Міністерство освіти на науки України

Львівський Національний Університет Природокористування

|  |  |  |
| --- | --- | --- |
| Факультет |  | Кафедра |
| механіки, енергетики та ІТ |  | Інформаційних технологій |

***КУРСОВА РОБОТА***

з дисципліни: ***Обʼєктно-орієнтоване програмування***

**на тему:** Розробка системи обліку роздрібної торгівлі мінеральними добривами

Студент гр. ІТ-31 Янкович Данило Ігорович \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_« » 2023.

(ПІБ, підпис, дата)

Керівник:\_\_\_\_\_\_\_\_\_к.т.н.,доцент Татомир А.В. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(наук. ступінь, ПІБ)

**Львівський національний університет природокористування**

Кафедра “Затверджую”

інформаційних систем завідувач кафедри ІСТ

та технологій д.т.н., доцент

\_\_\_\_\_\_\_\_\_\_\_\_ А.М. Тригуба

“\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_20\_\_\_р.

**ЗАВДАННЯ**

на курсову роботу з дисципліни

**“ОБ’ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ”**

Студенту групи Іт-31 Янковичy Данилy Ігоровичy

(прізвище та ініціали)

**1. Тема роботи** “Розробка системи обліку роздрібної торгівлі мінеральними добривами”

**2. Термін здачі студентом закінченої роботи** 27.11.23

**3. Індивідуальне завдання роботи**:

- реалізувати можливість додавання партій мінеральних добрив різних видів та їх продажу;

- виводити інформацію про залишки та обсяги продажу;

- врахувати обмежений термін зберігання добрив.

**4. Перелік питань, які необхідно розробити в роботі:**

*Реферат.* Вступ.

1. АНАЛІЗ СУЧАСНОГО СТАНУ ПИТАННЯ ТА ОБҐРУНТУВАННЯ ЗАВДАННЯ НА РОБОТУ.

1.1. Аналіз предмету проектування.  
1.2. Класифікація об’єктно-орієнтованих мов програмування.  
1.3. Огляд та аналіз сучасних технологій та засобів проектування програмного забезпечення.  
1.4. Універсальна мова проектування UML.

1.5. Уточнена постановка задачі на розробку програмного забезпечення.

2. РОЗРОБКА ЗАГАЛЬНОЇ СТРУКТУРИ ПРОГРАМИ.

2.1. Аналіз функцій системи.

2.2. Розроблення структурної моделі системи.

3. РОЗРОБЛЕННЯ ІЄРАРХІЇ КЛАСІВ.

3.1. Декомпозиція системи.

3.2. Аналіз взаємозв’язків між об’єктами.

3.3. Розроблення інтерфейсів класів.

3.4. UML-діаграма класів.

4. РОЗРОБКА СИСТЕМИ ВВОДУ-ВИВОДУ ДАНИХ.

5. ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.

6. РОЗРОБКА ДОКУМЕНТАЦІЇ ДЛЯ СУПРОВОДЖЕННЯ ПРОГРАМНОГО ПРОДУКТУ.

Висновки.

Бібліографічний список

Додатки

**5. Консультанти з розділів курсової роботи:** А. Татомир

**6. Дата видачі завдання** 15.02.23

**7. Дні і години консультацій**

**8. Завдання на курсову роботу затверджено на засіданні кафедри інформаційних систем і технологій**

**Протокол №\_\_\_ від \_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_р.**

Керівник роботи: А. Татомир

Завдання прийняв до виконання (дата, підпис) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Зміст

[Вступ 1](#_Toc151566388)

[1. АНАЛІЗ СУЧАСНОГО СТАНУ ПИТАННЯ ТА ОБҐРУНТУВАННЯ ЗАВДАННЯ НА РОБОТУ. 2](#_Toc151566389)

[1.1. Аналіз предмету проектування. 2](#_Toc151566390)

[1.2. Класифікація об’єктно-орієнтованих мов програмування 4](#_Toc151566391)

[1.3. Огляд та аналіз сучасних технологій та засобів проектування програмного забезпечення. 6](#_Toc151566392)

[1.4. Універсальна мова проектування UML. 8](#_Toc151566393)

[1.5. Уточнена постановка задачі на розробку програмного забезпечення. 9](#_Toc151566394)

[2. РОЗРОБКА ЗАГАЛЬНОЇ СТРУКТУРИ ПРОГРАМИ 12](#_Toc151566395)

[2.1. Аналіз функцій системи. 12](#_Toc151566396)

[2.2. Розроблення структурної моделі системи. 14](#_Toc151566397)

[3. РОЗРОБЛЕННЯ ІЄРАРХІЇ КЛАСІВ. 16](#_Toc151566398)

[3.1. Декомпозиція системи. 16](#_Toc151566399)

[3.2. Аналіз взаємозв’язків між об’єктами. 17](#_Toc151566400)

[3.3. Розроблення інтерфейсів класів 19](#_Toc151566401)

[3.4. UML-діаграма класів. 20](#_Toc151566402)

[4. РОЗРОБКА СИСТЕМИ ВВОДУ-ВИВОДУ ДАНИХ. 22](#_Toc151566403)

[5. ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ. 24](#_Toc151566404)

[6. РОЗРОБКА ДОКУМЕНТАЦІЇ ДЛЯ СУПРОВОДЖЕННЯ ПРОГРАМНОГО ПРОДУКТУ. 25](#_Toc151566405)

[Висновки. 26](#_Toc151566406)

[Бібліографічний список 27](#_Toc151566407)

[Додаток А 29](#_Toc151566408)

[Додаток Б 31](#_Toc151566409)

[Додаток В 33](#_Toc151566410)

# Вступ

Роздрібна торгівля мінеральними добривами є важливим ланцюгом у сучасному сільському господарстві, визначаючи результативність і продуктивність сільськогосподарських угідь. Щоб забезпечити ефективну управлінську діяльність у цій галузі, виникає необхідність у впровадженні сучасних інформаційних систем, спрямованих на автоматизацію та оптимізацію процесів обліку, продажу та управління запасами мінеральних добрив.

