Міністерство освіти і науки України
Чернівецький національний університет імені Юрія Федьковича

Програмного забезпечення комп'ютерних систем
(повна назва кафедри, циклової комісії)

КУРСОВИЙ ПРОЄКТ:
СТРУКТУРИ ДАНИХ, АЛГОРИТМИ ТА ОБ'ЄКТНО-ОРІЄНТОВАНА ПАРАДИГМА

на тему:**Система управління базою даних «Автомобіль»**

здобувач вищої освіти II курсу 243 групи
спеціальності 121 «Інженерія програмного забезпечення»
першого (бакалаврського) рівня вищої освіти
Бугера М.О.

Дата захисту «_____»_____20__р.
Оцінка:
за національною шкалою _____
(словами)
кількість балів _____
(цифра)
за шкалою ECTS _____
(літера)

Чернівці, 2025

ЧЕРНІВЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ЮРІЯ ФЕДЬКОВИЧА

Навчально-науковий інститут фізико-технічних та комп'ютерних наук
Кафедра програмного забезпечення комп'ютерних систем

ЗАВДАННЯ
на курсовий проєкт
здобувачу вищої освіти
першого (бакалаврського) рівня вищої освіти
Бугера Михайло Олександрович

1. Тема проєкту: Створення консольного застосунку за допомогою мови C++ для системи управління базою даних «Автомобіль»

2. Вихідні дані до проєкту:

- Створити базу даних – CSV, в який будуть вноситись дані про автомобілі;
- Передбачити ввід та вивід даних через консоль;
- Розробити консольне меню, для виконання функцій програми (додавання, редагування, видалення, перегляд);
- Передбачити сортування та фільтрацію автомобілів з бази даних;
- Передбачити перевірку коректності вводу даних та обробку виключень.

3. Зміст розрахунково-пояснювальної записки (перелік питань, які необхідно розробити):

- описати загальні вимоги до програми;
- описати призначення та область застосування;
- описати модулі проекту та алгоритми виконання;
- описати методи програми;
- описати користувацький інтерфейс.

4. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень):

- блок-схеми роботи з програмою;
- скріншоти роботи з програмою.

Керівники проєкту _____ Валь О.О.
(підпис керівника)

_____ Дяконенко Б.В.
(підпис керівника)

_____ Комісарчук В.В.
(підпис керівника)

Завдання взяв до виконання _____ Бугера М.О.
(підпис студента)

РЕФЕРАТ

У курсовому проєкті розроблено систему управління базою автомобілів для комп’ютерів, які працюють під керуванням операційної системи Windows.

Програмне забезпечення розраховане на користувачів, які не мають спеціальної комп’ютерної підготовки. Досвід роботи не вимагається.

Область застосування – програмне забезпечення для ноутбуків або персональних комп'ютерів, що працюють на операційних системах Windows та Linux.

Розробка реалізована засобами середовища Visual studio на мові C++. Дане середовище є зручним у використанні для швидкого та якісного створення додатків на ОС Windows.

Дана розробка у майбутньому може бути розширена із додаванням нового функціоналу і видозміненою логікою обробки.

Курсовий проєкт містить: 88 с., 28 рис., 7 табл., 1 додаток, 9 джерел.

ОПЕРАЦІЙНА СИСТЕМА, НОУТБУКИ, КОМП’ЮТЕРИ, VISUAL STUDIO, C++,КОНСОЛЬНИЙ ДОДАТОК.

SUMMARY

In the course project, a database management system of cars was developed for computers running the Windows operating system.

The software is intended for users who do not have special computer training. Work experience is not required.

The field of application is software for laptops or personal computers running Windows and Linux operating systems.

The development is implemented using the Visual studio environment in C++. This environment is convenient to use for quick and high-quality creation of applications on OC Windows.

This development can be expanded in the future with the addition of new functionality and modified processing logic.

The course project contains: 88 pages, 28 figures, 7 tables, 1 appendix, 9 references.

OPERATING SYSTEM, LAPTOPS, COMPUTERS, VISUAL STUDIO, C++,CONSOLE APP.

ЗМІСТ

ЗМІСТ 5

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 7

1 АРХІТЕКТУРА ТА ФУНКЦІОНАЛЬНІ ПОКАЗНИКИ 8

1.1 Загальні вимоги до програми 8

Вимоги до консольного інтерфейсу користувача 8

Вимоги до архітектури програми 8

Вимоги до функціональності програми 9

1.2 Призначення та область застосування 9

1.3 Функціональні вимоги 9

1.4 Опис сценаріїв використання програми 10

2 ОПИС ПРОГРАМИ 20

2.1 Структура програми 20

Модулі програми 20

Алгоритми роботи програми 21

2.2 Опис методів програми 26

2.3 Програмні засоби 31

2.4 Опис користувацького інтерфейсу 31

2.5 Тестування 39

3 ВИСНОВКИ 41

4 СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 42

5 ДОДАТКИ 43

ДОДАТОК А. Скролінг (текст) програми 43

Програмний модуль “ConsoleAplication.cpp” 43

Програмний модуль “AuthManager” 47
Вміст “AuthManager.h” 47
Вміст “AuthManager.cpp” 49
Програмний модуль “Car” 53
Вміст “Car.h” 53
Вміст “Car.cpp” 55
Програмний модуль “CarManager” 58
Вміст “CarManager.h” 58
Вміст “CarManager.cpp” 61
Програмний модуль “Configuration” 75
Вміст “Configuration.h” 75
Вміст “Configuration.cpp” 77
Програмний модуль “IPrintable” 81
Вміст “IPrintable.h” 81
Програмний модуль Vehicle” 82
Вміст “Vehicle.h” 82
Вміст “Vehicle.h” 85

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ОС – операційна система.

ПЗ – програмне забезпечення.

ID – ідентифікатор.

CRUD — це скорочення від англійських слів Create, Read, Update, Delete.

БД - База Даних.

1 АРХІТЕКТУРА ТА ФУНКЦІОНАЛЬНІ ПОКАЗНИКИ

1.1 Загальні вимоги до програми

Вимоги до консольного інтерфейсу користувача

1. Робоча мова інтерфейсу – українська.
2. Використання термінів, зрозумілих користувачеві.
3. Навігаційна панель (меню), яка забезпечує перегляд, редагування, додавання та зчитування даних.
4. «Посібник користувача», що містить інструкції з використання програми.
5. Повідомлення про некоректно введені дані.

Вимоги до архітектури програми

1) Використання класів, класів-нащадків, абстрактних класів та інтерфейсного класу які реалізують:
    - збереження даних у файли та зчитування даних з файлів.
    - методи вводу та виводу даних.
2) Стійкість програми:
    - програма повинна не втрачати працездатності при будь-яких діях користувача;
    - інформація, що вводиться, скрізь, де це можливо, піддається логічному забезпеченню цілісності даних;
    - при будь-яких діях користувача не повинні втрачатись дані або їх цілісність.

Вимоги до функціональності програми

1) База даних – це текстовий файл.
2) Збереження даних у БД, їх зчитування, зміна та видалення.
3) Обов’язкова перевірка коректності вводу даних та обробка виключень.
4) Сортування, фільтрація та пошук даних по самостійно обраному ключу.
5) Наявність посібника користувача, який ознайомить користувача з принципом взаємодії з програмою.
6) Можливість пошуку об’єктів.

1.2 Призначення та область застосування

Мета роботи полягає у розробці системи управління базою автомобілів для комп’ютерів, які працюють під керуванням ОС Windows та Linux.

Програмне забезпечення покращує управління автомобілями, а саме: додавання, редагування, видалення та отримання даних про авто за заданими фільтрами. А також робить зручним пошук авто за певними параметрами.

Область застосування – програмне забезпечення для ноутбуків або персональних комп'ютерів, що працюють на операційній системі Windows.

1.3 Функціональні вимоги

До програмного забезпечення висуваються такі функціональні вимоги:

1) Вхідні дані подаються користувачем методом введення із клавіатури у консоль.

- База даних – це CSV файл, тому передбачити:
- збереження даних у файл;
- зчитування даних з файлу.

2) Консольне меню повинне забезпечувати наступні можливості:
   - перегляд даних;
   - редагування даних;
   - додавання даних;
   - видалення даних;
   - перегляд посібника користувача.
3) обов’язкова перевірка коректності вводу даних та обробка виключень.

1.4 Опис сценаріїв використання програми

1. Сценарій: Авторизація користувача

Передумова: Користувач запустив програму
Основний сценарій
Крок 1: Система виводить меню авторизації
Крок 2: Користувач вводить логін і пароль
Крок 3: Система перевіряє дані
Крок 4: Якщо дані коректні — користувач авторизується, система відкриває головне меню
Альтернативний сценарій
Крок 1: Користувач вводить некоректні дані
Крок 2: Система виводить повідомлення “Користувача не знайдено”
Крок 3: Користувач повертається до меню авторизації

2. Сценарій: Додавання автомобіля

Передумова: Користувач авторизований, відкрито меню автомобілів
Основний сценарій
Крок 1: Користувач обирає пункт “2. Додати автомобіль”
Крок 2: Система викликає
Крок 3: Користувач вводить дані про авто (марка, рік, ціна, витрати пального тощо)
Крок 4: Система зберігає авто в колекцію та виводить повідомлення “Автомобіль успішно додано”
Альтернативний сценарій
Крок 1: Користувач вводить некоректні дані (наприклад, рік — буквами)
Крок 2: Система виводить повідомлення про помилку
Крок 3: Користувач повертається до вводу даних

3. Сценарій: Перегляд усіх автомобілів

Передумова: Користувач авторизований, база даних завантажена
Основний сценарій
Крок 1: Користувач обирає пункт “1. Переглянути всі автомобілі”
Крок 2: Система викликає
Крок 3: Виводиться список усіх авто з характеристиками
Альтернативний сценарій
Крок 1: База даних порожня
Крок 2: Система виводить повідомлення “Колекція автомобілів порожня!”

4. Сценарій: Видалення автомобіля

Передумова: Користувач авторизований, база даних завантажена
Основний сценарій
Крок 1: Користувач обирає пункт “6. Видалити автомобіль”
Крок 2: Вводить ID або індекс авто
Крок 3: Система викликає
Крок 4: Авто видаляється, система виводить повідомлення “Автомобіль успішно видалено”
Альтернативний сценарій
Крок 1: Користувач вводить ID, якого не існує
Крок 2: Система виводить повідомлення “Автомобіль не знайдено”

5. Сценарій: Пошук автомобіля за критеріями

Передумова: Користувач авторизований, база даних завантажена
Основний сценарій
Крок 1: Користувач обирає пункт “8. Пошук авто”
Крок 2: Вводить критерії (марка, рік, ціна тощо)
Крок 3: Система викликає
Крок 4: Виводиться список авто, що відповідають критеріям
Альтернативний сценарій
Крок 1: Користувач вводить некоректні критерії
Крок 2: Система виводить повідомлення “Немає результатів”

6. Сценарій: Сортування автомобілів

Передумова: Користувач авторизований, база даних завантажена
Основний сценарій
Крок 1: Користувач обирає пункт “15–19. Сортування”
Крок 2: Вказує критерій (ціна, рік, витрати пального)
Крок 3: Система викликає
Крок 4: Виводиться відсортований список авто
Альтернативний сценарій
Крок 1: Користувач вводить некоректний критерій
Крок 2: Система виводить повідомлення про помилку

7. Сценарій: Розрахунок вартості поїздки

Передумова: Користувач авторизований, авто обране
Основний сценарій
Крок 1: Користувач обирає пункт “14. Розрахунок вартості поїздки”
Крок 2: Вводить кількість км та ціну пального
Крок 3: Система викликає
Крок 4: Виводиться загальна вартість поїздки
Альтернативний сценарій
Крок 1: Користувач вводить буквене значення замість числа
Крок 2: Система виводить повідомлення про помилку

8. Сценарій: Редагування автомобіля

Передумова: Користувач авторизований, база даних завантажена
Основний сценарій
Крок 1: Користувач обирає пункт “5. Редагувати автомобіль”
Крок 2: Вводить ID або індекс авто
Крок 3: Система викликає
Крок 4: Користувач змінює потрібні поля
Крок 5: Система зберігає зміни та виводить повідомлення “Автомобіль успішно оновлено”
Альтернативний сценарій
Крок 1: Користувач вводить ID, якого не існує
Крок 2: Система виводить повідомлення “Автомобіль не знайдено”

9. Сценарій: Фільтрація автомобілів

Передумова: Користувач авторизований, база даних завантажена
Основний сценарій
Крок 1: Користувач обирає пункт “20. Фільтрація за кольором” або “21. Фільтрація за роком”
Крок 2: Вводить значення фільтра
Крок 3: Система виводить список авто, що відповідають фільтру
Альтернативний сценарій
Крок 1: Користувач вводить некоректне значення
Крок 2: Система виводить повідомлення “Немає результатів”

10. Сценарій: Перегляд економічних автомобілів

Передумова: Користувач авторизований, база даних завантажена
Основний сценарій
Крок 1: Користувач обирає пункт “10. Економічні авто”
Крок 2: Вводить порогове значення витрат пального
Крок 3: Система виводить список авто, які мають менші витрати
Альтернативний сценарій
Крок 1: Користувач вводить некоректне значення
Крок 2: Система виводить повідомлення про помилку

11.Сценарій: Застосування знижки

Передумова: Користувач авторизований, авто обране
Основний сценарій
Крок 1: Користувач обирає пункт “12. Знижка на авто”
Крок 2: Вводить відсоток знижки
Крок 3: Система викликає
Крок 4: Виводиться нова ціна авто
Альтернативний сценарій
Крок 1: Користувач вводить некоректне значення
Крок 2: Система виводить повідомлення про помилку

12.Сценарій: Перегляд посібника користувача

Передумова: Користувач авторизований
Основний сценарій
Крок 1: Користувач обирає пункт “22. Інструкція користування”
Крок 2: Система виводить текст посібника

13. Сценарій: Керування користувачами (для ADMIN)

Передумова: Користувач авторизований як адміністратор
Основний сценарій
Крок 1: Адміністратор обирає пункт “Додати/Видалити користувача”
Крок 2: Вводить логін нового або існуючого користувача
Крок 3: Система викликає або
Крок 4: Виводиться повідомлення про успішну дію
Альтернативний сценарій
Крок 1: Введено некоректні дані
Крок 2: Система виводить повідомлення про помилку

14. Сценарій: Завантаження даних з файлу

Передумова: Користувач авторизований
Основний сценарій
Крок 1: Користувач обирає пункт “9. Завантажити дані з файлу”
Крок 2: Система викликає
Крок 3: Дані з CSV-файлу завантажуються в колекцію
Крок 4: Система виводить повідомлення “Дані успішно завантажено”
Альтернативний сценарій
Крок 1: Файл не знайдено або пошкоджений
Крок 2: Система виводить повідомлення “Помилка завантаження файлу”

15.Сценарій: Збереження даних у файл

Передумова: Користувач авторизований, колекція авто не порожня
Основний сценарій
Крок 1: Користувач обирає пункт “23. Зберегти дані у файл”
Крок 2: Система викликає
Крок 3: Дані записуються у CSV-файл
Крок 4: Система виводить повідомлення “Дані успішно збережено”
Альтернативний сценарій
Крок 1: Виникає помилка запису (немає доступу до файлу)
Крок 2: Система виводить повідомлення “Помилка збереження даних”

16.Сценарій: Перехід між користувачами

Передумова: Користувач авторизований як адміністратор
Основний сценарій:
Крок 1: Адміністратор обирає пункт “Змінити користувача”
Крок 2: Система викликає
Крок 3: Виводиться меню авторизації
Крок 4: Новий користувач вводить логін і пароль

17. Сценарій : Перегляд списку користувачів (ADMIN)

Передумова: Користувач авторизований як адміністратор
Основний сценарій
Крок 1: Адміністратор обирає пункт “Переглянути список користувачів”
Крок 2: Система викликає
Крок 3: Виводиться список усіх зареєстрованих користувачів

18. Сценарій: Вихід з програми

Передумова: Користувач авторизований
Основний сценарій
Крок 1: Користувач обирає пункт “Вийти з програми”
Крок 2: Система завершує роботу програми

2 ОПИС ПРОГРАМИ

2.1 Структура програми

Модулі програми

Робота розробленого програмного забезпечення реалізується наступними модулями:

1. Vehicle – абстрактний клас що представляє транспортний засіб, у якому оголошені основні методи для дочірніх класів.
2. Car – клас нащадок Vehicle, клас що представляє автомобіль.
3. Configuration – клас нащадок Vehicle, який описує конфігурацію автомобілів.
4. CarManager – клас нащадок Car, менеджер-клас для керування колекцією автомобілів. Реалізує всю бізнес-логіку програми: CRUD-операції, завантаження/збереження, пошук, сортування, фільтрацію та аналітику.
5. AuthManager – клас для управління авторизаціє.
6. IPrintable – клас інтерфейс всі класи що наслідують його зообовязанні представити реалізацію методів Print() and ToCSV().

