**Міністерство освіти і науки України**

**Львівський національний університет Природокористування**

**факультет механіки, енергетики та інформаційних технологій**

**кафедра інформаційних технологій**

**Курсова робота**

із дисципліни «Об’єктно-орієнтоване програмування»

на тему:**«Розробка автоматизованої системи обліку роботи тваринницького підприємства»**

Виконав: студент 3 курсу групи Іт-31

Спеціальності 126 «Інформаційні системи та технології»

(шифр і назва)

Бунга В. Р.

(Прізвище та ініціали)

Керівник: к.т.н., професор Татомир А.В.

(Прізвище та ініціали)

**ДУБЛЯНИ-2023**

Львівський національний аграрний університет

# Кафедра

Інформаційних систем та технологій

# Кафедра

Інформаційних систем та технологій

# Кафедра

Інформаційних систем та технологій

# Кафедра

Інформаційних систем та технологій

# Кафедра

Інформаційних систем та технологій

# Кафедра

Інформаційних систем та технологій

# Кафедра

Інформаційних систем та технологій

# Кафедра

Інформаційних систем та технологій

**“Затверджую”**

завідувач кафедри ІСТ

д.т.н., в.о. проф.

\_\_\_\_\_\_\_\_\_\_\_\_\_А.М. Тригуба

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

**“Затверджую”**

завідувач кафедри ІСТ

д.т.н., в.о. проф.

\_\_\_\_\_\_\_\_\_\_\_\_\_А.М. Тригуба

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

**“Затверджую”**

завідувач кафедри ІСТ

д.т.н., в.о. проф.

\_\_\_\_\_\_\_\_\_\_\_\_\_А.М. Тригуба

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

**“Затверджую”**

завідувач кафедри ІСТ

д.т.н., в.о. проф.

\_\_\_\_\_\_\_\_\_\_\_\_\_А.М. Тригуба

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

**“Затверджую”**

завідувач кафедри ІСТ

д.т.н., в.о. проф.

\_\_\_\_\_\_\_\_\_\_\_\_\_А.М. Тригуба

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

**“Затверджую”**

завідувач кафедри ІСТ

д.т.н., в.о. проф.

\_\_\_\_\_\_\_\_\_\_\_\_\_А.М. Тригуба

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

**“Затверджую”**

завідувач кафедри ІСТ

д.т.н., в.о. проф.

\_\_\_\_\_\_\_\_\_\_\_\_\_А.М. Тригуба

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

**“Затверджую”**

завідувач кафедри ІСТ

д.т.н., в.о. проф.

\_\_\_\_\_\_\_\_\_\_\_\_\_А.М. Тригуба

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

**ЗАВДАННЯ**

на курсову роботу з дисципліни

***«***Об’єктно-орієнтоване програмування***»***

#### Студенту групи Iт-31 Бунзі В. Р.

*(прізвище та ініціали)*

**1. Тема роботи** «Розробка автоматизованої системи обліку роботи тваринницького підприємства»

**2. Термін здачі студентом закінченої роботи 26.05.23**

**3. Індивідуальне завдання роботи**:

- реалізувати систему обліку персноналу та тварин таким чином, щоби відображати розподіл відповідальності працівників за окремих тварин;

-передбачити можливість вводити показники ефективності кожної тварини;

-виводити основні метрики по фермі загалом та по персоналу;

**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) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# ВСТУП

ООП (об'єктно-орієнтоване прогpамування) — це парадигма програмування, яка дозволяє організовувати програми як набори взаємодіючих об'єктів. Це дозволяє розбивати складні завдання на менші, легші для розуміння частини, які можна впpоваджувати, тестувати та підтримувати .ООП спрощує розробку великих проектів за рахунок зменшення складності коду, підвищення гнучкості та зручності обслуговування програмного забезпечення та створення готових блоків для багаторазового використання. Загалом ООП дозволяє програмістам писати чистіший, структурований код, який можна підтримувати, забезпечуючи високу якість та ефективність програмного забезпечення. ООП може зайняти більше часу для написання коду на ранніх стадіях розробки,але це інвестується в час і зусилля для підтримки коду з часом. ООП також дозволяє створювати більш безпечні програмні продукти, оскільки дизайн базується на розподілі обов'язків та концепції захисту даних. Це допомагає запобігти помилкам і вразливостям,які можуть призвести до серйозних проблем у майбутньому.

Розвиток автоматизованих систем обліку в тваринницькій галузі є важливим питанням для оптимізації виробничих процесів і підвищення ефективності управління. Посилення ринкової конкуренції змушує тваринницькі підприємства вдосконалювати системи управління для забезпечення високої якості продукції та отримання максимального прибутку. Автоматизовані системи обліку на тваринницьких підприємствах дозволяють ефективно вести облік персоналу і тварин, контролювати виробничі процеси і забезпечувати моніторинг ефективності управління. Це дозволяє швидко і точно приймати рішення, знижує ризик виробничих помилок і підвищує ефективність використання ресурсів. Буде розглянуто процес розробки автоматизованої системи обліку для тваринницького підприємства. Буде детально проаналізовано вимоги до системи, дизайн та функціональну розробку, включаючи введення обліку персоналу та тварин, введення показників продуктивності для кожної тварини та виведення ключових показників для ферми в цілому та для персоналу. Також будуть детально розглянуті основні принципи об'єктно-орієнтованого програмування і те, як вони можуть бути використані для розробки ефективної системи обліку.

# РОЗДІЛ 1.

# АНАЛІЗ СУЧАСНОГО СТАНУ ПИТАННЯ ТА ОБГРУНТУВАННЯ ЗАВДАННЯ НА РОБОТУ

* 1. **Аналіз предмету проектування**

За даними Продовольчої та сільськогосподарської організації ООН (ФАО), виробництво м'яса зросте на 1,3% до 337 мільйонів тонн у 2020 році. Очікується подальше зростання споживчого попиту на м'ясо. У цьому контексті ефективне управління тваринницькими підприємствами стане надзвичайно важливим. Згідно з дослідженням MarketsandMarkets, світовий ринок систем автоматизації в сільському господарстві також очікується на рівні 5,5 млрд доларів США до 2023 року. Це свідчить про те, що тваринницькі підприємства все більше зацікавлені у використанні новітніх технологій для підвищення ефективності та збільшення прибутковості. Розробка автоматизованих систем обліку на тваринницьких підприємствах є одним із таких сучасних технологічних рішень, яке може підвищити ефективність роботи підприємств та збільшити їхні прибутки.

