**Львівський національний аграрний університет Факультет механіки та енергетики**

## кафедра інформаційних систем та технологій

**КУРСОВА РОБОТА**

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

на тему: «Розроблення реєстру транспортних засобів підприємства»

Студент 1 курсу КН-12сп групи денної форми навчання механіки та енергетики факультету

Базюк Андрій Романович

Науковий керівник \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

к. т. н., в. о. доцента, Татомир А.В.

Дубляни 2022

**ЗМІСТ**

ВСТУП 3

РОЗДІЛ 1. АНАЛІЗ СУЧАСНОГО СТАНУ ПИТАННЯ ТА ОБҐРУНТУВАННЯ ЗАВДАННЯ НА РОБОТУ 4

1.1. Аналіз предмету проектування 4  
1.2. Класифікація об’єктно-орієнтованих мов програмування 4  
1.3. Огляд та аналіз сучасних технологій та засобів проектування програмного забезпечення 8  
1.4. Універсальна мова проектування UML 9

1.5. Уточнена постановка задачі на розробку програмного забезпечення 10

РОЗДІЛ 2. РОЗРОБКА ЗАГАЛЬНОЇ СТРУКТУРИ ПРОГРАМИ 12

2.1. Аналіз функцій системи 12

2.2. Розроблення структурної моделі системи 12

РОЗДІЛ 3. РОЗРОБЛЕННЯ ІЄРАРХІЇ КЛАСІВ 14

3.1. Декомпозиція системи 14

3.2. Аналіз взаємозв’язків між об’єктами 15

3.3. Розроблення інтерфейсів класів 15

3.4. UML-діаграма класів 16

РОЗДІЛ 4. РОЗРОБКА СИСТЕМИ ВВОДУ-ВИВОДУ ДАНИХ 18

РОЗДІЛ 5. ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 20

РОЗДІЛ 6. РОЗРОБКА ДОКУМЕНТАЦІЇ ДЛЯ СУПРОВОДЖЕННЯ ПРОГРАМНОГО ПРОДУКТУ 28

ВИСНОВКИ 30

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 31

**ВСТУП**

Об’єктно-орієнтоване програмування (ООП) – це еволюційний крок, який випливає із розвитку програмування. ООП дає нам можливість відчути себе не тільки програмістом, а й архітектором, проектуючи структуру програми, створюючи красиві форми.

Ціллю є демонстрація основи ООП на прикладі програми «Реєстр транспортних засобів підприємства».

В сучасному світі людство оперує безмежною кількістю інформації, яку зберігати та сортувати без допоміжних засобів просто неможливо. Тому актуальність систем управління сьогодні дуже висока. Вони допомагають упорядковувати, знаходити та порівнювати певну структуру даних.

Задача розробки реєстру транспортних засобів також вимагає створення системи управління, щоб можна було легко, а головне, швидко знайти потрібний транспортний засіб, створити новий, редагувати інформацію про нього тощо.

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

* 1. **Аналіз предмету проектування**

Предметом проектування є реєстр транспортних засобів підприємства.

Реєстр – упорядкована інформаційна система про положення об’єктів, що

підлягають обліку в системі відповідного рівня управління.

Інформаційна система — сукупність організаційних і технічних засобів для збереження та [обробки інформації](https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%BE%D0%B1%D0%BA%D0%B0_%D1%96%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D1%96%D1%97) з метою забезпечення інформаційних потреб користувачів.

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

Класифікацій мов програмування існує багато, але наукової теорії поки що немає. Три основні класифікації: за функціональною силою, за предметною орієнтацією, за рівнем абстракції.

За функціональною силою:

1. Універсальні мови ( в них можна промоделювати, умовно кажучи, будь-який алгоритм).
2. Спеціалізовані мови (орієнтовані на певні класи задач).

За предметною орієнтацією:

Кожна мова програмування виникла в процесі розв’язання певного класу задач, наприклад, мови програмування для розв’язання задач символьної обробки (Lisp, Cobol, тощо)

За рівнем абстракції*:*

1. Мови низького рівня (машинно-залежні) – Assembler і т.п.
2. Мови високого рівня (орієнтовані на користувача до певної міри) - Pascal, C,  Fortran, тощо.

Мови програмування низького рівня- орієнтовані на конкретний тип процесора і враховують його особливості.

Переваги:

* з допомогою мов низького рівня створюються ефективні і компактні програми, оскільки розробник отримує доступ до всіх можливостей процесора.

Недоліки:

* програміст, що працює з мовами низького рівня, має бути високої кваліфікації, добре розуміти будову комп’ютера;
* результуюча програма не може бути перенесена на комп’ютер з іншим типом процесора.

Асемблер

Мова Асемблера поєднує в собі достоїнства мови машинних команд і деякі риси мов високого рівня. Асемблер забезпечує можливість застосування символічних імен у вихідній програмі й рятує програміста від стомлюючої праці (неминучого при програмуванні мовою машинних команд) по розподілі пам'яті комп'ютера для команд, змінних і констант. Асемблер дозволяє також гнучко й повно використати технічні можливості комп'ютера, як і мова машинних команд. Транслятор вихідних програм в Асемблері простіше транслятора, що вимагається для мови програмування високого рівня. На Асемблері можна написати настільки ж ефективну за розміром й часом виконання програму, як і програму мовою машинних команд. Ця перевага відсутня в мов високого рівня. Цю мову часто застосовують для програмування систем реального часу, технологічними процесами обладнання, забезпечення роботи інформаційно-вимірювальних комплексів. До таких систем звичайно пред'являються високі вимоги за обсягом займаної машинної пам'яті. Часто мова Асемблера доповнюється засобами формування макрокоманд, кожна з яких еквівалентна цілій групі машинних команд. Таку мову називають мовою макроасемблера.