Алгоритми роботи програми

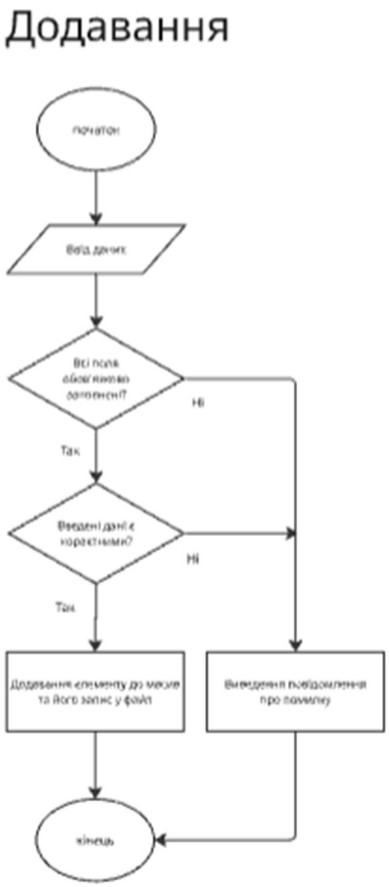

Рисунок 1 – Узагальнена блок-схема алгоритму додавання об’єкту

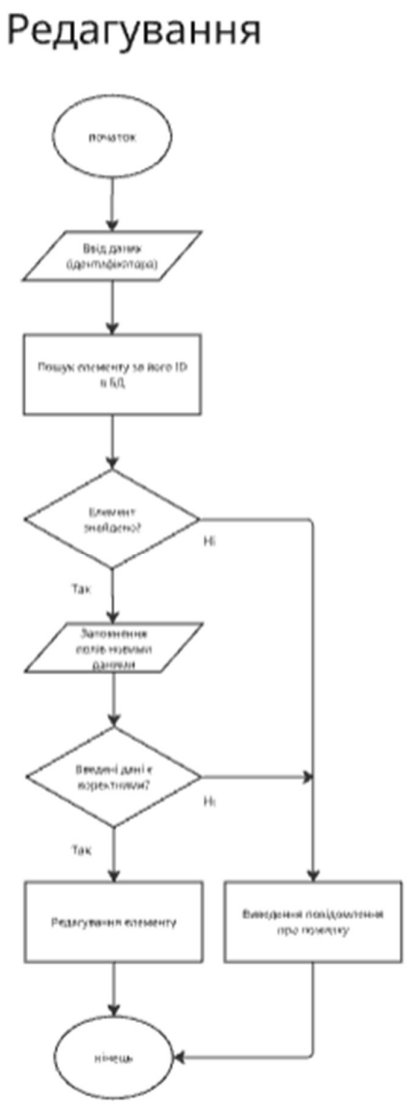

Рисунок 2 – Узагальнена блок-схема алгоритму редагування об’єкту

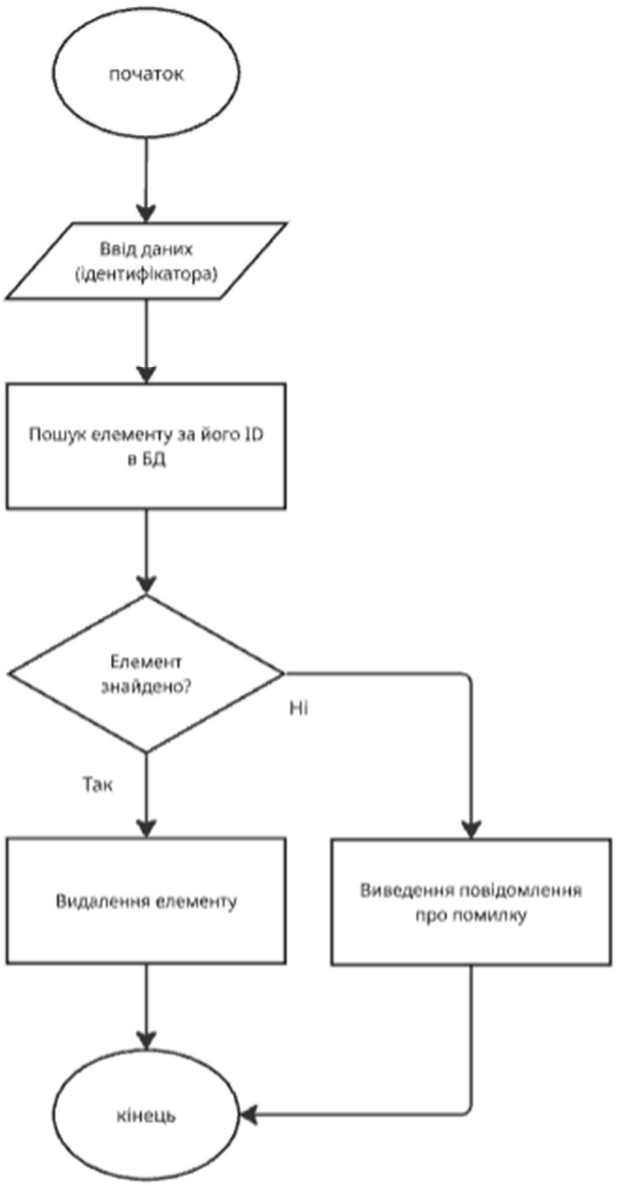

Рисунок 3 – Узагальнена блок-схема алгоритму видалення об’єкта

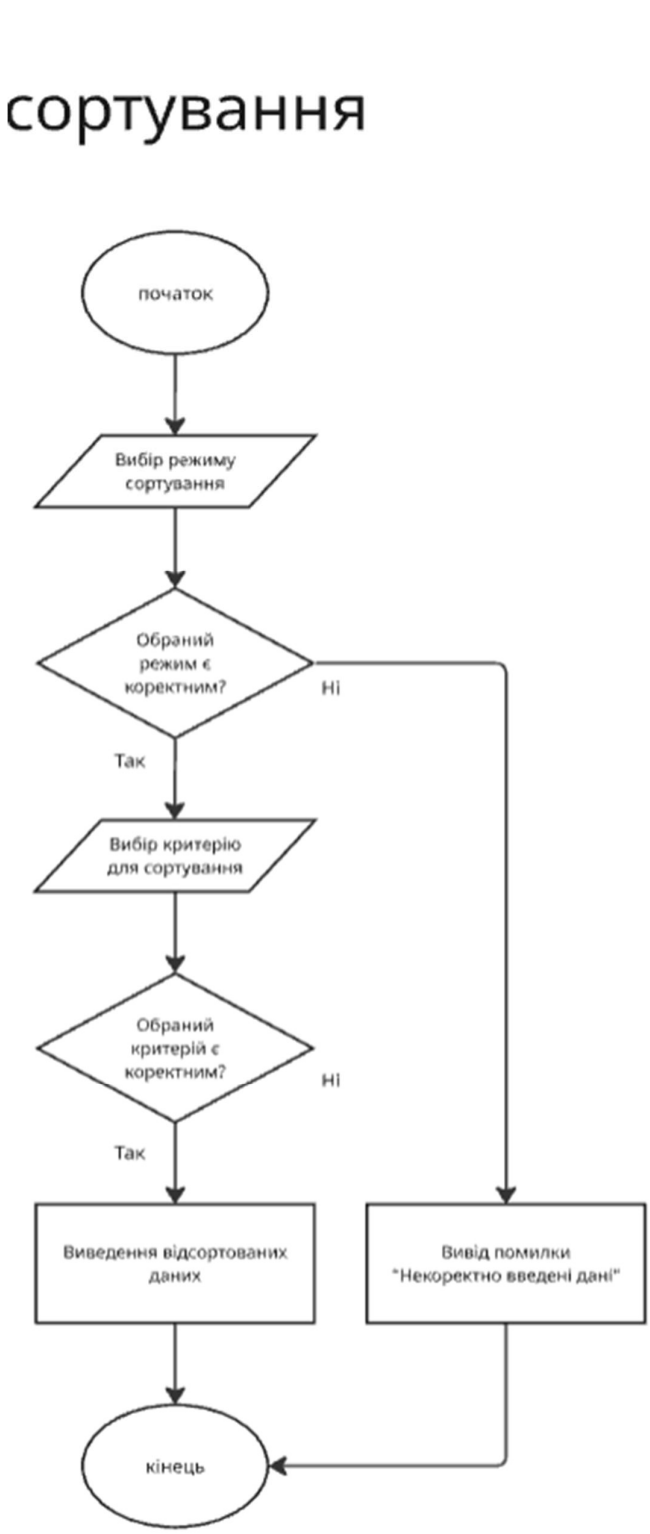

Рисунок 4 – Узагальнена блок-схема алгоритму перегляду доданих об’єктів та їх сортування

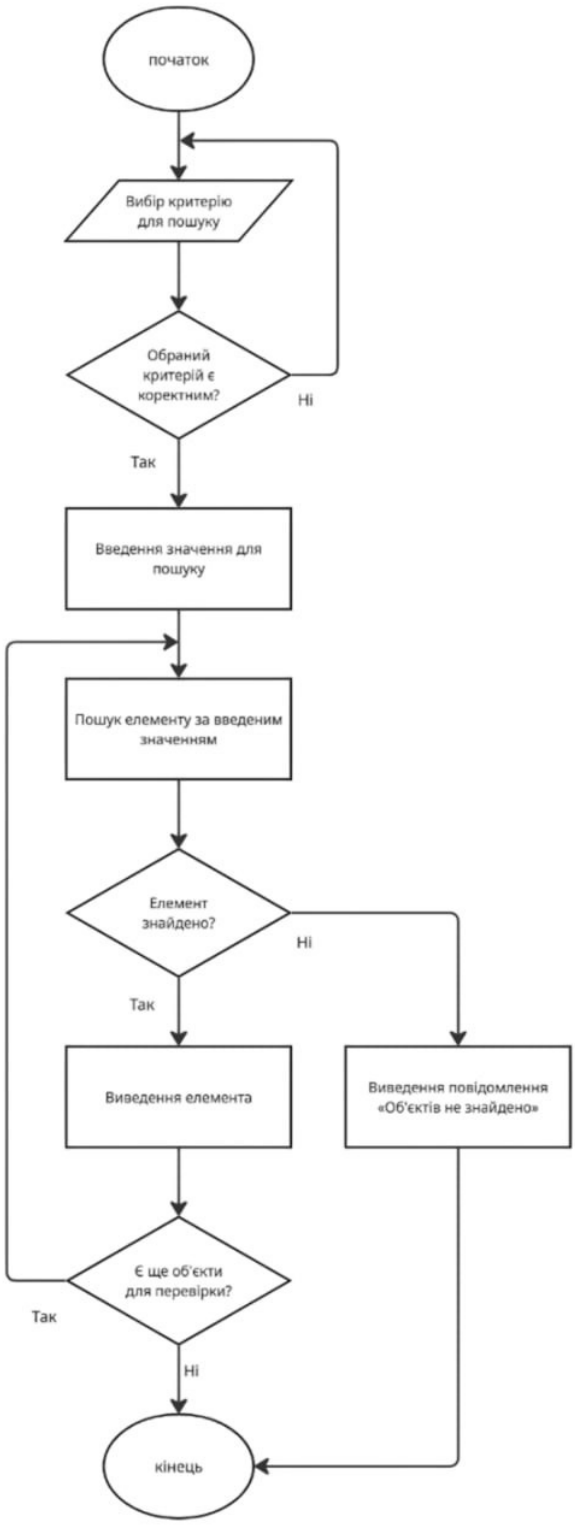

Рисунок 5 – Узагальнена блок-схема алгоритму перегляду доданих об’єктів та їх пошуку

2.2 Опис методів програми

Список методів класу IPrintable та їх опис наведено в табл.1.

Таблиця 1 – Основні методи класу IPrintable

№	Назва методу	Короткий опис
1	*Print()*	віртуальний метод, призначений для виводу елементів.
2	*ToCSV()*	віртуальний метод, для запису даних у БД

Список методів класу Vehicle та їх опис наведено в табл.2.

Таблиця 2 – Основні методи класу Vehicle

№	Назва методу	Короткий опис
1	*isOlderThan()*	метод для перевірки чи є транспортний засіб старшим за вказаний рік.
2	*isCheaperThan()*	метод для перевірки чи є транспортний засіб дешевшим за вказану ціну.
3	*age()*	Розраховує вік транспортного засобу.
4	*applyDiscount()*	Застосовує відсоткову знижку до ціни.
5	*isBrand()*	Перевіряє, чи співпадає марка авто із заданою.

Список методів класу Car та їх опис наведено в табл.3.

Таблиця 3 – Основні методи класу Car

№	Назва методу	Короткий опис
1	*isEconomicall()*	Перевіряє, чи є автомобіль економічним.
2	*isFamilyCar()*	Визначає, чи підходить автомобіль як сімейний.
3	*compareByFuel*	Порівнює цей автомобіль з іншим за витратами пального.
4	*isNewerThan*	Перевіряє, чи автомобіль новіший за вказаний рік.
5	*costPer100km*	Обчислює вартість пального для проїзду 100 км.
6	*print()*	Реалізація віртуального методу, виводу інформації, з батьківського класу.
7	*toCSV*	Реалізація віртуального методу, запису інформації, з батьківського класу.

Список методів класу Configuration та їх опис наведено в табл.4.