На сьогоднішній день багато тваринницьких підприємств ведуть ручний облік роботи та стану тварин. Це призвело до низки проблем, серед яких помилки в обліку, втрата даних, недостатній контроль за роботою персоналу та загальною ефективністю роботи тваринницьких підприємств. З іншого боку, автоматизація обліку роботи та стану тварин на тваринницьких підприємствах значно просунулася вперед. Існує низка програмних продуктів, які забезпечують автоматизований облік роботи та стану тварин. Такі системи не лише використовують бирки, чіпи та індивідуальні номери для ведення обліку тварин, але й збирають та аналізують дані про різні аспекти діяльності тваринницького підприємства. Однак не всі програмні продукти є ефективними і забезпечують необхідний рівень контролю та аналізу даних. Існують також проблеми інтеграції з іншими системами, такими як фінансовий та кадровий облік. Тому розробка власної автоматизованої системи обліку роботи та умов утримання тварин може бути корисним рішенням для тваринницьких підприємств, що дозволить точніше контролювати процеси та ефективніше використовувати ресурси на підприємстві. Сьогодні багато тваринницьких підприємств все ще ведуть облік роботи та умов утримання тварин вручну. Це спричиняє різноманітні проблеми, включаючи помилки в обліку, втрату даних, погане оперативне управління персоналом та загальну ефективність роботи тваринницьких підприємств. З іншого боку, автоматизація обліку роботи та стану тварин на тваринницьких підприємствах є досить розвиненою. Існує низка програмних продуктів, які забезпечують автоматизований облік роботи та стану тварин. Такі системи не тільки використовують бирки, чіпи та індивідуальні номери для ведення обліку тварин, але й збирають та аналізують дані про різні аспекти діяльності тваринницького підприємства. Однак не всі програмні продукти є ефективними і забезпечують необхідний рівень контролю та аналізу даних. Існують також проблеми інтеграції з іншими системами, такими як фінансовий та кадровий облік. Тому розробка власної автоматизованої системи обліку завдань та умов утримання тварин може бути корисним рішенням для тваринницьких підприємств, що дозволить точніше контролювати процеси та ефективніше використовувати ресурси на підприємстві.

Розробка автоматизованої системи обліку праці на тваринницьких підприємствах, яка має на меті спростити та оптимізувати процес ведення обліку даних про тварин та персонал. Для розуміння суті та обсягу цієї роботи необхідно проаналізувати існуючі процеси обліку праці на тваринницьких підприємствах та існуючі системи, що використовуються для цієї мети. Такий аналіз може виявити недоліки та проблеми з існуючою системою, а також можливості для покращення та вдосконалення процесу. Крім того, слід визначити основні потреби та вимоги, які повинна задовольняти система, що розробляється, зокрема збір та аналіз даних про тварин і персонал, зазначення розподілу відповідальності працівників за окремих тварин, введення показників продуктивності для кожної тварини та виведення ключових показників для всієї ферми.

Таким чином, аналізуючи сучасний стан автоматизованих систем обліку тваринницьких підприємств, можна дійти висновку, що більшість підприємств використовують застарілі методи обліку, що може призвести до помилок і затримок під час отримання необхідної інформації. Розробка автоматизованої системи обліку тваринницького підприємства розв'язала б ці проблеми і дала б змогу точніше та швидше вести облік персоналу та тварин, а також відображати основні показники по всій фермі та персоналу. Така система не тільки підвищить ефективність і прибутковість підприємства, а й сприятиме його розвитку.

* 1. **Класифікація об’єктно-орієнтованих мов програмування.**

Об'єктно-орієнтоване програмування (ООП) - це підхід до розробки програмного забезпечення, при якому реальний світ моделюється у вигляді об'єктів, які взаємодіють між собою для виконання різних завдань. ООП включає в себе ряд понять, таких як класи, об'єкти, успадкування, інкапсуляція і поліморфізм. В ООП клас - це шаблон для створення об'єктів, які містять властивості та методи для маніпулювання даними. Об'єкти є екземплярами класу і можуть існувати та взаємодіяти незалежно один від одного. Спадкування дозволяє створювати нові класи на основі існуючого класу, який успадковує властивості та методи від суперкласу. Інкапсуляція дозволяє приховати деталі реалізації від користувача, в той час як поліморфізм дозволяє використовувати одні й ті ж методи і властивості в різних класах. ООП є більш гнучким, а код - багаторазовим, оскільки об'єкти можна створювати, редагувати і повторно використовувати багато разів у різних контекстах. В результаті ООП стало основним методом розробки програмного забезпечення і використовується в багатьох сферах, включаючи веб-розробку, мобільну розробку та розробку ігор.

Об'єктно-орієнтоване програмування (ООП) - це підхід до розробки програмного забезпечення, заснований на концепції об'єктів, які об'єднують дані та пов'язані з ними операції. Об'єктно-орієнтовані мови програмування - це мови програмування, які підтримують цей підхід. Класифікація об'єктно-орієнтованих мов програмування зазвичай базується на чотирьох основних поколіннях

* Об'єктно-орієнтовані мови програмування першого покоління: Мова програмування Smalltalk, яка з'явилася в 1970-х роках і була першою мовою програмування, яка повністю використовувала підхід ООП.
* Мови програмування ООП другого покоління: C++, розроблена у 1980-х роках як розширення мови C, що дозволяє розробникам використовувати об'єктно-орієнтовані конструкції.
* Третє покоління ООП мов програмування: Java та Python, створені в 1990-х роках; ці мови призначені для розробки крос-платформного програмного забезпечення і мають вбудовану підтримку ООП.
* Мови ООП четвертого покоління: Це Ruby, C# та Swift, які з'явилися у 2000-х роках. Ці мови програмування є продовженням попереднього покоління і мають вищий рівень абстракції, що спрощує процес програмування та полегшує розробку складних програмних систем.

Щоб краще зрозуміти класифікацію об'єктно-орієнтованих мов програмування, давайте розглянемо деякі конкретні приклади мов з кожної класифікації:

* Клас-орієнтовані мови Java, C#, Python
* Прототипно-орієнтовані мови: JavaScript, Lua, Self
* Об'єктно-орієнтовані мови з множинною спадковістю: C++, Perl, Eiffel
* Об'єктно-орієнтовані мови з можливістю делегування: Smalltalk, Objective-C, Ruby
* Об'єктно-орієнтовані мови з комбінованим успадкуванням і делегуванням: Scala, Python (версія 3), Kotlin.

Кожна з цих категорій мов має свої унікальні особливості та переваги, залежно від потреб програміста та характеру проекту. Наприклад, клас-орієнтовані мови, такі як Java та C#, мають сувору типізацію та велику бібліотечну базу і дозволяють розробляти корпоративні додатки; прототипно-орієнтовані мови, такі як JavaScript, дозволяють швидко розробляти прототипи та прості додатки об'єктно-орієнтовані мови з поєднанням успадкування та делегування, такі як Scala, дозволяють писати більш гнучкі та складні додатки. Яку мову програмування обрати для конкретного проекту, залежить від низки факторів, зокрема від характеру проекту, вимог до швидкості та продуктивності, наявності розробників з необхідними навичками та доступності бібліотек.

