Черкаський національний університет імені Богдана Хмельницького

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ АВТОМАТИЗОВАНИХ СИСТЕМ

КУРСОВА РОБОТА

з дисципліни "Об'єктно-орієнтоване програмування"

НА ТЕМУ «Інформаційна система підтримки продажу транспортних засобів»

Студента 2 курсу, групи КС-202
напряму підготовки «Програмна інженерія»
спеціальності «121 - Інженерія програмного забезпечення»

Невмитого Олега Миколайовича

Керівник доцент, к.т.н.
Мисник Б.В.

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна шкала: _____

Кількість балів: _____ Оцінка: ECTS _____

Члени комісії _____ _____

(підпис) **(прізвище та ініціали)**

_____ _____

(підпис) **(прізвище та ініціали)**

_____ _____

(підпис) **(прізвище та ініціали)**

м. Черкаси – 2022 рік

Зміст

Вступ 3
Розділ 1. Огляд алгоритмів роботи та інструментарію для реалізації програми.. 5
1.1. Опис алгоритму роботи програми 5
1.2. Вибір структур даних для реалізації даного програмного продукту 5
1.3. Вибір графічної бібліотеки 6
1.4. Висновок до першого розділу 6
Розділ 2. Проектування інформаційної системи підтримки продажу транспортних засобів 7
2.1. Функції для реалізації програмного продукту 7
2.2. Узагальнена блок схема програми 7
2.3. Проектування класів та їх взаємодія 9
2.4. Висновок до другого розділу 10
Розділ 3. Інформаційна система підтримки продажу транспортних засобів 11
3.1. Опис реалізації ООП в програмі 11
3.2. Створення базових класів 13
3.4. Діаграма класів 18
3.5. Висновок до третього розділу 20
Загальний висновок 22
Джерела 23

Вступ

Мета цієї курсової роботи є створення додатку для системи підтримки продажу транспортних засобів, з використанням трьох основних принципів об’єктно-орієнтованого програмування (ООП), а саме:

1. Інкапсуляція
2. Наслідування
3. Поліморфізм

Об’єктно-орієнтоване програмування (ООП) - одна з перших парадигм в програмуванні. Вона вирішує головну проблему – що робити зі складною предметною областю і складним кодом. Суть цієї парадигми сприймати всю предметну область у вигляді об’єктів, які якимсь чином взаємодіють один з одним. Оскільки людському мозку легше мислити об’єктами, ми автоматично розуміємо, що і у якого об’єкта має бути. Людині легко зрозуміти, де розташувати ті чи інші методи в коді. Тому з ООП ми отримуємо найбільш зрозумілу структуру програми. Основні принципи об’єктно орієнтованого програмування включаються в себе інкапсуляцію, наслідування та поліморфізм. [1]

Інкапсуляція - це коли кожен об’єкт зберігає свій стан всередині класу приватним. Інші об’єкти не мають прямого доступу до цього стану. Замість цього вони можуть викликати лише список відкритих функцій, які називаються методами. Отже, об’єкт керує своїм власним станом за допомогою методів — і жоден інший клас не може використати його, якщо це явно не дозволено. Якщо ви хочете зв’язатися з об’єктом, ви повинні використовувати надані методи. Але ви не можете змінити стан. [2]

Наслідування - це здатність набувати властивостей існуючих класів і створювати нові. Воно дозволяє повторно використовувати код без необхідності переписувати його в іншому класі. Однією з найкращих особливостей успадкування є можливість скорочувати код. Тобто ви можете використовувати

цей принцип, щоб успадкувати код від іншого класу та повторно використовувати його в новому класі. [3]

Поліморфізм – це здатність однієї функції виконувати різні способи. Іншими словами, це стосується здатності об’єкта приймати більше ніж одну форму. Поліморфізм можна застосувати двома простими способами.

1. Перевантаження методу
2. Перевизначення методу

Перевантаження методу – це коли клас має кілька методів з однаковими іменами, але різним набором параметрів. Перевантаження можна використати лише якщо ваші методи задовольняють будь-які з наступних правил:

1. Мають різні параметри
2. Мать різний тип повертаючих даних. [4]

Перевизначення методу – це коли, дочірні класи реалізують при наслідуванні ті самі методи, що і батьківський, але логіка його роботи відрізняється він інших його нащадків, чи самого батьківського класу.

Завдання: написати систему підтримки продажу транспортних засобів, за допомогою якої можна буде підібрати собі транспортний засіб. Транспортні засоби сортуються за цілями використання (перевезення вантажів різного виду, перевезення людей різної кількості, тощо) та видами палива, що вони використовують (бензин, дизель, електрика). Також використати основні принципи ООП (інкапсуляція, наслідування, поліморфізм).

Для досягнення мети курсової роботи на тему “інформаційна система підтримки транспортних засобів” необхідно вирішити наступні завдання:

1. Спроектувати структуру класів, потрібних для роботи програми;
2. Сформувати реалізацію курсової роботи з використанням принципів ООП;
3. Побудувати узагальнену блок-схему роботи програми;
4. Побудувати діаграму класів, які використовують наслідування та поліморфізм;
5. Реалізувати інформаційну систему підтримки продажу транспортних засобів, із спроектованою структурою програми.

Розділ 1. Огляд алгоритмів роботи та інструментарію для реалізації програми.

Відповідно до поставленої мети курсової роботи, для її реалізації було обрано об’єктно орієнтовану мову програмування C# із зручним синтаксисом, та підтримкою бібліотеки WPF для створення десктопних додатків з користувацьким інтерфейсом, що значно спрощує реалізацію потрібних функцій. Розробка буде проводитися в рідному для C# інтегрованому середовищі розробки - Visual Studio 2022.

1.1. Опис алгоритму роботи програми

Під час запуску програми, при завантажені вікна програма спробує отримати дані з файлу JSON, де лежать серіалізовані об’єкти транспортних засобів, та завантажить ці дані у таблицю на головній сторінці. Після успішного запуску програми, вона матиме такі функції:

1. Сортувати отримані дані в таблиці по категоріям (тип транспортного засобу, тип пального та станом справності)
2. Отримати інформацію про конкретний транспортний засіб
3. Додати в список новий транспортний засіб
4. Додати нові типи пального

1.2. Вибір структур даних для реалізації даного програмного продукту

В реалізації потрібних нам функцій потрібно зберігати списки транспортних засобів та видів пального, оскільки дані будуть змінюватися динамічно, ми використаємо стандартну структуру даних – список (List<T>). Для видалення зі списку об’єкта, ми реалізуємо в ньому інтерфейс IEquatable і перепишемо метод Equals, щоб при використанні методу Remove, воно шукало об’єкт по унікальному ідентифікаторі.

Для додаткових можливих функцій, таких як список делегатів ми використаємо звичайний масив (Array). Зберігати такі дані як стан функціювання та тип вантажу, де стани повинні бути статичними і не змінюватися будемо в перерахуваннях (Enum).

1.3. Вибір графічної бібліотеки

Графічний інтерфейс для даного програмного продукту буде зроблений за допомогою Windows Presentation Foundation (WPF). Це фреймворк для розробки користувацьких інтерфейсів, з використанням мови розмітки XAML.

1.4. Висновок до першого розділу

В даному розділі було описано основний алгоритм роботи програмного продукту, обрано мову програмування (C#), структури даних (List<T>, Array, Enum) та фреймворк для написання графічного інтерфейску (WPF).

Розділ 2. Проектування інформаційної системи підтримки продажу транспортних засобів

2.1. Функції для реалізації програмного продукту

Використання програми буде проходити наступним чином:
в програмі при старті ініціалізуються дані про добавлені транспортні засоби, які зберігаються в JSON файлі. Користувацький інтерфейс буде мати три сторінки, та бокову панель для того, щоб переключатися між ними.

