МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

Комп‘ютерний практикум №2

з курсу «Основи розробки програмного забезпечення на платформі Microsoft.NET»

на тему: «LINQ to Objects»

Перевірила:
доцент,
Ліщук К. І.

Виконав:
студент 2 курсу
групи ІП-21 ФІОТ
Гриценко А. В.

Київ 2024

Варіант 19

Мета: ознайомитися з обробкою XML документів з використанням технології LINQ to XML

Постановка задачі комп‘ютерного практикуму № 2
При виконанні комп‘ютерного практикуму необхідно виконати наступні дії:

1) Розробити структуру XML для зберігання даних згідно варіантів, наведених нижче.

2) Створити XML-файл з використанням XmlWriter. Дані необхідно вводити з консолі, зберегти.

3) Продемонструвати завантаження створеного в п.2 файлу (або заздалегідь створеного) з використанням:
   a. XmlDocument;
   b. XmlSerializer.

4) LINQ to XML:
   a. Вивести зміст файлу, створеного в п.2 або заздалегідь створеного для демонстрації
   b. Розробити як мінімум 15 різних запитів, використовуючи різні дії над отриманими даними. Запити не повинні повторюватись.

5) Створити програмне забезпечення, котре реалізує обробку даних з використання бібліотеки LINQ to XML

6) Програмне забезпечення необхідно розробити у вигляді консольного застосування на мові C#.

7) Звіт повинен містити: опис архітектури проекту, словесний опис запитів, текст програмного коду, скріншоти результатів виконання.

Варіант індивідуального завдання:

19) Розробити структуру даних для зберігання інформації про реєстрацію транспортних засобів. Для кожного транспортного засобу зберігається як мінімум марка авто, виробник, модель, тип кузову, рік випуску, номер шасі (VIN-код), колір, номерний знак, технічний стан, власник автомобіля, перелік водіїв, котрі мають право керувати транспортним засобом, тощо. Для власників та тих персон, котрі мають право керувати транспортним засобом, - номер прав водія, прізвище, ім'я, по батькові, дата народження, адреса реєстрації. Необхідно врахувати, що транспортний засіб може мати декілька власників (тобто бути зареєстрованим декілька разів).

ER-diagram

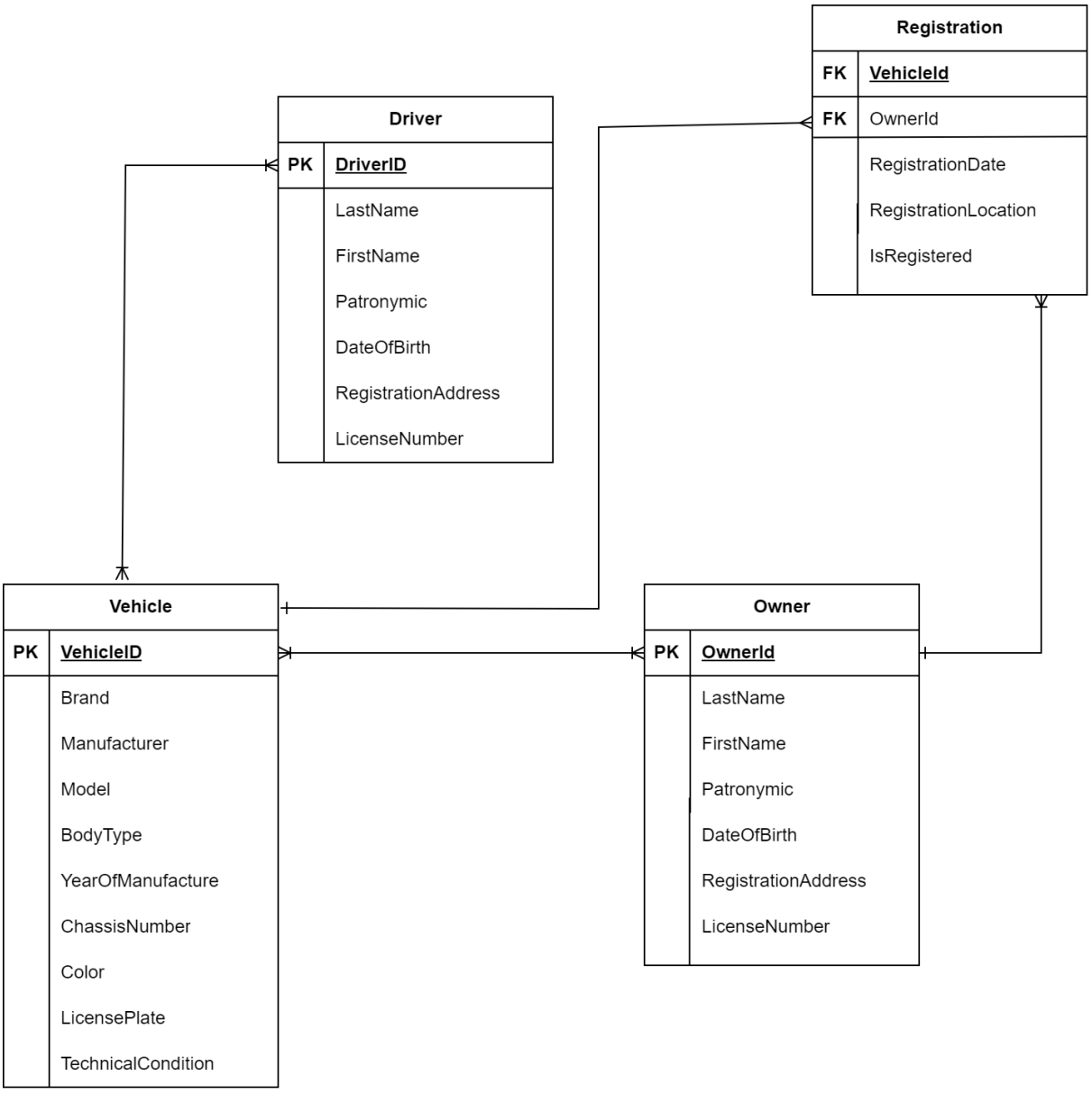


Рисунок 1. ER diagram

Vehicle - Driver (багато до багатьох): кілька водіїв можуть керувати кількома транспортними засобами, і навпаки. Наприклад, у службах оренди або сімейних автомобілях, де кілька членів сім’ї можуть керувати одним транспортним засобом.