Ця курсова робота присвячена розгляду ключових аспектів розробки системи обліку роздрібної торгівлі мінеральними добривами. Мета дослідження полягає у розкритті та аналізі важливих функцій цієї системи, визначенні вимог до її розвитку, виборі оптимальних технологій та стратегій забезпечення якості. Під час роботи ми також розглянемо вплив системи на довкілля та можливості впровадження екологічно чистих практик у роздрібній торгівлі мінеральними добривами.

Здійснюючи подорож у світ оптимізації та ефективного управління, ми спробуємо створити образ системи, яка відповідає сучасним вимогам та сприяє сталому розвитку галузі роздрібної торгівлі мінеральними добривами.

# 1. АНАЛІЗ СУЧАСНОГО СТАНУ ПИТАННЯ ТА ОБҐРУНТУВАННЯ ЗАВДАННЯ НА РОБОТУ.

Першочерговим завданням є ретельний аналіз поточного стану роздрібної торгівлі мінеральними добривами. Це включає в себе дослідження ринкових тенденцій, конкурентного середовища, законодавчого регулювання та споживчого попиту. Аналіз цих аспектів дозволить визначити ключові проблеми та можливості, з якими стикається галузь.

## 1.1. Аналіз предмету проектування.

Предметом проектування є система обліку роздрібної торгівлі мінеральними добривами, що виступає ключовим елементом в сучасному управлінні підприємствами цієї галузі. Перед початком розробки такої системи, важливо провести докладний аналіз предмету проектування для чіткого визначення функцій, вимог та потреб бізнесу.

Обрання системи обліку обумовлено необхідністю підвищення ефективності роздрібної торгівлі мінеральними добривами. Зростання обсягів операцій та складність управління вимагають автоматизації та оптимізації процесів для забезпечення точності обліку, збільшення продуктивності та вдосконалення стратегічного прийняття рішень.

Система обліку роздрібної торгівлі мінеральними добривами буде включати в себе функції ведення точного обліку залишків на складах, реєстрації операцій купівлі та продажу, а також автоматизацію формування звітів про обсяги продажів, прибуток та залишки. Забезпечення доступу до актуальної інформації та її аналіз дозволить оптимізувати управлінський процес.

Основні функції системи будуть включати в себе:

* Ведення точного обліку залишків та урахування змін цін на ринку.
* Реєстрація та аналіз операцій купівлі та продажу.
* Автоматизація генерації звітів для забезпечення оперативного аналізу даних.

Система має відповідати високим стандартам надійності та ефективності. Вимоги включають в себе використання баз даних для зберігання інформації, розробку інтуїтивно зрозумілого інтерфейсу для користувачів, інтеграцію з іншими інформаційними системами, а також захист від несанкціонованого доступу.

Система повинна враховувати вплив на довкілля та спрямовуватися на впровадження екологічно чистих практик. Зокрема, це може включати зменшення використання паперу через використання електронних звітів та оптимізацію транспортних маршрутів.

Розглядаючи сучасні технології, можливості використання IoT для відстеження обсягів залишків на складах, аналізу даних для прогнозування попиту та використання штучного інтелекту для оптимізації процесів управління можуть значно підвищити ефективність системи.

Загальний аналіз предмету проектування визначає стратегічне завдання розробки системи обліку роздрібної торгівлі мінеральними добривами, яке повинне відповідати потребам бізнесу та забезпечувати стале покращення управлінських процесів.

## 1.2. Класифікація об’єктно-орієнтованих мов програмування

Об’єктно-орієнтовані мови програмування (ООП) використовують концепції об’єктів та класів для організації коду. Класифікація об’єктно-орієнтованих мов програмування включає різні типи мов, які можуть відрізнятися за своєю спрямованістю, філософією, архітектурою та властивостями. Виокремимо кілька основних класифікаційних аспектів:

1. За ступенем підтримки ООП:
   * Повністю об’єктно-орієнтовані мови (Fully Object-Oriented Languages): Вони базуються виключно на концепціях ООП і не мають інших парадигм програмування. Приклади: Smalltalk.
   * Мови з об’єктно-орієнтованими можливостями (Object-Based Languages): Ці мови мають підтримку об’єктів, але можуть також використовувати інші парадигми. Приклади: Python, JavaScript.
2. За рівнем абстракції об’єктів:
   * Мови з класами (Class-Based Languages): Вони використовують класи для створення об’єктів та спадкування між класами. Приклади: Java, C++.
   * Мови з прототипами (Prototype-Based Languages): Вони використовують прототипи об’єктів для створення нових об’єктів. Приклади: JavaScript.
3. За областю використання:
   * Мови для веб-розробки: Наприклад, JavaScript, яке використовується для створення динамічного веб-змісту.
   * Мови для системного програмування: Наприклад, C++, яке використовується для створення оптимізованих системних додатків.
   * Мови для штучного інтелекту: Наприклад, Python, яке широко використовується в області машинного навчання.
4. За підтримкою множинного успадку:
   * Мови з одинарним успадку (Single Inheritance Languages): Вони дозволяють кожному класу мати лише одного батька. Приклади: Java.
   * Мови з багаторівневим успадку (Multiple Inheritance Languages): Вони дозволяють класам мати більше одного батька. Приклади: C++.
5. За підтримкою або відсутністю збірки сміття:
   * Мови з автоматичною збіркою сміття (Languages with Garbage Collection): Вони автоматично видаляють непотрібні об’єкти, спрощуючи управління пам’яттю. Приклади: Java, C#.
   * Мови без автоматичної збірки сміття: Розробник повинен вручну керувати пам'яттю, видаляючи необхідні об'єкти. Приклади: C++.