Роберт Мартін у книзі «Чистий код : створення, аналіз та рефакторинг» писав «Об'єктно-орієнтований дизайн є процесом знаходження правильних способів декомпозиції системи на зв'язкові та єдиноцільні класи. І це не мистецтво, а наука. Це досить точна наука, з логічними правилами, обмеженнями та прогресом. Якщо дизайн нашої системи ООП неправильний, ми платимо високу ціну за це. Так само, як ми платимо за погану архітектуру.»

* 1. **Огляд та аналіз сучасних технології та засобів проектування програмного забезпечення.**

Сучасний стан інструментів та технологій проектування програмного забезпечення є дуже добрим і постійно розвивається. Зокрема, багато інструментів та технологій проектування програмного забезпечення зосереджені на тому, щоб зробити розробку програмного забезпечення ефективнішою, безпечнішою та простішою.

Однак із впровадженням нових технологій та інструментів розробки програмного забезпечення з'явилися нові виклики та проблеми. Наприклад, стає дедалі важче забезпечити безпеку програмного забезпечення, оскільки злочинці постійно вдосконалюють свої методи атак.

Крім того, значною проблемою залишається забезпечення якості програмного забезпечення. З цієї причини багато компаній використовують автоматизоване тестування та інші методи для забезпечення якості програмного забезпечення. Загалом, стан інструментів та технологій проектування програмного забезпечення є дуже добрим і постійно покращується, але вони потребують постійного оновлення та вдосконалення для забезпечення ефективності та безпеки розробки програмного забезпечення.

Існує низка інструментів для проектування програмного забезпечення, які допомагають розробникам створювати додатки швидше та ефективніше. До найпоширеніших інструментів проектування програмного забезпечення належать:

* UML (Unified Modelling Language - уніфікована мова моделювання): стандартна мова моделювання, яка дозволяє розробникам створювати діаграми, що описують різні аспекти програми, такі як структура, функціональність і процеси.
* ERD (Entity-Relationship Diagrams) - тип діаграм, що дозволяє розробникам описувати зв'язки між сутностями в базі даних.
* CASE-системи (Computer Aided Software Engineering) - інтегроване середовище розробки дозволяє розробникам виконувати різні етапи проектування програмного забезпечення, включаючи аналіз вимог, проектування системи, розробку та тестування.
* IDE (Integrated Development Environments) - інтегровані середовища розробки, які дозволяють розробникам створювати, тестувати та налагоджувати додатки у зручному для користувача інтерфейсі.
* Редактори коду - програми, такі як Sublime Text, Visual Studio Code та Notepad++, що дозволяють розробникам писати та редагувати програмний код.
* Системи контролю версій - системи контролю версій, такі як Git, що дозволяють розробникам зберігати та керувати версіями коду, співпрацювати з іншими розробниками та вирішувати конфліктні зміни.
* Системи відстеження помилок - системи відстеження помилок, такі як JIRA, дозволяють розробникам відстежувати та управляти помилками в додатках.

Ці інструменти проектування програмного забезпечення допомагають розробникам зосередитися на різних аспектах розробки, скорочуючи час і зусилля, необхідні для створення додатків, і забезпечуючи якість та ефективність виконання завдань з розробки програмного забезпечення.

Метою та завданням сучасних методів та інструментів проектування програмного забезпечення є забезпечення ефективної та безпечної pозробки програмного забезпечення, скорочення часу та витрат на розробку, покращення якості та функціональності програмного забезпечення, а також забезпечення простоти обслуговування та масштабованості. Основними цілями є

* Підвищення ефективності: сучасні методи та інструменти проектування програмного забезпечення дозволяють розробникам більш ефективно використовувати ресурси та скорочувати час розробки додатків.
* Забезпечення безпеки: є найважливішою метою сучасних методів та інструментів проектування програмного забезпечення. Забезпечення безпеки означає зменшення можливості скомпрометувати програмне забезпечення, зменшення можливості скомпрометувати ресурси, встановлення належного контролю доступу та захист даних.
* Якість є ще однією важливою метою. Використання сучасних методів та інструментів розробки програмного забезпечення може зменшити кількість помилок, спростити процес тестування та підвищити якість програмного забезпечення.
* Масштабованість: використання сучасних методів та інструментів проектування програмного забезпечення забезпечує масштабованість програмного забезпечення для задоволення зростаючих потреб користувачів.
* Простота підтримки: забезпечення простоти підтримки є ще однією метою. Сучасні програмні технології та інструменти проектування дозволяють легко орієнтуватися в програмному забезпеченні.
  1. **Універсальна мова проектування UML.**

Уніфікована мова моделювання (UML) - це стандартизована мова моделювання програмного забезпечення, яка використовується для візуального представлення дизайну програмного забезпечення. UML є дуже популярною технікою проектування програмного забезпечення і є необхідною для більшості розробників програмного забезпечення. UML містить різні типи діаграм, які використовуються для моделювання різних аспектів програмного забезпечення, таких як структура, поведінка та взаємодія системи з іншими системами. Ці діаграми включають діаграми класів, діаграми послідовності, діаграми діяльності та інші. UML також має деякі переваги, такі як спрощення процесу проектування програмного забезпечення, підвищення якості та ефективності, а також покращення комунікації між розробниками та клієнтами. Одним з основних принципів UML є принцип моделювання системи як набору інтерактивних об'єктів, що забезпечує максимальну гнучкість і масштабованість системи.

Крім того, UML підтримує методи програмування на основі класів, що дозволяє розробляти більш компактний і структурований код.Таким чином, UML є потужним і популярним інструментом моделювання програмного забезпечення, який може значно спростити процес проектування та підвищити якість і ефективність розробки.

UML також включає в себе інструменти для формулювання вимог до програмного забезпечення, моделювання архітектури програмних систем, проектування взаємодії між об'єктами, визначення структур класів, діаграм станів, діаграм послідовності та діаграм зв'язку. Однією з головних переваг UML є те, що це універсальна мова, яку можна використовувати для всіх типів проектування програмного забезпечення. UML також дозволяє розробникам програмного забезпечення легко спілкуватися один з одним, використовуючи спільну мову, зменшуючи кількість помилок і покращуючи якість продукту. UML є потужним інструментом для проектування програмного забезпечення, але для початківців він може бути складним у вивченні та розумінні. Однак, маючи достатній досвід і підготовку, UML може бути дуже корисною для проектування програмного забезпечення будь-якого рівня складності.