Таблиця 4 – Основні методи класу Configuration

№	Назва методу	Короткий опис
1	*isLuxury()*	Перевіряє, чи є комплектація люксовою.
2	*toggleAirConditioner()*	Змінює стан кондиціонера (включає/виключає).
3	*toggleMultimedia()*	Змінює стан мультимедійної системи (включає/виключає).
4	*toggleSafety()*	Змінює стан системи безпеки (включає/виключає).
5	*print()*	Реалізація віртуального методу, виводу інформації, з батьківського класу.
6	*toCSV*	Реалізація віртуального методу, запису інформації, з батьківського класу.

Список методів класу AuthManager та їх опис наведено в табл.5.

Таблиця 5 – Основні методи класу AuthManager

	Назва методу	Короткий опис
	loadUsers()	Завантажує дані користувачів з файлу 'users.txt'.
	saveUsers()	Зберігає поточні дані користувачів у файл 'users.txt'.
	login()	Запускає процес входу користувача в систему.
	isAdmin()	Перевіряє, чи є поточний користувач адміністратором.
	reauthenticate()	Дозволяє змінити поточного користувача.
	getCurrentUser()	Отримує логін поточного залогованого користувача.
	addUser()	Додає нового користувача до системи (тільки для адміністратора).
	deleteUser()	Видаляє існуючого користувача (тільки для адміністратора).
	listUsers()	Виводить список всіх зареєстрованих користувачів.

Список методів класу CarManager та їх опис наведено в табл.6.

Таблиця 6 – Основні методи класу CarManager

№	Назва методу	Короткий опис
1	*loadFromFile()*	Завантажує дані про автомобілі з файлу.
2	*saveToFile()*	Зберігає поточну колекцію автомобілів у файл.
3	*addCar()*	Ініціює процес додавання нового автомобіля через діалог з користувачем.
4	*showAllCars()*	Виводить повний список автомобілів з їх характеристиками в консоль.
5	*editCar()*	Дозволяє користувачеві редагувати дані існуючого автомобіля за індексом.
6	*removeCar()*	Видаляє автомобіль з колекції за вказаним індексом.
7	*searchCars()*	Запускає розширений пошук за декількома критеріями.
8	*sortBy(X)(ascending)* *X = (Year, Price, FuelConsumption)*	Сортує список автомобілів за (роком випуску, ціною, витратами пального) (ascending true для сортування за зростанням, false - за спаданням).
9	*safeInput()*	безпечний ввід для int
10	*safeInputDouble()*	безпечний ввід для double
11	*checker()*	перевіряє коректність даних

2.3 Програмні засоби

Розробка реалізована засобами середовища Visual Studio на мові C++. Дане середовище є потужною та інтегрованою платформою для швидкого та якісного створення додатків на ОС Windows.

Microsoft – це компанія, яка створює одне з провідних інтегрованих середовищ розробки (IDE), Visual Studio [1], що надає глибоку підтримку C++. Дане ПЗ надає комплексний набір інструментів (включно з потужним відлагоджувачем та інструментами тестування), які значно полегшують процес створення, налагодження та тестування програмного забезпечення. Для автоматизації процесу побудови Visual Studio використовує власну систему MSBuild, а також має першокласну інтеграцію з CMake [2], що дозволяє легко керувати проєктами та адаптувати їх до різних середовищ.

2.4 Опис користувацького інтерфейсу

Після запуску програми, в консоль виводитися меню авторизації користувача (рис. 6).

Рисунок 6 - меню авторизації

Після успішної авторизації, як адміністратор ми бачимо меню, яке складається з шести пунктів (рис. 7). Адміністратор може додати, видалити користувача, переглянути список користувачів, перейти до меню автомобілів, змінити користувача та вийти з програми.

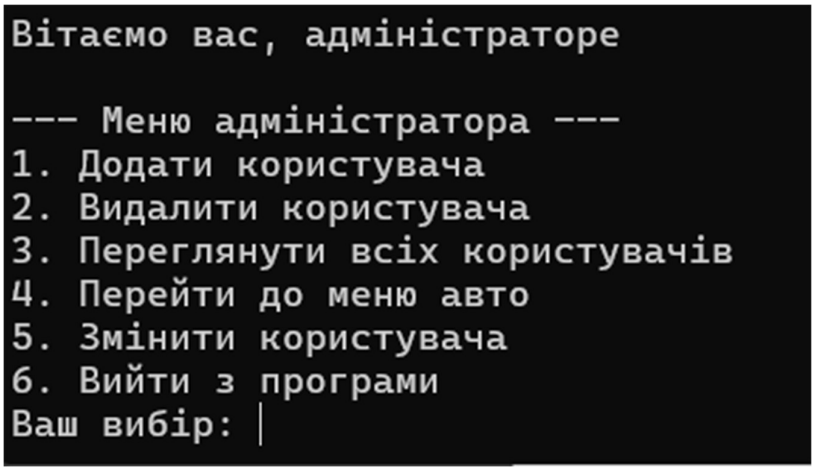

Рисунок 7 – меню адміністратора

Обравши пункт 4, консоль виводить меню з автомобілями, яке складається з 23 пунктів (рис. 8).

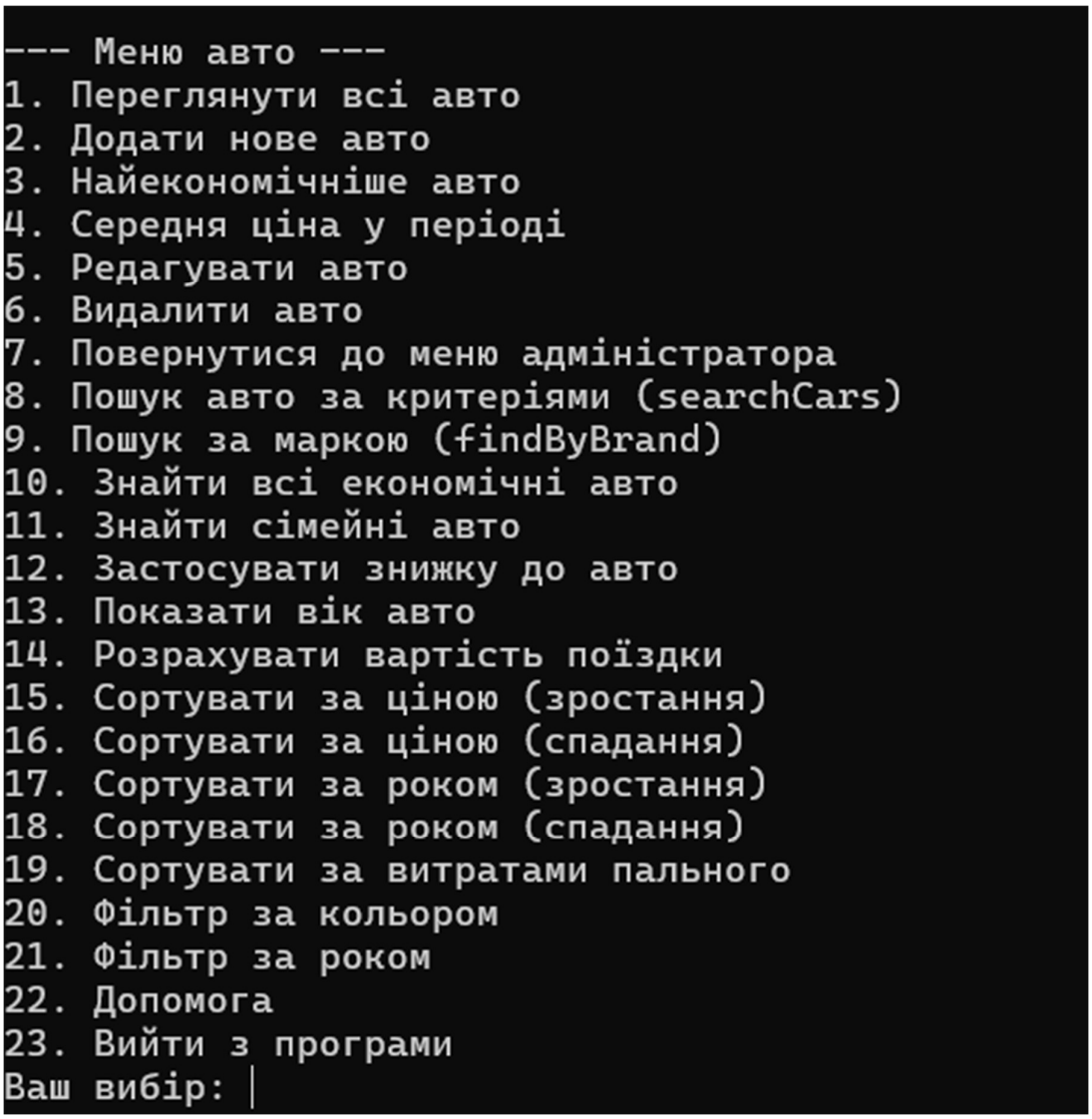

Рисунок 8 – меню автомобілів

Обравши пункт 22 меню, виводиться інструкція користування програмою (рис. 9) виводиться довідка.

```
================ Довідка для адміністратора ================

--> Пояснення роботи програми <--
Ця програма є системою управління автосалоном. Вона дозволяє
адміністраторам та користувачам працювати зі спільною базою даних автомобілів,
що зберігається у файлі 'cars.csv'.
Адміністратор має повний доступ до всіх функцій, включаючи керування
користувачами та редагування каталогу авто.

--> Правила вводу даних <--
1. При введенні числових даних (ціна, рік, кількість дверей) використовуйте
   лише цифри. Введення тексту призведе до помилки.
2. При введенні тексту (марка, модель, колір) уникайте пробілів.
   Наприклад, вводьте 'Volkswagen', а не 'Volks wagen'.
3. Для позначення 'так' або 'ні' (наприклад, наявність кондиціонера)
   використовуйте 1 для 'так' і 0 для 'ні'.

--> Короткий опис команд <--

--- Керування каталогом ---
1.  Переглянути всі авто  - Виводить повний список автомобілів у базі.
2.  Додати нове авто      - Запускає процес додавання нового авто в каталог.
3.  Редагувати авто       - Дозволяє змінити дані існуючого автомобіля за його індексом.
5.  Видалити авто         - Видаляє автомобіль з бази за його індексом.
12. Застосувати знижку    - Застосовує відсоткову знижку до ціни обраного авто.

--- Аналітика та пошук ---
3.  Найекономічніше авто  - Знаходить авто з найменшими витратами пального.
4.  Середня ціна у періоді - Розраховує середню ціну авто за вказаний період років.
5.  Пошук за критеріями   - Розширений пошук за маркою, кольором та роком.
9.  Пошук за маркою       - Швидкий пошук та фільтрація всіх авто конкретної марки.
13. Показати вік авто     - Визначає вік обраного автомобіля.
14. Розрахувати вартість  - Розраховує вартість поїздки на задану відстань.

--- Сортування та фільтрація ---
15-18. Сортувати          - Сортує список авто за ціною або роком (за зростанням/спаданням).
19-21. Фільтрувати        - Показує авто, що відповідають заданому кольору або року.

--- Керування акаунтом ---
7.  Меню адміністратора   - Повернення до головного меню адміністратора (керування користувачами).
23. Вийти з програми      - Зберігає всі зміни та завершує роботу програми.
============================================================
```

Рисунок 9 – інструкція з користування програмою

Обравши пункт 1 головного меню, користувачу буде виведено весь список автомобілів (рис 10)

```
--- Всі автомобілі ---
Марка: Supra, Модель: 1, Колір: white, Витрати бензину: 14 л/100км, Кількість дверей: 4, Рік: 1999, Ціна: 1900 грн
Configuration: Lux | AC: Yes | Multimedia: Yes | Safety: Yes | Price: 100
Марка: Supra, Модель: 2, Колір: white, Витрати бензину: 11 л/100км, Кількість дверей: 4, Рік: 2000, Ціна: 2000 грн
Configuration: Luxary | AC: Yes | Multimedia: Yes | Safety: Yes | Price: 500
Марка: Supra, Модель: 2, Колір: white, Витрати бензину: 11 л/100км, Кількість дверей: 4, Рік: 2000, Ціна: 2000 грн
Configuration: Luxary | AC: Yes | Multimedia: Yes | Safety: Yes | Price: 0
Марка: Supra, Модель: 4, Колір: White, Витрати бензину: 10 л/100км, Кількість дверей: 4, Рік: 2004, Ціна: 4000 грн
Configuration: Luxary | AC: Yes | Multimedia: Yes | Safety: Yes | Price: 100
```

Рисунок 10 – перегляд всіх автомобілів

Обравши пункт 2 меню автомобілів, користувач зможе добавити автомобіль(рис 11).

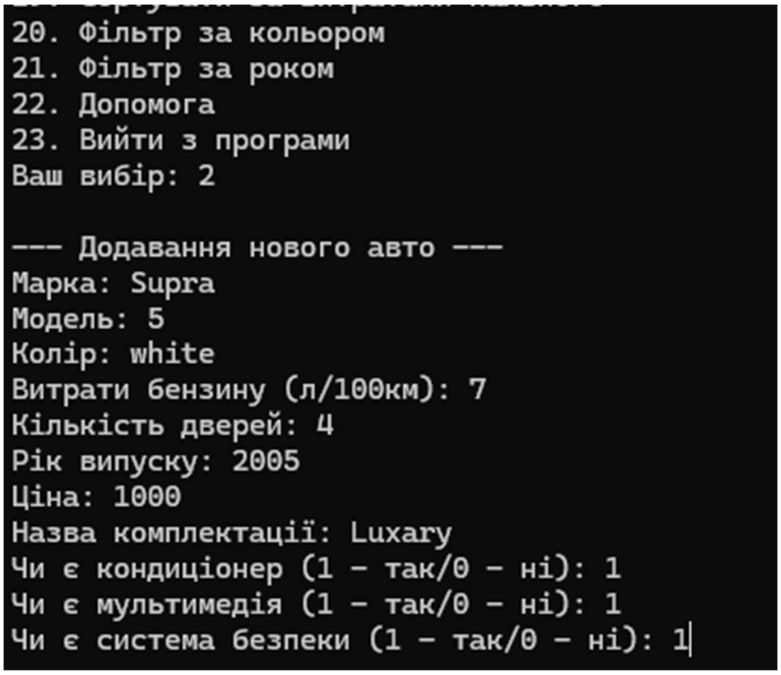

Рисунок 11 – додавання авто

Обравши пункт 6 меню автомобілів, користувач зможе видалити авто (рис 12). Після, потрібно ввести ідентифікатор авто для видалення(рис. 13).