Перша сторінка, тут будуть знаходяться фільтри, по яким буде вибір які саме транспортні засоби показувати та сама таблиця з ними. В рядку таблиці будуть такі дані як назва, модель, ціна, та кількість сидячих місць у цьому транспортному засобі. Також кнопка “Details”, при натисканні на яку буде відкрите додаткове вікно з повною інформацією про вибраний транспортний засіб.

Друга і третя сторінки для запису нових транспортних засобів та запису нових видів палива відповідно, після чого програма збереже додані дані в файл JSON, за допомогою серіалізації.

Для збереження даних будуть написані DTO (Data Transfer Object), та адаптери, які будуть конвертувати дані з моделей в dto, та навпаки. Це потрібно для того, щоб можна було правильно серіалізувати дані у JSON формат.

2.2. Узагальнена блок схема програми

Отже, вся програма складається з формування потрібних списків з даними, обробки натискань користувача та переключення сторінок.

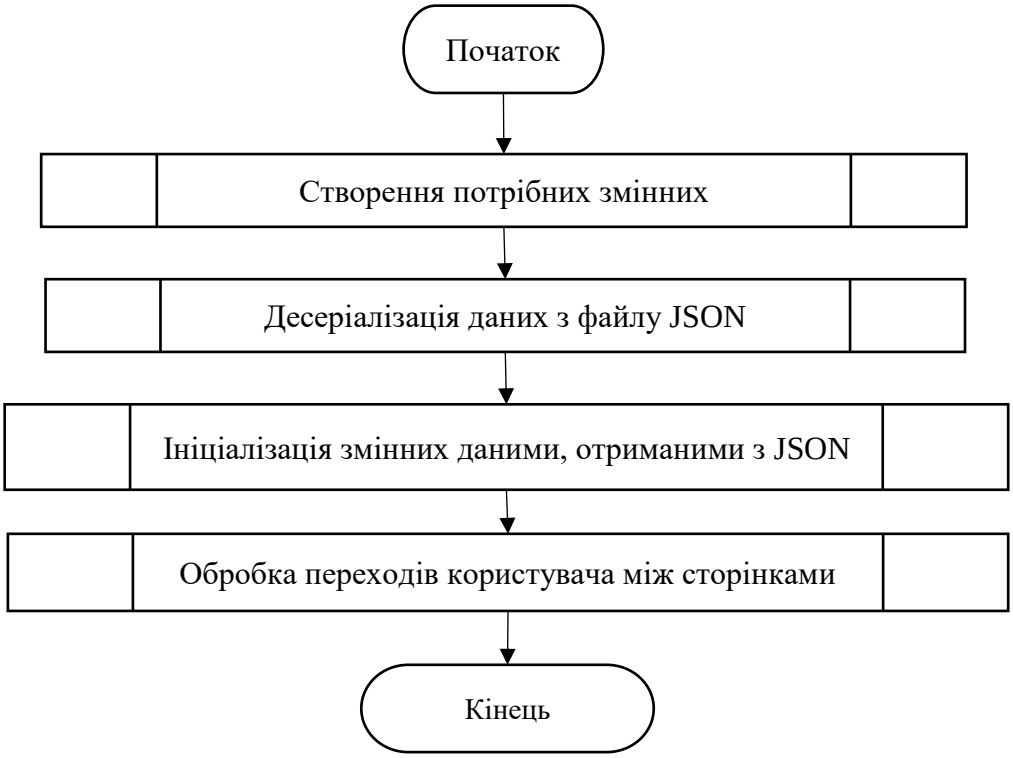

Рис. 2.1. Узагальнена блок схема.

Далі наведені блок схеми додавання (рис. 2) та видалення (рис. 3) зі списку нових транспортних. Також додавання нових видів пального (рис. 4).

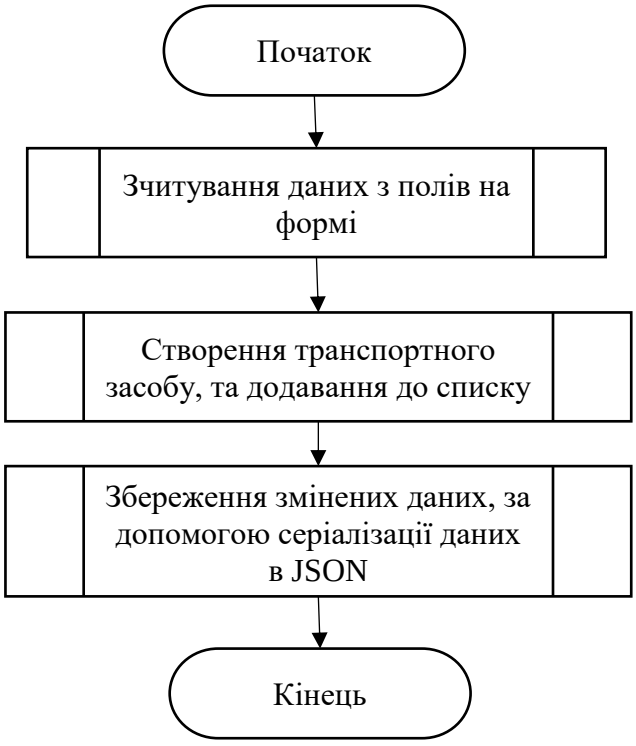

Рис. 2.2. Додавання транспортних засобів.

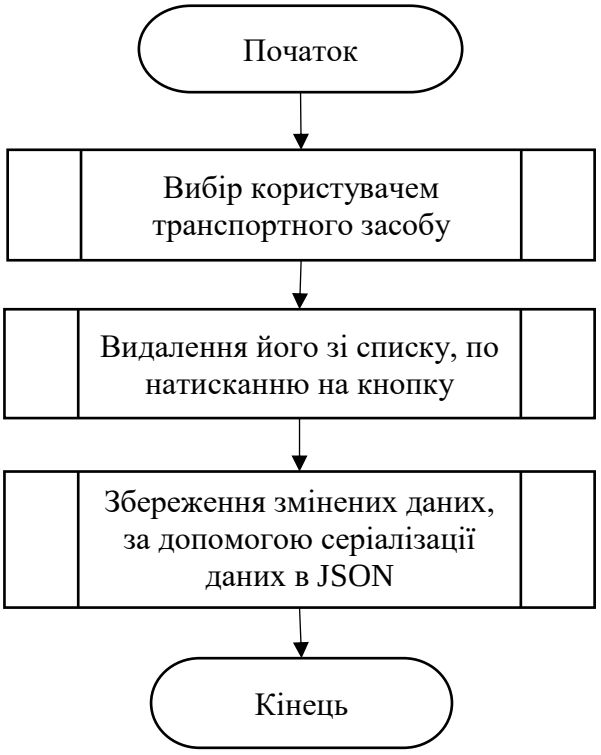

Рис. 2.3. Видалення транспортних засобів.

Рис. 2.4. Додавання нових видів палива.

2.3. Проектування класів та їх взаємодія

Для реалізації програми потрібно вирішити з яких класів вона буде складатися, та як вони будуть пов’язані між собою.

Спочатку створимо базовий абстрактний клас Vehicle, який буде мати спільні поля та абстрактні методи для його нащадків.

Далі будуть створюватися класи Car, Plane для відокремлення різновидів транспорту, вони будуть наслідувати в себе Vehicle, та реалізувати абстрактні методи.