Фаза проектування UML може відрізнятися в залежності від методології розробки програмного забезпечення, але, як правило, включає в себе наступні етапи :

* Визначення вимог до програмного забезпечення: На цьому етапі визначаються вимоги до програми, яку потрібно розробити. Цей етап може включати аналіз бізнес-процесів і збір вимог від клієнтів та інших зацікавлених сторін.
* Створення моделі вимог: На цьому етапі створюється модель вимог до програмного забезпечення, яка описує, що потрібно реалізувати в додатку.
* Аналіз та проектування архітектури: На цьому етапі аналізуються вимоги до програми та розробляється архітектура програми. У цьому процесі визначаються основні компоненти програми та їх взаємодія.
* Розробка моделей: На цьому етапі розробляються різні моделі програми для опису різних аспектів програми. Наприклад, розробляються моделі класів, послідовності виконання та діаграми діяльності.
* Генерація коду: на основі розроблених моделей генерується вихідний код програми. Цей етап можна автоматизувати за допомогою спеціальних інструментів.
* Тестування: після того, як код згенеровано, він тестується на відповідність вимогам та на наявність помилок. Розгортання: після успішного тестування програмне забезпечення розгортається у виробничому середовищі.
* Підтримка та модифікації: після розгортання програмне забезпечення може потребувати підтримки та модифікацій.

* 1. **Уточнена постановка задачі на розроблення програмного забезпечення.**

Ідея проєкту полягає в тому, щоб створити програмне забезпечення, яке допоможе волонтерам орієнтуватися у їхній діяльності.

Для створення додатку для обліку волонтерської діяльності можна врахувати наступні пункти:

* Опрацювання завдання розробки автоматизованої системи обліку праці в галузі тваринництва передбачає наступні кроки
* Аналіз потреб користувачів: необхідно з'ясувати, які потреби мають користувачі (менеджери, працівники підприємства), які процеси повинна обробляти система, які функції повинні використовуватися тощо.
* Визначення функціональних вимог: необхідно визначити, які функції повинна виконувати система, які дані вона повинна зберігати та обробляти.
* Визначення нефункціональних вимог Необхідно визначити нефункціональні вимоги, які необхідно враховувати при розробці системи, такі як швидкість, безпека та надійність.
* Розробка архітектури системи Необхідно розробити архітектуру системи, наприклад, які компоненти системи повинні бути розроблені, як вони повинні взаємодіяти і які інтерфейси вони повинні мати.
* Реалізація системи: цей етап передбачає розробку та тестування коду системи.
* Введення системи в експлуатацію Після успішного тестування система вводиться в експлуатацію і стає доступною для користувачів.

# РОЗДІЛ 2.

# РОЗРОБКА ЗАГАЛЬНОЇ СТРУКТУРИ ПРОГРАМИ

**2.1 Аналіз функцій системи.**

Функціональний аналіз системи - це процес вивчення функцій, які повинна виконувати система, і розуміння взаємозв'язків між ними. Цей аналіз допомагає розробляти високоякісні, ефективні системи, які відповідають вимогам користувачів і бізнесу.Аналіз функціональності системи є важливим кроком у розробці та впровадженні будь-якої інформаційної системи. Цей аналіз допомагає зрозуміти, які функції повинні виконуватися, щоб система була ефективною і відповідала потребам користувачів і бізнесу.

Система обліку роботи тваринницького підприємства має на меті забезпечити ефективне ведення даних . Для досягнення цієї мети система має наступні функції : додавання даних про певну особу(її — ід , ім’я ,з якими тваринами працює, скільки приросто молока, яєць з цих тварин, в якому саме відділі вони працюють), видалення даних по ід з баз даних, перегляд, сортування.

Нижче вказаний детальніший розбір функцій:

Додавання даних:

1. Вибираємо в головному меню « Додати нові дані»;
2. В наступному меню вибираємо групу тварин, дані яких хочемо добавити;
3. Після цього вводимо : ід , ім’я , якщо вибрали корову ,то вводимо коди корів , якщо курей , то кількість курей за яких ви відповідаєте.

Видалення даних:

1. В головному меню вибираємо «Видалення даних»;
2. В наступному меню вибираємо групу тварин, з якої потрібно видалити дані про певного співробітника.

Перегляд даних:

1. В головному меню вибираємо «Переглянути дані»;
2. В наступному меню вибираємо групу тварин, дані яких хочете переглянути.

**2.2 Розроблення структурної моделі системи.**

Розробка моделі структури системи - це створення схематичного представлення компонентів системи та їх взаємозв'язків. Модель використовується для розуміння функціонування системи, підвищення ефективності та зручності використання, а також для детального опису компонентів. Процес розробки моделі структури системи складається з наступних кроків

Основна мета облікової системи тваринницького підприємства - забезпечити ефективне управління даними. Для цього система повинна мати такі компоненти

Компонент "Додавання даних про працівників":

* Функція "Додати нового працівника". Поля "Ідентифікатор", "Ім'я", "Прізвище", "Посада", "Відділ".
* Функція "Додати тварину до співробітника".
* Поля: ідентифікатор тварини, тип тварини (наприклад, корова, курка), кількість тварин, відділ обслуговування.

Компонент "Видалення даних про працівника".

* Функція "Видалити працівника"
* Поле Ідентифікатор працівника
* Функція "Видалити тварину у працівника"
* Поле Ідентифікатор тварини, ідентифікатор працівника.

Компонент "Перегляд даних співробітників":

* Функція "Переглянути список співробітників".
* Поля Ідентифікатор, Ім'я, Прізвище, Посада, Відділ.
* Функція "Переглянути список тварин співробітників".
* Поле: ідентифікатор тварини, тип тварини, кількість тварин, відділ обслуговування.
* Функція "Відобразити загальну інформацію ".
* Поля Літри молока, кількість яєць.

# РОЗДІЛ 3.

# РОЗРОБЛЕННЯ ІЄРАРХІЇ КЛАСІВ

**3.1 Декопозиція системи**

Декомпозиція системи - це процес розбиття великої системи на менші компоненти для полегшення її аналізу та управління. Цей процес дозволяє розбити складні системи на менші, простіші компоненти, кожен з яких може бути проаналізований і керований окремо. Декомпозиція системи передбачає розкладання її на підсистеми, модулі, функції та процеси. Декомпозиція зменшує складність системи, забезпечує більшу стабільність та ефективність роботи системи, а також сприяє більш ефективному використанню ресурсів.

SOLID - це абревіатура, що означає п'ять принципів об'єктно-орієнтованого проектування програмного забезпечення. Ці принципи були сформульовані Робертом Мартіном.