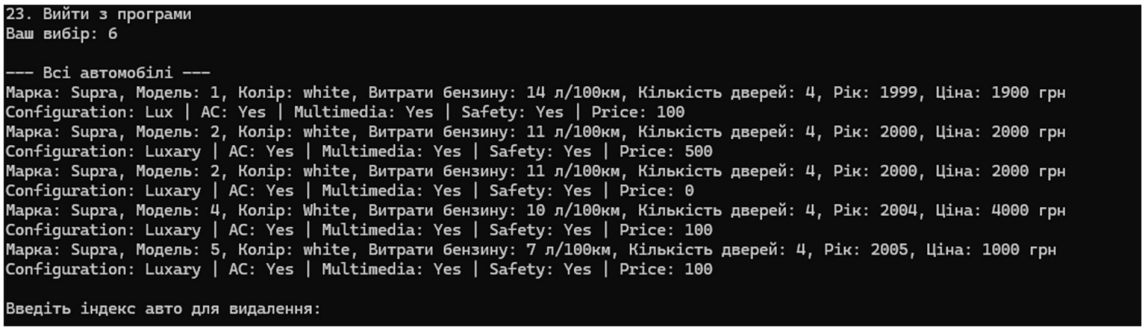

Рисунок 12 – видалення авто

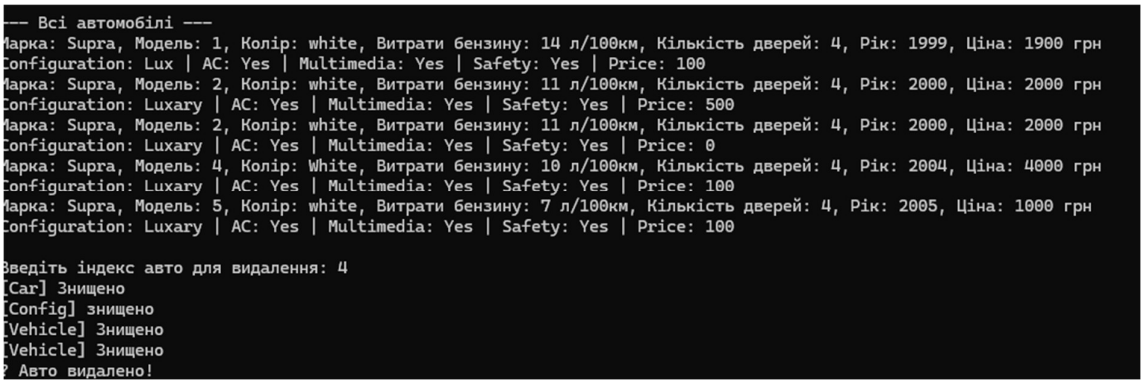

Рисунок 13 – видалення

Обравши пункт 8 меню автомобілів, користувач зможу шукати авто за багатьма критеріями(рис 14). Обравши пункт 10 меню автомобілів, у користувача буде змога вивести на всі економічні, тобто авто в яких розхід менший аніж той що задав користувач(рис 15).

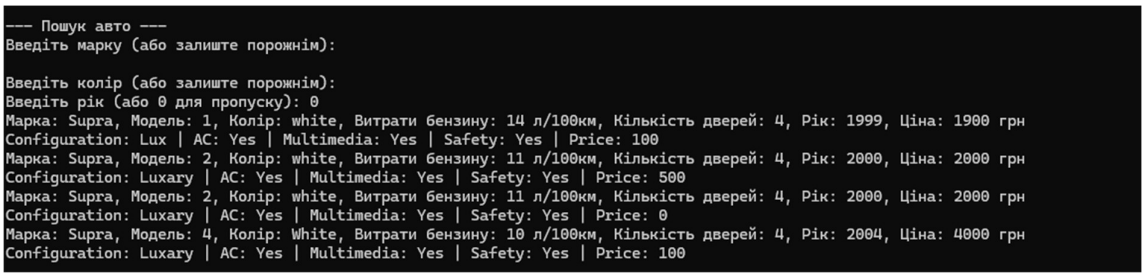

Рисунок 14 – пошук авто за критеріями

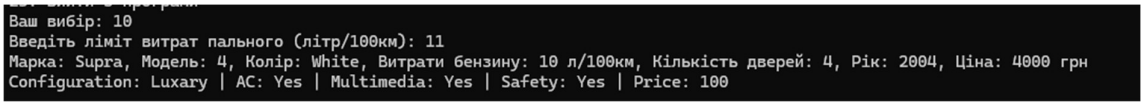

Рисунок 15 – пошук економічного авто

Обравши пункт 12 меню автомобілів, користувач може вивести на екран яка ціна буде автомобіля зі знижкою яка буде введена(рис 16). Обравши пункт 14, користувач зможе розрахувати вартість поїздки на обраному авто(рис 17). Обравши пункт 15-19, користувач зможе вивести на екран відсортований список авто за різними критеріями, такими як ціна(зростання та спадання), рік(зростання та спадання) а також сортувати за витратами пального у спадані(рис 18).

Рисунок 16 – Знижка на авто

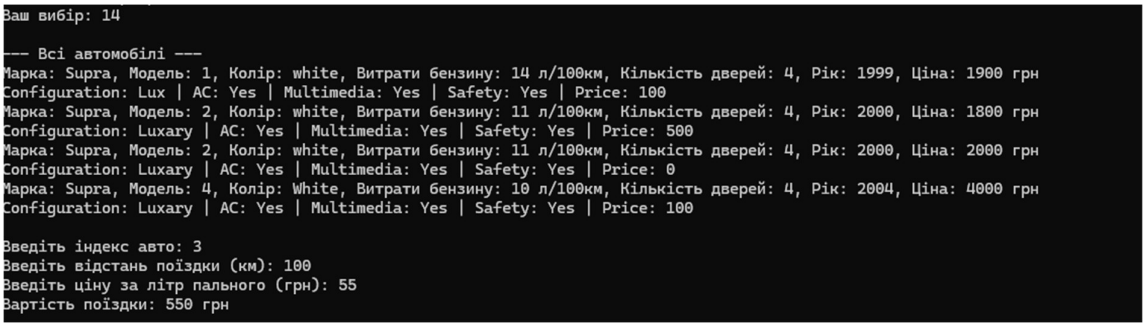

Рисунок 17 – вартість поїздки

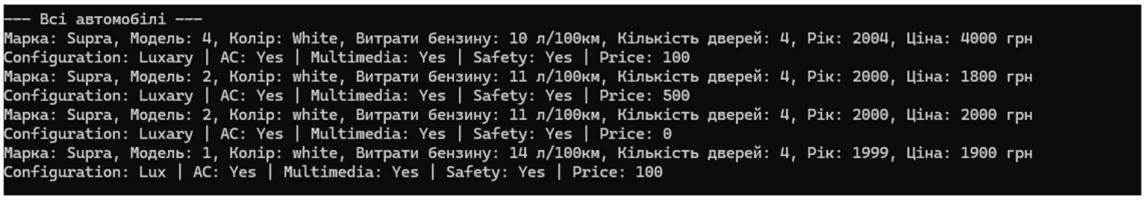

Рисунок 18 – відсортований список авто за витгратами пального

Обравши пункт 20 головного меню, користувачеві буде надано можливість відфільтрувати всі авто за кольором(рис 19). Обравши пункт 21 користувач зможе відфільтрувати за роком(рис. 20)

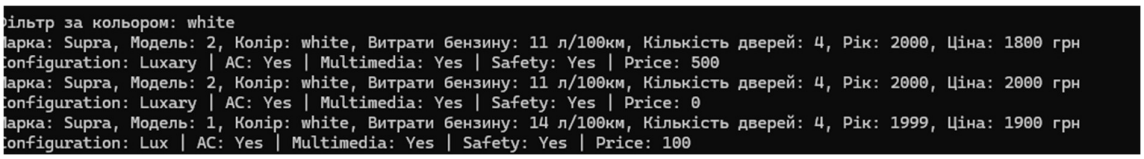

Рисунок 19 – фільтраціяза кольором

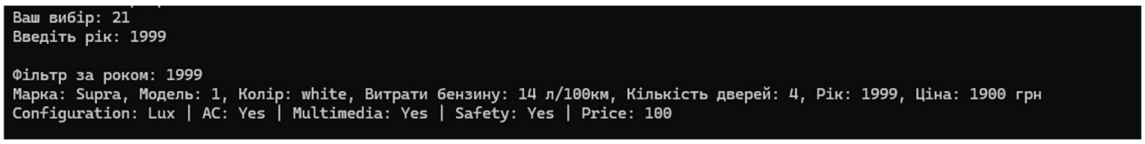

Рисунок 20 – фільтрація за роком

Якщо це буде простий користувач без прав адміністратора в нього буде трішки менший функціонал(рис. 21)

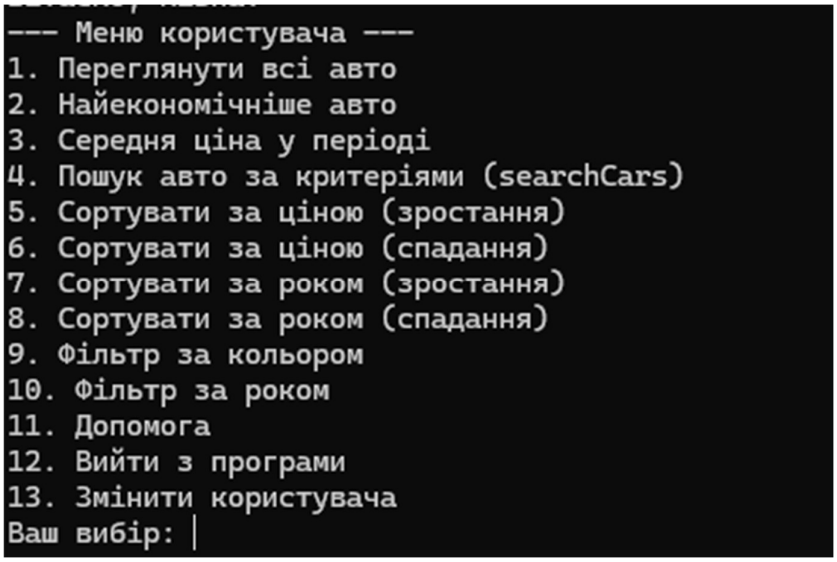

Рисунок 21 – функціонал простого користувача

2.5 Тестування

При введені інформації при авторизації про користувача якого не існує викидає помилку (рис.22)

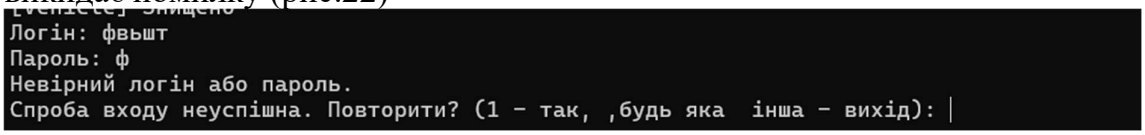

Рисунок 22 – некоректна інформація при авторизації

При додаванні та редагуванні авто якщо в поле дата ввести буквене значення то викидає помилку (рис. 23)

Рисунок 23- некоректне значення в полі рік

При введені часового проміжку в роках для сортування, якщо ввести некоректне значення викидає помилку (рис.24)

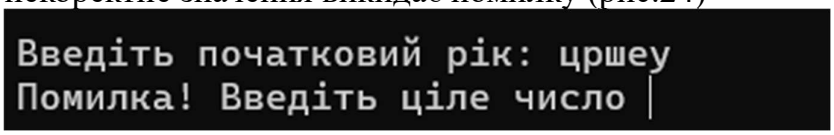

Рисунок 24- помилка при сортуванні

При введені некоретного значення в головному меню автомобілів так само як в меню автомобілів не для адміністратора видає повідомлення про некоректність вводу (рис.25)

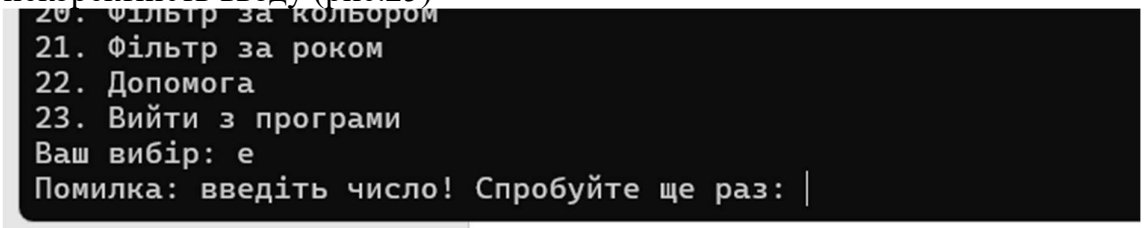

Рисунок 25- некоректність вводу в меню авто

При розрахунку ціни поїздки введення буков у поле кількість кілометрів та ціна палива викидає виняток (рис.26)

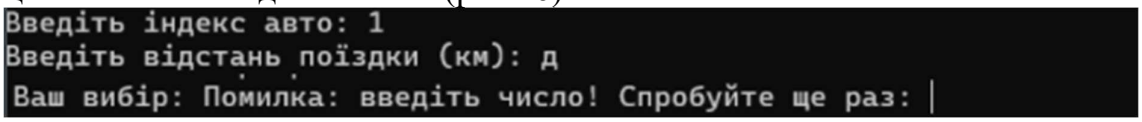

Рисунок 26- буквений вираз у функції розрахунок ціни поїздки

Прівведені буквеного значення у поле рік при фільтрації за роком програма йде у захист і не виключаєть, видається повідомлення (рис.28) про помилку (рис.27)

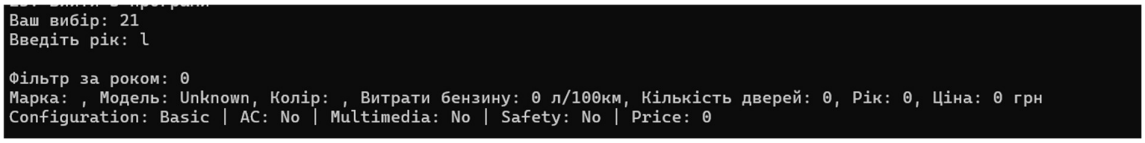

Рисунок 27 – помилка при фільтрації

Рисунок 28- повідомлення про помилку

3 ВИСНОВКИ

У результаті виконання курсового проєкту розроблена система управління базою автомобілів для комп’ютерів, які працюють під керуванням ОС Windows.

У розробленому програмному забезпечені передбачено:

1) Подання даних користувачем у консолі.
2) Перевірку коректності введення даних та обробку виключень.
3) Збереження даних у файл.
4) Зчитування даних з файлу.
5) Перегляд доданих об’єктів.
6) Редагування доданих об’єктів.
7) Видалення об’єктів.
8) Сортування об’єктів по самостійно вибраному ключу.
9) Фільтрацію об’єктів по самостійно вибраному ключу.
10) Пошук об’єктів по самостійно обраному ключу.

Дана розробка у майбутньому може бути розширена із додаванням нового функціоналу і видозміненою логікою обробки.

4 СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

Microsoft Learn: https://learn.microsoft.com/en-us/cpp

cppreference.com -https://en.cppreference.com/

Doxygen Manual - https://www.doxygen.nl/manual/index.html

STL documentation - https://cplusplus.com/reference

Visual Studio Code - https://code.visualstudio.com.

Bjarne Stroustrup, "The C++ Programming Language," 4th Edition, Addison-Wesley, 2013.

Scott Meyers, "Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14," O'Reilly Media, 2014.