Vehicle - Owner (багато до багатьох): кілька власників можуть мати право власності на декілька транспортних засобів, і навпаки.

Vehicle - Registration (один до багатьох): кожен транспортний засіб може мати декілька реєстраційних записів з часом, але кожен реєстраційний запис пов’язаний лише з одним транспортним засобом. У реальних сценаріях транспортні засоби можуть проходити кілька реєстрацій через зміну власності, місцезнаходження чи інші фактори, але кожна реєстрація стосується окремого автомобіля.

Owner - Registration (один до багатьох): кожен власник може мати кілька реєстраційних записів протягом певного часу, але кожен реєстраційний запис пов’язано лише з одним власником.

Основні класи, що реалізують дану діаграму:

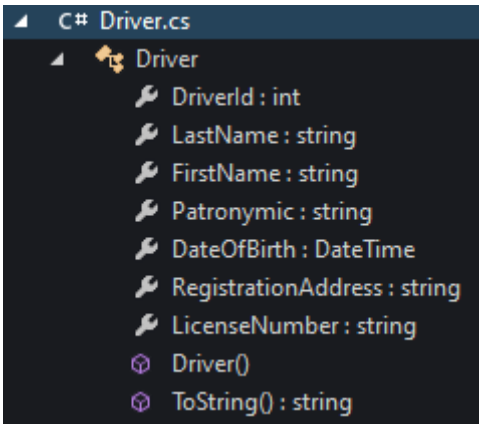


Рисунок 2. Клас водія

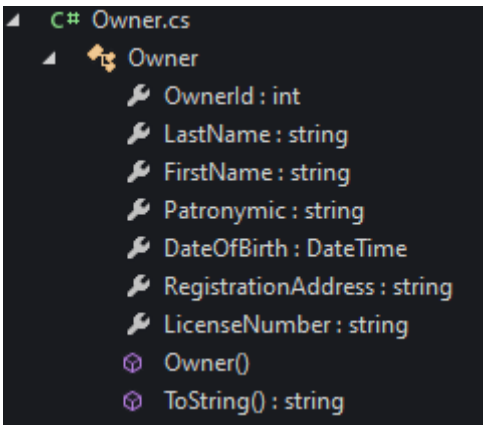


Рисунок 3. Клас власника