Після цього потрібно буде створини вже реалізацію конкретних транспортних засобів, таких як – Bus, Truck, PassengerPlane, TransportPlane. Bus та Truck будуть наслідувати Car, а PassengerPlane і TransportPlane – Plane.

Класи для реалізації пального будуть також поділені на абстрактний клас FuelType, і класи наслідники CarFuel, AviationFuel.

Зобразимо схематично відношення класів (Рис 2.5).

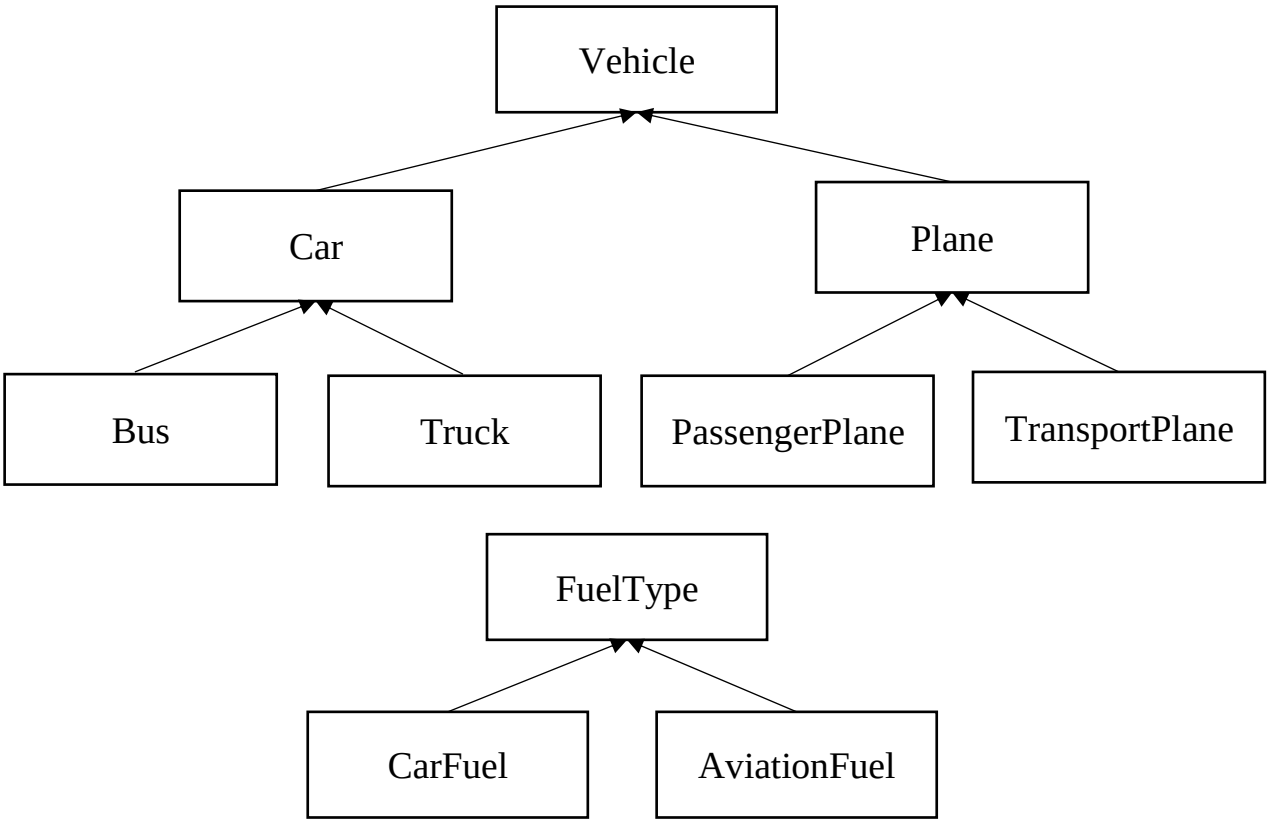

Рис 2.5. Діаграма взаємодії класів.

2.4. Висновок до другого розділу

У другому розділі було описано які функції потрібно реалізувати в програмі, відношення класів та взаємодію між ними, побудовано загальні блок схеми та діаграма класів.

Розділ 3. Інформаційна система підтримки продажу транспортних засобів

3.1. Опис реалізації ООП в програмі

Розглянемо де буде реалізовано в програмі принципи об’єктно орієнтованого програмування.

1. **Інкапсуляція.**

   Усі властивості класу Vehicle, та інших класів наслідників будуть мати геттери з модифікатором доступу – public, але сеттери – protected, що дозволяє змінювати данні через властивість в класах наслідниках, але не дає змогу змінювати дані зовні.

```
public Guid Id { get; protected set; }
public string Name { get; protected set; }
public string Model { get; protected set; }
public double Price { get; protected set; }
public int NumberOfSeats { get; protected set; }
public FuelType FuelType { get; protected set; }
public IFunctionality Functionality { get; protected set; }
```

2. **Наслідування.**

   Від абстрактного класу Vehicle будуть наслідуватися в два класи, Car та Plane. Вони унаслідують спільні між всима наслідниками. Car буде наслідуватися в Bus та Truck, а Plane в PassengerPlane та TransportPlane.

```
public abstract class Vehicle {}
public class Car : Vehicle {}
public class Plane : Vehicle {}
public class Bus : Car {}
public class Truck : Car {}
public class PassengerPlane : Plane {}
public class TransportPlane : Plane {}
```

3. **Поліморфізм.**

   Перша частина поліморфізма реалізована через базовий клас Vehicle, він включає в себе абстрактний метод:

```
public abstract bool CheckFunctionality();
```

   Цей абстрактний метод буде перевизначатися в класах наслідниках Car та Plane. Також у базовому класу буде знаходитися властивість:

```
public IFunctionality Functionality { get; protected set; }
```

Ця властивість знаходиться в класі під типом даних IFunctionaly, це інтерфейс, який реалізуть класи CarFunctionaly та PlaneFunctionaly.

```
public class CarFunctionality : IFunctionality
public class PlaneFunctionality : IFunctionality
```

Інтерфейс IFunctionaly включає один метод – IsNormalFunctionality, цей метод буде викликатися через абстрактний метод в класах Car, Plane.

```
public interface IFunctionality
{
  bool IsNormalFunctionality();
}

public override bool CheckFunctionality()
{
  return Functionality.IsNormalFunctionality();
}
```

Друга частина поліморфізма. Класи Bus, Truck, PassengerPlane, TransportPlane реалізують інтерфейс IInformationDetails, який має метод GetInformation(), що буде повертати повну інформацію про об’єкь в string форматі.

```
public interface IInformationDetails
{
  string GetInformation();
}
```

Приклад реалізації методу в класі Bus

```
public string GetInformation()
{
      return $"Type: Bus\n" +
             $"Name: {Name}\n" +
             $"Price: ${Price}\n" +
             $"Number of seats: {NumberOfSeats}pcs.\n" +
             $"Max capacity of people: {MaxCapacityOfPeople}pcs.\n" +
             $"Fuel type: Car's fuel: {FuelType.Name}\n" +
             $"Functionality  State:  {(Functionality.IsNormalFunctionality()  ?
             FunctionalyState.Good.ToString()                                  :
             FunctionalyState.Bad.ToString())}";
}
```

Виклик методу буде проходити через каст класу до інтерфейсу IInformationDetails, при отриманні його з DataGrid

```
((IInformationDetails)VehicleList.SelectedItem).GetInformation()
```

Третя реалізація поліморфізма. Для кожного UserControl, які будуть використовуватися для додавання нових транспортних засобів реалізувати