Herb Sutter, "Exceptional C++: 47 Engineering Puzzles, Programming Problems, and Solutions," Addison-Wesley, 2000.c

5 ДОДАТКИ

ДОДАТОК А. Скролінг (текст) програми

Програмний модуль “ConsoleAplication.cpp”

Вміст “ConsoleAplication.cpp”

```
#include "CarManager.h"
#include "AuthManager.h"
#include <windows.h>
#include <iostream>
#include <stdlib.h>
#include <limits>
using namespace std;

int safeInput() {
    int value;
    while (true) {
        cin >> value;

        if (cin.fail()) {
            cin.clear();
            cin.ignore((std::numeric_limits<std::streamsize>::max)(), '\n');
            cout << "Помилка: введіть число! Спробуйте ще раз: ";
        }
        else {
            cin.ignore((std::numeric_limits<std::streamsize>::max)(), '\n');
            return value;
        }
    }
}



int main() {
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    AuthManager auth;
    CarManager manager;
    manager.loadFromFile("cars.csv");

    while (true) {
        if (!auth.login()) {
            cout << "Спроба входу неуспішна. Повторити? (1 - так, ,будь яка  інша - вихід): ";
            int retry; cin >> retry;
            if (retry != 1) break;
```

```
        continue;
    }
    bool activeSession = true;

    while (activeSession) {

        if (auth.isAdmin()) {
            cout << "\nВітаємо вас, адміністраторе\n";
            cout << "\n--- Меню адміністратора ---\n";
            cout << "1. Додати користувача\n";
            cout << "2. Видалити користувача\n";
            cout << "3. Переглянути всіх користувачів\n";
            cout << "4. Перейти до меню авто\n";
            cout << "5. Змінити користувача\n";
            cout << "6. Вийти з програми\n";
            cout << "Ваш вибір: ";
            int choice;
            choice = safeInput();

            switch (choice) {
            case 1: auth.addUser(); break;
            case 2: auth.deleteUser(); break;
            case 3: auth.listUsers(); break;
            case 4: {
                bool carMenuActive = true;
                while (carMenuActive) {
                    cout << "\n--- Меню авто ---\n";
                    cout << "1. Переглянути всі авто\n";
                    cout << "2. Додати нове авто\n";
                    cout << "3. Найекономічніше авто\n";
                    cout << "4. Середня ціна у періоді\n";
                    cout << "5. Редагувати авто\n";
                    cout << "6. Видалити авто\n";
                    cout << "7. Повернутися до меню адміністратора\n";
                    cout << "8. Пошук авто за критеріями (searchCars)\n";
                    cout << "9. Пошук за маркою (findByBrand)\n";
                    cout << "10. Знайти всі економічні авто\n";
                    cout << "11. Знайти сімейні авто\n";
                    cout << "12. Застосувати знижку до авто\n";
                    cout << "13. Показати вік авто\n";
                    cout << "14. Розрахувати вартість поїздки\n";
                    cout << "15. Сортувати за ціною (зростання)\n";
                    cout << "16. Сортувати за ціною (спадання)\n";
                    cout << "17. Сортувати за роком (зростання)\n";
                    cout << "18. Сортувати за роком (спадання)\n";
                    cout << "19. Сортувати за витратами пального\n";
                    cout << "20. Фільтр за кольором\n";
                    cout << "21. Фільтр за роком\n";
                    cout << "22. Допомога\n";
                    cout << "23. Вийти з програми\n";
                    cout << "Ваш вибір: ";
                    int carChoice;
                    carChoice = safeInput();
```

```
            switch (carChoice) {
            case 1: manager.showAllCars(); break;
            case 2: manager.addCar(); break;
            case 3: manager.findMostEconomicalCar(); break;
            case 4: manager.averagePriceInPeriod(); break;
            case 5: manager.editCar(); break;
            case 6: manager.removeCar(); break;
            case 7: carMenuActive = false; break;
            case 8: manager.searchCars(); break;
            case 9: manager.findByBrand(); break;
            case 10: manager.findEconomicalCars(); break;
            case 11: manager.findFamilyCars(); break;
            case 12: manager.applyDiscountToCar(); break;
            case 13: manager.showCarAge(); break;
            case 14: manager.calcTripCost(); break;
            case 15: manager.sortByPrice(true); break;
            case 16: manager.sortByPrice(false); break;
            case 17: manager.sortByYear(true); break;
            case 18: manager.sortByYear(false); break;
            case 19: manager.sortByFuelConsumption(true); break;
            case 20: {
               string color;
               cout << "Введіть колір: "; cin >> color;
               manager.filterByColor(color);
               break;
            }
            case 21: {
               int year;
               cout << "Введіть рік: "; cin >> year;
               manager.filterByYear(year);
               break;
            }
            case 22: manager.showHelp(); break;
            case 23: manager.saveToFile("cars.csv"); return 0;
            default: cout << "Невірний вибір.\n"; break;
            }
         }
         break;
      }
      case 5: activeSession = false; break;
      case 6: manager.saveToFile("cars.csv"); return 0;
      default: cout << "Невірний вибір.\n"; break;
      }
   }
   else {
      cout << "Вітаємо, " << auth.getCurrentUser() << "!\n";
      cout << "--- Меню користувача ---\n";
      cout << "1. Переглянути всі авто\n";
      cout << "2. Найекономічніше авто\n";
      cout << "3. Середня ціна у періоді\n";
      cout << "4. Пошук авто за критеріями (searchCars)\n";
      cout << "5. Сортувати за ціною (зростання)\n";
```

```
            cout << "6. Сортувати за ціною (спадання)\n";
            cout << "7. Сортувати за роком (зростання)\n";
            cout << "8. Сортувати за роком (спадання)\n";
            cout << "9. Фільтр за кольором\n";
            cout << "10. Фільтр за роком\n";
            cout << "11. Допомога\n";
            cout << "12. Вийти з програми\n";
            cout << "13. Змінити користувача\n";
            cout << "Ваш вибір: ";

            int choice;
            choice = safeInput();

            switch (choice) {
            case 1: manager.showAllCars(); break;
            case 2: manager.findMostEconomicalCar(); break;
            case 3: manager.averagePriceInPeriod(); break;
            case 4: manager.searchCars(); break;
            case 5: manager.sortByPrice(true); break;
            case 6: manager.sortByPrice(false); break;
            case 7: manager.sortByYear(true); break;
            case 8: manager.sortByYear(false); break;
            case 9: {
               string color;
               cout << "Введіть колір: "; cin >> color;
               manager.filterByColor(color);
               break;
            }
            case 10: {
               int year;
               cout << "Введіть рік: "; cin >> year;
               manager.filterByYear(year);
               break;
            }
            case 11: manager.showHelpnotadm(); break;
            case 12: manager.saveToFile("cars.csv"); return 0;
            case 13: activeSession = false; break;
            default: cout << "Невірний вибір.\n"; break;
            }
         }
      }

   }

   return 0;
}
```

Програмний модуль “AuthManager”

Вміст “AuthManager.h”

```
#pragma once
#include <vector>
#include <string>
#include <unordered_map>

/**
 * @class AuthManager
 * @brief Клас для керування автентифікацією та обліковими записами користувачів.
 * @details Відповідає за завантаження, збереження, перевірку та керування
 * даними користувачів, включаючи логін, пароль та права доступу (адміністратор).
 */
class AuthManager {
public:
    /**
     * @brief Конструктор за замовчуванням.
     * @details Ініціалізує менеджер та завантажує користувачів з файлу 'users.txt'.
     */
    AuthManager();

    /**
     * @brief Конструктор з параметрами для ініціалізації з вектора.
     * @param users Вектор рядків у форматі "username:password".
     */
    AuthManager(const std::vector<std::string>& users);

    /**
     * @brief Конструктор копіювання.
     */
    AuthManager(const AuthManager& other);

    /**
     * @brief Конструктор переміщення.
     */
    AuthManager(AuthManager&& other) noexcept;

    /**
     * @brief Деструктор.
     * @details Автоматично зберігає всі зміни у файлі користувачів при знищенні об'єкта.
     */
    ~AuthManager();

    /**
     * @brief Завантажує дані користувачів з файлу 'users.txt'.
     */
    void loadUsers();

    /**
     * @brief Зберігає поточні дані користувачів у файл 'users.txt'.
```

```
    */
    void saveUsers();

    /**
     * @brief Запускає процес входу користувача в систему.
     * @return true, якщо вхід успішний, інакше false.
     */
    bool login();

    /**
     * @brief Перевіряє, чи є поточний користувач адміністратором.
     * @return true, якщо поточний користувач - 'admin', інакше false.
     */
    bool isAdmin() const;

    /**
     * @brief Дозволяє змінити поточного користувача.
     * @return true, якщо вхід нового користувача успішний, інакше false.
     */
    bool reauthenticate();

    /**
     * @brief Отримує логін поточного залогованого користувача.
     * @return Рядок з логіном поточного користувача.
     */
    std::string getCurrentUser() const;

    /**
     * @brief Додає нового користувача до системи (тільки для адміністратора).
     */
    void addUser();

    /**
     * @brief Видаляє існуючого користувача (тільки для адміністратора).
     */
    void deleteUser();

    /**
     * @brief Виводить список всіх зареєстрованих користувачів.
     */
    void listUsers();

private:
    std::string filename;
    std::unordered_map<std::string, std::string> users;
    std::string currentUser;
    int deleteAttempts;
};
```

Вміст “AuthManager.cpp”

```
#include "AuthManager.h"
#include <iostream>
#include <fstream>
#include <sstream>
#include <stdexcept>
#include <unordered_map>

using namespace std;

AuthManager::AuthManager() : filename("users.txt"), deleteAttempts(0) {
    loadUsers();
}

AuthManager::AuthManager(const vector<string>& usersVec) {
    for (const auto& entry : usersVec) {
        auto pos = entry.find(':');
        if (pos != string::npos) {
            string username = entry.substr(0, pos);
            string password = entry.substr(pos + 1);
            users[username] = password;
        }
    }
    saveUsers();
}

AuthManager::AuthManager(const AuthManager& other)
    : filename(other.filename), users(other.users), currentUser(other.currentUser), deleteAttempts(other.deleteAttempts) {
}

AuthManager::AuthManager(AuthManager&& other) noexcept
    : filename(move(other.filename)), users(move(other.users)), currentUser(move(other.currentUser)),
deleteAttempts(other.deleteAttempts) {
}


AuthManager::~AuthManager() {
    saveUsers();
    cout << "[authmanager] знищено\n";
}

void AuthManager::loadUsers() {
    try {
        ifstream file(filename);
        if (!file.is_open()) throw runtime_error("Неможливов відкрити users.txt");
        string line;
        while (getline(file, line)) {
            istringstream iss(line);
            string username, password;
            if (getline(iss, username, ':') && getline(iss, password)) {
                users[username] = password;
            }
```

```
        }
        file.close();

        // If file is empty, add default admin
        if (users.empty()) {
            users["admin"] = "admin123";
            saveUsers();
        }
    }
    catch (const exception& ex) {
        cerr << "Помилка завантаження користувачів " << ex.what() << endl;
        users.clear();
        users["admin"] = "admin123";
        saveUsers();
    }
}

void AuthManager::saveUsers() {
    try {
        ofstream file(filename);
        if (!file.is_open()) throw runtime_error("Неможливо відкрити users.txt для запису");
        for (const auto& pair : users) {
            file << pair.first << ':' << pair.second << '\n';
        }
        file.close();
    }
    catch (const exception& ex) {
        cerr << "Помилка збереження користувачів " << ex.what() << endl;
    }
}

bool AuthManager::login() {
    try {
        if (users.empty()) throw runtime_error("users.txt пустий");
        if (users.count("admin") == 0)
            throw runtime_error("Фдмін не знайдено");
    }
    catch (const exception& ex) {
        cerr << "Помилка входу " << ex.what() << endl;
        return false;
    }

    string username, password;
    cout << "Логін: ";
    cin >> username;
    cout << "Пароль: ";
    cin >> password;

    auto it = users.find(username);
    if (it != users.end() && it->second == password) {
        currentUser = username;
        return true;
    }
```

```
    else {
        cout << "Невірний логін або пароль.\n";
        return false;
    }
}

bool AuthManager::isAdmin() const {
    return currentUser == "admin";
}

string AuthManager::getCurrentUser() const {
    return currentUser;
}

void AuthManager::addUser() {
    try {
        string username, password;
        cout << "Новий логін: ";
        cin >> username;
        if (users.count(username)) {
            cout << "Такий користувач вже існує.\n";
            return;
        }
        cout << "Пароль: ";
        cin >> password;
        users[username] = password;
        saveUsers();
        cout << "Користувача додано.\n";
    }
    catch (const exception& ex) {
        cerr << "Помилка додавання користувача " << ex.what() << endl;
            }
}

void AuthManager::deleteUser() {
    string username;
    cout << "Введіть логін для видалення: ";
    cin >> username;
    if (username == "admin") {
        deleteAttempts += 1;
        switch (deleteAttempts) {
        case 1: cout << "Неможливо видалити адміністратора.\n"; return;
        case 2: cout << "Неможна це робити програма капут якщо адмін видалити\n"; return;
        case 3: cout << "Я ж сказав заборонено машина розумніша і знає як краще\n"; return;
        default: cout << "Відчепися глек\n"; return;
        }
    }
    if (users.erase(username)) {
        saveUsers();
        cout << "Користувача видалено.\n";
    }
    else {
        cout << "Користувача не знайдено.\n";
```

```
    }
}

void AuthManager::listUsers() {
    cout << "\nСписок користувачів:\n";
    for (const auto& pair : users) {
        cout << pair.first << ':' << pair.second << '\n';
    }
}

bool AuthManager::reauthenticate() {
    cout << "\n--- Зміна користувача ---\n";
    return login();
}
```

Програмний модуль “Car”

Вміст “Car.h”

```
#pragma once
#include "Vehicle.h"
#include "Configuration.h"
#include <sstream>

/**
 * @class Car
 * @brief Клас, що представляє автомобіль.
 * @details Є нащадком класу Vehicle і містить специфічну для авто
 * інформацію: витрати пального, кількість дверей та комплектацію.
 */
class Car : public Vehicle {
public:
    // --- Конструктори та деструктор ---
    Car();
    Car(const std::string& b, const std::string& c, const std::string& m, double fuel, int doors, int y, double p, const Configuration& config);
    Car(const Car& other);
    Car(Car&& other) noexcept;
    ~Car();

    // --- Getters ---
    double getFuelConsumption() const;
    int getDoorCount() const;
    std::string getModel() const;

    // --- Setters ---
    void setFuelConsumption(double f);
    void setDoorCount(int d);
    void setModel(const std::string& m);

    /**
     * @brief Перевіряє, чи є автомобіль економічним.
     * @param limit Поріг витрат пального у літрах на 100 км (L/100km).
     * @return true, якщо витрати пального цього автомобіля менші або рівні @p limit; інакше false.
     */
    bool isEconomical(double limit) const;

    /**
     * @brief Визначає, чи підходить автомобіль як сімейний.
     * @details У прикладі реалізації можуть використовуватися критерії: кількість дверей (наприклад, >=4),
     * наявність систем безпеки в конфігурації тощо.
     * @return true, якщо автомобіль вважається сімейним згідно з критеріями; інакше false.
     */
    bool isFamilyCar() const;

    /**
     * @brief Порівнює цей автомобіль з іншим за витратами пального.
```

```
    * @param other Інший автомобіль для порівняння.
    * @return true, якщо цей автомобіль має менші витрати пального (краща економія), ніж @p other; інакше false.
    */
   bool compareByFuel(const Car& other) const;

   /**
    * @brief Перевіряє, чи автомобіль новіший за вказаний рік.
    * @param y Рік для порівняння.
    * @return true, якщо рік випуску автомобіля строго більший за @p y; інакше false.
    */
   bool isNewerThan(int y) const;

   /**
    * @brief Обчислює вартість пального для проїзду 100 км.
    * @param fuelPrice Ціна пального за літр (у тій же валюті, що й ціна автомобіля).
    * @return Оцінкова вартість проїзду 100 км, обчислена як fuelConsumption (L/100km) * @p fuelPrice (валюта/літр).
    */
   double costPer100km(double fuelPrice) const;

   /**
    * @brief Повертає вік автомобіля в повних роках.
    * @details Вік обчислюється відносно поточного календарного року.
    * @return Кількість повних років від року випуску автомобіля до поточного року.
    */
   int getAge() const;

   // --- Реалізація абстрактних методів ---
   void print() const override;
   std::string toCSV() const override;

   /**
    * @brief Створює об'єкт Car з рядка CSV.
    * @param line Рядок у форматі CSV.
    * @return Новий об'єкт Car.
    */
   static Car fromCSV(const std::string& line);

private:
   Configuration config;
   double fuelConsumption;
   int doorCount;
   std::string model;
};
```