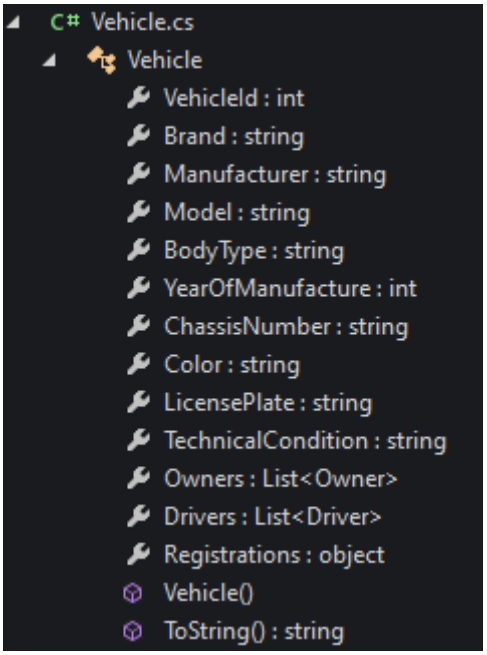


Рисунок 4. Клас транспортного засобу

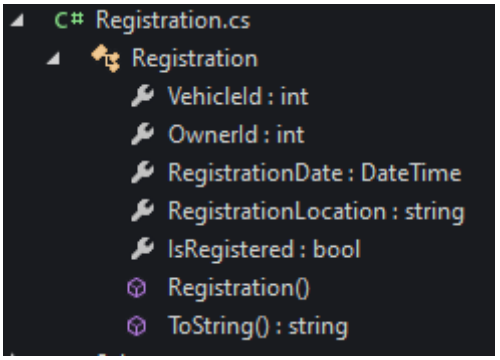


Рисунок 5. Клас реєстрації

Допоміжні класи, що реалізують зв’язок багато-до-багатьох

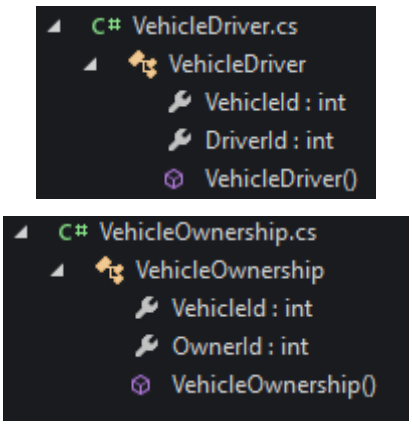


Рисунок 6. Допоміжні класи для зв’язків

XML-файли

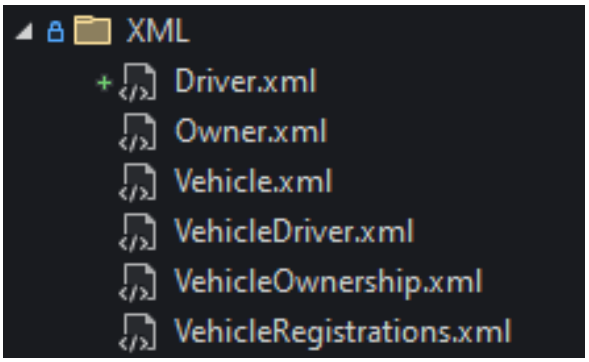


Рисунок 7. Файли XML

Допоміжні класи, що реалізують логіку обробки XML документів

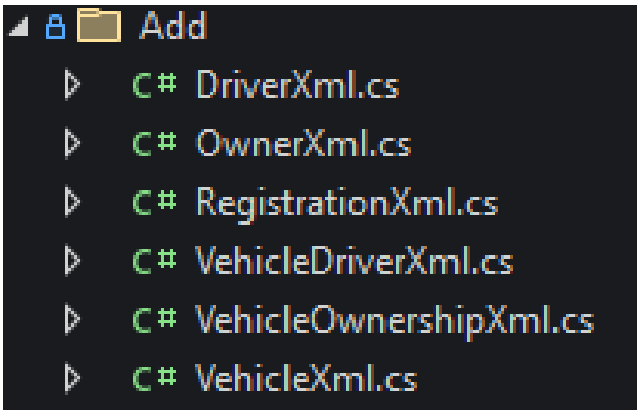
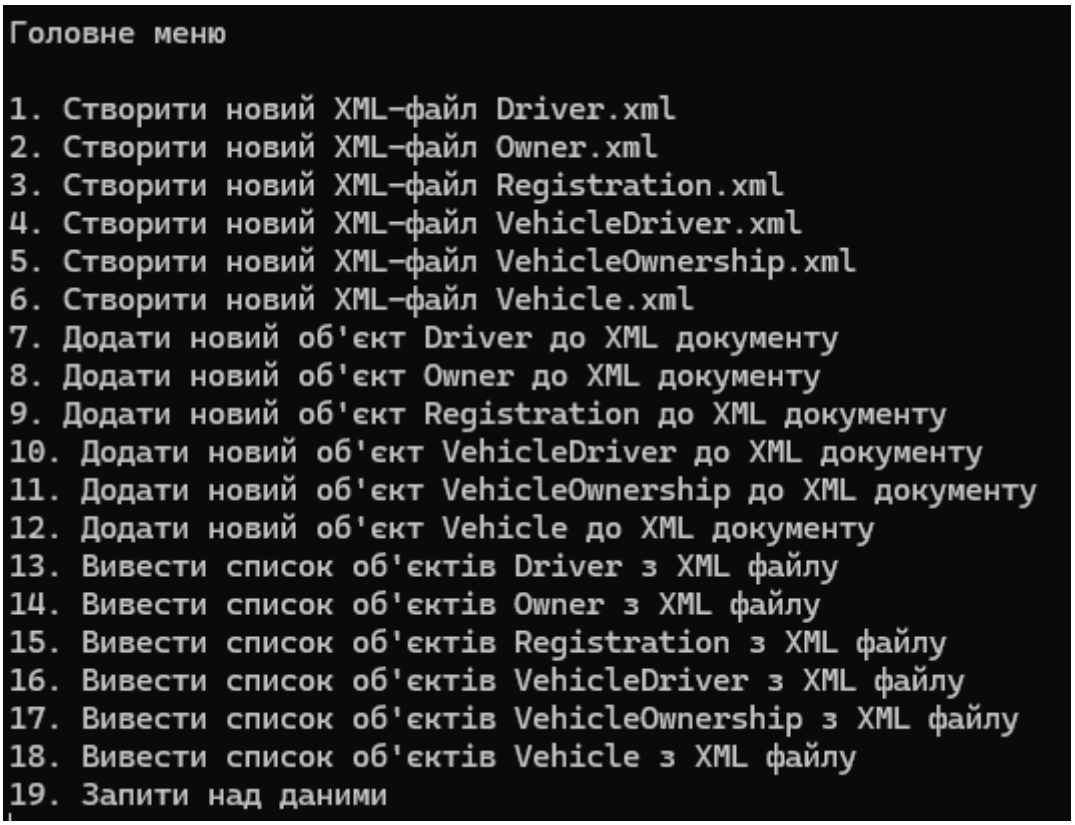


Рисунок 8. Допоміжні класи для обробки XML документів

Приклад роботи програми



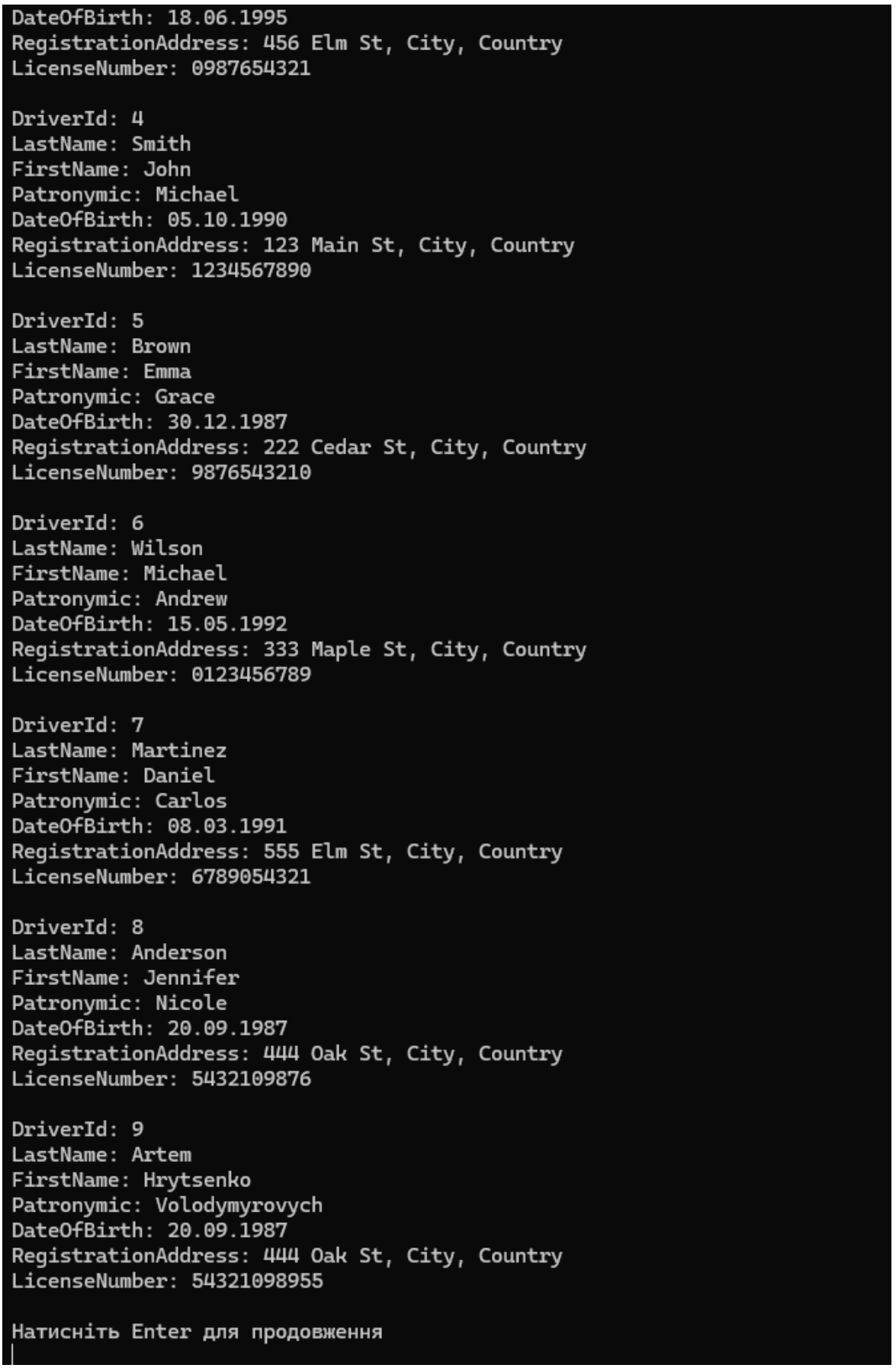


Рисунок 9. Приклади роботи програми

Словесний опис запитів

GetSUVVehicles : цей запит отримує всі транспортні засоби, які мають тип кузова "SUV". Потім відображається список цих транспортних засобів, включаючи їх марку, модель і рік випуску, якщо такі є.