Ці класифікації надають загальний огляд різних об’єктно-орієнтованих мов програмування та допомагають вибрати мову, яка найкраще відповідає конкретним потребам проекту.

## 1.3. Огляд та аналіз сучасних технологій та засобів проектування програмного забезпечення.

В сучасному інформаційному суспільстві, ефективне та якісне проектування програмного забезпечення вимагає використання передових технологій та засобів. Огляд та аналіз таких інструментів є важливою частиною розробки будь-якого програмного продукту. Розглянемо ключові аспекти цього процесу:

1. Інтегровані середовища розробки (IDE):
   * Огляд: Сучасні IDE, такі як Visual Studio, IntelliJ IDEA, або VS Code, надають розробникам зручне середовище для написання коду, відлагодження, тестування та управління проектами.
   * Аналіз: Вони підтримують широкий спектр мов програмування та мають багатий набір інструментів для автоматизації рутинних завдань.
2. Системи контролю версій:
   * Огляд: Git, Mercurial, SVN - ці системи дозволяють ефективно ведення роботи над кодом в команді та відстеження змін.
   * Аналіз: Git вирізняється розподіленою структурою, галузями (branches), та можливістю працювати без підключення до мережі.
3. Мови програмування:
   * Огляд: Python, JavaScript, Java, C# - ці мови отримали широке визнання у різних областях розробки.
   * Аналіз: Вибір мови залежить від завдань проекту: Python - для швидкості розробки, Java - для масштабованих додатків, JavaScript - для веб-розробки.
4. Фреймворки та бібліотеки:
   * Огляд: Django, React, Spring, .NET - ці фреймворки та бібліотеки полегшують розробку та надають готові рішення для спільних завдань.
   * Аналіз: Використання фреймворків прискорює процес розробки, сприяє стандартизації та забезпечує високу ступінь переносимості коду.
5. Хмарні сервіси:
   * Огляд: AWS, Azure, Google Cloud - ці платформи надають інфраструктуру та сервіси для розгортання, масштабування та керування додатками.
   * Аналіз: Використання хмарних сервісів дозволяє підтримувати високу доступність, забезпечує зручний моніторинг та ефективне використання ресурсів.
6. Методології розробки:
   * Огляд: Agile, Scrum, Kanban - ці методології дозволяють розробляти програмне забезпечення ітеративно та адаптивно.
   * Аналіз: Agile-підходи дозволяють гнучко реагувати на зміни вимог та покращувати якість продукту протягом усього циклу

## 1.4. Універсальна мова проектування UML.

Універсальна мова моделювання (UML) - це стандарт, який використовується для візуалізації, специфікації, побудови та документування програмних систем. UML став важливим інструментом для інженерії програмного забезпечення та визначення відносин між різними елементами великих та складних систем. Розглянемо ключові аспекти універсальної мови проектування UML:

1. Огляд та Історія:
   * Огляд: UML була створена в 1997 році за ініціативи об'єднання лідерів галузі об'єктно-орієнтованого програмування (ООП).
   * Історія: Її розробка була спрямована на об'єднання кращих практик та підходів до моделювання.
2. Основні Елементи UML:
   * Структурні елементи: Класи, об'єкти, компоненти, пакети.
   * Поведінкові елементи: Діаграми активностей, станів, послідовностей та співпраці.
   * Групуючі елементи: Пакети, підсистеми.
3. Види Діаграм:
   * Діаграма класів: Показує структуру системи за допомогою класів, взаємодії та атрибутів.
   * Діаграма послідовності: Відображає взаємодію між об'єктами в конкретний час.
   * Діаграма діяльності: Моделює бізнес-процеси та алгоритми.
4. Використання в Розробці ПЗ:
   * UML дозволяє розробникам, архітекторам та іншим учасникам проекту використовувати спільну мову для визначення, розуміння та спілкування структури та поведінки системи.
   * Забезпечує високий рівень абстракції для великих та складних систем.
5. Інструменти та Середовища:
   * Для створення діаграм UML використовуються спеціалізовані інструменти, такі як Enterprise Architect, Visual Paradigm, Lucidchart, які надають розширені можливості редагування та аналізу моделей.
6. Підтримка Різних Парадигм:
   * UML підтримує об'єктно-орієнтовану, компонентну, структурну та інші парадигми проектування.

UML став необхідним інструментом для розробників програмного забезпечення та архітекторів, який полегшує процес проектування, сприяє зрозумінню та спілкуванню у межах розробкових команд.