інтерфейс IInterfaceDataReceiver<T> загального типу, який включає в себе метод GetData, щоб створити потрібний тип транспорту.

```
public interface IInterfaceDataReceiver<T>
{
  T GetData();
}
```

```
public partial class BusCreatUserControl : UserControl, IInterfaceDataReceiver<Vehicle>
```

```
public partial class PassengerPlaneCreatUserControl : UserControl, IInterfaceDataReceiver<Vehicle>
```

```
public partial class TransportPlaneCreatUserControl : UserControl, IInterfaceDataReceiver<Vehicle>
```

```
public partial class TruckCreatUserControl : UserControl, IInterfaceDataReceiver<Vehicle>
```

Приклад реалізації методу в класі `BusCreatUserControl`

```
public Vehicle GetData()
{
      return new Bus(    Guid.NewGuid(),
                         Name.Text,
                         Model.Text,
                         Convert.ToDouble(Price.Text),
                         Convert.ToInt32(NumbersOfSeats.Text),
                         (CarFuel)FuelNameTypeFilter.SelectedItem,
                         new CarFunctionality(),
                         Convert.ToInt32(PeopleCapacity.Text)
                       );
}
```

Виклик методу GetData буде проходити в методі GetDataFromList, який шукає об’єкт реалізований через інтерфейс IInterfaceReceiver, кастить цей об’єкт до цього типу, та викликає метод GetData.

```
private T GetDataFromList<T>(IEnumerable<object> list)
{
      foreach (var control in list)
      {
          if (control is IInterfaceDataReceiver<T> receiver)
          {
              return receiver.GetData();
          }
      }

      throw new Exception("User Control is not defined!");
}
```

3.2. Створення базових класів

Спочатку треба створити перерахування FunctionalityState, яке буде відповідати за можливі стани функціювання, та перерахування TypeOfCargo, що буде відповідати за типи вантажу який може перевозити грузовий транспортний засіб. Далі потрібно створити абстрактний клас FuelType для створення різних видів пального та наслідувати його в два класи AviationFuel і CarFuel. Останнім з додаткових даних буде реалізація інтерфейсу IFunctionality в класах CarFunctionality та PlaneFunctionality, реалізацію цих класів можна переглянути в Додаток А

Далі необхідно реалізувати базовий абстрактний клас Vehicle, який буде потім наслідуватися в інші класи. У ньому потрібно створити 7 властивостей, з модифікатором доступу сетера protected та гетера - public:

1. Ідентифікатор (Id) з типом даних Guid, що зробити записи унікальними, та порівнювати унікальність по цьому полю.
2. Ім'я транспорту (Name), з типом даних string.
3. Модель (Model), з типом даних string
4. Ціна (Price), з типом даних double
5. Кількість сидінь (NumberOfSeats) – тип даних int
6. Тип пального (FuelType) – тип даних абстрактний клас FuelType
7. Функціювання (Functionality) – тип даних IFunctionality

Окрім властивостей, потрібно також об’явити абстрактний метод CheckFunctionality, та реалізувати інтерфейс IEquatable<Vehicle>, щоб при видаленні об’єкта зі списку, при пошуку використовувався ідентифікатор. Лістниг класу Vehicle можна переглянути в Додаток Б.Додаток Б

Після того як ми створили абстрактний метод Vehicle, треба створити класи Car та Plane(лістинг можна розглянути в Додаток В), в які буде наслідуватися Vehicle, та буде реалізовано абстрактний метод CheckFunctionality. **Додаток В**

Далі реалізуємо класи Bus, Truck, PassengerPlane, TransportPlane. Кожному також реалізуємо метод GetInformation інтерфейсу IInformationDetails. Класам TransportPlane та Truck добавимо дві властивості:

1. Максимальна вага (MaxWeightOfCargo) – типу даних double
2. Тип вантажу, який перевозить транспорт (CargoType) – тип даних перерахування TypeOfCargo.

Класу Bus потрібно добавити властивості

1. Максимальна кількість людей яка влазить в транспортний засіб (MaxCapacityOfPeople) – тип даних int.

Для PassengerPlane нічого додаткового реалізувати не потрібно, лістинг даних класів можна переглянути в Додаток Г.

Для реалізації потрібних функцій напишемо класи GasStation, який буде містити в собі список (зберігання буде саме під абстрактним типом List<FuelType>) пального і мати відповідні методи для роботи з ним(додавання\видалення\отримання конкретних видів пального) та VehicleShop, який вже буде працювати з списком транспортних засобі (також список з абстрактного класу List<Vehicle>) і також мати відповідні методи для роботи з ним (Додавання\видалення\отримання отфільтрованих транспортних засобів), реалізацію якого можна подивитися в Додаток Д.

3.3. Створення інтерфейсу

Створимо головне вікно з бічною панеллю, яке буде відповідати за переключення між сторінками, так поле на якому будуть розміщуватися вміст цих сторінок.(Рис. 3.1)

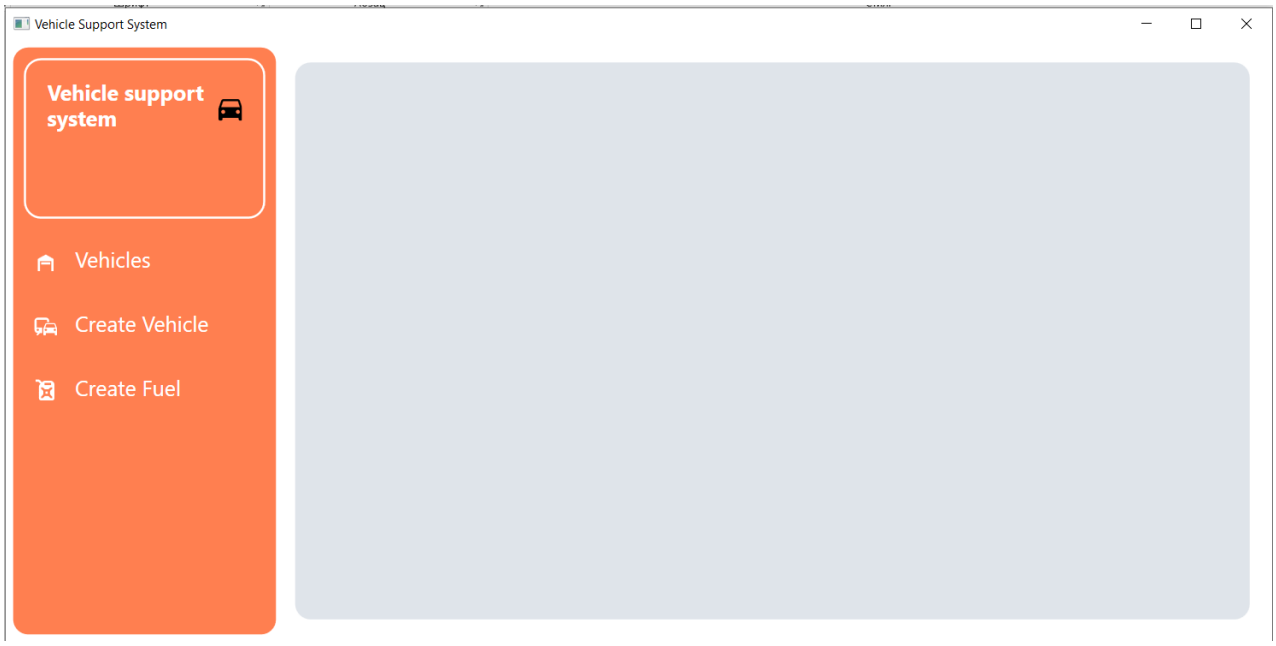

Рис. 3.1. Інтерфейс головного вікна

Далі необхідно створити три сторінки:

1. Сторінка із списком транспортних засобів. (Рис. 3.2)
2. Сторінка для додавання нових транспортних засобів. (Рис. 3.3)
3. Сторінка для додавання нових видів пального. (Рис. 3.4)

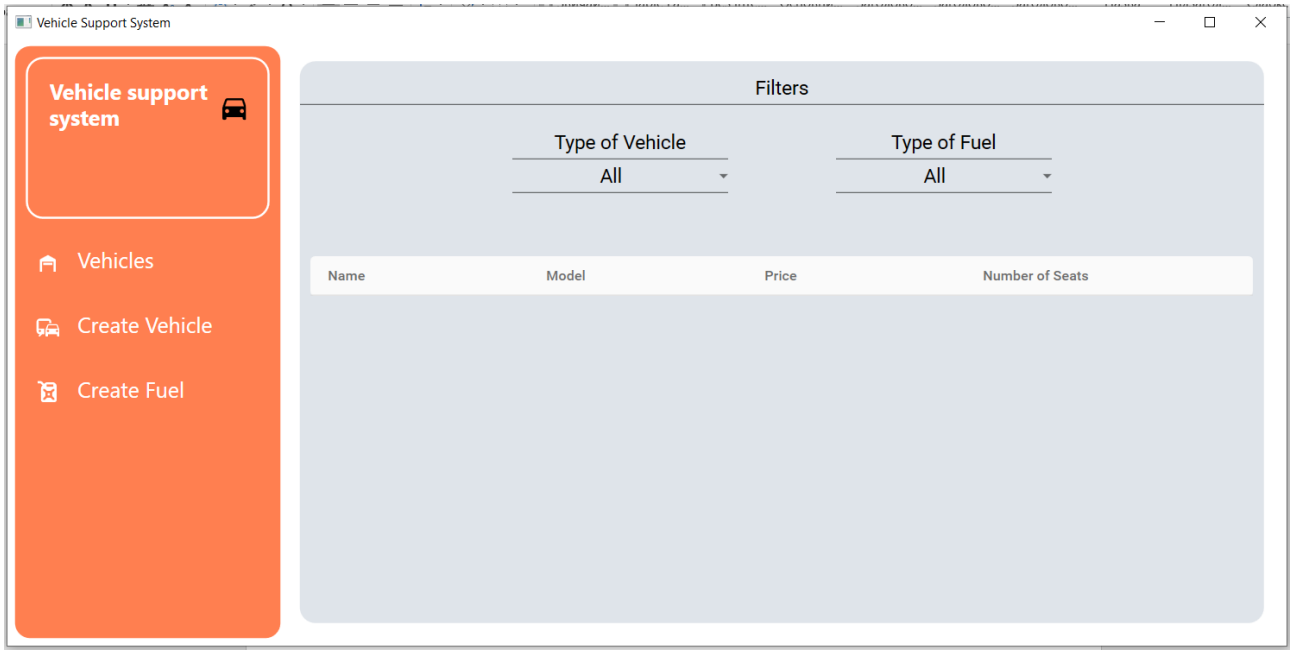

Рис. 3.2. Список транспортних засобів

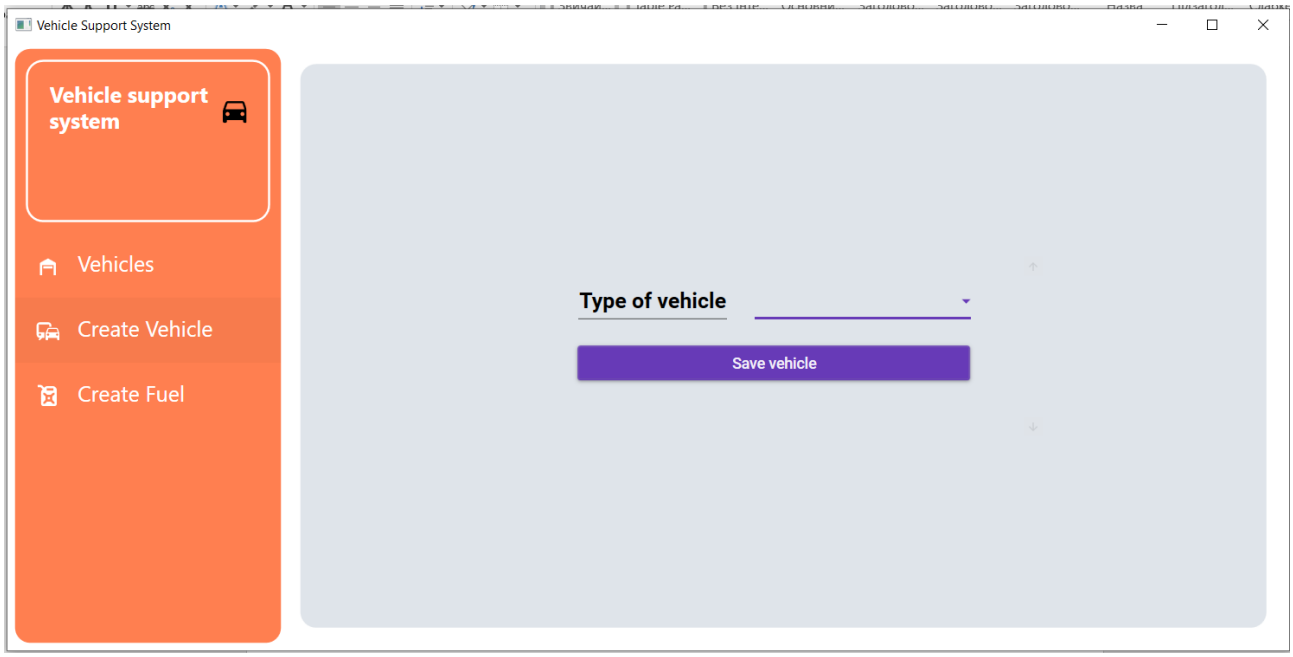

Рис. 3.3. Сторінка додавання транспортних засобів

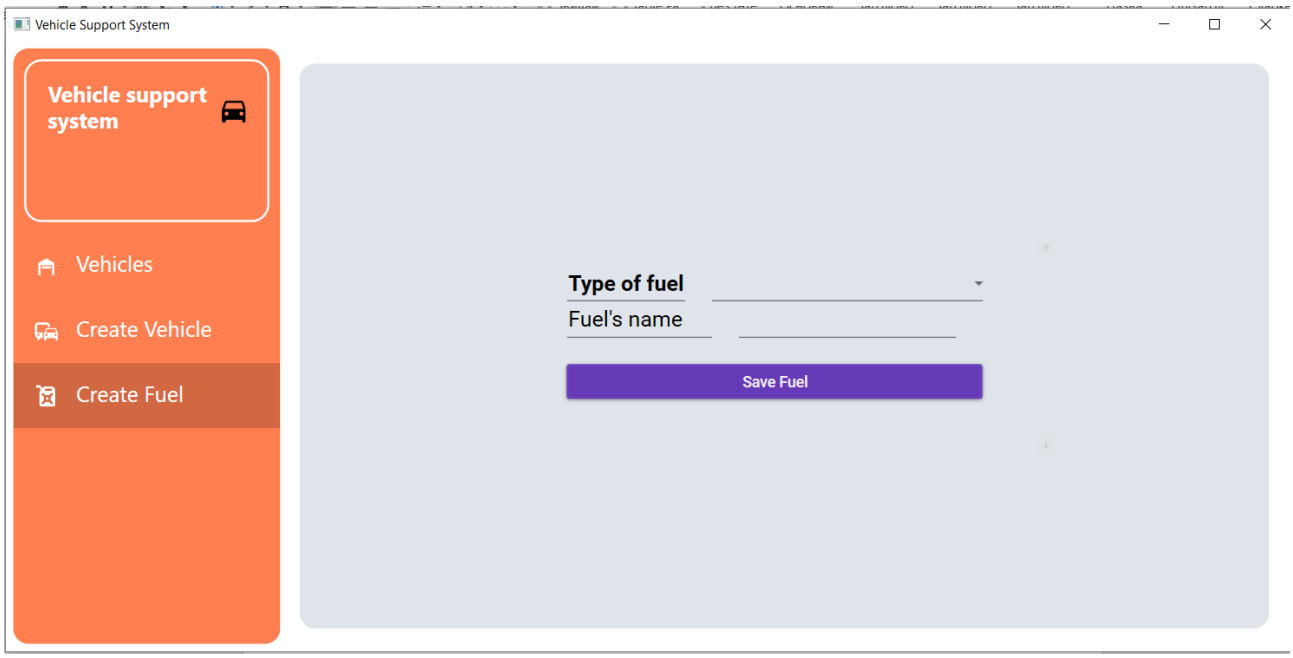

Рис. 3.4. Сторінка додавання пального