GetAllSUVsWithOwners : цей запит отримує всі позашляховики разом із відповідними власниками. Він об’єднує інформацію з кількох таблиць, щоб зіставити кожен SUV з його власником, а потім відображає цю інформацію, включаючи марку та модель автомобіля, а також ім’я та прізвище власника.

GetMachinesInExcellentCondition : цей запит отримує список транспортних засобів, які знаходяться у відмінному технічному стані. Він фільтрує транспортні засоби за ознакою їх технічного стану та відображає марку та модель кожного автомобіля, який відповідає цьому критерію.

GetOwnersWithinAgeRange : цей запит отримує список власників, вік яких входить у вказаний діапазон. Він обчислює діапазон дат народження на основі наданих мінімальних і максимальних років, а потім відображає імена та дати народження власників, які відповідають цим критеріям.

GetDriverWithLongestName : цей запит знаходить водія із найдовшою назвою. Він розраховує довжину повного імені кожного водія (ім’я, прізвище та по батькові) і вибирає водія з максимальною довжиною імені, а потім відображає його повне ім’я.

GetVehicleWithEarliestManufactureYear : цей запит отримує транспортний засіб із найранішим роком виробництва. Він упорядковує транспортні засоби за роками випуску та вибирає той із найранішим роком, а потім відображає його марку, модель і рік виробництва.

SortDriversByDateOfBirthAscendingWithBoundaries : цей запит отримує відсортований список водіїв за датою народження в межах зазначеного діапазону. Він фільтрує водіїв за роками їх народження, що входять у вказані межі, сортує їх за датою народження, а потім відображає їхні імена та дати народження в порядку зростання.

GetUnregisteredVehicles : цей запит отримує список транспортних засобів, які не зареєстровані. Він перевіряє, чи є будь-які реєстрації, пов’язані з кожним транспортним засобом, і вибирає транспортні засоби без будь-яких реєстрацій, а потім відображає їх марку, модель та ідентифікатор автомобіля.

GetMachinesReleasedBeforeYear : цей запит отримує список транспортних засобів, які були випущені до вказаного року. Він фільтрує транспортні засоби на основі того, що роки їх виробництва менші або дорівнюють указаному року, і відображає їх марку, модель і рік виробництва.

GetVehiclesRegisteredInLocalOffice : цей запит отримує список транспортних засобів, зареєстрованих у розташуванні «Місцеве реєстраційне відділення». Він поєднує інформацію з реєстраційних таблиць і таблиць транспортних засобів, щоб відповідати транспортним засобам, зареєстрованим у цьому місці, а потім відображає їх марку та модель.

GetVehiclesNotDrivenByDrivers : отримує список транспортних засобів, якими не керують водії. Він перевіряє зв’язок між транспортними засобами та водіями в таблиці vehicleDriver і вибирає транспортні засоби, які не мають відповідних записів водіїв.

GetUniqueOwnersAndDriversNames : отримує список унікальних імен шляхом поєднання імен власників і водіїв. Він окремо отримує імена та прізвища власників і водіїв, об’єднує їх, видаляє дублікати та повертає унікальний список імен.

GetOwnersWithMultipleVehicles : отримує список власників, які володіють кількома транспортними засобами. Він групує записи про власність транспортного засобу за ідентифікатором власника, фільтрує власників із кількома транспортними засобами та повертає їх деталі.

SortVehiclesByModelYearDescendingWithBoundaries : отримує список транспортних засобів, відсортованих за роком виробництва в порядку спадання в межах зазначеного діапазону. Він вибирає транспортні засоби,

вироблені між заданими роками, упорядковує їх за роком виробництва в порядку спадання та повертає відсортований список.

FindDriversWithoutVehicles : отримує список водіїв, які не мають пов’язаних транспортних засобів. Він перевіряє наявність водіїв, які не мають відповідних записів у таблиці vehicleDriver, і повертає їхні дані.

GetAverageAgeOfDrivers : обчислює середній вік водіїв. Він обчислює вік кожного водія на основі дати його народження, підсумовує вік, обчислює середнє значення та повертає його.

GetDriversYoungerThanAge : отримує список водіїв, які молодші за вказаний вік. Він розраховує максимальну дату народження на основі даного віку, фільтрує водіїв, молодших за цей вік, і повертає їхні дані.

GetVehiclesRegisteredInYear : отримує список транспортних засобів, зареєстрованих у певному році. Він фільтрує реєстраційні записи за роком реєстрації та поєднує їх із записами транспортних засобів, щоб отримати список транспортних засобів, зареєстрованих у цьому році.

GetDriversBornInYear : отримує список водіїв, які народилися в певний рік. Він фільтрує водіїв за роком народження та повертає їхні дані.

GetCarsOwnedByOwner : отримує список автомобілів, які належать певному ідентифікатору власника. Він поєднує записи про володіння транспортним засобом із записами про транспортні засоби, щоб ідентифікувати автомобілі, що належать зазначеному власнику, і повертає їх марку та модель.

Висновок

У даній роботі ми ознайомилися з обробкою XML документів за допомогою технології LINQ to XML в середовищі C#. Було розроблено структуру XML, створено XML-файлу з використанням XmlWriter, завантажено та оброблено XML-даних за допомогою XmlDocument та XmlSerializer, а також реалізовано запити. Реалізація програмного забезпечення у вигляді консольного застосування на мові C#, яке демонструє обробку даних з використанням LINQ to XML.