Ось п'ять принципів SOLID:

* Принцип єдиної відповідальності (Single Responsibility Principle, SRP) - клас повинен мати лише одну причину для змін. Наприклад, клас, що відповідає за роботу з базою даних, повинен мати можливість лише зберігати та отримувати дані, але не повинен мати можливості обробляти дані або виводити їх на екран. Щоб досягти цього, можна створити окремі класи, які відповідають за обробку та відображення даних. Таким чином, два різні класи будуть виконувати тільки свої власні завдання, а контексти, які вони використовують, будуть окремими.
* Принцип відкритості/закритості (Open/Closed Principle, OCP) - програмні об'єкти повинні бути відкритими для розширення, але закритими для змін. Оскільки зміни в існуючих класах можуть мати непередбачувані наслідки у програмі, ця політика допомагає зберегти та розширити код. Замість цього ви можете створити новий клас або інтерфейс, який успадковує функціональність існуючого класу, і розширити його. Наприклад, якщо у вас є клас, який працює з різними типами платежів, ви можете розширити його, створивши новий клас, який успадкує функціональність батьківського класу і додасть нові типи платежів. Таким чином, ви можете додати нову функціональність до вашого додатку, не змінюючи вихідний код базового класу і не впливаючи на поведінку інших частин програми.
* Принцип заміщення Ліскова (LSP) - об'єкти підкласів можуть замінювати свої батьківські класи без зміни основної функціональності програми. Це означає, що в програмному коді об'єкти базового класу можуть бути замінені об'єктами підкласу без зміни функціональності програми. Крім того, підкласи не повинні додавати нові передумови або зменшувати постумови базового класу, оскільки це може призвести до некоректної поведінки програми. Наприклад, якщо базовий клас має метод, який повертає площу фігури, успадкований клас не повинен повертати від'ємне значення або взагалі не повертати площу. Це було б порушенням принципу LSP. Цей принцип допомагає створювати більш гнучкі та масштабовані програми, які легше модифікувати та розширювати.
* Принцип розділення інтерфейсів (ISP) - клієнти не повинні залежати від інтерфейсів, які вони не використовують. Цей принцип передбачає створення більшої кількості інтерфейсів з меншою кількістю методів. Він також передбачає, що класи повинні бути прив'язані до абстракції, а не до конкретної реалізації. Це дозволяє використовувати різні реалізації для однієї і тієї ж функціональності, надаючи програмному забезпеченню більшої гнучкості.
* Принцип інверсії залежностей (Dependency Inversion Principle, DIP) - модулі верхнього рівня не залежать від модулів нижнього рівня, і обидва типи модулів залежать від абстракцій. Іншими словами, DIP вимагає, щоб програмний код був прив'язаний до абстракції, а не до конкретної реалізації. Це дозволяє змінювати реалізацію без зміни модуля верхнього рівня, який використовує цю реалізацію. Щоб бути сумісним з DIP, ви повинні визначити абстракцію, яка визначає функціональність модуля, і створити залежність від цієї абстракції.

Система обліку роботи тваринницького підприємства буде містити :

* Клас «DataBase» , який буде містити методи для запиту до бази даних, додавання, видалення та перегляд даних;
* Клас «Animal», який буде містити дані про персону , яка працює з певними видами тваринами. Містить атрибути id , name , much( кількість надою молока, або знесених яєць), та номер відділу в якому працює дана людина;
* Клас «Cow», який буде наслідувати вище вказані атрибути , також додається новий атрибут «a\_name» , де будуть міститись коди корів ,з якими працює людина;
* Клас «Chicken» , який буде наслідувати вище вказані атрибути , також додається новий атрибут «a\_name1» , де буде міститись кількість курей , з якими працює людина.
* Клас «UserInterface» , який буде містити методи для вводу та виводу даних зібраних в базі даних.

**3.2** **Аналіз взаємозв’язків між об’єктами**

Аналіз взаємозв'язків між об'єктами - це процес визначення того, як об'єкти в системі взаємодіють один з одним. Цей аналіз може виявити необхідність покращення дизайну системи, зокрема зменшення залежностей між об'єктами, підвищення незалежності та модульності, а також підвищення ефективності взаємодії між об'єктами.

Аналіз взаємозв'язків між об'єктами є важливою частиною процесу проектування програмного забезпечення. Цей аналіз допомагає зрозуміти, які об'єкти взаємодіють, як вони обмінюються даними і як вони підтримують функціональність системи. Одним із способів аналізу взаємозв'язків між об'єктами є створення діаграми класів. Діаграма класів показує, які класи (об'єкти) в системі впливають один на одного і які частини обмінюються даними. Це допомагає визначити, які об'єкти потребують вдосконалення та оптимізації. Аналіз взаємозв'язків також дозволяє виявити та усунути залежності між об'єктами, які погіршують функціональність та зручність використання системи. Наприклад, якщо в системі є залежність між класом A і класом B, зміна класу A може вплинути на клас B. Це може викликати проблеми при внесенні змін і погіршити ремонтопридатність системи. Реляційний аналіз може виявити такі залежності та зменшити їх. Тому аналіз взаємозв'язків між об'єктами є важливим кроком у процесі розробки програмного забезпечення для покращення функціональності та зручності використання системи, а також для зменшення залежностей між об'єктами та збільшення їх незалежності.

Містить два класи: Database та UserInterface Клас UserInterface додає нові записи до бази даних, отримує всі записи з бази даних, використовуючи об'єкти класу Database, а клас UserInterface використовує об'єкти класу Database для додавання нових записів до бази даних та отримання всіх записів з бази даних. Об'єкти класів Cow і Chicken передаються методам класу Database і зберігаються в базі даних; у класах Cow і Chicken реалізовано метод from\_input(), що дозволяє користувачеві вводити дані про нову людину відповідальну за певною групою тварин за допомогою методу From\_input(). Клас UserInterface також містить метод для відображення всіх записів про корів та курчат, що зберігаються в базі даних. Функція main виконує нескінченний цикл, поки користувач не вирішить вийти з програми. Користувач може додати нові записи про корів або курчат, переглянути всі записи про корів або курчат або вийти з програми. Зверніть увагу, що метод програми close закриває з'єднання з базою даних шляхом виклику методу close\_database() класу UserInterface. Таким чином, між класами встановлюється залежність, де клас UserInterface залежить від класу Database для доступу до бази даних, а класи Cow і Chicken використовуються методами класу Database для зберігання даних про тварин .