## 

## 1.5. Уточнена постановка задачі на розробку програмного забезпечення.

Уточнена постановка задачі на розробку програмного забезпечення (ПЗ) є важливим етапом у процесі розпочатку проекту. Цей документ визначає вимоги та очікування від розробленого програмного продукту, надаючи основну інформацію для команди розробників. В ньому зазвичай містяться такі елементи:

1. Опис Проблеми або Задачі:
   * Зазначте конкретну проблему чи задачу, яку програмне забезпечення має вирішити. Визначте її контекст та важливість для бізнесу чи користувачів.
2. Мета та Об'єктиви:
   * Сформулюйте загальну мету проекту та конкретні об'єктиви, які повинні бути досягнуті. Це може бути полегшення роботи певного процесу, покращення продуктивності чи надання нового функціоналу.
3. Сфера Застосування:
   * Визначте, де та як буде використовуватися розроблене ПЗ. Це може бути внутрішнє використання в організації або продукт для зовнішніх клієнтів.
4. Вимоги до Функціональності:
   * Опишіть конкретні функції, які повинні бути реалізовані в ПЗ. Розділіть їх на основні та додаткові функції для кращого розуміння пріоритетів.
5. Вимоги до Надійності та Безпеки:
   * Зазначте вимоги до надійності, стійкості та безпеки програмного продукту. Це може включати в себе запобігання витокам інформації, забезпечення доступу лише авторизованим користувачам і т.д.
6. Інтерфейси та Взаємодія:
   * Опишіть, як буде виглядати інтерфейс користувача та як користувачі будуть взаємодіяти з програмним продуктом. Розгляньте можливість інтеграції з іншими системами.
7. Вимоги до Використання Ресурсів:
   * Визначте, які ресурси (пам'ять, обчислювальна потужність, мережеві ресурси) повинні бути доступні для програмного забезпечення.
8. Вимоги до Відповідності:
   * Зазначте будь-які нормативи, стандарти чи вимоги до відповідності, якими повинно керуватися програмне забезпечення.
9. Графік Розробки та Етапи Проекту:
   * Визначте очікувані терміни завершення кожного етапу розробки та остаточного продукту.
10. Технічні Вимоги:
    * Зазначте технічні обмеження та вимоги, такі як використання певних технологій, мов програмування, баз даних і т.д.

Цей документ слід виробляти взаємодії з усіма зацікавленими сторонами та слугує важливим інструментом для забезпечення спільного розуміння та виконання вимог проекту.

# 2. РОЗРОБКА ЗАГАЛЬНОЇ СТРУКТУРИ ПРОГРАМИ

Розробка загальної структури програми - це ключовий етап у створенні програмного продукту. Загальна структура визначає організацію програми та взаємодію її компонентів.

## 2.1. Аналіз функцій системи.

Аналіз функцій системи - це процес визначення, розгляду та документування всіх функцій, які має виконувати програмне забезпечення чи інша система. Цей процес включає в себе розгляд бізнес-вимог, потреб користувачів та інших зацікавлених сторін, щоб забезпечити повне та точне зображення функціональності системи. Ось кроки, які можна виконати при аналізі функцій системи:

1. Збір Вимог:
   * Зібрати вимоги від різних зацікавлених сторін, таких як замовники, користувачі, бізнес-аналітики та інші. Це може включати сесії визначення вимог, інтерв'ю з клієнтами та аналіз існуючих документів.
2. Ідентифікація Функцій:
   * Визначити основні функції, які система повинна виконувати. Розділіть їх на основні та додаткові функції для легшого управління пріоритетами.
3. Створення Опису Функцій:
   * Опишіть кожну функцію докладно. Використовуйте структурований підхід, включаючи вхідні та вихідні дані, процеси та умови.
4. Визначення Параметрів Функцій:
   * Деталізуйте вимоги до вхідних та вихідних параметрів для кожної функції. Це може включати формати даних, обсяги, обмеження та інші характеристики.
5. Створення Діаграм Функцій:
   * Використовуйте діаграми, такі як діаграми потоку даних (DFD) або діаграми використання, для візуалізації взаємодії між різними функціями та акторами.
6. Визначення Сценаріїв Використання:
   * Розгляньте різні сценарії використання системи та визначте, які функції будуть активовані в різних ситуаціях.
7. Оцінка Пріоритетів:
   * Встановіть пріоритети для кожної функції на основі їх важливості для бізнес-цілей та потреб користувачів.
8. Визначення Інтеграції:
   * Розгляньте, як функції будуть взаємодіяти між собою та які інтеграції можуть бути необхідні з іншими системами чи сервісами.
9. Документування:
   * Забезпечте документування всіх аспектів аналізу функцій, використовуючи стандартні формати та засоби документації.

Аналіз функцій системи є важливим етапом у розробці програмного продукту та допомагає забезпечити відповідність системи вимогам та очікуванням користувачів.

## 2.2. Розроблення структурної моделі системи.

Структурна модель системи складається з наступних основних компонентів:

1. Клас Batch - представляє окрему партію мінерального добрива з такими атрибутами:
   * name - назва добрива
   * expiration\_date - кінцева дата зберігання
   * quantity - кількість в партії (кг)
   * batch\_number - унікальний номер партії

Цей клас успадковує від базового класу MineralFertilizer методи перевірки терміну придатності is\_expired() та виведення інформації про партію display\_info().

1. Клас Inventory - представляє склад з запасами добрив. Містить:
   * список партій batches
   * список продажів sales

Передбачені методи для:

* + додавання нової партії на склад add\_batch()
  + продажу добрива з вказаної партії sell\_fertilizer()
  + виведення повного складського звіту display\_inventory()

1. Клас RetailSale - зберігає інформацію про окремий продаж:
   * посилання на об'єкт партії fertilizer
   * кількість проданого quantity\_sold
   * дата продажу sale\_date

Передбачено метод виведення інформації про продаж display\_sale\_info()

1. Клас UI - реалізує інтерфейс користувача, взаємодію з оператором.