Програмний код

Реалізація основних класів:

Driver.cs:

```
using System;
using System.Collections.Generic;
using System.Globalization;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Serialization;
namespace laba2
{
   public class Driver
   {
      public int DriverId { get; set; }
      public string? LastName { get; set; }
      public string? FirstName { get; set; }
      public string? Patronymic { get; set; }
      public string? RegistrationAddress { get; set; }
      public string? LicenseNumber { get; set; }

      [XmlIgnore]
      public DateTime DateOfBirth { get; set; }

      [XmlElement("DateOfBirth")]
      public string DateOfBirthString
      {
         get { return DateOfBirth.ToString("dd.MM.yyyy"); }
         set { DateOfBirth = DateTime.ParseExact(value, "dd.MM.yyyy", null);
}
      }
```

```
        public Driver(int driverId, string? lastName, string? firstName, string?
patronymic, DateTime dateOfBirth, string? registrationAddress, string?
licenseNumber)
        {
            DriverId = driverId;
            LastName = lastName;
            FirstName = firstName;
            Patronymic = patronymic;
            DateOfBirth = dateOfBirth;
            RegistrationAddress = registrationAddress;
            LicenseNumber = licenseNumber;
        }

        public Driver()
        {

        }

        public static Driver CreateDriver()
        {
            int driverId;
            string? lastName, firstName, patronymic, registrationAddress,
licenseNumber;
            DateTime dateOfBirth;

            Console.WriteLine("Enter Driver ID:");
            while (!int.TryParse(Console.ReadLine(), out driverId) || driverId <= 0)
            {
                Console.WriteLine("Invalid Driver ID. It should be a positive
integer.");
                Console.WriteLine("Enter Driver ID:");
            }

            Console.WriteLine("Enter Last Name:");
            lastName = Console.ReadLine();

            Console.WriteLine("Enter First Name:");
```

```
            firstName = Console.ReadLine();

            Console.WriteLine("Enter Patronymic:");
            patronymic = Console.ReadLine();

            Console.WriteLine("Enter Date of Birth (dd.MM.yyyy):");
            while (!DateTime.TryParseExact(Console.ReadLine(), "dd.MM.yyyy",
CultureInfo.InvariantCulture, DateTimeStyles.None, out dateOfBirth))
            {
                Console.WriteLine("Invalid Date. Please enter in dd.MM.yyyy
format:");
            }

            Console.WriteLine("Enter Registration Address:");
            registrationAddress = Console.ReadLine();

            Console.WriteLine("Enter License Number:");
            licenseNumber = Console.ReadLine();

            return new Driver(driverId, lastName, firstName, patronymic,
dateOfBirth, registrationAddress, licenseNumber);
        }
    }
}
```

Owner.cs:

```
using System;
using System.Collections.Generic;
using System.Globalization;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Serialization;
namespace laba2
{
    public class Owner
```

```
    {
        public int OwnerId { get; set; }
        public string? LastName { get; set; }
        public string? FirstName { get; set; }
        public string? Patronymic { get; set; }

        [XmlIgnore]
        public DateTime DateOfBirth { get; set; }

        [XmlElement("DateOfBirth")]
        public string DateOfBirthString
        {
            get { return DateOfBirth.ToString("dd.MM.yyyy"); }
            set { DateOfBirth = DateTime.ParseExact(value, "dd.MM.yyyy", null);
}
        }

        public string? RegistrationAddress { get; set; }
        public string? LicenseNumber { get; set; }

        public Owner(int ownerId, string? lastName, string? firstName, string?
patronymic, DateTime dateOfBirth, string? registrationAddress, string?
licenseNumber)
        {
            OwnerId = ownerId;
            LastName = lastName;
            FirstName = firstName;
            Patronymic = patronymic;
            DateOfBirth = dateOfBirth;
            RegistrationAddress = registrationAddress;
            LicenseNumber = licenseNumber;
        }

        public Owner() { }

        public static Owner CreateOwner()
        {
```

```
        int ownerId;
        string? lastName, firstName, patronymic, registrationAddress,
licenseNumber;
        DateTime dateOfBirth;

        Console.WriteLine("Enter Owner ID:");
        while (!int.TryParse(Console.ReadLine(), out ownerId) || ownerId <= 0)
        {
            Console.WriteLine("Invalid Owner ID. It should be a positive
integer.");
            Console.WriteLine("Enter Owner ID:");
        }

        Console.WriteLine("Enter Last Name:");
        lastName = Console.ReadLine();

        Console.WriteLine("Enter First Name:");
        firstName = Console.ReadLine();

        Console.WriteLine("Enter Patronymic:");
        patronymic = Console.ReadLine();

        Console.WriteLine("Enter Date of Birth (dd.MM.yyyy):");
        while (!DateTime.TryParseExact(Console.ReadLine(), "dd.MM.yyyy",
CultureInfo.InvariantCulture, DateTimeStyles.None, out dateOfBirth))
        {
            Console.WriteLine("Invalid Date. Please enter in dd.MM.yyyy
format:");
        }

        Console.WriteLine("Enter Registration Address:");
        registrationAddress = Console.ReadLine();

        Console.WriteLine("Enter License Number:");
        licenseNumber = Console.ReadLine();
```

```
            return new Owner(ownerId, lastName, firstName, patronymic,
dateOfBirth, registrationAddress, licenseNumber);
        }
    }
}
}
```

Vehicle.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace laba2
{
    public class Vehicle
    {
        public int VehicleId { get; set; }
        public string? Brand { get; set; }
        public string? Manufacturer { get; set; }
        public string? Model { get; set; }
        public string? BodyType { get; set; }
        public int YearOfManufacture { get; set; }
        public string? ChassisNumber { get; set; }
        public string? Color { get; set; }
        public string? LicensePlate { get; set; }
        public string? TechnicalCondition { get; set; }
        public List<Registration>? Registrations { get; set; }

        public Vehicle(int vehicleId, string? brand, string? manufacturer, string?
model, string? bodyType, int yearOfManufacture, string? chassisNumber,
string? color, string? licensePlate, string? technicalCondition,
List<Registration>? registrations)
        {
            VehicleId = vehicleId;
```