Мови програмування високого рівня.

Мови програмування високого рівня дозволяють писати програми в формі, більш наближеній до звичайної мови. Програму, написаною мовою високого рівня, можна більш легко читати і модифікувати, і вони значно полегшують роботу програміста порівняно з написанням машинного коду. Для перекладу програм, написаних мовою високого рівня, в машинні коди, повинні існувати спеціальні програми. Такі програми називаються трансляторами. Важливою особливістю мов високого рівня є їх відносна незалежність від машини. Це означає, що правила запису програм не залежать або мало залежать від особливостей конкретної машини. Тоді для перенесення програми на іншу машину програму не обов'язково переписувати заново, достатньо лише відтранслювати її в коди, специфічні для цієї машини. В крайньому разі, зміни в програмі повинні бути мінімальними.

С++

Мова C++ з'явився на початку 80-х років.

Створений Б’єрном Страуструпом з первісною метою позбавити себе й своїх друзів від програмування на асемблері, Сі або різних інших мовах високого рівня.

Очевидно, що найбільше C++ запозичив з мови Сі, а також з безпосереднього його попередника мови BCPL.

Ці запозичення забезпечили C++ потужними засобами низького рівня, що дозволяють вирішувати складні завдання системного програмування. Але що в першу чергу відрізняє C++ від Сі - це різний ступінь уваги до типів і структур даних. Це пов'язано з появою понять класу, похідного класу й віртуальної функції, перейнятих у свою чергу з мови Симула 67.

Це дає в C++ більш ефективні можливості для контролю типів і забезпечує модульність програми.

На думку автора мови, розходження між ідеологією С й C++ полягає приблизно в наступному: програма на С відображає “спосіб мислення” процесора, а C++ - спосіб мислення програміста. Відповідаючи вимогам сучасного програмування, C++ наголошує на розробці нових типів даних найбільше повно відповідним концепціям обраної області знань. Клас є ключовим поняттям C++. Опис класу містить опис даних, що вимагаються для подання об'єктів цього типу й набір операцій для роботи з подібними об'єктами.

На відміну від традиційних структур С й Паскаля, членами класу є не тільки дані, але й функції. Функції - члени класу мають привілейований доступ до даних усередині об'єктів цього класу й забезпечують інтерфейс між цими об'єктами й іншою програмою. При подальшій роботі зовсім не обов'язково пам'ятати про внутрішню структуру класу й механізм роботи убудованих функцій. У цьому змісті клас подібний до електричного приладу - мало хто знає про його пристрій, але всі знають, як ним користуватися.

Мова С++ є засобом об'єктного програмування, новітньої методики проектування й реалізації програм, що у поточному десятилітті, швидше за все, замінить традиційне процедурне програмування.

Головною метою творця мови доктора Б’єрна Страустрапа було оснащення мови С++ конструкціями, що дозволяють збільшити продуктивність праці програмістів і полегшити процес оволодіння великими програмними продуктами.

Абстракція, реалізація, спадкування й поліморфізм є необхідними властивостями якими володіє мова С++, завдяки чому він не тільки універсальний, як і мова С, але і є об'єктною мовою.

Java

Мова Java зародилась як частина проекту створення передового програмного забезпечення (ПО) для різних побутових приладів. Реалізація проекту була почата мовою С++, але незабаром виник ряд проблем, найкращим засобом боротьби з якими була зміна самого інструмента - мови програмування. Стало очевидним, що необхідно платформо-незалежна мова програмування, що дозволяє створювати програми, які не доводилося б компілювати окремо для кожної архітектури й можна було б використати на різних процесорах під різними операційними системами.

Три ключових елементи об'єдналися в технології мови Java:

1. Java надає для широкого використання свої аплети (applets) — невеликі, надійні, динамічні, що не залежать від платформи активні мережні додатки, що вбудовують у сторінки Web. Аплети Java можуть настроюватися й поширюватися споживачам з такою же легкістю, як будь-які документи HTML.
2. Java вивільняє міць об’єктно-орієнтованої розробки додатків, сполучаючи простий і знайомий синтаксис із надійним й зручним в роботі з середовищем розробки. Це дозволяє широкому колу програмістів швидко створювати нові програми й нові аплети.
3. Java надає програмістові багатий набір класів об'єктів для ясного абстрагування багатьох системних функцій, використовуваних при роботі з вікнами, мережею й для вводу-виводу. Ключова риса цих класів полягає в тім, що вони забезпечують створення незалежних від використовуваної платформи абстракцій для широкого спектра системних інтерфейсів.

**1.3 Огляд та аналіз сучасних технологій та засобів проектування програмного забезпечення**

Комп'ютерна система управління (КСУ) — автоматизована система, що ґрунтується на комплексному використанні технічних, математичних, інформаційних та організаційних засобів для управління складними технічними й економічними об'єктами. КСУ - це сукупність керованого об’єкта й автоматичних вимірювальних та керуючих пристроїв, у якій частину функцій виконує людина.

Створені за тридцятилітню історію впровадження ЕОМ у сферу управлінської діяльності численні КСУ різняться призначенням, проблемною орієнтацією, місцем застосування, автоматизованими функціями і т. ін. З метою підвищення ефективності витрат на розвиток діючих систем та проектування нових, усунення паралелізму і дублювання в проведенні наукових досліджень і проектно-конструкторських робіт, створення типових проектних рішень і типових КСУ зроблено їх класифікацію.