3.4. Діаграма класів

Розглянемо діграму абстрактного класу Vehicle та його нащадків.(Рис. 3.5)

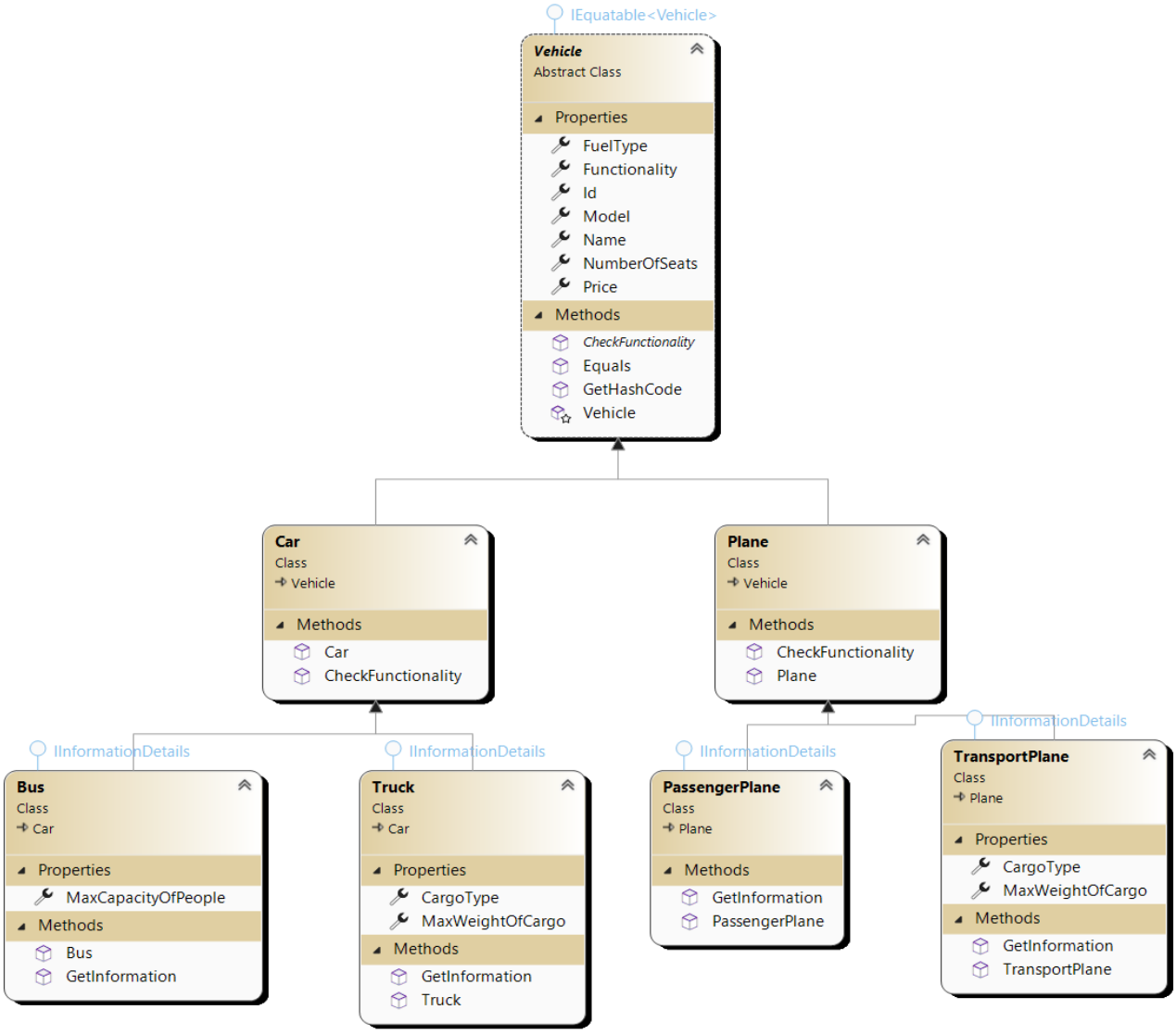

Рис. 3.5. Діаграма зв’язку абстрактного класу
Vehicel, та його наслідників

Діаграма абстрактного класу FuelType із його нащадками. (Рис. 3.6)

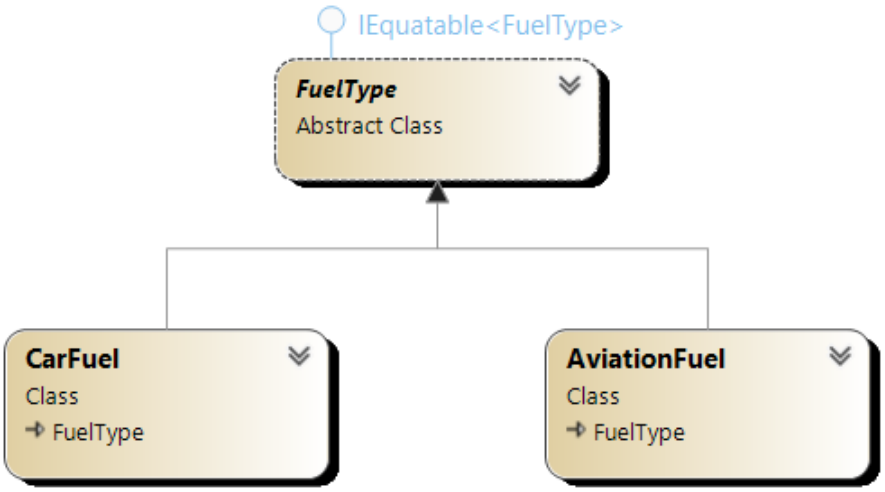

Рис. 3.6. Абстрактний клас FuelType, та його класи нащадки

Діаграма класів, що реалізували інтерфейс IFunctionaly. (Рис. 3.7)