```
            Brand = brand;
            Manufacturer = manufacturer;
            Model = model;
            BodyType = bodyType;
            YearOfManufacture = yearOfManufacture;
            ChassisNumber = chassisNumber;
            Color = color;
            LicensePlate = licensePlate;
            TechnicalCondition = technicalCondition;
            Registrations = registrations;
        }

        public Vehicle() { }

        public static Vehicle CreateVehicle()
        {
            int vehicleId, yearOfManufacture;
            string? brand, manufacturer, model, bodyType, chassisNumber, color,
licensePlate, technicalCondition;

            Console.WriteLine("Enter Vehicle ID:");
            while (!int.TryParse(Console.ReadLine(), out vehicleId) || vehicleId <=
0)
            {
                Console.WriteLine("Invalid Vehicle ID. It should be a positive
integer.");
                Console.WriteLine("Enter Vehicle ID:");
            }

            Console.WriteLine("Enter Brand:");
            brand = Console.ReadLine();

            Console.WriteLine("Enter Manufacturer:");
            manufacturer = Console.ReadLine();

            Console.WriteLine("Enter Model:");
            model = Console.ReadLine();
```

```
            Console.WriteLine("Enter Body Type:");
            bodyType = Console.ReadLine();

            Console.WriteLine("Enter Year of Manufacture:");
            while (!int.TryParse(Console.ReadLine(), out yearOfManufacture) ||
yearOfManufacture <= 0)
            {
                Console.WriteLine("Invalid Year of Manufacture. It should be a
positive integer.");
                Console.WriteLine("Enter Year of Manufacture:");
            }

            Console.WriteLine("Enter Chassis Number:");
            chassisNumber = Console.ReadLine();

            Console.WriteLine("Enter Color:");
            color = Console.ReadLine();

            Console.WriteLine("Enter License Plate:");
            licensePlate = Console.ReadLine();

            Console.WriteLine("Enter Technical Condition:");
            technicalCondition = Console.ReadLine();

            List<Registration>? registrations = null;

            return new Vehicle(vehicleId, brand, manufacturer, model, bodyType,
yearOfManufacture, chassisNumber, color, licensePlate, technicalCondition,
registrations);
        }
    }
}
```

Registration.cs:

```
using System;
```

```
using System.Collections.Generic;
using System.Globalization;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Serialization;

namespace laba2
{
   public class Registration
   {
      public int VehicleId { get; init; }
      public int OwnerId { get; init; }

      [XmlIgnore]
      public DateTime RegistrationDate { get; set; }

      [XmlElement("RegistrationDate")]
      public string RegistrationDateString
      {
         get { return RegistrationDate.ToString("dd.MM.yyyy"); }
         set { RegistrationDate = DateTime.ParseExact(value, "dd.MM.yyyy",
null); }
      }
      public string? RegistrationLocation { get; init; }
      public bool IsRegistered { get; set; }

      public Registration(int vehicleId, int ownerId, DateTime registrationDate,
string? registrationLocation, bool isRegistered)
      {
         VehicleId = vehicleId;
         OwnerId = ownerId;
         RegistrationDate = registrationDate;
         RegistrationLocation = registrationLocation;
         IsRegistered = isRegistered;
      }
```

```
    public Registration() { }

    public static Registration CreateRegistration()
    {
        int vehicleId, ownerId;
        DateTime registrationDate;
        string? registrationLocation;
        bool isRegistered;

        Console.WriteLine("Enter Vehicle ID:");
        while (!int.TryParse(Console.ReadLine(), out vehicleId) || vehicleId <=
0)
        {
            Console.WriteLine("Invalid Vehicle ID. It should be a positive
integer.");
            Console.WriteLine("Enter Vehicle ID:");
        }

        Console.WriteLine("Enter Owner ID:");
        while (!int.TryParse(Console.ReadLine(), out ownerId) || ownerId <= 0)
        {
            Console.WriteLine("Invalid Owner ID. It should be a positive
integer.");
            Console.WriteLine("Enter Owner ID:");
        }

        Console.WriteLine("Enter Registration Date (dd.MM.yyyy):");
        while (!DateTime.TryParseExact(Console.ReadLine(), "dd.MM.yyyy",
CultureInfo.InvariantCulture, DateTimeStyles.None, out registrationDate))
        {
            Console.WriteLine("Invalid Date. Please enter in dd.MM.yyyy
format:");
        }

        Console.WriteLine("Enter Registration Location:");
        registrationLocation = Console.ReadLine();
```

```
            Console.WriteLine("Is Registered? (true/false):");
            while (!bool.TryParse(Console.ReadLine(), out isRegistered))
            {
                Console.WriteLine("Invalid input. Please enter 'true' or 'false':");
            }

            return new Registration(vehicleId, ownerId, registrationDate,
registrationLocation, isRegistered);
        }
    }
}
```

Реалізація допоміжних класів:

VehicleDriver.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace laba2
{
    public class VehicleDriver
    {
        public int VehicleId { get; set; }
        public int DriverId { get; set; }

        public VehicleDriver(int vehicleId, int driverId)
        {
            VehicleId = vehicleId;
            DriverId = driverId;
        }

        public VehicleDriver()
```

```
        {

        }

        public static VehicleDriver CreateVehicleDriver()
        {
            int vehicleId, driverId;

            Console.WriteLine("Enter Vehicle ID:");
            while (!int.TryParse(Console.ReadLine(), out vehicleId) || vehicleId <=
0)
            {
                Console.WriteLine("Invalid Vehicle ID. It should be a positive
integer.");
                Console.WriteLine("Enter Vehicle ID:");
            }

            Console.WriteLine("Enter Driver ID:");
            while (!int.TryParse(Console.ReadLine(), out driverId) || driverId <= 0)
            {
                Console.WriteLine("Invalid Driver ID. It should be a positive
integer.");
                Console.WriteLine("Enter Driver ID:");
            }

            return new VehicleDriver(vehicleId, driverId);
        }
    }
}
```

VehicleOwnership.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
using System.Threading.Tasks;

namespace laba2
{
   public class VehicleOwnership
   {
      public int VehicleId { get; set; }
      public int OwnerId { get; set; }

      public VehicleOwnership(int vehicleId, int ownerId)
      {
         VehicleId = vehicleId;
         OwnerId = ownerId;
      }

      public VehicleOwnership()
      {

      }

      public static VehicleOwnership CreateVehicleOwnership()
      {
         int vehicleId, ownerId;

         Console.WriteLine("Enter Vehicle ID:");
         while (!int.TryParse(Console.ReadLine(), out vehicleId) || vehicleId <=
0)
         {
            Console.WriteLine("Invalid Vehicle ID. It should be a positive
integer.");
            Console.WriteLine("Enter Vehicle ID:");
         }