КСУ дає змогу розв'язувати задачі перспективного та оперативного планування виробництва, оперативного розподілу завантаження обладнання, оптимального розподілу обладнання та використання ресурсів і інше. АСК належить до класу людино-машинних систем і складається з функціональної і забезпечувальної частин.

Функціональна частина КСУ включає систему моделей планово-економічних і управлінських задач, забезпечувальна частина — інформаційну і технічну бази, математичне забезпечення, економіко-організаційну базу та інше.

Спеціальне математичне забезпечення включає пакети прикладних програм, що здійснюють організацію й обробку даних з метою реалізації необхідних функцій управління в рамках певних економіко-математичних та організаційних моделей. Програмне забезпечення КСУ (ПЗ) містить сукупність програм на носіях, даних і програмних документів, яка призначена для відлагодження, функціонування й перевірки роботоздатності КСУ.

* 1. **Універсальна мова проектування UML**

UML (англ. *Unified Modeling Language*) — уніфікована мова моделювання, використовується у парадигмі об'єктно-орієнтованого програмування. Є невід'ємною частиною уніфікованого процесу розробки програмного забезпечення. UML є мовою широкого профілю, це відкритий стандарт, що використовує графічні позначення для створення абстрактної моделі системи, називаної *UML-моделлю*. UML був створений для визначення, візуалізації, проектування й документування в основному програмних систем. UML не є мовою програмування, але в засобах виконання UML-моделей як інтерпретованого коду можлива кодогенерація. Перша версія (1.0) UML вийшла 13 січня 1997, вона була створена за запитом ObjectManagementGroup (OMG) — організації, відповідальної за прийняття стандартів в галузі об'єктних технологій і баз даних. Після обговорення, у вересні 1997 року, версія 1.1 UML була представлена на голосування в OMG. Розробку UML підтримали і вже тоді використовували як стандарт такі гранди ринку інформаційних технологій, як Microsoft, IBM, Hewlett-Packard, Oracle,DEC, Sybase, Logic Works й інші.

UML може бути застосовано на всіх етапах життєвого циклу аналізу бізнес-систем і розробки прикладних програм. Різні види діаграм які підтримуються UML, і найбагатший набір можливостей представлення певних аспектів системи робить UML універсальним засобом опису як програмних, так і ділових систем. Діаграми дають можливість представити систему (як ділову, так і програмну) у такому вигляді, щоб її можна було легко перевести в програмний код.

Основною причиною використання мови UML є спілкування розробників між собою.

Крім того, UML спеціально створювалася для оптимізації процесу розробки програмних систем, що дозволяє збільшити ефективність їх реалізації у кілька разів і помітно поліпшити якість кінцевого продукту.

UML прекрасно зарекомендувала себе в багатьох успішних програмних проектах. Засоби автоматичної генерації кодів дозволяють перетворювати моделі мовою UML у вихідний код об’єктно-орієнтованих мов програмування, що ще більш прискорює процес розробки.

Практично усі CASE-засоби (програми автоматизації процесу аналізу і проектування) мають підтримку UML. Моделі розроблені в UML, дозволяють значно спростити процес кодування і направити зусилля програмістів безпосередньо на реалізацію системи.

Діаграми підвищують супроводжуваність проекту і полегшують розробку документації.

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

На основі виконаного аналізу та огляду літературних джерел можна сформулювати постановку задачі.

Організаційна сутність задачі

Розробити програмне забезпечення інформаційної системи «Реєстр транспортних засобів підприємства», основними функціями якої є: можливість додавання транспортних засобів користувачем, можливість перегляду всіх транспортних засобів реєстру, зміна та видалення існуючих транспортних засобів, можливість задання станів техніки («доступний», «використовується», «в ремонті» тощо).

Розробити зручний графічний інтерфейс для роботи з інформацією. Основні дії та взаємодія між користувачем та системою повинні супроводжуватися відповідними повідомленнями для користувача.

Програма «Реєстр транспортних засобів підприємства» повинна бути універсальною, такою, щоб підходила як для домашнього користування, так і для підприємств.

Опис вхідної інформації

Подання вхідної інформації здійснюється через консоль. При додаванні та редагуванні транспортного засобу, наприклад, потрібно ввести інформацію про транспортний засіб. Вхідна інформація проходить валідацію на правильність після введення.

Опис вихідної інформації

Дані про транспортні засоби зберігаються в файл. Вихідна інформація також представлена в консолі. Можливе виведення списку наявних транспортних засобів. Форма подання інформації представлена у вигляді списку з таких стрічок: «Id <ідентифікатор транспортного засобу>: <Бренд>, <Модель>, <Статус>».

Опис алгоритму розв’язання задачі

З запуском програми в консоль виводиться інформація про можливі дії користувача, а саме: додати, редагувати, видалити, показати та вийти. Ці команди дозволяють користувачу взаємодіяти з реєстром транспортних засобів.

Додати – надає можливість користувачу вказати інформацію про транспортний засіб (бренд, модель, статус) та додати його в реєстр. Після введення інформації про транспортний засіб, реєстр створює об’єкт, присвоює його ідентифікатор та зберігає його у файл.

Редагувати – ­­­­­­­­­­­­­­­­­­­­­надає можливість користувачу змінити існуючу інформацію про транспортний засіб. Користувач може змінити як всі параметри транспортного засобу, так і окремі, залишаючи значення пустим.

Видалити – ­­­­­­­­­­­­­­­­­­­­­надає можливість користувачу видалити транспортний засіб з реєстру вказавши його ідентифікатор.