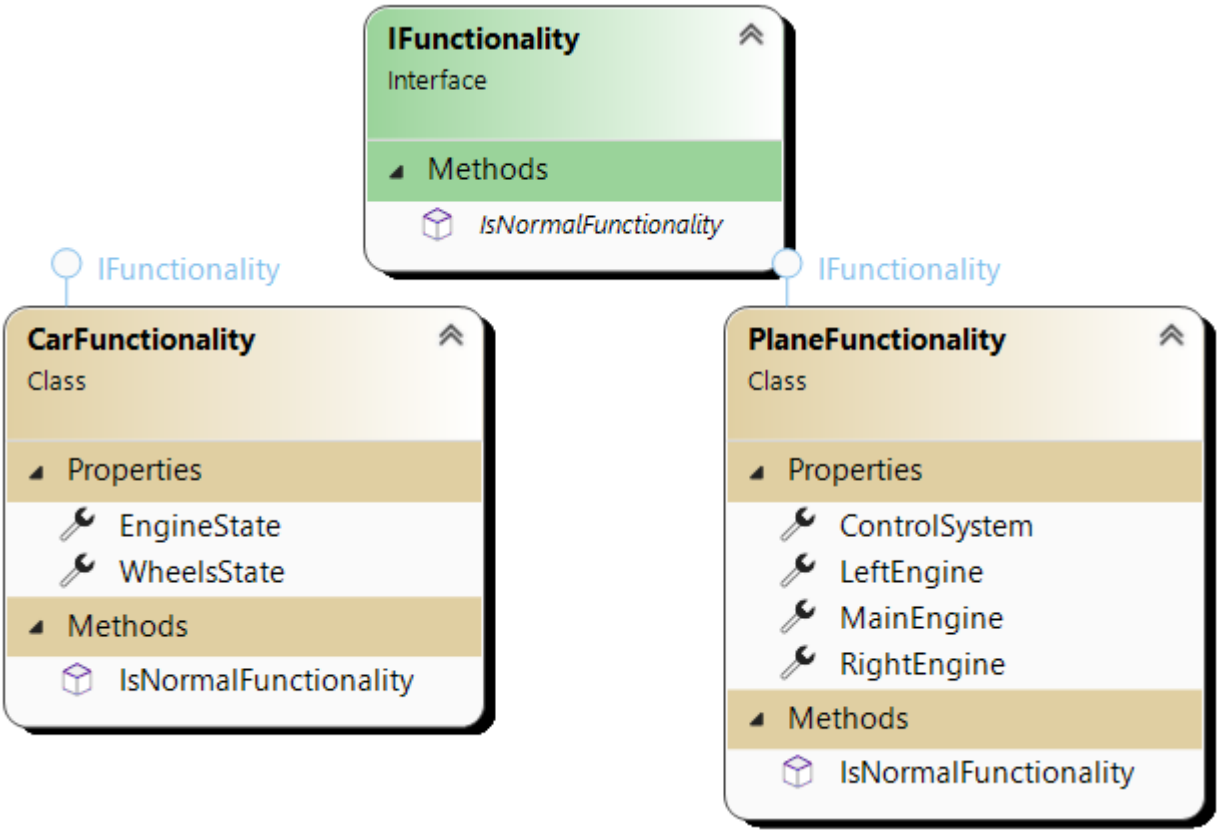

Рис. 3.7. Діаграма класів, що реалузують інтерфейс IFunctionaly

Діаграма реалізації інтерфейсу IInterfaceDataReceiver. (Рис. 3.8)

Рис. 3.8. Діаграма класів, що реалізують
інтерфейс IInterfaceDataReceiver

3.5. Висновок до третього розділу

В даному розділі було описано реалізація потрібних класів для роботи інформаційної системи підтримки продажу транспортних засобі та зроблено користувацький інтерфейс.. Описано реалізацію трьох концепцій ООП:

1. Інкапсуляція, через модифікатори доступу полів private і protected.
2. Наслідування. Класи Car, Plane успадковують Vehicle, та Bus, Truck і PassengerPlane, TransportPlane успадковують Car і Plane відповідно.

3. Поліморфізм був реалізований через абстрактний класс Vehicle та його метод CheckFunctionaly і поле Functionality. Також поліморфізм використався реалізації методу GetData інтерфейсу IInterfaceDataReceiver.

Загальний висновок

Для досягнення мети курсової роботи було вирішено поставленні завдання. Зокрема описано основний алгоритм роботи програми, та функції які потрібно реалізувати з використанням об’єктно орієнтованого програмування. Були сформовані блок схеми загальної реалізації основних функцій програми.

Після сформованих алгоритмів та побудованих узагальнених блок схем, було розглянуто реалізацію інформаційної системи. Описано, та наведено приклади реалізації інкапсуляції, наслідування, поліморфізма у даній роботі. Розглянуто створення користувацького інтерфейсу, та які функції він буде мати. Створено діаграму класів, які реалізують наслідування та поліморфізм.

Отже було реалізовано програмний продукт “Інформаційна система підтримки продажу транспортних засобів”, з використанням трьох концепцій об'єктно орієнтованого програмування, яка має функції ведення списків транспортних засобів, та їх фільтрування.

Джерела

1. ООП [Електронний ресурс]. Режим доступу: https://foxminded.ua/ua/shho-take-oop-ob-iektno-oriientovane-programuvannja
2. Інкапсуляція [Електронний ресурс]. Режим доступу: https://www.freecodecamp.org/news/object-oriented-programming-concepts-21bb035f7260
3. Наслідування [Електронний ресурс]. Режим доступу: https://www.partech.nl/en/publications/2020/10/basic-principles-of-object-oriented-programming#
4. Поліморфізм [Електронний ресурс]. Режим доступу: https://www.partech.nl/en/publications/2020/10/basic-principles-of-object-oriented-programming#
5. Список [Електронний ресурс]. Режим доступу: https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1?view=net-6.0

Додаток А

```
    public enum FunctionalyState
    {
        Good,
        Normal,
        Bad
    }
public enum TypeOfCargo
    {
        Humanitarian,
        Flammable,
        Fragile
    }


public interface IFunctionality
    {
        bool IsNormalFunctionality();
    }


public class CarFunctionality : IFunctionality
    {
        public FunctionalyState WheelsState { get; private set; }
        public FunctionalyState EngineState { get; private set; }

        public bool IsNormalFunctionality()
        {
            return (WheelsState == FunctionalyState.Normal || WheelsState ==
FunctionalyState.Good) &&
                   EngineState == FunctionalyState.Good;
        }
    }
public class PlaneFunctionality : IFunctionality
    {
        public FunctionalyState LeftEngine { get; private set; }
        public FunctionalyState RightEngine { get; private set; }
        public FunctionalyState MainEngine { get; private set; }
        public FunctionalyState ControlSystem { get; private set; }

        public bool IsNormalFunctionality()
        {
            return LeftEngine == FunctionalyState.Good && RightEngine ==
FunctionalyState.Good &&
                  MainEngine == FunctionalyState.Good && ControlSystem ==
FunctionalyState.Good;
        }
    }
```

```
public abstract class FuelType : IEquatable<FuelType>
    {
        public Guid Id { get; private set; }
        public string Name { get; private set; } = string.Empty;

        protected FuelType(Guid id, string name)
        {
            Id = id;
            Name = name;
        }

        public bool Equals(FuelType? other)
        {
            if (other == null)
                return false;

            return Id.Equals(other.Id);
        }

        public override int GetHashCode()
        {
            return Id.GetHashCode();
        }

        public override string ToString()
        {
            return Name;
        }
    }

public class AviationFuel : FuelType
    {
        public AviationFuel(Guid id, string name) : base(id, name)
        {

        }
    }

public class CarFuel : FuelType
    {
        public CarFuel(Guid id, string name) : base(id, name)
        {

        }
    }
```