Вміст “Car.cpp”

```
#include "Car.h"
using namespace std;

Car::Car()
    : Vehicle(), model("Unknown"), fuelConsumption(0.0), doorCount(0), config() {
}

Car::Car(const string& b, const string& c, const string& m,
    double fuel, int doors, int y, double p, const Configuration& cfg)
    : Vehicle(b, c, y, p, m),
    model(m),
    fuelConsumption(fuel),
    doorCount(doors),
    config(cfg) {
}

Car::Car(const Car& other)
    : Vehicle(other),
    model(other.model),
    fuelConsumption(other.fuelConsumption),
    doorCount(other.doorCount),
    config(other.config) {
}

Car::Car(Car&& other) noexcept
    : Vehicle(move(other)),
    model(move(other.model)),
    fuelConsumption(other.fuelConsumption),
    doorCount(other.doorCount),
    config(move(other.config)) {
}

Car::~Car() {
    cout << "[Car] Знищено " << endl;
}

void Car::print() const {
    cout << "Марка: " << getBrand()
        << ", Модель: " << model
        << ", Колір: " << getColor()
        << ", Витрати бензину: " << fuelConsumption << " л/100км"
        << ", Кількість дверей: " << doorCount
        << ", Рік: " << getYear()
        << ", Ціна: " << getPrice() << " грн" << endl;
    config.print();
}

Car Car::fromCSV(const string& line) {
    try {
        istringstream iss(line);
```

```
        string brand, color, model;
        double fuelConsumption, price;
        int doorCount, year;

        getline(iss, brand, ',');
        getline(iss, color, ',');
        getline(iss, model, ',');
        iss >> fuelConsumption; iss.ignore();
        iss >> doorCount; iss.ignore();
        iss >> year; iss.ignore();
        iss >> price;

        if (iss.peek() == ',') iss.get(); // пропускаємо кому
        string configData;
        getline(iss, configData);

        Configuration cfg;
        if (!configData.empty()) {
            cfg = Configuration::fromCSV(configData);
        }

        return Car(brand, color, model, fuelConsumption, doorCount, year, price, cfg);

    }
    catch (const exception& e) {
        cerr << "Помилка при розборі CSV рядка: " << e.what() << endl;
        return Car();
                }
}

string Car::toCSV() const {
        ostringstream oss;
        oss << getBrand() << "," << getColor() << "," << model << ","
            << fuelConsumption << "," << doorCount << ","
            << getYear() << "," << getPrice() << ","
            << config.toCSV();
        return oss.str();

}

// Getters
double Car::getFuelConsumption() const { return fuelConsumption; }
int Car::getDoorCount() const { return doorCount; }

// Setters
void Car::setFuelConsumption(double f) { fuelConsumption = f; }
void Car::setDoorCount(int d) { doorCount = d; }

bool Car::isEconomical(double limit) const { return fuelConsumption < limit; }
bool Car::isFamilyCar() const { return doorCount >= 4; }
bool Car::compareByFuel(const Car& other) const { return fuelConsumption < other.fuelConsumption; }
bool Car::isNewerThan(int y) const { return getYear() > y; }
double Car::costPer100km(double fuelPrice) const { return (fuelConsumption / 100.0) * fuelPrice * 100; }
```

```
int Car::getAge() const {
    time_t now = time(0);
    tm ltm;
    localtime_s(&ltm, &now);
    int currentYear = 1900 + ltm.tm_year;
    return currentYear - getYear();
}

string Car::getModel() const { return model; }
void Car::setModel(const string& m) { model = m; }
```

Програмний модуль “CarManager”

Вміст “CarManager.h”

```
#pragma once
#include <vector>
#include <memory>
#include <string>
#include "Car.h"

/**
 * @class CarManager
 * @brief Менеджер-клас для керування колекцією автомобілів.
 * @details Реалізує всю бізнес-логіку програми: CRUD-операції,
 * завантаження/збереження, пошук, сортування, фільтрацію та аналітику.
 * Є центральним компонентом, що взаємодіє з користувачем.
 */
class CarManager {
public:
    CarManager();
    CarManager(const CarManager& other);
    CarManager(CarManager&& other) noexcept;
    ~CarManager();

    /**
     * @brief Завантажує дані про автомобілі з файлу.
     * @param filename Ім'я файлу для завантаження (напр., "cars.csv").
     */
    void loadFromFile(const std::string& filename);

    /**
     * @brief Зберігає поточну колекцію автомобілів у файл.
     * @param filename Ім'я файлу для збереження.
     */
    void saveToFile(const std::string& filename);

    /**
     * @brief Ініціює процес додавання нового автомобіля через діалог з користувачем.
     */
    void addCar();

    /**
     * @brief Виводить повний список автомобілів з їх характеристиками в консоль.
     */
    void showAllCars() const;

    /**
     * @brief Дозволяє користувачеві редагувати дані існуючого автомобіля за індексом.
     */
    void editCar();

    /**
```

```
 * @brief Видаляє автомобіль з колекції за вказаним індексом.
 */
void removeCar();

// --- Аналітика та розрахунки ---
void findMostEconomicalCar() const;
void averagePriceInPeriod() const;
        // --- Методи для додаткової інформації по окремих авто ---
void calcTripCost() const;
void showCarAge() const;
void applyDiscountToCar();

// --- Сортування ---

/**
 * @brief Сортує список автомобілів за ціною.
 * @param ascending true для сортування за зростанням, false - за спаданням.
 */
void sortByPrice(bool ascending = true);

/**
 * @brief Сортує список автомобілів за роком випуску.
 * @param ascending true для сортування за зростанням, false - за спаданням.
 */
void sortByYear(bool ascending = true);

/**
 * @brief Сортує список автомобілів за витратами пального.
 * @param ascending true для сортування за зростанням, false - за спаданням.
 */
void sortByFuelConsumption(bool ascending = true);

// --- Пошук та фільтрація ---

/**
 * @brief Фільтрує та виводить автомобілі за вказаною маркою.
 * @param b Марка для фільтрації.
 */
void filterByBrand(const std::string& b) const;

/**
 * @brief Фільтрує та виводить автомобілі за вказаним кольором.
 * @param c Колір для фільтрації.
 */
void filterByColor(const std::string& c) const;

/**
 * @brief Фільтрує та виводить автомобілі за вказаним роком.
 * @param y Рік для фільтрації.
 */
void filterByYear(int y) const;

/**
```

```
    * @brief Фільтрує та виводить автомобілі за вказаною моделлю.
    * @param m Модель для фільтрації.
    */
    void filterByModel(const std::string& m) const;

    /**
    * @brief Запускає діалог пошуку авто за маркою.
    */
    void findByBrand() const;

    /**
    * @brief Запускає розширений пошук за декількома критеріями.
    */
    void searchCars() const;

    /**
    * @brief Знаходить та виводить економічні авто (витрати пального нижче ліміту).
    */
    void findEconomicalCars() const;

    /**
    * @brief Знаходить та виводить "сімейні" авто (4+ дверей).
    */
    void findFamilyCars() const;

    void showHelp() const;
    void showHelpnotadm() const;

/**
 * @brief Валідатор введених значень за ключем.
 * @tparam Type Тип значення (наприклад, `int`, `double`).
 * @param value Значення для перевірки.
 * @param flag Ключ правила ("year","fuel","AC","multi","safe","price","doors","distance","fuelPrice" тощо).
 * @return `true`, якщо значення коректне.
 */
template <typename Type>
bool checker(Type value, const std::string& flag) const;

/**
 * @brief Безпечне зчитування цілого числа з `std::cin`.
 * @return Коректно введене ціле число.
 */
int safeInput() const;

/**
 * @brief Безпечне зчитування `double` з `std::cin`.
 * @return Коректно введене число з плаваючою комою.
 */
        double safeInputDouble() const;

private:
    std::vector<std::shared_ptr<Car>> cars;
    void printCarIndexList() const;
};
```