Показати – виводить список транспортних засобів в консоль. Реєстр проходить через всі зареєстровані транспортні засобі і формує стрічку з її параметрів у вигляді «Id <ідентифікатор транспортного засобу>: <Бренд>, <Модель>, <Статус>».

Вийти – дозволяє вийти з програми.

**РОЗДІЛ 2. РОЗРОБКА ЗАГАЛЬНОЇ СТРУКТУРИ ПРОГРАМИ**

* 1. **Аналіз функцій системи**

Програмний продукт оперує даними про транспортні засоби.

Завдання системи.

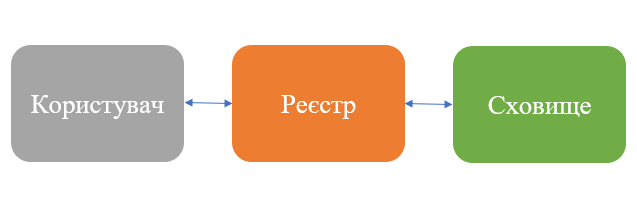
Система повинна застосовуватися для отримання інформації про транспортні засоби, додавання нової, редагування та видалення техніки. Система повинна виконувати основні функції:

* можливість додавання нової техніки;
* пошук та виведення наявної техніки;
* можливість зміни одиниць техніки;
* можливість видалення одиниць техніки;
* можливість задання станів техніки («доступний», «використовується», «в ремонті» тощо);
* можливість фільтрування техніки за станами.

Не функціональні вимоги:

* для звичайної роботи програми достатньо комп’ютера з монітором клавіатурою та мишкою;
* простий і зрозумілий інтерфейс.
  1. **Розроблення структурної моделі системи**

Структурна модель системи повинна передбачати роботу з користувацькими запитами (див. рис. 2.1).



*Рис. 2.1. Структурна модель системи*

Пояснення до рисунка:

* користувач може робити запити до реєстру, а саме «додати», «редагувати», «видалити», «показати» тощо;
* реєстр обробляє запити користувача та взаємодіє зі сховищем транспортних засобів. Він може записувати в сховище нову техніку, діставати існуючу, редагувати та видаляти одиниці техніки. Також може надавати інформацію користувачу;
* сховище зберігає транспортні засоби та дозволяє зчитувати та записувати їх.

**РОЗДІЛ 3. РОЗРОБЛЕННЯ ІЄРАРХІЇ КЛАСІВ**

* 1. **Декомпозиція системи**

Декомпозиція — науковий метод, що використовується структуру завдання і дозволяє змінити вирішення одного великого завдання рішенням серії менших завдань, нехай і взаємопов’язаних, але більш простих. Декомпозиція, як процес розділення, дозволяє розглядати будь-яку досліджувану систему як складну, що складається з окремих взаємопов'язаних підсистем, які, в свою чергу, також можуть бути розділеними на частини.

Метою декомпозиції є отримання вичерпного переліку класів, які необхідно реалізувати для побудови інформаційної системи з дотриманням принципів об’єктно-орієнтованого програмування.

Під час проведення декомпозиції слід дотримуватися принципів SOLID. Принципи SOLID використовуються для дизайну та розробки програмних систем, які, з великою ймовірністю, зможуть тривалий час розвиватися, розширюватися та підтримуватися. Дана концепція включає:

* принцип єдиного обов'язку (Single responsibility principle);
* принцип відкритості/закритості (Open/closed principle);
* принцип підстановки Лісков (Liskov substitution principle);
* принцип розділення інтерфейсу (Interface segregation principle);
* принцип інверсії залежностей (Dependency inversion principle).

Декомпозиція системи:

Клас Vehicle – представляє транспортний засіб, містить інформацію про нього та методи для роботи з ним.

Клас RegistrySystem – представляє реєстр, який буде виконувати основні функції системи, такі як створення, зміну, видалення транспортних засобів. Безпосередньо буде використовувати клас Vehicle.

Клас Storage – абстрактний клас, який описує базові методи для роботи з файловим сховищем, щоб зберігати транспортні засоби.

Клас JsonStorage – унаслідується від класу Storage. Зберігає дані в форматі json.

Клас PlainTextStorage – унаслідується від класу Storage. Зберігає дані в форматі txt, тобто як звичайний текст.

Клас Interface – абстрактний клас, який представляє собою інтерфейс для взаємодії з користувачем. Описує всі методи для можливості взаємодії.

Клас UserInputInterface – унасідується від класу Interface. Взаємодіє з користувачем через консоль.

* 1. **Аналіз взаємозв’язків між об’єктами**

Структура класу VehicleStatus (Enum):

AVAILABLE: str – статус «доступний»;

IN\_USE: str – статус «використовується»;

UNDER\_REPAIR: str – статус «в ремонті»;

NOT\_AVAILABLE: str – статус «недоступний».

Структура класу Vehicle:

brand: str – назва бренду транспортного засобу;

model: str – назва моделі транспортного засобу;

status: VehicleStatus – статус транспортного засобу;

vehicles\_count: int – кількість створених транспортних засобів.

Структура класу JsonStorage:

jsonfile: Path – шлях до файлу сховища.

Структура класу RegistrySystem:

storage: Storage – об’єкт класу Storage (або унаслідуваних від нього класів);

vehicles: dict – словник, в якому зберігаються транспортні засоби.

Структура класу UserInputInterface:

registry: RegistrySystem – об’єкт класу RegistrySystem.

* 1. **Розроблення інтерфейсів класів**

Інтерфейс класу Vehicle:

set\_status – задає статус транспортного засобу;

from\_string: classmethod – створює об’єкт класу зі стрічки.