**3.3** **Розроблення інтерфейсів класів**

Розробка інтерфейсу класу - це процес проектування програмного забезпечення, який передбачає створення загального інтерфейсу для класу, що визначає, як клас взаємодіє з іншими класами та зовнішніми компонентами. При розробці інтерфейсів класів важливо, щоб вони були зрозумілими і простими у використанні. Інтерфейси класів повинні бути достатньо простими, щоб їх могли легко зрозуміти і використовувати інші розробники. Інтерфейс класу повинен бути добре продуманий і враховувати вимоги всіх користувачів. У той же час, важливо не визначати занадто багато методів і атрибутів, які можуть змінитися в майбутньому. Краще визначити лише ті методи та атрибути, які дійсно необхідні та корисні. Інтерфейс класу також повинен бути стійким до змін. Це означає, що якщо внутрішня реалізація класу змінюється, його інтерфейси повинні залишатися незмінними. Це гарантує збереження сумісності з іншими компонентами системи, які використовують клас. Також важливо надати документацію для інтерфейсу класу, включаючи опис кожного методу та атрибуту, параметрів, значень, що повертаються, та можливих винятків.

Загалом, розробка інтерфейсів класів є важливим етапом проектування програмного забезпечення і може гарантувати, що класи в системі використовуються належним чином і ефективно.

Клас «DataBase» містить в собі методи , які надсилають запит до бази даних . Запити які містить клас :

* Запит на додавання нових даних;
* Запит на перегляд всієї інформації;
* Запит на видалення даниз з таблиці баз даних.

Клас «UserInterface» містить методи :

* Додавати дані;
* Переглядати дані;
* Видаляти дані.

**3.4** **UML-діаграма класів.**

UML (Unified Modelling Language) - це мова моделювання, що використовується для візуального представлення програмних систем; UML містить набір графічних символів і правил для створення діаграм, які можна використовувати для опису різних аспектів програмного проекту (наприклад, архітектури, поведінки, функціональності, взаємодії між компонентами системи тощо). UML - це стандарт, розроблений спільнотою програмістів і прийнятий міжнародними організаціями, такими як OMG (Object Management Group). Він дозволяє командам програмістів уникати неправильного розуміння технічних специфікацій і забезпечує чіткість у спілкуванні між різними людьми та командами. Діаграми UML використовуються для моделювання різних елементів програмної системи, таких як класи, об'єкти, компоненти, взаємодії та поведінки. Кожен тип діаграм має свої особливості, які можна знайти в документації UML, і діаграми UML можна використовувати на різних етапах життєвого циклу програмного продукту, таких як аналіз вимог, проектування та тестування.

Діаграма класів - це тип діаграми UML, яка показує структуру системи або додатку у вигляді класів, їх атрибутів, методів і взаємодій між класами. Діаграми класів допомагають візуалізувати основні компоненти системи та їх залежності один від одного. На діаграмі класів класи зображуються у вигляді прямокутників з назвою класу в центрі. У центрі прямокутника можна вказати атрибути класу, такі як ім'я, тип і значення за замовчуванням. Методи класу відображаються як функції з іменем, параметрами і типом повернення. Діаграми класів також показують взаємодію між класами, наприклад, залежності, об'єднання, композиції та успадкування. Залежності показують, що клас залежить від іншого класу, але не має з ним прямого зв'язку. Агрегація та композиція показують зв'язок між частиною та цілим, тоді як композиція - це сильніший зв'язок, в якому частина не може існувати окремо від цілого. Успадкування показує зв'язок між батьківським і дочірнім класами, де дочірній клас успадковує атрибути і методи батьківського класу. Діаграми класів є потужним інструментом для розробки програмного забезпечення, оскільки вони допомагають візуалізувати та зрозуміти структуру системи та залежності між компонентами.

Клас «Animal» містить такі атрибути:

* id :integer;
* name: string;
* much: integer;
* dep\_number: integer;

Клас «Chicken» та «Cow» наслідують всі атрибути від класу «Animal», та один додатковий атрибут a\_name.

Клас «Database» містить в собі операції :

* add\_chicken
* add\_cow
* get\_all\_chickens
* get\_all\_cows
* remove\_chicken
* remove\_cow

# РОЗДІЛ 4.

# РОЗРОБКА СИСТЕМИ ВВОДУ-ВИВОДУ ДАНИХ

# Розробка систем введення/виведення даних - це процес створення програмного забезпечення, яке дозволяє користувачам взаємодіяти з базами даних та іншими системами за допомогою користувацького інтерфейсу. Системи введення/виведення даних дозволяють користувачам вводити дані в бази даних, читати і змінювати існуючі дані, а також отримувати звіти та іншу інформацію.

# При розробці такої системи можна виділити наступні етапи Визначення вимог:

# визначення функціональних і нефункціональних вимог до системи введення/виведення даних.

# Проектування інтерфейсу користувача: створення інтерфейсу, який дозволяє користувачам взаємодіяти з базою даних та іншими системами.

# Проектування бази даних: створення бази даних, яка буде використовуватися системою вводу/виводу.

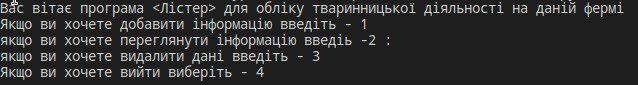
# Розробка програмного забезпечення: розробка програмного забезпечення, яке буде взаємодіяти з базою даних і забезпечувати функціональність системи вводу/виводу.

# Тестування: Перевірка функціональності та надійності системи вводу/виводу.

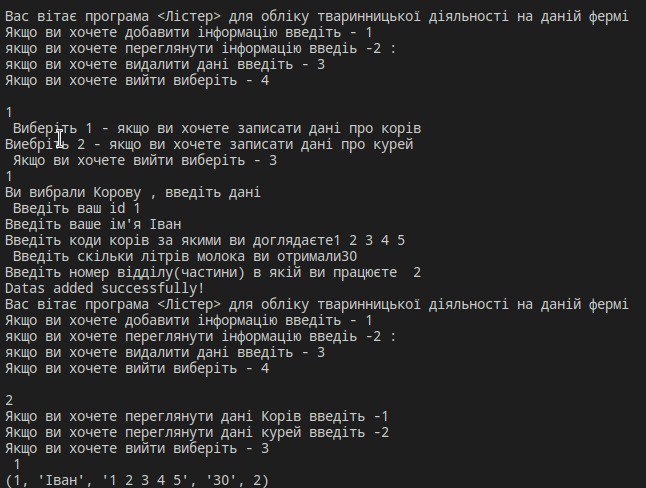
# Випуск та підтримка: випуск програмного забезпечення та надання підтримки користувачам.

# При розробці систем вводу/виводу даних важливо пам'ятати, що необхідно дотримуватися принципів безпеки та конфіденційності даних. Для цього можна використовувати різні технології, такі як шифрування даних та механізми автентифікації користувачів.

Стартовий екран, який ми бачимо при запуску показано на рис. 4.1.Тут ми можемо вибрати один із 4 варіантів: перегляд даних, додавання даних, видалення даних та вихід з програми.

Рис. 4.1 Стартовий екран

Для навігації в даній програмі використовуються цифри, при вводі певної цифри відкривається наступне меню. При виборі «2», відкривається меню зображене на (рис 4.2). Тут ми можемо переглянути дані певної групи тварин.