         Console.WriteLine("Enter Owner ID:");
         while (!int.TryParse(Console.ReadLine(), out ownerId) || ownerId <= 0)
         {
```

```
            Console.WriteLine("Invalid Owner ID. It should be a positive
integer.");
            Console.WriteLine("Enter Owner ID:");
        }

        return new VehicleOwnership(vehicleId, ownerId);
    }
  }
}
```

Основна реалізація запитів:

Queries.cs:

```
using System;
using System.Collections.Generic;
using System.Globalization;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Linq;

namespace laba2
{
  public class Queries
  {
    private readonly string _driver=
"C:\\Users\\User\\source\\repos\\laba2\\laba2\\XML\\Driver.xml";
    private readonly string _owner=
"C:\\Users\\User\\source\\repos\\laba2\\laba2\\XML\\Owner.xml";
    private readonly string _registration =
"C:\\Users\\User\\source\\repos\\laba2\\laba2\\XML\\VehicleRegistrations.xml";
    private readonly string _vehicle =
"C:\\Users\\User\\source\\repos\\laba2\\laba2\\XML\\Vehicle.xml";
    private readonly string _vehicleDriver =
"C:\\Users\\User\\source\\repos\\laba2\\laba2\\XML\\VehicleDriver.xml";
```

```
        private readonly string _vehicleOwnership =
"C:\\Users\\User\\source\\repos\\laba2\\laba2\\XML\\VehicleOwnership.xml";

        // #1. Отримати усі машини з типом кузову SUV
        public void GetSUVVehicles()
        {
            XDocument xdoc = XDocument.Load(_vehicle);

            var suvVehicles = from vehicle in xdoc.Descendants("Vehicle")
                              where (string?)vehicle.Element("BodyType") == "SUV"
                              select vehicle;

            Console.WriteLine("Список машин типу SUV:");

            if (suvVehicles.Any())
            {
                foreach (var vehicle in suvVehicles)
                {
                    if (vehicle != null)
                    {
                        Console.WriteLine($"Машина:
{(string?)vehicle.Element("Brand")} {(string?)vehicle.Element("Model")}, Рік
виробництва: {(int?)vehicle.Element("YearOfManufacture")}");
                    }
                }
            }
            else
            {
                Console.WriteLine("Немає машин типу SUV в базі даних.");
            }
        }

        // #2. Отримати усі машини та відповідно їх власників
        public IEnumerable<(XElement vehicle, XElement owner)>
GetAllSUVsWithOwners()
        {
            XDocument vehicleDoc = XDocument.Load(_vehicle);
```

```
        XDocument ownershipDoc = XDocument.Load(_vehicleOwnership);
        XDocument ownerDoc = XDocument.Load(_owner);

        var suvsWithOwners = from ownership in
ownershipDoc.Descendants("VehicleOwnership")
                             join vehicle in
vehicleDoc.Descendants("Vehicle").Where(v =>
(string)v.Element("BodyType") == "SUV")
                             on (int)ownership.Element("VehicleId") equals
(int)vehicle.Element("VehicleId")
                             join owner in ownerDoc.Descendants("Owner")
                             on (int)ownership.Element("OwnerId") equals
(int)owner.Element("OwnerId")
                             select (
                                vehicle: vehicle,
                                owner: owner
                             );

        Console.WriteLine("List of SUVs with their respective owners:");
        foreach (var pair in suvsWithOwners)
        {
           Console.WriteLine($"Vehicle:
{pair.vehicle.Element("Brand")?.Value}
{pair.vehicle.Element("Model")?.Value}, Owner:
{pair.owner.Element("FirstName")?.Value}
{pair.owner.Element("LastName")?.Value}");
        }

        return suvsWithOwners;
    }

    // #3. Отримати перелік машин, що мають відмінні технічні показники
    public IEnumerable<XElement> GetMachinesInExcellentCondition()
    {
        XDocument vehicleDoc = XDocument.Load(_vehicle);
```

```
            var machinesInExcellentCondition = from vehicle in
vehicleDoc.Descendants("Vehicle")
                                               where
(string?)vehicle.Element("TechnicalCondition") == "Excellent"
                                               select vehicle;

            Console.WriteLine("List of vehicles in excellent condition:");
            foreach (var machine in machinesInExcellentCondition)
            {
                Console.WriteLine($"Vehicle: {machine.Element("Brand")?.Value}
{machine.Element("Model")?.Value}");
            }

            return machinesInExcellentCondition;
        }

        // #4. Отримати перелік людей (власників), вік яких входить у заданий
діапазон
        public IEnumerable<XElement> GetOwnersWithinAgeRange(int minYear,
int maxYear)
        {
            XDocument ownerDoc = XDocument.Load(_owner);

            var currentDate = DateTime.Now;
            var minDateOfBirth = currentDate.AddYears(-maxYear);
            var maxDateOfBirth = currentDate.AddYears(-minYear);

            var ownersWithinAgeRange = from owner in
ownerDoc.Descendants("Owner")
                                       let dateOfBirthStr =
owner.Element("DateOfBirth")?.Value
                                       where dateOfBirthStr != null &&
                                             DateTime.TryParseExact(dateOfBirthStr,
"dd.MM.yyyy", CultureInfo.InvariantCulture, DateTimeStyles.None, out
DateTime dateOfBirth) &&
                                             dateOfBirth >= minDateOfBirth && dateOfBirth
<= maxDateOfBirth
```

```
                                select owner;

            Console.WriteLine("Owners within the specified age range:");
            foreach (var owner in ownersWithinAgeRange)
            {
                Console.WriteLine($"Owner: {owner.Element("FirstName")?.Value}
{owner.Element("LastName")?.Value}, Date of Birth:
{owner.Element("DateOfBirth")?.Value}");
            }

            return ownersWithinAgeRange;
        }


        // #5. Отримати водія, що має найдовше ім'я
        public void GetDriverWithLongestName(IEnumerable<Driver> drivers)
        {
            XDocument driverDoc = XDocument.Load(_driver);

            var driverWithLongestName = driverDoc.Descendants("Driver")
                                        .OrderByDescending(driver =>
                                        {
                                            var fullName =
((string?)driver.Element("FirstName") + (string?)driver.Element("LastName") +
(string?)driver.Element("Patronymic"));
                                            return fullName.Length;
                                        })
                                        .FirstOrDefault();

            if (driverWithLongestName != null)
            {
                Console.WriteLine($"Driver with the longest name:
{driverWithLongestName.Element("FirstName").Value}
{driverWithLongestName.Element("LastName").Value}
{driverWithLongestName.Element("Patronymic").Value}");
            }
            else
```