Інтерфейс класу JsonStorage:

save – зберігає транспортний засіб в сховище;

remove – видаляє транспортний засіб зі сховища;

read – повертає словник з даними про транспортні засоби;

\_write – записує словник з даними в файл (private).

Інтерфейс класу RegistrySystem:

read\_storage – зчитує транспортні засоби, записані в файл;

add\_vehicle – додає транспортний засіб в реєстр;

find\_vehicle – знаходить транспортний засіб по ідентифікатору;

edit\_vehicle – змінює дані транспортного засобу;

delete\_vehicle – видаляє транспортний засіб з реєстру;

show\_vehicles – виводить інформацію про техніку.

Інтерфейс класу UserInputInterface:

get\_input\_vehicle\_info – запитує в користувача дані про транспортний засіб;

get\_input\_vehicle\_id – запитує в користувача ідентифікатор транспортного засобу;

add\_vehicle – запитує потрібні дані для створення техніки;

edit\_vehicle – запитує потрібні дані для редагування транспортного засобу;

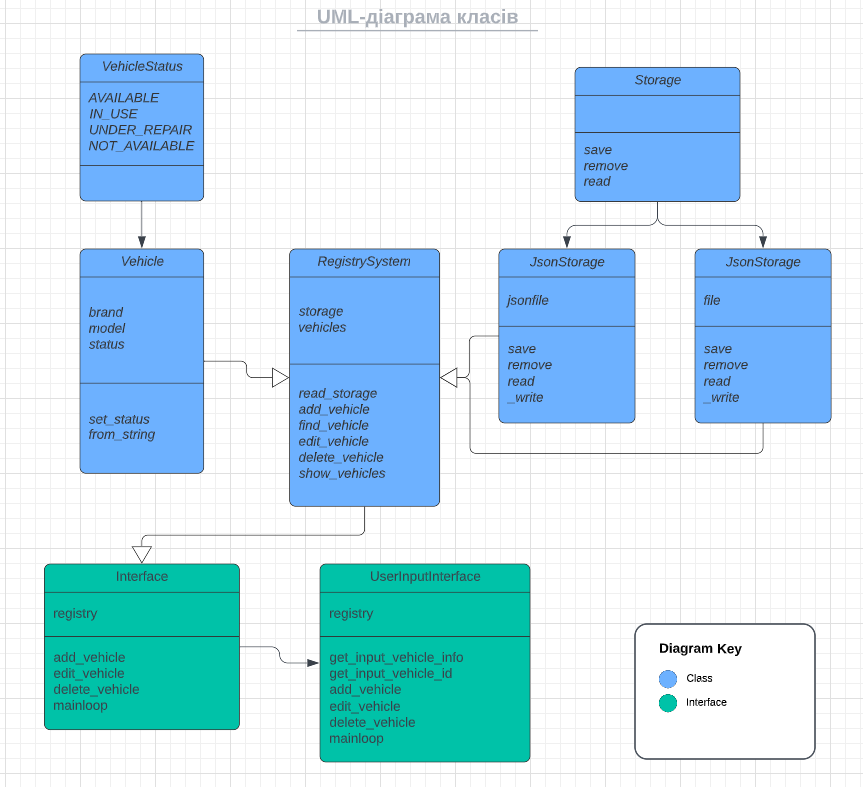
delete\_vehicle – запитує ідентифікатор для видалення одиниці техніки;

mainloop – основний метод, який викликає інші методи класу при потребі користувача.

* 1. **UML-діаграма класів**

Діаграма класів — статичне представлення структури моделі. Відображає статичні (декларативні) елементи, такі як класи, типи, їх зміст та відношення. Діаграма класів, також, може містити позначення для пакетів та може містити позначення для вкладених пакетів. Також, діаграма класів може містити позначення деяких елементів поведінки, однак їх динаміка розкривається в інших типах діаграм.

Діаграма класів (class diagram) служить для представлення статичної структури моделі системи в термінології класів об'єктно-орієнтованого програмування. На цій діаграмі показують класи, інтерфейси, об'єкти й кооперації, а також їхні відносини (див. рис. 3.1).



*Рис. 3.1. UML-діаграма класів*

**РОЗДІЛ 4. РОЗРОБКА СИСТЕМИ ВВОДУ-ВИВОДУ ДАНИХ**

Принципом взаємодії користувача з розробленим програмним забезпеченням являється запит від користувача та надання необхідних даним користувачем при потребі.

При запуску програми інтерфейс пропонує вибрати можливу дію користувачу (див. рис. 4.1).

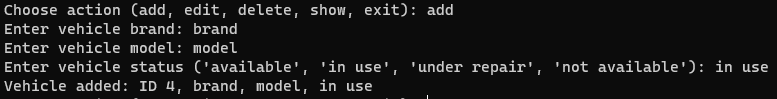


*Рис. 4.1. Початковий інтерфейс програми*

Користувач може вибрати один з наведених дій зі списку, а саме «add», «edit»,

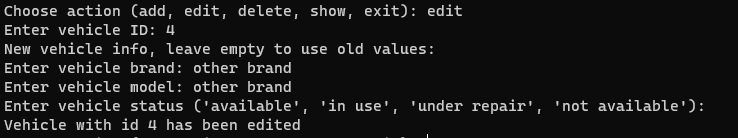
«delete», «show» і «exit». Кожна з цих дій повинна бути прописана користувачем в консолі, для її виклику.

Add ­­­­­­­­­­– просить надати користувача необхідні дані для створення транспортного засобу, створює та додає його до реєстру після перевірки введених даних. Виводить ідентифікатор та інформацію про транспортний засіб після додавання техніки до реєстру (див. рис. 4.2).