Така модель дозволяє гнучко зберігати дані про наявні партії добрив, обробляти надходження нових партій та продажі, а також формувати необхідні звіти по складських запасах і продажам.

# 3. РОЗРОБЛЕННЯ ІЄРАРХІЇ КЛАСІВ.

Розроблення ієрархії класів - це важливий етап в об'єктно-орієнтованому програмуванні, де визначаються класи, їхні взаємозв'язки та ієрархічна структура. Цей процес спрямований на створення чіткої та ефективної моделі об'єктів, які будуть використовуватися в програмному продукті.

## 3.1. Декомпозиція системи.

Ось приклад опису декомпозиції системи обліку роздрібної торгівлі мінеральними добривами:

1. Підсистема обліку товарно-матеріальних запасів

Відповідає за облік наявності та руху партій мінеральних добрив на складі. Включає класи:

* + Batch - представлення окремої партії добрив
  + Inventory - складські запаси, список об'єктів Batch

Основні функції:

* + Додавання нової партії на склад
  + Вивід переліку партій з поточними залишками
  + Оновлення залишку партії після продажу

1. Підсистема обліку роздрібних продажів

Відповідає за реєстрацію кожного факту продажу добрива зі складу. Включає класи:

* + RetailSale - дані про окремий продаж
  + Inventory - список продажів

Основні функції:

* + Реєстрація нового продажу
  + Вивід історії продажів

1. Користувацький інтерфейс

Забезпечує взаємодію користувача з системою. Реалізує всі сценарії роботи:

* + Додавання партій
  + Формування звітів
  + Оформлення продажу

Розмежовує бізнес-логіку системи від представлення та обробки даних.

Така декомпозиція дозволяє логічно розділити компоненти системи на слабопов'язані частини для спрощення подальшої розробки та супроводження.

## 3.2. Аналіз взаємозв’язків між об’єктами.

Ось приклад розширеного опису аналізу взаємозв'язків між об'єктами для системи обліку роздрібної торгівлі мінеральними добривами:

Для спроектованої системи можна виділити такі основні об'єкти та зв'язки між ними:

1. Batch - Партія товару
   * Це ключовий об'єкт предметної області, що уособлює партію конкретного мінерального добрива з певними характеристиками. Зберігає дані про назву, кількість, термін придатності тощо.
2. Inventory - Складський облік
   * Об'єкт агрегує в собі дані про всі наявні партії товарів. Відповідає за ведення складських залишків.

Зв'язок між обʼєктами:

* + Агрегація Inventory \*-> \* Batch

1. RetailSale - Роздрібний продаж
   * Представляє факт продажу певної кількості товару з конкретної партії.

Зв'язки:

* + Асоціація Batch -<> \* RetailSale
  + Агрегація Inventory \*-> \* RetailSale

Залежності:

* + RetailSale залежить від Inventory оскільки продаж може бути зареєстрований лише з наявного на складі товару.

Такий аналіз дає чітке уявлення про структуру та взаємовідносини ключових сутностей предметної області. На його основі здійснюється подальше проєктування системи та бази даних.

## 3.3. Розроблення інтерфейсів класів

Ось детальний приклад опису розроблення інтерфейсів класів для системи обліку роздрібної торгівлі мінеральними добривами:

1. Інтерфейс ІBatch

Визначає спільну поведінку для всіх класів, що представляють партію товару.

Методи:

* + getExpirationDate(): DateTime - отримати дату закінчення придатності
  + getBatchNumber(): string - отримати унікальний номер партії
  + changeQuantity(float) - змінити залишок товару в партії

1. Інтерфейс ІSale

Визначає спільну поведінку для класів, пов'язаних з продажем товару.

Методи:

* + registerSale(IBatch, float, DateTime) - зареєструвати факт продажу
  + getSalesReport(): string - сформувати звіт по продажам

1. Клас Batch : IBatch

Реалізує інтерфейс ІBatch.

1. Клас RetailSale
   * fertilizer: IBatch - посилання на партію добрива
   * quantity: float - кількість проданого
   * date: DateTime - дата продажу

Реалізує інтерфейс ІSale.

1. Клас Inventory
   * batches: List<IBatch>
   * sales: List<RetailSale>

Така модель дозволяє розширювати функціонал через єдині інтерфейси та міняти конкретні реалізації класів.

## 3.4. UML-діаграма класів.

UML-діаграма класів - це графічне представлення структури та взаємодії класів в системі програмного забезпечення. Вона надає абстрактний погляд на об'єктно-орієнтовану систему, визначаючи класи, їх атрибути, методи та взаємозв'язки між ними.

PlantUML - це інструмент для створення UML-діаграм з текстового опису, що дозволяє швидко та зручно генерувати діаграми. Основні кроки для створення UML-діаграми класів через PlantUML:

1. Встановлення PlantUML:
   * Завантажимо та встановимо PlantUML, доступний на офіційному веб-сайті.
2. Написання Коду UML:
   * Створимо текстовий файл і напишемо код UML для класів, атрибутів, методів та взаємозв'язків. Використовуємо синтаксис PlantUML для класів.
3. Використання PlantUML для Генерації Діаграми:
   * Запустимо PlantUML і відкриємо свій текстовий файл з кодом. PlantUML автоматично згенерує графічну діаграму на основі нашого опису.
4. Перегляд та Збереження:
   * Перегляньмо згенеровану діаграму та вносимо необхідні корективи. Потім збережемо готову діаграму у форматі зображення або іншому необхідному форматі.