Додаток Б

```
public abstract class Vehicle : IEquatable<Vehicle>
    {
        public Guid Id { get; protected set; }
        public string Name { get; protected set; }
        public string Model { get; protected set; }
        public double Price { get; protected set; }
        public int NumberOfSeats { get; protected set; }
        public FuelType FuelType { get; protected set; }
        public IFunctionality Functionality { get; protected set; }

        protected Vehicle(Guid id, string name, string model, double price, int
        numberOfSeats, FuelType fuelType, IFunctionality functionality)
        {
            if (name == "" || model == "")
                throw new Exception("Incorrect name or model!");
            if(price <= 0)
                throw new Exception("Incorrect price!");
            if(numberOfSeats <= 0)
                throw new Exception("Incorrect number Of Seats!");
            if (fuelType == null)
                throw new Exception("Fuel type can't be null!");

            Id = id;
            Name = name;
            Price = price;
            NumberOfSeats = numberOfSeats;
            FuelType = fuelType;
            Functionality = functionality;
            Model = model;
        }

        public abstract bool CheckFunctionality();

        public bool Equals(Vehicle? other)
        {
            if (other == null) return false;
            return Id.Equals(other.Id);
        }

        public override int GetHashCode()
        {
            return Id.GetHashCode();
        }
    }
```

Додаток В

```
public class Plane : Vehicle
    {
      public Plane(Guid id, string name,  string model, double price, int
      numberOfSeats, FuelType fuelType, IFunctionality functionality)
                    : base(id, name, model, price, numberOfSeats, fuelType,
                    functionality)
        {

        }

        public override bool CheckFunctionality()
        {
            return Functionality.IsNormalFunctionality();
        }
    }


public class Car : Vehicle
    {
      public Car(Guid id, string name, string model, double price, int numberOfSeats,
      FuelType fuelType, IFunctionality functionality)
                  : base(id, name, model, price, numberOfSeats, fuelType,
                  functionality)
        {

        }

        public override bool CheckFunctionality()
        {
            return Functionality.IsNormalFunctionality();
        }
    }
```

Додаток Г

```
public class Bus : Car, IInformationDetails
    {
        public Bus(Guid id, string name, string model, double price, int
numberOfSeats, FuelType fuelType, IFunctionality functionality, int
maxCapacityOfPeople)
                    : base(id, name, model, price, numberOfSeats, fuelType,
functionality)
        {
            MaxCapacityOfPeople = maxCapacityOfPeople;
        }

        public int MaxCapacityOfPeople { get; private set; }

        public string GetInformation()
        {
            return $"Type: Bus\n" +
                   $"Name: {Name}\n" +
                   $"Price: ${Price}\n" +
                   $"Number of seats: {NumberOfSeats}pcs.\n" +
                   $"Max capacity of people: {MaxCapacityOfPeople}pcs.\n" +
                   $"Fuel type: Car's fuel: {FuelType.Name}\n" +
                   $"Functionality State: {(Functionality.IsNormalFunctionality() ?
FunctionalyState.Good.ToString() : FunctionalyState.Bad.ToString())}";
        }
    }


public class Truck : Car, IInformationDetails
    {
        public Truck(Guid id, string name, string model, double price, int
numberOfSeats, FuelType fuelType,
                    IFunctionality functionality, double maxWeightOfCargo, TypeOfCargo
cargoType)
                    : base(id, name, model,price, numberOfSeats, fuelType,
functionality)
        {
            MaxWeightOfCargo = maxWeightOfCargo;
            CargoType = cargoType;
        }

        public double MaxWeightOfCargo { get; private set; }
        public TypeOfCargo CargoType { get; private set; }

        public string GetInformation()
        {
            return $"Type: Truck\n" +
                   $"Name: {Name}\n" +
                   $"Price: ${Price}\n" +
                   $"Number of seats: {NumberOfSeats}pcs.\n" +
                   $"Fuel type: Car's fuel: {FuelType.Name}\n" +
                   $"Type of Cargo: {CargoType}\n" +
                   $"Max Weight Of Cargo: {MaxWeightOfCargo}kg.\n" +
                   $"Functionality State: {(Functionality.IsNormalFunctionality() ?
FunctionalyState.Good.ToString() : FunctionalyState.Bad.ToString())}";
        }
    }
```

```
public class PassengerPlane : Plane, IInformationDetails
    {
        public PassengerPlane(Guid id, string name, string model, double price, int
numberOfSeats, FuelType fuelType, IFunctionality functionality) :
                                    base(id, name, model, price, numberOfSeats, fuelType,
functionality)
        {

        }

        public string GetInformation()
        {
            return $"Type: Passenger Plane\n" +
                   $"Name: {Name}\n" +
                   $"Price: ${Price}\n" +
                   $"Number of seats: {NumberOfSeats}pcs.\n" +
                   $"Fuel type: Aviation's fuel: {FuelType.Name}\n" +
                   $"Functionality State: {(Functionality.IsNormalFunctionality() ?
FunctionalyState.Good.ToString() : FunctionalyState.Bad.ToString())}";
        }
    }


public class TransportPlane : Plane, IInformationDetails
    {
        public TransportPlane(Guid id, string name, string model, double price, int
numberOfSeats, FuelType fuelType,
                                IFunctionality functionality, double maxWeightOfCargo,
TypeOfCargo cargoType)
                               : base(id, name, model, price, numberOfSeats, fuelType,
functionality)
        {
            MaxWeightOfCargo = maxWeightOfCargo;
            CargoType = cargoType;
        }

        public double MaxWeightOfCargo { get; private set; }
        public TypeOfCargo CargoType { get; private set; }

        public string GetInformation()
        {
            return $"Type: TransportPlane\n" +
                   $"Name: {Name}\n" +
                   $"Price: ${Price}\n" +
                   $"Number of seats: {NumberOfSeats}pcs.\n" +
                   $"Fuel type: Car's fuel: {FuelType.Name}\n" +
                   $"Type of Cargo: {CargoType}\n" +
                   $"Max Weight Of Cargo: {MaxWeightOfCargo}kg.\n" +
                   $"Functionality State: {(Functionality.IsNormalFunctionality() ?
FunctionalyState.Good.ToString() : FunctionalyState.Bad.ToString())}";
        }
    }
```

Додаток Д

```
public class GasStation
    {
        public List<FuelType> Fuels { get; private set; }

        public GasStation(List<FuelType> fuels)
        {
            Fuels = fuels;
        }

        public void AddFuel(FuelType obj)
        {
            Fuels.Add(obj);
        }

        public List<FuelType> GetCarFuels()
        {
            List<FuelType> fuels = new List<FuelType>();

            foreach (var fuel in Fuels)
            {
                if(fuel is CarFuel)
                {
                    fuels.Add((CarFuel)fuel);
                }
            }

            return fuels;
        }

        public List<FuelType> GetPlaneFuels()
        {
            List<FuelType> fuels = new List<FuelType>();

            foreach (var fuel in Fuels)
            {
                if (fuel is AviationFuel)
                {
                    fuels.Add((AviationFuel)fuel);
                }
            }

            return fuels;
        }


    }
```

```
public class VehiclesShop
    {
        private List<Vehicle> _vehicleList;

        public VehiclesShop(List<Vehicle> cars)
        {
            if (cars == null)
                throw new Exception("Car list can't be a null");

            _vehicleList = cars;
        }

        public List<Vehicle> VehicleList
        {
            get
            {
                return _vehicleList;
            }
            private set
            {
                _vehicleList = value;
            }
        }

        public void AddVehicle(Vehicle car)
        {
            VehicleList.Add(car);
        }

        public void RemoveVehicle(Vehicle car)
        {
            VehicleList.Remove(car);
        }

        public List<Vehicle> GetVehiclesByFilters(List<FilterCarsDelegate> filterCars)
        {
            List<Vehicle> list = new List<Vehicle>();

            foreach (var vehicle in VehicleList)
            {
                bool canToAdd = false;

                foreach(var method in filterCars)
                {
                    canToAdd = method(vehicle);
                    if (!canToAdd)
                        break;
                }

                if(canToAdd)
                    list.Add(vehicle);
            }

            return list;
        }

    }
```