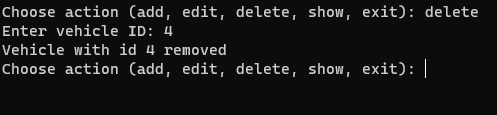
*Рис. 4.2. Вигляд дії «add»*

Edit – вимагає ввести ідентифікатор транспортного засобу, для його редагування. Якщо такий ідентифікатор існує – просить ввести нові дані техніки, або залишити пустим для використання попередніх даних (див. рис. 4.3).



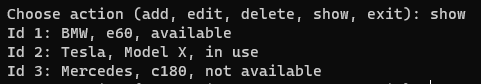
*Рис. 4.3. Вигляд дії «edit»*

Delete – просить ввести ідентифікатор транспортного засобу, для його видалення з реєстру. Виводить результат видалення (див. рис. 4.4).



*Рис. 4.4. Вигляд дії «delete»*

Show – виводить всі зареєстровані транспортні засоби в системі в спеціальному форматі, який забезпечується окремою функцією (див. рис. 4.5).



*Рис. 4.5. Вигляд дії «show»*

Exit – завершує роботу системи, виходячи з програми.

**РОЗДІЛ 5. ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

Тестування програмного забезпечення - процес перевірки відповідності заявлених до продукту вимог і реально реалізованої функціональності, здійснюваний шляхом спостереження за його роботою в штучно створених ситуаціях і на обмеженому наборі тестів, обраних певним чином.

Може оцінюватись:

* відповідність вимогам, якими керувалися проектувальники та розробники;
* правильна відповідь для усіх можливих вхідних даних;
* виконання функцій за прийнятний час;
* практичність;
* сумісність з програмним забезпеченням та операційними системами;
* відповідність задачам замовника.

Оскільки число можливих тестів навіть для нескладних програмних компонент практично нескінченне, тому стратегія тестування полягає в тому, щоб провести всі можливі тести з урахуванням наявного часу та ресурсів. Як результат програмне забезпечення тестується стандартним виконанням програми з метою виявлення помилок або інших дефектів.

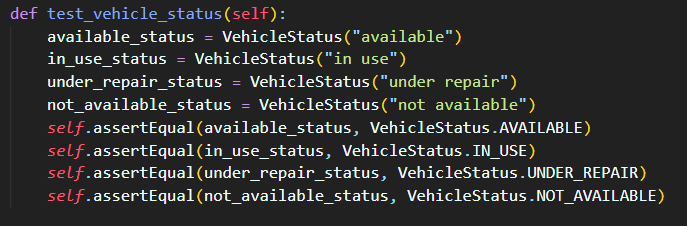
Тестування ПЗ може надавати об'єктивну, незалежну інформацію про якість ПЗ, ризики відмови, як для користувачів, так і для замовників.

Тестування може проводитись, як тільки створено виконуваний код (навіть частково завершений). Процес розробки зазвичай передбачає, коли та як буде відбуватися тестування. Наприклад, при поетапному процесі, більшість тестів відбувається після визначення системних вимог і тоді вони реалізуються в тестових програмах. На противагу цьому, відповідно до вимог гнучкої розробки ПЗ, програмування і тестування часто відбувається одночасно.

Тестування програми «Реєстр транспортних засобів підприємства»:

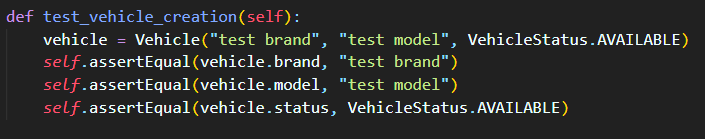
Vehicles Test Case: містить набір тестів, які перевіряють коректність створення транспортних засобів та методів класу Vehicle.

* Тест test\_vehicle\_status: перевіряє значення статусів (див. рис. 5.1).



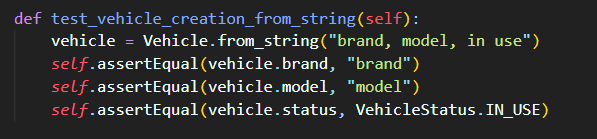
*Рис. 5.1. test\_vehicle\_status*

* Тест test\_vehicle\_creation: тестує створення об’єкту та його атрибутів (див. рис. 5.2).



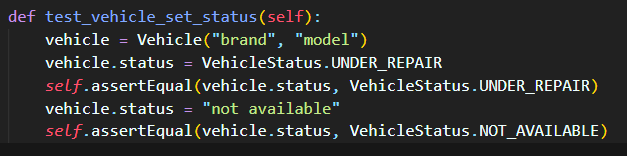
*Рис. 5.2. test\_vehicle\_creation*

* Тест test\_vehicle\_creation\_from\_string: перевіряє створення транспортного засобу зі стрічки (див. рис. 5.3).



*Рис. 5.3. test\_vehicle\_creation\_from\_string*

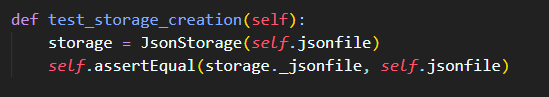
* Тест test\_vehicle\_set\_status: тестує задання значення статус транспортного засобу (див. рис. 5.4).



*Рис. 5.4. test\_vehicle\_set\_status*

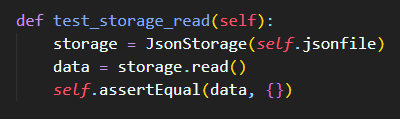
Json Storage Test Case: перевіряє роботу методів класу JsonStorage, тим самим перевіряється коректність роботи сховища Json.