PlantUML дозволяє простим та зрозумілим способом візуалізувати структуру класів, полів, методів та їх взаємозв'язки, полегшуючи розуміння архітектури програмного забезпечення.

Ось приклад опису UML-діаграми класів для системи обліку роздрібної торгівлі мінеральними добривами:

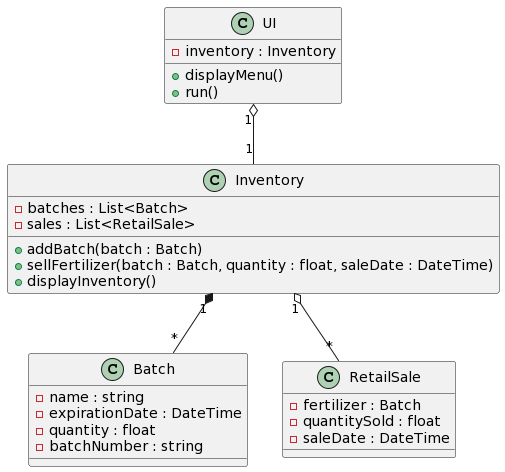


Рисунок 3.1 UML-діаграма класів

Така UML-діаграма дозволяє наочно представити структуру класів системи та зв’язки між ними — ієрархію, асоціації, агрегації, що полегшує розуміння її архітектури.

# 4. РОЗРОБКА СИСТЕМИ ВВОДУ-ВИВОДУ ДАНИХ.

**ВВЕДЕННЯ ДАНИХ**

Введення даних реалізовано в класі UI, в методі run().

Код можна побачити в «Додатку А»

Користувачу виводиться меню вибору дій. Для введення інформації про нову партію товару передбачено пункт меню 1. Додати партію. Користувач вводить з консолі такі дані для створення об'єкту класу Batch:

- Назва добрива

- Кінцева дата зберігання

- Кількість в партії

- Номер партії

Дані валідуються. Наприклад, перевіряється коректність введеної дати. Після цього створюється об'єкт Batch і додається до складських запасів Inventory.

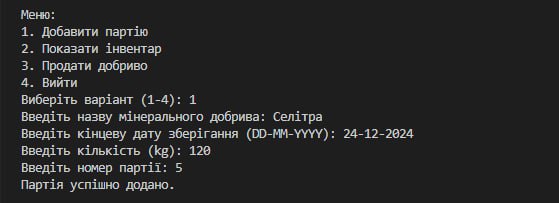


Рисунок 4.1 Робота програми – введення інформації

**ВИВЕДЕННЯ ДАНИХ**

Передбачено такі способи виведення даних:

1. Перегляд поточного складського інвентарю та історії продажів. Здійснюється за пунктом меню 2. Виводиться інформація про всі наявні партії на складі та інформація з усіх зареєстрованих продажів: дата, кількість, назва добрива.

2. Вивід повідомлення про результат операції (додавання партії, продажу).

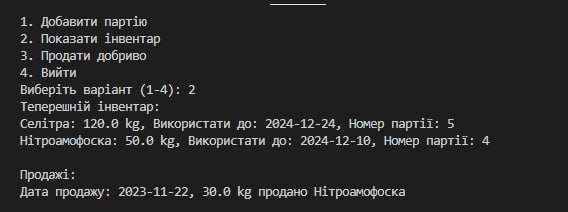


Рисунок 4.2 Робота програми – виведення інформації

Таким чином реалізовано введення всіх необхідних даних і вивід актуальної облікової інформації для користувача системи.

# 5. ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.

Тестування програмного забезпечення - це важливий етап у життєвому циклі розробки програм, який спрямований на перевірку його коректності, ефективності та відповідності вимогам. Цей процес допомагає виявити помилки та проблеми ще до випуску програми в експлуатацію.

Для тестування нашого додатку було проведено мануальне тестування.

Ось опис покрокового мануального тестування для системи обліку роздрібного продажу добрив:

1. Перевірка запуску та базової функціональності
   * Запускаємо головний файл main.py
   * Перевіряємо вивід меню користувача на екран
   * Обираємо пункт меню, перевіряємо реакцію системи
2. Тестування додавання нової партії
   * Обираємо п.1 Додати партію
   * Вводимо дані нової партії (назва, кількість, дата, номер)
   * Перевіряємо появу повідомлення про успішне додавання
   * Перевіряємо оновлення складського обліку (п.2 меню)
3. Перевірка продажу з існуючої партії
   * За п.3 продаємо частину товару з раніше створеної партії
   * Перевіряємо коректне списання заданої кількості
   * Перевіряємо реєстрацію продажу (п.2)
4. Спроба продати більше, ніж є на складі, та обробка помилки
   * Виконуємо спробу продати більшу кількість
   * Перевіряємо вивід повідомлення про недостатній залишок
   * Перевіряємо, що дані на складі не змінились

# 6. РОЗРОБКА ДОКУМЕНТАЦІЇ ДЛЯ СУПРОВОДЖЕННЯ ПРОГРАМНОГО ПРОДУКТУ.

Розробка документації для супроводження програмного продукту є важливим етапом у життєвому циклі розробки програми. Ці документи допомагають забезпечити ефективне супроводження та управління продуктом під час його експлуатації. Ось приклад інструкції зі встановлення для інформаційної системи обліку роздрібної торгівлі мінеральними добривами:

1. Передумови. Переконайтеся, що на комп'ютері встановлено:
   * Операційна система Windows 7 або новіша
   * Python версії 3.6 або новіший
   * СУБД MySQL версії 5.7 або новішої
2. Встановлення
   * Завантажте архів з кодом програми та розпакуйте його
   * Відкрийте консоль та перейдіть у папку з програмою
   * Встановіть залежності: *pip install -r requirements.txt*
3. Запуск. Для запуску програми виконайте в консолі: *python main.py*
4. Після чого можна працювати з програмою за допомогою консольного інтерфейсу.

