Store.cs

```

using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Course_OOP_Csharp
{
    public class Store:IEnumerable<ProductInStore>
    {
        private List<ProductInStore> products;
        public Store()
        {
            products = new List<ProductInStore>();
        }
        public void AddProduct(ProductInStore product)
        {
            products.Add(product);
        }

        public IEnumerator<ProductInStore> GetEnumerator()
        {
            return new StoreIterator(products.ToArray());
        }

        public double GetTotalProfit()
        {
            double profit = 0;
            foreach (ProductInStore product in products)
            {
                profit += product.GetPrice();
            }
            return profit;
        }
    }
}

```

```

        IEnumerator IEnumerable.GetEnumerator()
        {
            return new StoreIterator(products.ToArray());
        }
    }
}

```

ProductOnStock.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
namespace Course_OOP_Csharp
{
    public class ProductOnStock
    {
        protected string productName;
        protected string manufacturingDate;
        protected string qualityCertificate;
        protected double cost;
        public ProductOnStock(string name, string date, string certificate, double cost)
        {
            ProductName = name;
            ManufacturingDate = date;
            QualityCertificate = certificate;
            Cost = cost;
        }
        ~ProductOnStock()
        { }

        public string ProductName
        {
            set
            {
                productName = value;
            }
            get
            {
                return productName;
            }
        }
        public string ManufacturingDate
        {
            set
            {
                validateDate(value);
                manufacturingDate = value;
            }
            get
            {
                return manufacturingDate;
            }
        }
        public string QualityCertificate
        {
            set
            {
                qualityCertificate = value;
            }
        }
    }
}

```

```

        get
        {
            return qualityCertificate;
        }
    }
    public double Cost
    {
        set
        {
            cost = value;
        }
        get
        {
            return cost;
        }
    }
    public virtual double GetPrice()
    {
        return cost;
    }
    public static ProductOnStock operator +(ProductOnStock product, double value)
    {
        product.cost += value;
        return product;
    }
    public static ProductOnStock operator -(ProductOnStock product, double value)
    {
        product.cost -= value;
        return product;
    }

    protected void validateDate(string date)
    {
        DateTime.Parse(date);
    }
}
}

```