* Тест test\_storage\_creation: тестує створення сховища та його атрибутів (див. рис. 5.5).



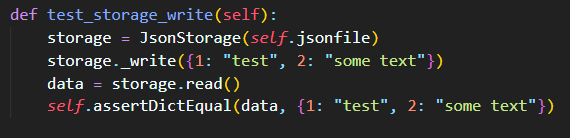
*Рис. 5.5. test\_storage\_creation*

* Тест test\_storage\_read: перевіряє метод read, для отримання даних в сховищі (див. рис. 5.6).



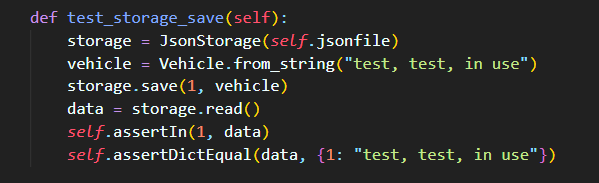
*Рис. 5.6. test\_storage\_read*

* Тест test\_storage\_write: тестує запис даних у сховище (див. рис. 5.7).



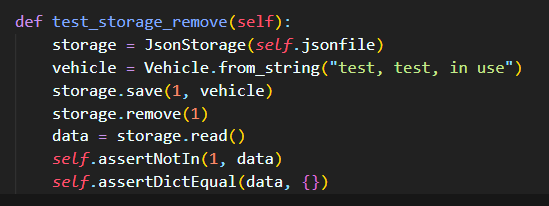
*Рис. 5.7. test\_storage\_write*

* Тест test\_storage\_save: перевіряє збереження транпсортного засобу в сховище (див. рис. 5.8).



*Рис. 5.8. test\_storage\_save*

* Тест test\_storage\_remove: тестує коректне видалення транспортного засобу зі сховища (див. рис. 5.9).



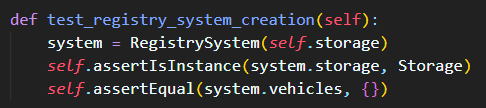
*Рис. 5.9. test\_storage\_remove*

Plain Text Storage Test Case: перевіряє роботу методів класу PlainTextStorage, тим самим перевіряється коректність роботи сховища Txt.

Цей тест кейс містить ті самі тести, що і попередній тест кейс, тільки замість сховища Json, тут присутнє відповідне сховище Txt.

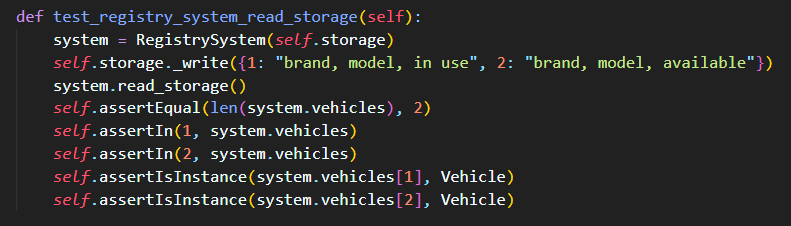
Registry System Test Case: набір тестів, які слугують для перевірки коректності роботи класу Registry System, його методів та взаємодії з іншими класами.

* Тест test\_registry\_system\_creation: перевіряє принцип створення реєстру та його атрибутів (див. рис. 5.10).



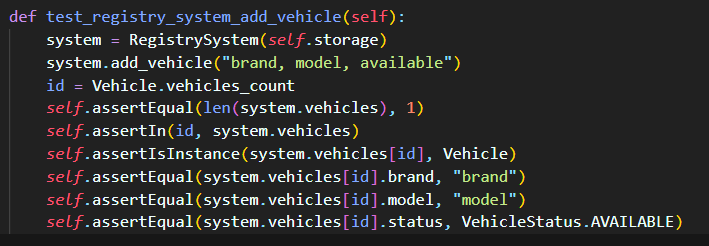
*Рис. 5.10.test\_registry\_system\_creation*

* Тест test\_registry\_system\_read\_storage: тестує читання сховища реєстром і правильність зберігання транспортних засобів (див. рис. 5.11).



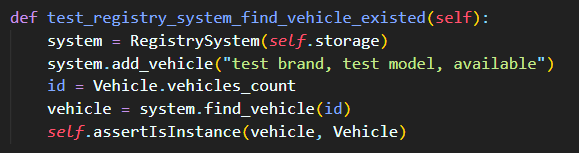
*Рис. 5.11. test\_registry\_system\_read\_storage*

* Тест test\_registry\_system\_add\_vehicle: перевіряє додавання транспортного засобу в реєстр та коректність збережених даних (див. рис. 5.12).



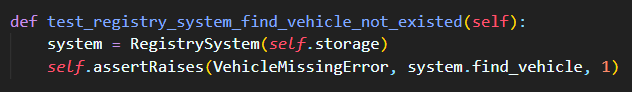
*Рис. 5.12. test\_registry\_system\_add\_vehicle*

* Тест test\_registry\_system\_find\_vehicle\_existed: тестує роботу методу find\_vehicle, якщо транспортний засіб існує (див. рис. 5.13).



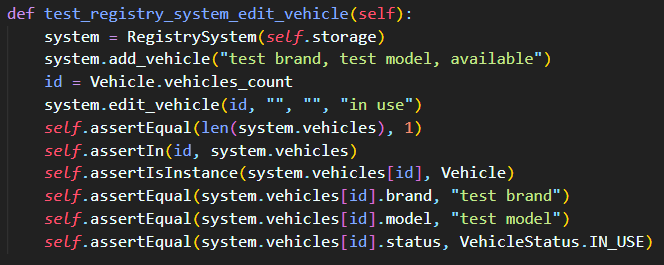
*Рис. 5.13. test\_registry\_system\_find\_vehicle\_existed*

* Тест test\_registry\_system\_find\_vehicle\_not\_existed: перевіряє роботу методу find\_vehicle, якщо транспортний засіб не існує (див. рис. 5.14).