Забезпечення повної та зрозумілої документації сприяє ефективному супроводженню та використанню програмного продукту, як користувачами, так і технічним персоналом.

# Висновки.

Під час розробки цього додатку було проведено аналіз сучасного стану питання, визначено завдання на роботу та проведено докладний огляд сучасних технологій та інструментів для проектування програмного забезпечення. Була вивчена універсальна мова проектування UML та ії застосування для створення моделей системи. В рамках проекту розроблена загальна структура програми, проведений аналіз функцій системи та створена ієрархія класів.

Також була проведена декомпозиція системи та аналіз взаємозв'язків між об'єктами. Були розроблені інтерфейси класів та створена UML-діаграма класів для кращого розуміння структури системи.

Після цього була уточнена постановка задачі на розробку програмного забезпечення та написані тести для перевірки модуля на завантаження музики. Завершальним етапом була розробка документації для супроводження програмного продукту, яка включала в себе технічну документацію, інструкції користувача.

Всі ці етапи спрямовані на створення високоякісного програмного продукту, який задовольнить потреби користувачів та буде легко супроводжуватися та розвиватися в майбутньому. Розробка програмного забезпечення є процесом, що вимагає уваги до деталей, досліджень та системного підходу, і цей проект не є винятком.

Нехай цей додаток стане корисним та піднесе радість людям, які будуть ним користуватися.

# Бібліографічний список

1. Павленко М. М. Інформаційні системи і технології в обліку : навчальний посібник / М. М. Павленко. – К. : Центр учбової літератури, 2018. – 252 с.
2. Глинський Я. М. Інформаційні системи в управлінні : навчальний посібник / Я. М. Глинський. – Львів : Растр-7, 2017. – 140 с.
3. Гордієнко І. В. Інформаційні системи і технології в управлінні організаціями: конспект лекцій / І. В. Гордієнко, М. М. Колосова, С. В. Карцов. – Харків : ХНУРЕ, 2018. – 124 с.
4. Ткаченко А. М. Інформаційні системи та технології в обліку : опорний конспект лекцій / А. М. Ткаченко. – Київ : КНУТД, 2016. – 76 с.
5. Інформаційні системи і технології в економіці : посібник для студентів вищих навчальних закладів / за ред.: В. С. Пономаренка. – К. : Академвидав, 2010. – 544 с.
6. Черемисина Н. В. Інформаційні системи та технології в управлінні проектами : конспект лекцій / Н. В Черемисина. – Харків : ХНАДУ, 2015. – 108 с.
7. Мезенцев К. М. Автоматизовані інформаційні системи бухгалтерського обліку, контролю, аналізу та аудиту : навчальний посібник / К. М. Мезенцев. – Київ : Центр учбової літератури, 2020. – 288 с.
8. Литвин В. В. Проектування баз даних : навчальний посібник / В. В. Литвин. – Львів : Магнолія 2006, 2018. – 208 с.
9. Гужва В. М. Інформаційні системи і технології на підприємствах : навчальний посібник / В. М. Гужва. – Київ : КНТЕУ, 2001. – 400 с.
10. Бурячок В. Л. Сучасні інформаційні технології в управлінні підприємницькою діяльністю : навчальний посібник / В. Л. Бурячок, О. В. Мельников. – Київ : Кондор, 2015. – 242 с.

# Додаток А

Код програми

*Main.py*

from **ui** import **UI**

*# Початок виконання програми*

if \_\_name\_\_ == "\_\_main\_\_":

ui = **UI**()

ui.**run**()

*UI.py*

from **batch** import \*

from **inventory** import **Inventory**

class **UI**():

def **\_\_init\_\_**(self) -> None:

self.inventory = **Inventory**()

*# Функція для виводу меню і отримання вибору користувача*

def **display\_menu**(self):

**print**("\nМеню:")

**print**("1. Добавити партію")

**print**("2. Показати інвентар")

**print**("3. Продати добриво")

**print**("4. Вийти")

return **input**("Виберіть варіант (1-4): ")

def **run**(self):

while True:

choice = self.**display\_menu**()

if choice == "1":

name = **input**("Введіть назву мінерального добрива: ")

expiration\_date = **datetime**.**strptime**(**input**("Введіть кінцеву дату зберігання (DD-MM-YYYY): "), '%d-%m-%Y')

quantity = **float**(**input**("Введіть кількість (kg): "))

batch\_number = **input**("Введіть номер партії: ")

batch = **Batch**(name, expiration\_date, quantity, batch\_number)

self.inventory.**add\_batch**(batch)

**print**("Партія успішно додано.")

elif choice == "2":

self.inventory.**display\_inventory**()

elif choice == "3":

batch\_number = **input**("Введіть номер партії для продажу: ")

quantity\_sold = **float**(**input**("Введіть кількість, яку плануєте продати (kg): "))

sale\_date = **datetime**.**now**().**strftime**("%Y-%m-%d %H:%M:%S")

for batch in self.inventory.batches:

if batch.batch\_number == batch\_number:

self.inventory.**sell\_fertilizer**(batch, quantity\_sold, sale\_date)

break

else:

**print**("Партія не знайдена.")

elif choice == "4":

**print**("Завершення програми.")

break

else:

**print**("Неправильний вибір. Виберіть між 1-4.")

# Додаток Б

Код роботи з БД

*database.py*

import **sqlite3**

from **batch** import **Batch**

from **retail\_sale** import **RetailSale**

from **datetime** import **datetime**

class **Database**:

def **\_\_init\_\_**(self):

self.**create\_tables**()

def **create\_tables**(self):

connection = **sqlite3**.**connect**('inventory.db')

c = connection.**cursor**()

c.**execute**('''

CREATE TABLE IF NOT EXISTS batches

(id INTEGER PRIMARY KEY, name TEXT, expiration\_date TEXT,

quantity REAL, batch\_number TEXT)

''')

c.**execute**('''

CREATE TABLE IF NOT EXISTS sales

(id INTEGER PRIMARY KEY, batch\_id INTEGER, quantity\_sold REAL,

sale\_date TEXT, FOREIGN KEY(batch\_id) REFERENCES batches(id))

''')

connection.**commit**()

def **post\_sql\_query**(func):

def **wrapper**(\*args, \*\*kwargs):

with **sqlite3**.**connect**('inventory.db') as connection:

cursor = connection.**cursor**()

try:

res = func(\*(args + (cursor,)), \*\*kwargs)

return res

except **Exception** as ex:

**print**(ex)

result = cursor.**fetchall**()

return result

return **wrapper**

**@post\_sql\_query**

def **add\_batch**(self, batch, c):

c.execute("INSERT INTO batches (name, expiration\_date, quantity, batch\_number) VALUES (?, ?, ?, ?)", (batch.name, batch.expiration\_date, batch.quantity, batch.batch\_number))

**@post\_sql\_query**

def **register\_sale**(self, batch, quantity\_sold, sale\_date, c):

c.execute("INSERT INTO sales (batch\_id, quantity\_sold, sale\_date) VALUES (?, ?, ?)", (batch.batch\_number, quantity\_sold, sale\_date))

c.execute("UPDATE batches SET quantity= quantity - ? WHERE batch\_number=?", (quantity\_sold, batch.batch\_number))

**@post\_sql\_query**

def **load\_batches**(self, c):

c.execute("SELECT \* FROM batches")

batches = []

for row in c:

batch = **Batch**(row[1], **datetime**.**strptime**(row[2], "%Y-%m-%d %H:%M:%S"), row[3], row[4])

batches.**append**(batch)

return batches

**@post\_sql\_query**

def **get\_batch\_by\_id**(self, bid, c):

c.execute("SELECT \* FROM batches WHERE batch\_number=?", (bid,))

row = c.fetchone()

batch = **Batch**(row[1], **datetime**.**strptime**(row[2], "%Y-%m-%d %H:%M:%S"), row[3], row[4])

return batch

**@post\_sql\_query**

def **load\_sales**(self, c):

c.execute("SELECT \* FROM sales")

sales = []

for row in c:

batch = self.**get\_batch\_by\_id**(row[1])

sale = **RetailSale**(batch, row[2], **datetime**.**strptime**(row[3], "%Y-%m-%d %H:%M:%S"))

sales.**append**(sale)

return sales

# Додаток В

Класи програми

*inventory.py*

from **retail\_sale** import **RetailSale**

from **database** import **Database**

class **Inventory**:

def **\_\_init\_\_**(self):

self.db = **Database**()

self.batches = self.db.**load\_batches**()

self.sales = self.db.**load\_sales**()

def **add\_batch**(self, batch):

self.batches.append(batch)

self.db.**add\_batch**(batch)

def **sell\_fertilizer**(self, batch, quantity\_sold, sale\_date):

if batch.quantity >= quantity\_sold:

batch.quantity -= quantity\_sold

sale = **RetailSale**(batch, quantity\_sold, sale\_date)

self.db.**register\_sale**(batch, quantity\_sold, sale\_date)

self.sales.append(sale)

**print**(f"{quantity\_sold} kg {batch.name} успішно продано.")

else:

**print**("Не достатньо добрива для продажі.")

def **display\_inventory**(self):

**print**("Теперешній інвентар:")

for batch in self.batches:

batch.display\_info()

**print**("\nПродажі:")

for sale in self.sales:

sale.display\_sale\_info()

*retail\_sale.py*

class **RetailSale**:

def **\_\_init\_\_**(self, fertilizer, quantity\_sold, sale\_date):

self.fertilizer = fertilizer

self.quantity\_sold = quantity\_sold

self.sale\_date = sale\_date

def **display\_sale\_info**(self):

**print**(f"Дата продажу: {self.sale\_date.strftime('%Y-%m-%d')}, {self.quantity\_sold} kg продано {self.fertilizer.name}")

*batch.py*

from **datetime** import **datetime**

class **MineralFertilizer**:

def **\_\_init\_\_**(self, name, expiration\_date, quantity):

self.name = name

self.expiration\_date = expiration\_date

self.quantity = quantity

def **is\_expired**(self):

return **datetime**.**now**() **>** self.expiration\_date

def **display\_info**(self):

**print**(f"{self.name}: {self.quantity} kg, Використати до: {self.expiration\_date.strftime('%Y-%m-%d')}, Номер партії: {self.batch\_number}")

class **Batch**(**MineralFertilizer**):

def **\_\_init\_\_**(self, name, expiration\_date, quantity, batch\_number):

**super**().**\_\_init\_\_**(name, expiration\_date, quantity)

self.batch\_number = batch\_number