Вміст “CarManager.cpp”

```
#include "CarManager.h"
#include <iostream>
#include <fstream>
#include <algorithm>
#include <sstream>
#include <limits>
#include <stdexcept>
#include <climits>

using namespace std;

CarManager::CarManager() {}
CarManager::CarManager(const CarManager& other) : cars(other.cars) {}
CarManager::CarManager(CarManager&& other) noexcept : cars(move(other.cars)) {}
CarManager::~CarManager() {
    cout << "[CarManager] Деструктор\n";
}

string toLower(const string& str) {
    string result = str;
    transform(result.begin(), result.end(), result.begin(),
        [](unsigned char c) { return tolower(c); });
    return result;
}

int CarManager::safeInput() const {
    int value;
    while (true) {
        cin >> value;

        if (cin.fail()) {
            cin.clear();
            cin.ignore((numeric_limits<streamsize>::max)(), '\n');
            cout << "Помилка: введіть число! Спробуйте ще раз: ";
        }
        else {
            cin.ignore((numeric_limits<streamsize>::max)(), '\n');
            return value;
        }
    }
}

double CarManager::safeInputDouble() const {
    double value;
    while (true) {
        cin >> value;
        if (cin.fail()) {
            cin.clear();
            cin.ignore((numeric_limits<streamsize>::max)(), '\n');
            cout << "Помилка: введіть число! Спробуйте ще раз: ";
        }
```

```
        else {
            cin.ignore((numeric_limits<streamsize>::max)(), '\n');
            return value;
        }
    }
}

template <typename Type>
bool CarManager::checker(Type value, const std::string& flag) const {
    if (flag == "year" && value < 1900 && value > 2025) {
        cout << "Помилка: Некоректно введений рік. Має бути більшим за 1900 та меншим за 2025." << "!\n";
        return false;
    }
    else if (flag == "fuel") {
        if (value < 1 || value > 100) {
            cout << "Помилка: Некоректно введене значення." << "!\n";
            return false;
        }
    }
    else if (flag == "AC") {
        if (value < 0 || value > 1) {
            cout << "Помилка: Некоректно введене значення." << "!\n";
            return false;
        }
    }
    else if (flag == "multi") {
        if (value < 0 || value > 1) {
            cout << "Помилка: Некоректно введене значення." << "!\n";
            return false;
        }
    }
    else if (flag == "safe") {
        if (value < 0 || value > 1) {
        cout << "Помилка: Некоректно введене значення." << "!\n";
        return false;
        }
    }
    else if (flag == "price") {
        if (value < 1 || value > INT_MAX) {
            cout << "Помилка: Некоректно введене значення." << "!\n";
            return false;
        }
    }
    else if(flag == "doors") {
        if (value < 2 || value > 6) {
            cout << "Помилка: Некоректно введене значення." << "!\n";
            return false;
        }
            }
    else if (flag == "distance") {
        if (value <= 0.0) {
            cout << "Помилка: Некоректно введене значення." << "!\n";
            return false;
```

```
        }
    }
    else if (flag == "fuelPrice") {
        if (value <= 0.0) {
            cout << "Помилка: Некоректно введене значення." << "!\n";
            return false;
        }
    }
    else if (flag == "none") {
        cout << "ERROR" << endl;
                        return false;
    }

            return true;
}


void CarManager::printCarIndexList() const {
    cout << "\nІндекс\tМарка\n";
    for (size_t i = 0; i < cars.size(); ++i) {
        cout << i << "\t";
        cars[i]->print();
    }
}

void CarManager::loadFromFile(const string& filename) {
    ifstream file(filename);
    if (!file.is_open()) {
        throw runtime_error(string("[loadFromFile] Failed to open file: ") + filename);
    }

    string line;
    size_t lineNo = 0;
    while (getline(file, line)) {
        ++lineNo;
        if (!line.empty()) {
            try {
                cars.push_back(make_shared<Car>(Car::fromCSV(line)));
            }
            catch (const std::exception& ex) {
                cerr << "[loadFromFile] Error parsing line " << lineNo << ": " << ex.what() << '\n';
            }
            catch (...) {
                cerr << "[loadFromFile] Unknown error parsing line " << lineNo << '\n';
            }
        }
    }

    if (file.bad()) {
        throw runtime_error(string("[loadFromFile] I/O error while reading file: ") + filename);
    }
}
```

```
void CarManager::saveToFile(const string& filename) {
    ofstream file(filename);
    if (!file.is_open()) {
        throw runtime_error(string("[saveToFile] Failed to open file for writing: ") + filename);
    }

    for (const auto& car : cars) {
        file << car->toCSV() << '\n';
        if (!file) {
            throw runtime_error(string("[saveToFile] Failed while writing to file: ") + filename);
        }
    }
}

void CarManager::addCar() {
    string brand, color, model;
    double fuel, price;
    int doors, year;

    cout << "\n--- Додавання нового авто ---\n";
    cout << "Марка: "; cin >> brand;
    cout << "Модель: "; cin >> model;
    cout << "Колір: ";
    bool valid = false;
    while (!valid) {
        std::cin >> color;
        valid = true;
        for (char c : color) {
            if (!isalpha(c)) {
                std::cerr << "Помилка: введено недопустимий символ '" << c << "'!" << std::endl;
                valid = false;
                break;
            }
        }
    }


    cout << "Витрати бензину (л/100км): ";
    fuel = safeInput();
    if (!checker(fuel, "fuel")) return;

    cout << "Кількість дверей: ";
    while (!(cin >> doors)) {
        cout << "Помилка! Введіть ціле число (напр. 4): ";
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
    }

    cout << "Рік випуску: ";
    year = safeInput();
    if (!checker(year, "year")) return;

    cout << "Ціна: ";
```

```
    price = safeInput();
            if (!checker(price, "price")) return;

    string packageName;
    bool hasAC, hasMultimedia, hasSafety;
    double configPrice;
    cin.ignore();
    cout << "Назва комплектації: "; getline(cin, packageName);
            cin.ignore();
    cout << "Чи є кондиціонер (1 - так/0 - ні): ";
    hasAC=safeInput();
    if (!checker(hasAC, "AC")) return;
    cout << "Чи є мультимедія (1 - так/0 - ні): ";
    hasMultimedia = safeInput();
    if (!checker(hasMultimedia, "multi")) return;
    cout << "Чи є система безпеки (1 - так/0 - ні): ";
    hasSafety = safeInput();
    if (!checker(hasSafety, "safe")) return;
    cout << "Нова ціна конфігурації: ";
    while (!(cin >> configPrice)) {
       cout << "Помилка! Введіть числове значення (напр. 550000): ";
       cin.clear();
       cin.ignore(numeric_limits<streamsize>::max(), '\n');
    }

    try {
       Configuration config(brand, model, packageName, hasAC, hasMultimedia, hasSafety, configPrice);
       cars.push_back(make_shared<Car>(brand, toLower(color), model, fuel, doors, year, price, config));
       cout << "? Авто додано!\n";
    }
    catch (const std::exception& ex) {
       cerr << "[addCar] Error creating car/configuration: " << ex.what() << '\n';
       cout << "? Помилка при створенні авто.\n";
       return;
    }
    catch (...) {
       cerr << "[addCar] Unknown error creating car/configuration\n";
       cout << "? Невідома помилка при створенні авто.\n";
       return;
    }

    try {
       saveToFile("cars.csv");
    }
    catch (const std::exception& ex) {
       cerr << "[addCar] Failed to save to file: " << ex.what() << '\n';
       cout << "? Помилка збереження у файл.\n";
    }
    catch (...) {
       cerr << "[addCar] Unknown error while saving to file\n";
       cout << "? Невідома помилка збереження у файл.\n";
    }
}
```

```
void CarManager::showAllCars() const {
    if (cars.empty()) {
        cout << "База порожня.\n";
        return;
    }
    cout << "\n--- Всі автомобілі ---\n";
            int index = 1;
    for (const auto& car : cars) {
        cout << index << endl;
        car->print();
        index++;
    }
}

void CarManager::editCar() {
    int index;
    showAllCars();
    cout << "\nВведіть індекс авто для редагування: ";
    index =  safeInput() - 1;;

    if (index >= 0 && index < cars.size()) {
        string brand, color, model;
        double fuel, price;
        int doors, year;

        cout << "--- Редагування авто ---\n";
        cout << "Нова марка: "; cin >> brand;
        cout << "Нова модель: "; cin >> model;
        cout << "Новий колір: ";
        bool valid = false;
        while (!valid) {
            cin >> color;
            valid = true;
            for (char c : color) {
                if (!isalpha(c)) {
                    cerr << "Помилка: введено недопустимий символ '" << c << "'!" <<endl<< "неможна вводити числове значення в колір"<<endl;
                                                                cout << "Введіть колір ще раз: ";
                    valid = false;
                    break;
                }
            }
        }
        cout << "Нові витрати бензину: ";
                        fuel = safeInput();
        if (!checker(fuel, "fuel")) return;

        cout << "Нова кількість дверей: ";
                        doors = safeInput();
                        if (!checker(doors, "doors")) return;

        cout << "Новий рік випуску: ";
```

```
year = safeInput();
if (!checker(year, "year")) return;

cout << "Нова ціна: ";
                price = safeInput();
                if (!checker(price, "price")) return;

double configPrice;
string packageName;
bool hasAC, hasMultimedia, hasSafety;
cout << "Назва комплектації: "; getline(cin, packageName);
cout << "Чи є кондиціонер (1 - так/0 - ні): ";
hasAC = safeInput();
if (!checker(hasAC, "AC")) return;
cout << "Чи є мультимедія (1 - так/0 - ні): ";
hasMultimedia = safeInput();
if (!checker(hasMultimedia, "multi")) return;
cout << "Чи є система безпеки (1 - так/0 - ні): ";
hasSafety = safeInput();
if (!checker(hasSafety, "safe")) return;
cout << "Нова ціна: ";
while (!(cin >> configPrice)) {
   cout << "Помилка! Введіть числове значення (напр. 550000): ";
   cin.clear();
   cin.ignore(numeric_limits<streamsize>::max(), '\n');
}

try {
   Configuration config(brand, model, packageName, hasAC, hasMultimedia, hasSafety, configPrice);
   cars[index] = make_shared<Car>(brand, toLower(color), model, fuel, doors, year, price, config);
   cout << "? Авто оновлено!\n";
}
catch (const std::exception& ex) {
   cerr << "[editCar] Error creating car/configuration: " << ex.what() << '\n';
   cout << "? Помилка при оновленні авто.\n";
   return;
}
catch (...) {
   cerr << "[editCar] Unknown error creating car/configuration\n";
   cout << "? Невідома помилка при оновленні авто.\n";
   return;
}

try {
   saveToFile("cars.csv");
}
catch (const std::exception& ex) {
   cerr << "[editCar] Failed to save to file: " << ex.what() << '\n';
   cout << "? Помилка збереження у файл.\n";
}
catch (...) {
   cerr << "[editCar] Unknown error while saving to file\n";
   cout << "? Невідома помилка збереження у файл.\n";
```

```
        }
    }
    else {
        cout << "? Невірний індекс!\n";
    }
}

void CarManager::removeCar() {
    int index;
    showAllCars();
    cout << "\nВведіть індекс авто для видалення: ";
    index =  safeInput() - 1;;
    if (index >= 0 && index < cars.size()) {
        cars.erase(cars.begin() + index);
        cout << "? Авто видалено!\n";
    }

    else {
        cout << "? Невірний індекс!\n";
    }
}

void CarManager::findMostEconomicalCar() const {
    if (cars.empty()) {
        cout << "База порожня.\n"; return;
    }
    auto best = min_element(cars.begin(), cars.end(),
        [](auto& a, auto& b) { return a->getFuelConsumption() < b->getFuelConsumption(); });
    cout << "\nНайекономічніший автомобіль:\n";
    (*best)->print();
}

void CarManager::averagePriceInPeriod() const {
    int start, end;
    cout << "\nВведіть початковий рік: ";
    while (!(cin >> start)) {
        cout << "Помилка! Введіть ціле число ";
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
    }

    cout << "Введіть кінцевий рік : ";
    while (!(cin >> end)) {
        cout << "Помилка! Введіть ціле число (напр. 2023): ";
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
    }

    double sum = 0; int count = 0;
    for (const auto& car : cars) {
        if (car->getYear() >= start && car->getYear() <= end) {
            sum += car->getPrice();
            count++;
```

```
        }
    }
    if (count == 0) cout << "? Немає авто у цьому періоді.\n";
    else cout << "Середня ціна: " << sum / count << " грн\n";
}

// ---------- Сортування ----------
void CarManager::sortByPrice(bool ascending) {
    sort(cars.begin(), cars.end(), [ascending](auto& a, auto& b) {
        return ascending ? a->getPrice() < b->getPrice() : a->getPrice() > b->getPrice();
        });
    cout << "? Відсортовано за ціною.\n";
}

void CarManager::sortByYear(bool ascending) {
    sort(cars.begin(), cars.end(), [ascending](auto& a, auto& b) {
        return ascending ? a->getYear() < b->getYear() : a->getYear() > b->getYear();
        });
    cout << "? Відсортовано за роком.\n";
}

void CarManager::sortByFuelConsumption(bool ascending) {
    sort(cars.begin(), cars.end(), [ascending](auto& a, auto& b) {
        return ascending ? a->getFuelConsumption() < b->getFuelConsumption()
            : a->getFuelConsumption() > b->getFuelConsumption();
        });
    cout << "? Відсортовано за витратами пального.\n";
}

// ---------- Фільтрація ----------
void CarManager::filterByBrand(const string& b) const {
    cout << "\nФільтр за маркою: " << b << "\n";
            string brandLower = toLower(b);
    bool found = false;
    for (const auto& car : cars) {
        if (car->isBrand(b)) {
            car->print();
            found = true;
        }
    }
    if (!found) cout << "? Немає авто з такою маркою.\n";
}

void CarManager::filterByColor(const string& color) const {
    cout << "\nФільтр за кольором: " << color << "\n";
            string colorLower = toLower(color);
    bool found = false;
    for (const auto& car : cars) {
        if (car->getColor() == colorLower) {
            car->print();
            found = true;
        }
    }
```

```
    if (!found) cout << "? Немає авто з таким кольором.\n";
}

void CarManager::filterByYear(int year) const {
    cout << "\nФільтр за роком: " << year << "\n";
    bool found = false;
    for (const auto& car : cars) {
        if (car->getYear() == year) {
            car->print();
            found = true;
        }
    }
    if (!found) cout << "? Немає авто цього року.\n";
}

void CarManager::showCarAge() const {
    double index;
    showAllCars();
    cout << "\nВведіть індекс авто для перевірки віку: ";
    index =  safeInput() - 1;;
    if (index >= 0 && index < cars.size()) {
        cout << "Вік авто: " << cars[index]->getAge() << " років\n";
    }
    else {
        cout << "? Невірний індекс.\n";
    }
}

void CarManager::findByBrand() const {
    string b;
    cout << "\nВведіть марку для пошуку: "; cin >> b;
    filterByBrand(b);
}

void CarManager::searchCars() const {
    if (cars.empty()) {
        cout << "База порожня.\n";
        return;
    }

    string brandFilter, colorFilter;
    int yearFilter = 0;

    cout << "\n--- Пошук авто ---\n";
    cout << "Введіть марку (або залиште порожнім): ";
    cin.ignore();
    getline(cin, brandFilter);

    cout << "Введіть колір (або залиште порожнім): ";
    getline(cin, colorFilter);

    cout << "Введіть рік (або 0 для пропуску): ";
    cin >> yearFilter;
```

```
    brandFilter = toLower(brandFilter);
    colorFilter = toLower(colorFilter);

    bool found = false;
    for (const auto& car : cars) {
        string carStr = toLower(car->toCSV());
        bool matches = true;

        if (!brandFilter.empty() && carStr.find(brandFilter) == string::npos)
            matches = false;
        if (!colorFilter.empty() && carStr.find(colorFilter) == string::npos)
            matches = false;
        if (yearFilter != 0 && car->getYear() != yearFilter)
            matches = false;

        if (matches) {
            car->print();
            found = true;
        }
    }

    if (!found)
        cout << "Нічого не знайдено за заданими критеріями.\n";
}


void CarManager::findEconomicalCars() const {
    double limit;
    cout << "Введіть ліміт витрат пального (літр/100км): "; cin >> limit;
    bool found = false;
    for (const auto& car : cars) {
        if (car->isEconomical(limit)) {
            car->print();
            found = true;
        }
    }
    if (!found) cout << "? Немає авто економніших за " << limit << " л/100км.\n";
}

void CarManager::findFamilyCars() const {
    cout << "\nСімейні авто (>=4 дверей):\n";
    bool found = false;
    for (const auto& car : cars) {
        if (car->isFamilyCar()) {
            car->print();
            found = true;
        }
    }
    if (!found) cout << "? Сімейних авто немає.\n";
}

void CarManager::filterByModel(const string& m) const {
```

```
    cout << "\nФільтр за моделлю: " << m << "\n";
    bool found = false;
    for (const auto& car : cars) {
        if (car->getModel() == m) {
            car->print();
            found = true;
        }
    }
    if (!found) cout << "? Немає авто з такою моделлю.\n";
}

void CarManager::applyDiscountToCar() {
    int index;
    double percent;
    showAllCars();
    cout << "\nВведіть індекс авто для знижки: "; index =  safeInput() - 1;
    cout << "Введіть відсоток знижки: "; cin >> percent;

    if (index >= 0 && index < cars.size()) {
        cars[index]->applyDiscount(percent);
        cout << "? Знижку застосовано!\n";
        cars[index]->print();
    }
    else {
        cout << "? Невірний індекс.\n";
    }
}

void CarManager::calcTripCost() const {
    int index;
    double distance, fuelPrice;
    showAllCars();
    cout << "\nВведіть індекс авто: "; index =  safeInput() - 1;;
    cout << "Введіть відстань поїздки (км): ";
            distance = safeInputDouble();
    if (!checker(distance, "distance")) return;
    cout << "Введіть ціну за літр пального (грн): ";
            fuelPrice = safeInputDouble();
    if (!checker(fuelPrice, "fuelPrice")) return;

    if (index >= 0 && index < cars.size()) {
        double cost = cars[index]->costPer100km(fuelPrice) * (distance / 100.0);
        cout << "Вартість поїздки: " << cost << " грн\n";
    }
    else {
        cout << "? Невірний індекс.\n";
    }
}

void CarManager::showHelp() const {
    cout << "\n================ Довідка для адміністратора ================\n";

    cout << "\n--> Пояснення роботи програми <--\n";
```

```
    cout << "Ця програма є системою управління автосалоном. Вона дозволяє\n";
    cout << "адміністраторам та користувачам працювати зі спільною базою даних автомобілів,\n";
    cout << "що зберігається у файлі 'cars.csv'.\n";
    cout << "Адміністратор має повний доступ до всіх функцій, включаючи керування\n";
    cout << "користувачами та редагування каталогу авто.\n";

    cout << "\n--> Правила вводу даних <--\n";
    cout << "1. При введенні числових даних (ціна, рік, кількість дверей) використовуйте\n";
    cout << "   лише цифри. Введення тексту призведе до помилки.\n";
    cout << "2. При введенні тексту (марка, модель, колір) уникайте пробілів.\n";
    cout << "   Наприклад, вводьте 'Volkswagen', а не 'Volks wagen'.\n";
    cout << "3. Для позначення 'так' або 'ні' (наприклад, наявність кондиціонера)\n";
    cout << "   використовуйте 1 для 'так' і 0 для 'ні'.\n";

    cout << "\n--> Короткий опис команд <--\n\n";
    cout << "--- Керування каталогом ---\n";
    cout << "1.  Переглянути всі авто  - Виводить повний список автомобілів у базі.\n";
    cout << "2.  Додати нове авто      - Запускає процес додавання нового авто в каталог.\n";
    cout << "5.  Редагувати авто       - Дозволяє змінити дані існуючого автомобіля за його індексом.\n";
    cout << "6.  Видалити авто         - Видаляє автомобіль з бази за його індексом.\n";
    cout << "12. Застосувати знижку    - Застосовує відсоткову знижку до ціни обраного авто.\n\n";

    cout << "--- Аналітика та пошук ---\n";
    cout << "3.  Найекономічніше авто  - Знаходить авто з найменшими витратами пального.\n";
    cout << "4.  Середня ціна у періоді - Розраховує середню ціну авто за вказаний період років.\n";
    cout << "8.  Пошук за критеріями   - Розширений пошук за маркою, кольором та роком.\n";
    cout << "9.  Пошук за маркою       - Швидкий пошук та фільтрація всіх авто конкретної марки.\n";
    cout << "13. Показати вік авто     - Визначає вік обраного автомобіля.\n";
    cout << "14. Розрахувати вартість  - Розраховує вартість поїздки на задану відстань.\n\n";

    cout << "--- Сортування та фільтрація ---\n";
    cout << "15-18. Сортувати          - Сортує список авто за ціною або роком (за зростанням/спаданням).\n";
    cout << "19-21. Фільтрувати        - Показує авто, що відповідають заданому кольору або року.\n\n";

    cout << "--- Керування акаунтом ---\n";
    cout << "7.  Меню адміністратора   - Повернення до головного меню адміністратора (керування користувачами).\n";
    cout << "23. Вийти з програми      - Зберігає всі зміни та завершує роботу програми.\n";
    cout << "============================================================\n";
}
void CarManager::showHelpnotadm() const {
    cout << "\n================== Довідка для користувача ==================\n";

    cout << "\n--> Пояснення роботи програми <--\n";
    cout << "Ця програма є каталогом автомобілів автосалону. Ви можете\n";
    cout << "переглядати, шукати, сортувати та аналізувати доступні автомобілі.\n";
    cout << "Усі дані завантажуються зі спільної бази даних.\n";

    cout << "\n--> Правила вводу даних <--\n";
    cout << "1. При введенні числових даних (наприклад, рік) використовуйте лише цифри.\n";
    cout << "2. При введенні тексту (марка, колір) пишіть одне слово без пробілів.\n";

    cout << "\n--> Короткий опис команд <--\n\n";
    cout << "--- Перегляд та аналітика ---\n";
```

```
    cout << "1.  Переглянути всі авто  - Виводить повний список автомобілів у базі.\n";
    cout << "2.  Найекономічніше авто   - Знаходить авто з найменшими витратами пального.\n";
    cout << "3.  Середня ціна у періоді - Розраховує середню ціну авто за вказаний період років.\n";
    cout << "4.  Пошук за критеріями    - Дозволяє знайти авто за маркою, кольором або роком.\n\n";

    cout << "--- Сортування та фільтрація ---\n";
    cout << "5-8.  Сортувати            - Сортує список авто за ціною або роком.\n";
    cout << "9-10. Фільтрувати          - Показує авто, що відповідають заданому кольору або року.\n\n";

    cout << "--- Загальні команди ---\n";
    cout << "11. Допомога (цей список)  - Виводить цю довідку.\n";
    cout << "12. Вийти з програми       - Завершує роботу програми.\n";
    cout << "============================================================\n";
}
```