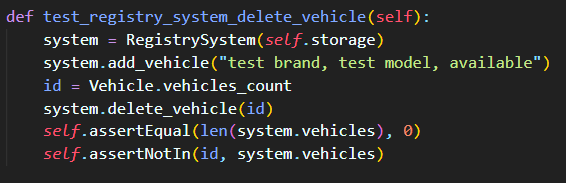
*Рис. 5.14. test\_registry\_system\_find\_vehicle\_not\_existed*

* Тест test\_registry\_system\_edit\_vehicle: тестує редагування транспортного засобу реєстру (див. рис. 5.15).



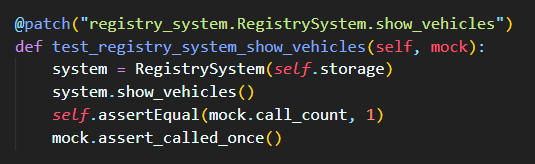
*Рис. 5.15. test\_registry\_system\_edit\_vehicle*

* Тест test\_registry\_system\_delete\_vehicle: перевіряє видалення одиниці техніки з реєстру (див. рис. 5.16).



*Рис. 5.16. test\_registry\_system\_delete\_vehicle*

* Тест test\_registry\_system\_show\_vehicles: перевіряє роботу методу show\_vehicles, який виводить зареєстровані транспортні засоби в системі (див. рис. 5.17).



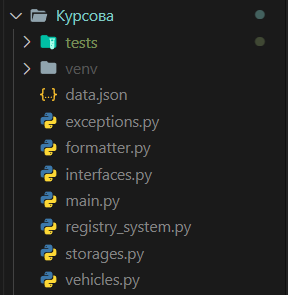
*Рис. 5.17. test\_registry\_system\_show\_vehicles*

**РОЗДІЛ 6. РОЗРОБКА ДОКУМЕНТАЦІЇ ДЛЯ СУПРОВОДЖЕННЯ ПРОГРАМНОГО ПРОДУКТУ**

Документація до проекту:

Опис стеку технологій: проект виконаний на мові програмування Python в середовищі Visual Studio Code.

Файлова структура проекту: проект розділений на модулі. Це обумовлюється зручністю використання, швидкістю знаходження помилок та легкою заміною модуля на інший при потребі. Структура містить такі модулі: vehicles.py, storages.py, registry\_system.py, interfaces.py, formatter.py, exceptions.py, main.py, папку tests, в якій знаходить набір тест кейсів та data.<ext>, який слугує сховищем (див. рис. 6.1).



*Рис. 6.1. Структура файлової системи*

Запуск проекту:

Для стабільної роботи рекомендується встановити віртуальне оточення Python. Це можливо зробити за допомогою команди «python -m venv <назва віртуального оточення>». Після виконання команди створиться папка з вказаною назвою. Віртуальне оточення потрібно активувати, за допомогою команди «<назва віртуального оточення>\Scripts\activate>» (Windows), або «source < назва віртуального оточення >/bin/activate» (MacOS, Linux).

Запуск виконається за допомогою команди «python main.py» в консолі. Для запуску тестів потрібно ввести команду «python -m unittest discover tests».

Вимоги до системи:

Проект можливо запустити на любій сучасній операційній системі, такій як Windows, MacOS, Linux, з встановленим інтерпретатором Python. Версія Python повинна відповідати 3.10, або вище, для коректної роботи проекту.

**ВИСНОВКИ**

В ході виконання роботи мовою Python в середовищі Visual Studio Code реалізовано програму "Реєстр транспортних засобів підприємства". Також продемонстровано на прикладі можливості об’єктно-орієнтованого програмування.

Реалізовано такі функціональні можливості:

* Здійснено запис та зчитування даних у файл (з файлу);
* Можливість додавання транспортного засобу в реєстр;
* Можливість пошуку транспортного засобу;
* Можливість редагування транспортного засобу;
* Можливість видалення транспортного засобу з реєстру;
* Можливість виводу зареєстрованих транспортних засобів;
* Можливість задання станів техніки («доступний», «використовується», «в ремонті», «недоступний»).

Об’єктно-орієнтоване програмування дозволяє створювати програму, як набір користувацьких типів даних (класів), приховувати деталі реалізації, використовувати повторний код, інтерпретувати виклики процедур та функцій на етапі виконання (втілюючи основи ООП - інкапсулювання, поліморфізм, успадкування).

На закінчення терміну на виконання курсової роботи залишились не реалізованими такі функціональні вимоги: можливість фільтрування транспортних засобів в реєстрі за станами.

Універсальність програми полягає в тому, що її інтерфейс простий та зрозумілий, кожен користувач може створити свій реєстр та мати інформацію про транспортні засоби, якою зручно користуватись.

**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ**

1. **John M. Zelle «Python Programming: An Introduction to Computer Science».**
2. **Ел Свейгарт «Автоматизація рутинних задач за допомогою Python. Практиче керівництво для початківців».**
3. **Ден Бейдер «Чистий Python. Тонкощі програмування для профі».**
4. [https://lucid.app/](https://lucid.app/documents#/dashboard) - Побудова UML-діаграми.
5. <https://studfile.net/> - Теоретична інформація.
6. <https://www.learnpython.org/> - Adavanced Tutorial.