Рис. 4.2 Меню «Перегляд даних»

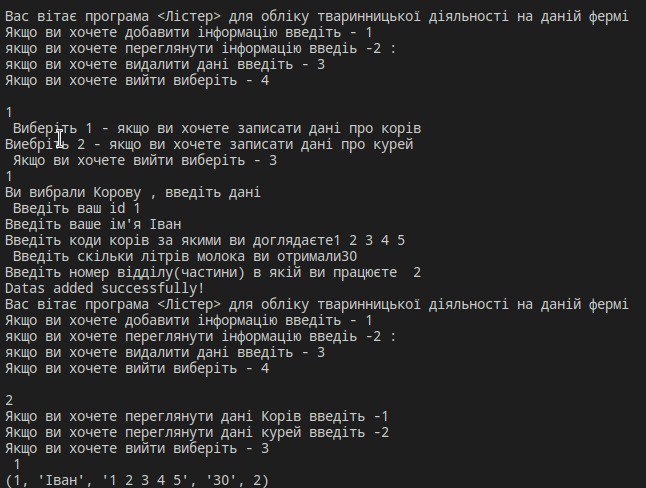
Після того як ми переглянули дані, виходимо з даного та вибираємо наступний пункт , якщо ми хочемо добавити дані вводимо «1»,після цього відкривається меню з вибором групи тварин, після цього вибираємо групу , яку нам потрібно та вводимо дані , приклад зображено на (рис 4.3).

Рис. 4.3 Меню «Додавання даних»

Після того як ми переглянули дані, виходимо з даного та вибираємо наступний пункт, якщо ми хоче видалити вибираємо «3», відкривється меню зображене на (рис 4.4). Вибираємо групу , після цього вводимо «id» користувача ,дані якого ми хочемо видалити.

Рис 4.4. Меню пункту «Видалити волонтера»

Розробка системи вводу-виводу даних є важливою складовою будь-якої інформаційної системи. Її основна мета полягає в забезпеченні ефективного взаємодії між користувачем та системою, що дозволяє збирати, обробляти та зберігати дані відповідно до потреб користувача.

# РОЗДІЛ 5.

# ТЕСТУВАННЯ ПРОГРАМНОГО

# ЗАБЕЗПЕЧЕННЯ

# Тестування програмного забезпечення - це процес перевірки, верифікації та валідації програмних продуктів для забезпечення їхньої якості та належного функціонування. Основною метою тестування програмного забезпечення є виявлення помилок і дефектів у програмному продукті до того, як його буде введено в експлуатацію. Це забезпечує якість продукту, знижує ризик збоїв, підвищує надійність і задовольняє потреби користувачів.

# Тестування програмного забезпечення можна класифікувати за різними критеріями, такими як :

# За метою: функціональне, негативне, регресійне, інтеграційне, приймальне, навантажувальне, безпекове, інше

# За методом: ручне, автоматичне, напівавтоматичне, комбіноване, інше

# За обсягом: тестування "чорного ящика", тестування "білого ящика", тестування "сірого ящика".

# За фазою розробки: модульне тестування, інтеграційне тестування, системне тестування, приймальне тестування.

# Тестування програмного забезпечення може бути дуже складним процесом і вимагає висококваліфікованих тестувальників, ефективної організації роботи, відповідних інструментів і методологій. Однак добре організоване тестування допомагає забезпечити високу якість і надійність програмних продуктів.

# Основні етапи тестування програмного забезпечення зазвичай включають наступні види діяльності

# Планування тестування: Цей етап включає аналіз вимог до продукту, вибір методів та інструментів тестування, створення тестової документації та планування часових рамок.

# Підготовка тестових даних: Цей етап включає створення тестових скриптів і наборів тестових даних, які будуть використовуватися під час тестування.

# Виконання тестів: Цей етап включає виконання тестових сценаріїв і наборів тестових даних для перевірки функціональності, надійності та безпеки програмного продукту.

# Аналіз результатів тестування: Після завершення тестування результати аналізуються та документуються. Це допомагає виявити помилки та дефекти, які необхідно виправити до випуску продукту.

# Виправлення дефектів: Після аналізу результатів тестування виявлені дефекти виправляються, і продукт повторно тестується, щоб переконатися, що вони були виправлені.

# Повторне тестування: після того, як дефекти виправлені, тест повторюється, щоб переконатися, що виправлення були правильними і що немає нових дефектів.

# Тестування програмного забезпечення є важливим етапом життєвого циклу розробки програмного забезпечення. Воно забезпечує високу якість, надійність і безпеку програмного продукту, що підвищує задоволеність користувачів і довіру до бренду.

Після запуску програми ми бачимо перед собою стартове меню зображене на (рис 5.1).

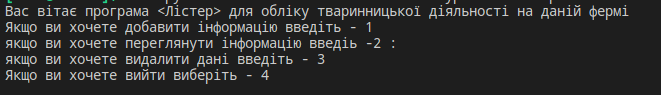


Рис. 5.1 Стартове меню

Для початку на потрібно добавити дані вводимо «1», наступний крок це вибір групи тварин , до якої ми хочемо ввести дані , далі вводимо дані про нашого працівника.

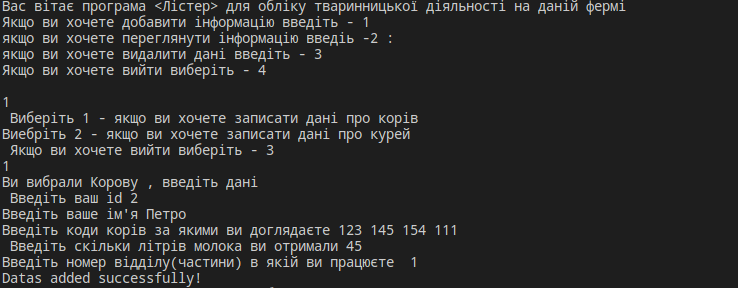


Рис. 5.2 Меню «Добавити волонтера»

Після введення даних , на екрані ми бачимо текст , який нам вказує на те, що дані успішно добавлені. Тепер ми можемо перевірити чи добавились наші дані в базу даних . В головному меню вибирає пункт «2», на (рис 5.3), ми бачимо ,що наші дані успішно добавлені в базу даних.