Програмний модуль “Configuration”

Вміст “Configuration.h”

```
#pragma once
#include "Vehicle.h"
#include "IPrintable.h"
#include <string>
#include <iostream>

/**
 * @class Configuration
 * @brief Клас, що представляє комплектацію автомобіля.
 * @details Зберігає інформацію про пакет опцій (кондиціонер, мультимедіа),
 * його назву та ціну. Є нащадком Vehicle та реалізує IPrintable.
 */
class Configuration : public Vehicle, public IPrintable {
public:
    Configuration& operator=(const Configuration& other);
    Configuration();
    Configuration(const std::string& brand, const std::string& model,
        const std::string& packageName, bool ac, bool multimedia, bool safety, double price);
    Configuration(const Configuration& other);
    Configuration(Configuration&& other) noexcept;
    ~Configuration();

    // --- Get/Set ---
    std::string getPackageName() const;
    void setPackageName(const std::string& name);
    double getPrice() const;
    void setPrice(double p);


    /**
     * @brief Перевіряє, чи є комплектація люксовою.
     * @return true, якщо всі опції включені, інакше false.
     */
    bool isLuxury() const;

    /**
     * @brief Змінює стан кондиціонера (включає/виключає).
     * @details Якщо поле `hasAirConditioner` було встановлене в `true`, метод
     * переключить його в `false`. Якщо було `false` — переключить в `true`.
     */
    void toggleAirConditioner();

    /**
     * @brief Змінює стан мультимедійної системи (включає/виключає).
     * @details Якщо поле `hasMultimedia` було встановлене в `true`, метод
     * переключить його в `false`. Якщо було `false` — переключить в `true`.
     */
```

```
    void toggleMultimedia();

    /**
     * @brief Змінює стан системи безпеки (включає/виключає).
     * @details Якщо поле `hasSafetySystem` було встановлене в `true`, метод
     * переключить його в `false`. Якщо було `false` — переключить в `true`.
     */
    void toggleSafety();

    // --- Реалізація інтерфейсу ---
    void print() const override;
    std::string toCSV() const override;

    /**
     * @brief Створює об'єкт Configuration з рядка CSV.
     * @param line Рядок у форматі CSV.
     * @return Новий об'єкт Configuration.
     */
    static Configuration fromCSV(const std::string& line);

private:
    std::string packageName;
    bool hasAirConditioner;
    bool hasMultimedia;
    bool hasSafetySystem;
    double price;
};
```

Вміст “Configuration.cpp”

```
#include "Configuration.h"
#include <string>
#include <sstream>
#include <iostream>
#include <stdexcept>
#include <algorithm>
#include <cctype>
#include <vector>

using namespace std;

static string trim(const string& s) {
    size_t a = 0;
    while (a < s.size() && isspace(static_cast<unsigned char>(s[a]))) ++a;
    size_t b = s.size();
    while (b > a && isspace(static_cast<unsigned char>(s[b - 1]))) --b;
    return s.substr(a, b - a);
}

static bool parseBoolToken(const string& tok) {
    string t = tok;
    transform(t.begin(), t.end(), t.begin(), [](unsigned char c) { return tolower(c); });
    t = trim(t);
    if (t == "1" || t == "true" || t == "yes") return true;
    if (t == "0" || t == "false" || t == "no") return false;
    throw invalid_argument(string("Invalid boolean token: '") + tok + "'");
}

Configuration::Configuration()
    : Vehicle("Unknown", "", 0, 0.0, "Unknown"),
    packageName("Basic"),
    hasAirConditioner(false),
    hasMultimedia(false),
    hasSafetySystem(false),
    price(0.0) {
}

Configuration::Configuration(const string& brand, const string& model,
    const string& packageName, bool ac, bool multimedia, bool safety, double price)
    : Vehicle(brand, "", 0, 0.0, model),
    packageName(packageName),
    hasAirConditioner(ac),
    hasMultimedia(multimedia),
    hasSafetySystem(safety),
    price(price) {
    // Validate parameters that could be invalid at runtime
    if (packageName.empty()) {
        throw invalid_argument("Configuration::Configuration - packageName cannot be empty");
    }
    if (price < 0.0) {
        throw invalid_argument("Configuration::Configuration - price cannot be negative");
```

```
    }
}

Configuration::Configuration(const Configuration& other)
    : Vehicle(other),
    packageName(other.packageName),
    hasAirConditioner(other.hasAirConditioner),
    hasMultimedia(other.hasMultimedia),
    hasSafetySystem(other.hasSafetySystem),
    price(other.price) {
}

Configuration::Configuration(Configuration&& other) noexcept
    : Vehicle(move(other)),
    packageName(move(other.packageName)),
    hasAirConditioner(other.hasAirConditioner),
    hasMultimedia(other.hasMultimedia),
    hasSafetySystem(other.hasSafetySystem),
    price(other.price) {
}

Configuration::~Configuration() {
    cout << "[Config] знищено " << endl;
}

string Configuration::getPackageName() const { return packageName; }
void Configuration::setPackageName(const string& name) {
    if (name.empty()) throw invalid_argument("Configuration::setPackageName - name cannot be empty");
    packageName = name;
}

double Configuration::getPrice() const { return price; }
void Configuration::setPrice(double p) {
    if (p < 0.0) throw invalid_argument("Configuration::setPrice - price cannot be negative");
    price = p;
}

bool Configuration::isLuxury() const {
    return hasAirConditioner && hasMultimedia && hasSafetySystem;
}

void Configuration::toggleAirConditioner() { hasAirConditioner = !hasAirConditioner; }
void Configuration::toggleMultimedia() { hasMultimedia = !hasMultimedia; }
void Configuration::toggleSafety() { hasSafetySystem = !hasSafetySystem; }

void Configuration::print() const {
    cout << "Configuration: " << packageName
        << " | AC: " << (hasAirConditioner ? "Yes" : "No")
        << " | Multimedia: " << (hasMultimedia ? "Yes" : "No")
        << " | Safety: " << (hasSafetySystem ? "Yes" : "No")
        << " | Price: " << price << endl;
}
```

```
string Configuration::toCSV() const {
    ostringstream oss;
    oss << packageName << ","
        << hasAirConditioner << ","
        << hasMultimedia << ","
        << hasSafetySystem << ","
        << price;
    return oss.str();
}

Configuration Configuration::fromCSV(const string& line) {
    try {
        // Split into tokens by comma
        vector<string> tokens;
        string token;
        istringstream iss(line);
        while (getline(iss, token, ',')) {
            tokens.push_back(token);
        }

        if (tokens.size() != 5) {
            throw invalid_argument(string("fromCSV: expected 5 fields, got ") + to_string(tokens.size()));
        }

        string packageNameTok = trim(tokens[0]);
        bool ac = parseBoolToken(tokens[1]);
        bool multimedia = parseBoolToken(tokens[2]);
        bool safety = parseBoolToken(tokens[3]);

        string priceTok = trim(tokens[4]);
        double priceVal = 0.0;
        try {
            size_t idx = 0;
            priceVal = stod(priceTok, &idx);
            if (idx != priceTok.length()) {
                throw invalid_argument(string("fromCSV: trailing characters in price token '") + priceTok + "'");
            }
        }
        catch (const invalid_argument& e) {
            throw invalid_argument(string("fromCSV: invalid price value: '") + priceTok + "'");
        }
        catch (const out_of_range& e) {
            throw invalid_argument(string("fromCSV: price out of range: '") + priceTok + "'");
        }

        // Validate parsed values
        if (packageNameTok.empty()) {
            throw invalid_argument("fromCSV: packageName is empty");
        }
        if (priceVal < 0.0) {
            throw invalid_argument("fromCSV: price cannot be negative");
        }
```

```
        // Construct and return Configuration. Use "Unknown" for brand/model as before.
        return Configuration("Unknown", "Unknown", packageNameTok, ac, multimedia, safety, priceVal);
    }
    catch (const exception& ex) {
        // Re-throw with context to help callers identify the problematic line
        throw invalid_argument(string("Configuration::fromCSV error parsing line: '") + line + "' -> " + ex.what());
    }
    catch (...) {
        throw invalid_argument(string("Configuration::fromCSV unknown error parsing line: '") + line + "'");
    }
}

Configuration& Configuration::operator=(const Configuration& other) {
    if (this != &other) {
        Vehicle::operator=(other);
        packageName = other.packageName;
        hasAirConditioner = other.hasAirConditioner;
        hasMultimedia = other.hasMultimedia;
        hasSafetySystem = other.hasSafetySystem;
        price = other.price;
    }
    return *this;
}
```

Програмний модуль “IPrintable”

Вміст “IPrintable.h”

```
#pragma once
#include <string>

/**
 * @class IPrintable
 * @brief Інтерфейс, що визначає контракт для об'єктів, які можна роздрукувати.
 * @details Будь-який клас, що реалізує цей інтерфейс, зобов'язаний
 * надати реалізацію методів print() та toCSV().
 */
class IPrintable {
public:
    /**
     * @brief Чисто віртуальний метод для виводу інформації в консоль.
     */
    virtual void print() const = 0;

    /**
     * @brief Чисто віртуальний метод для серіалізації об'єкта в CSV-формат.
     * @return Рядок у форматі CSV.
     */
    virtual std::string toCSV() const = 0;

    /**
     * @brief Віртуальний деструктор для коректного поліморфного видалення.
     */
    virtual ~IPrintable() {}
};
```

Програмний модуль Vehicle”

Вміст “Vehicle.h”

```
#pragma once
#include <iostream>
#include <string>

/**
 * @class Vehicle
 * @brief Абстрактний базовий клас, що представляє транспортний засіб.
 * @details Визначає загальні властивості для всіх транспортних засобів,
 * такі як марка, колір, рік та ціна, а також базовий інтерфейс.
 */
class Vehicle {
public:
    /**
     * @brief Перевантажений оператор присвоювання.
     */
    Vehicle& operator=(const Vehicle& other);

    /**
     * @brief Конструктор за замовчуванням.
     */
    Vehicle();

    /**
     * @brief Конструктор з параметрами.
     * @param b Марка.
     * @param c Колір.
     * @param y Рік випуску.
     * @param p Ціна.
     * @param m Модель.
     */
    Vehicle(const std::string& b, const std::string& c, int y, double p, const std::string& m);

    /**
     * @brief Конструктор копіювання.
     */
    Vehicle(const Vehicle& other);

    /**
     * @brief Конструктор переміщення.
     */
    Vehicle(Vehicle&& other) noexcept;

    /**
     * @brief Віртуальний деструктор.
     */
    virtual ~Vehicle();
```

```
// --- Getters ---
std::string getBrand() const;
std::string getColor() const;
int getYear() const;
double getPrice() const;
std::string getModel() const;

// --- Setters ---
void setBrand(const std::string& b);
void setColor(const std::string& c);
void setYear(int y);
void setPrice(double p);

// --- 5 власних методів ---

/**
 * @brief Перевіряє, чи є транспортний засіб старшим за вказаний рік.
 * @param y Рік для порівняння.
 * @return true, якщо авто старше, інакше false.
 */
bool isOlderThan(int y) const;

/**
 * @brief Перевіряє, чи є транспортний засіб дешевшим за вказану ціну.
 * @param p Ціна для порівняння.
 * @return true, якщо авто дешевше, інакше false.
 */
bool isCheaperThan(double p) const;

/**
 * @brief Розраховує вік транспортного засобу.
 * @param currentYear Поточний рік.
 * @return Вік у роках.
 */
int age(int currentYear) const;

/**
 * @brief Застосовує відсоткову знижку до ціни.
 * @param percent Відсоток знижки.
 */
void applyDiscount(double percent);

/**
 * @brief Перевіряє, чи співпадає марка авто із заданою.
 * @param b Марка для порівняння.
 * @return true, якщо марки співпадають, інакше false.
 */
bool isBrand(const std::string& b) const;

// --- Чисто віртуальні методи ---

/**
 * @brief Чисто віртуальний метод для виводу інформації про об'єкт.
```

```
     */
    virtual void print() const = 0;

    /**
     * @brief Чисто віртуальний метод для конвертації даних у CSV-рядок.
     * @return Рядок у форматі CSV.
     */
    virtual std::string toCSV() const = 0;

protected:
    std::string model;

private:
    std::string brand;
    std::string color;
    int year;
    double price;
};
```

Вміст “Vehicle.h”

```
#include "Vehicle.h"
using namespace std;

// Конструктори
Vehicle::Vehicle() : brand(""), color(""), year(0), price(0.0) {}

Vehicle::Vehicle(const string& b, const string& c, int y, double p, const string& m)
    : brand(b), color(c), year(y), price(p), model(m) {}

Vehicle::Vehicle(const Vehicle& other)
    : brand(other.brand), color(other.color), year(other.year), price(other.price) {}

Vehicle::Vehicle(Vehicle&& other) noexcept
    : brand(move(other.brand)), color(move(other.color)), year(other.year), price(other.price) {}

// Деструктор
Vehicle::~Vehicle() {
    cout << "[Vehicle] Знищено " << endl;
}

// Getters
string Vehicle::getBrand() const { return brand; }
string Vehicle::getColor() const { return color; }
int Vehicle::getYear() const { return year; }
double Vehicle::getPrice() const { return price; }

// Setters
void Vehicle::setBrand(const string& b) { brand = b; }
void Vehicle::setColor(const string& c) { color = c; }
void Vehicle::setYear(int y) { year = y; }
void Vehicle::setPrice(double p) { price = p; }

// Методи предметної області
bool Vehicle::isOlderThan(int y) const { return year < y; }
bool Vehicle::isCheaperThan(double p) const { return price < p; }
int Vehicle::age(int currentYear) const { return currentYear - year; }
void Vehicle::applyDiscount(double percent) { price -= price * (percent / 100.0); }
bool Vehicle::isBrand(const string& b) const { return brand == b; }
string Vehicle::getModel() const {
    return model;
}

Vehicle& Vehicle::operator=(const Vehicle& other) {
    if (this != &other) {
        brand = other.brand;
        model = other.model;
        color = other.color;
        year = other.year;
        price = other.price;
    }
    return *this;
```