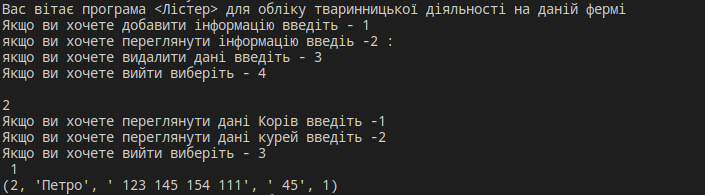


Рис. 5.3 Меню «Перегляд інформації »

Якщо у нас з’явилась потреба видалити дані користувача , в головному меню вибираємо «3» після чого вводимо «id» працівника . Далі ми перевіряємо через меню «Перегляд інформації» перевіряємо , чи видалились дані з бази даних . Як ми бачимо дані успішно видалені.

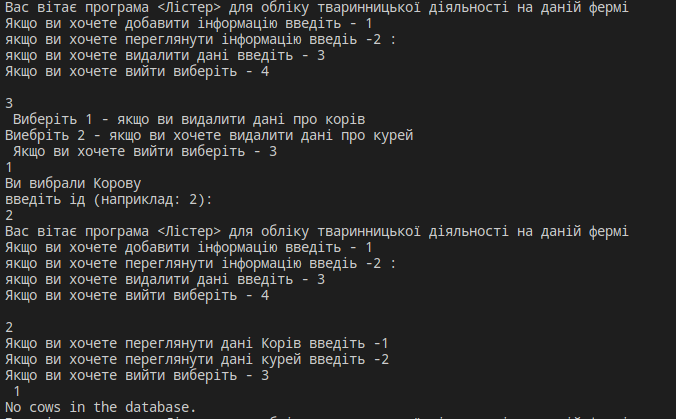


Рис. 5.4 Меню пункту «Видалити інформацію про волонтера»

# РОЗДІЛ 6.

# РОЗРОБКА ДОКУМЕНТАЦІЇ ДЛЯ СУПРОВОДЖЕННЯ ПРОГРАМНОГО ПРОДУКТУ

Клас ‘Animal’ містить атрибути:

- ‘id’ (int): ідентифікатор працівника.

- ‘name’ (str): ім'я .

- ‘much’ (str): кількість приросту (молока, яєць).

- ‘dep\_number’ (str): відділ в якому працює людина.

Клас ‘Cow’

Наслідує атрибути з класу ‘Animal’, та добавлено ще один атрибут ‘a\_name1’, який вказує на коди корів з якими працює людина.

Клас ‘Chicken’

Наслідує атрибути з класу ‘Animal’, та добавлено ще один атрибут ‘a\_name’, який вказує кількість курей з якими працює людина.

Клас ‘Database’

Клас працює з базою даних sqlite3, містить методи , які надсилають запити на те, щоб добавити , видалити, переглянути дані.

- add\_cow (self, cow): надсилає запит на додавання даних до таблиці під назвою «cows».

- add\_chicken (self, chicken): надсилає запит на додавання даних до таблиці під назвою «chickens».

- get\_all\_chickens (self): переглядає всі записи у таблиці «chickens» .

- get\_all\_cow (self): переглядає всі записи у таблиці «cows» .

- remove\_cow (self, value): видаляє запис про певного робітника

- remove\_chickens (self, value): видаляє запис про певного робітника

Клас UserInterface

- remove\_cows: запитує користувача «id» працівника, якого потрібно видалити з таблиці «cows»

- remove\_chickens: запитує користувача «id» працівника, якого потрібно видалити з таблиці «chickens»

- add\_cow: запитує користувача про дані нового волонтера групи Sunrise та додає відповідний запис до бази даних.

- add\_chicken: вимагає вводу даних працівника, який працює з групою тварин «chickens» та додає запис до бази даних.

- add\_cow: вимагає вводу даних працівника, який працює з групою тварин «cow» та додає запис до бази даних.

- view\_cows: виводить на екран усі записи про групу тварин «cows»

- view\_chickens: виводить на екран усі записи про групу тварин «chickens»

# ВИСНОВКИ

Об'єктно-орієнтоване програмування - важлива парадигма програмування, яка дозволяє розбивати складні завдання на менші, більш зрозумілі частини і сприяє підвищенню ефективності та якості програмного забезпечення. У тваринницькій галузі розробка автоматизованих систем обліку на основі принципів ООП може зробити значний внесок у підвищення ефективності виробничих процесів та якості продукції. Впровадження обліку персоналу і тварин, введення показників продуктивності по кожній тварині і виведення ключових показників по фермі і персоналу в цілому може значно полегшити процес управління; використання ООП може забезпечити більш безпечні програмні продукти і знизити ризик виробничих помилок. Тому розробка автоматизованих систем обліку для тваринницьких підприємств на основі ООП може зробити значний внесок у підвищення ефективності та якості виробничих процесів у цій галузі.

Крім того, використання ООП може зменшити кількість помилок у програмному забезпеченні та вразливостей у системі безпеки. Це особливо важливо у тваринницькій галузі, де некоректні дані або помилки в програмному коді можуть призвести до непередбачуваних наслідків. Тому дуже раціонально використовувати об'єктно-орієнтоване програмування в процесі розробки автоматизованих систем обліку на тваринницьких підприємствах. Воно зменшує складність коду, забезпечує гнучкість і простоту супроводу програмного забезпечення, а також дозволяє створювати більш безпечні та ефективні програмні продукти. Як результат, використання ООП може допомогти тваринницьким підприємствам підвищити ефективність управління, зменшити кількість помилок та підвищити якість і надійність продукції.

# БІБЛІОГРАФІЧНИЙ СПИСОК

1. Downey A., Elkner J., Meyers Ch. How to Think Like a Computer Scientist Learning with Python. - Wellesley, Massachusetts: Green Tea Press, 2002. 290 pp..
2. Tutorialspoint / Python [Електронний ресурс]. Режим доступу: <http://www.tutorialspoint.com/python/>
3. Програмування числових методів мовою Python : підруч. / А. В. Анісімов, А. Ю. Дорошенко, С. Д. Погорілий, Я. Ю. Дорогий ; за ред. А. В. Анісімова. К. : Видавничо-поліграфічний центр «Київський університет», 2014. 640 с.
4. Інформаційна система «Agri-footprint» [Електронний ресурс]. Режим доступу: <https://www.agri-footprint.com/products-services/>.
5. Python learning [Електронний ресурс]. Режим доступу: https://www.w3schools.com/python/default.asp
6. Хахаев И.А. Практикум по алгоритмізації і програмуванню на Python. М.: Альт Лінукс, 2010. 126 с.
7. Dusty Phillips «Python 3 Object-Oriented Programming» , 2010
8. Офіційна документація sqlite3 [Електронний ресурс]. Режим доступу: https://docs.python.org/3/library/sqlite3.html
9. SQLite in Python: Writing Your First Program [Електронний ресурс]. Режим доступу: https://www.datacamp.com/tutorial/sqlite-in-python
10. Інформаційний портал про аграрний сектор "ПроАгро" [Електронний ресурс]. Режим доступу: https://proagro.com.ua/