```
        {
            Console.WriteLine("No driver found.");
        }
    }


    // #6. Отримати машину, що має найстаріший рік випуску
    public void
GetVehicleWithEarliestManufactureYear(IEnumerable<Vehicle> vehicles)
    {
        XDocument vehicleDoc = XDocument.Load(_vehicle);

        var vehicleWithEarliestManufactureYear =
vehicleDoc.Descendants("Vehicle")
                                        .OrderBy(vehicle =>
(int)vehicle.Element("YearOfManufacture"))
                                        .FirstOrDefault();

        if (vehicleWithEarliestManufactureYear != null)
        {
            Console.WriteLine($"Vehicle with the earliest manufacture year:
{vehicleWithEarliestManufactureYear.Element("Brand").Value}
{vehicleWithEarliestManufactureYear.Element("Model").Value}, Year of
Manufacture:
{vehicleWithEarliestManufactureYear.Element("YearOfManufacture").Value}")
;
        }
        else
        {
            Console.WriteLine("No vehicle found.");
        }
    }

    // #7. Отримати відсортований перелік водіїв за датою народження в
порядку зростання
```

```
    public void
SortDriversByDateOfBirthAscendingWithBoundaries(IEnumerable<Driver>
drivers, int startYear, int endYear)
    {
        XDocument driverDoc = XDocument.Load(_driver);

        var sortedDrivers = driverDoc.Descendants("Driver")
                                .Where(driver => {
                                    var dateOfBirthStr =
(string)driver.Element("DateOfBirth");
                                    return dateOfBirthStr != null &&
                                        DateTime.TryParseExact(dateOfBirthStr,
"dd.MM.yyyy", CultureInfo.InvariantCulture, DateTimeStyles.None, out
DateTime dateOfBirth) &&
                                        dateOfBirth.Year >= startYear &&
dateOfBirth.Year <= endYear;
                                })
                                .OrderBy(driver => {
                                    var dateOfBirthStr =
(string)driver.Element("DateOfBirth");
                                    return DateTime.ParseExact(dateOfBirthStr,
"dd.MM.yyyy", CultureInfo.InvariantCulture);
                                });

        Console.WriteLine($"Sorted list of drivers by date of birth between
{startYear} and {endYear}:");

        foreach (var driver in sortedDrivers)
        {
            Console.WriteLine($"Driver: {driver.Element("FirstName").Value}
{driver.Element("LastName").Value}, Date of Birth:
{driver.Element("DateOfBirth").Value}");
        }
    }

    // #8. Отримати перелік машин, що не є зареєстрованими
```

```
    public void GetUnregisteredVehicles(IEnumerable<Vehicle> vehicles,
IEnumerable<Registration> registrations)
    {
        XDocument vehicleDoc = XDocument.Load(_vehicle);
        XDocument registrationDoc = XDocument.Load(_registration);

        var unregisteredVehicles = from vehicle in
vehicleDoc.Descendants("Vehicle")
                                   where !registrationDoc.Descendants("Registration")
                                                .Any(registration =>
(int)registration.Element("VehicleId") == (int)vehicle.Element("VehicleId"))
                                   select vehicle;

        Console.WriteLine("List of unregistered vehicles:");

        foreach (var vehicle in unregisteredVehicles)
        {
            Console.WriteLine($"Vehicle: {vehicle.Element("Brand").Value}
{vehicle.Element("Model").Value}, Vehicle ID:
{vehicle.Element("VehicleId").Value}");
        }
    }

    // #9. Отримати перелік машин, випущених до певного року (включно)
    public void GetMachinesReleasedBeforeYear(IEnumerable<Vehicle>
vehicles, int year)
    {
        XDocument vehicleDoc = XDocument.Load(_vehicle);

        var machinesReleasedBeforeYear = from vehicle in
vehicleDoc.Descendants("Vehicle")
                                         where (int)vehicle.Element("YearOfManufacture")
<= year
                                         select vehicle;

        Console.WriteLine($"List of machines released before year {year}:");
```

```
        foreach (var vehicle in machinesReleasedBeforeYear)
        {
            Console.WriteLine($"Vehicle: {vehicle.Element("Brand").Value}
{vehicle.Element("Model").Value}, Year of Manufacture:
{vehicle.Element("YearOfManufacture").Value}");
        }
    }

    // #10. Отримати машини, зареєстрованих в локації "Local Registration
Office"
    public IEnumerable<object>
GetVehiclesRegisteredInLocalOffice(IEnumerable<Vehicle> vehicles,
IEnumerable<Registration> registrations)
    {
        XDocument vehicleDoc = XDocument.Load(_vehicle);
        XDocument registrationDoc = XDocument.Load(_registration);

        var vehiclesRegisteredInLocalOffice = from registration in
registrationDoc.Descendants("Registration")
                                              where
((string)registration.Element("RegistrationLocation")).Trim().Equals("Local
Registration Office", StringComparison.OrdinalIgnoreCase)
                                              join vehicle in
vehicleDoc.Descendants("Vehicle") on (int)registration.Element("VehicleId")
equals (int)vehicle.Element("VehicleId")
                                              select new
                                              {
                                                  Brand = vehicle.Element("Brand")?.Value,
                                                  Model = vehicle.Element("Model")?.Value
                                              };

        Console.WriteLine("List of vehicles registered in the local office:");

        foreach (var vehicle in vehiclesRegisteredInLocalOffice)
        {
            Console.WriteLine($"Vehicle: {vehicle.Brand} {vehicle.Model}");
        }
```

```
            return vehiclesRegisteredInLocalOffice;
        }


        // #11. Отримати машини, які не керуються
        public IEnumerable<object>
GetVehiclesNotDrivenByDrivers(IEnumerable<Vehicle> vehicles,
IEnumerable<Driver> drivers, IEnumerable<VehicleDriver> vehicleDrivers)
        {
            XDocument vehicleDriverDoc = XDocument.Load(_vehicleDriver);

            var vehiclesNotDrivenByDrivers = from vehicle in vehicles
                                             where !drivers.Any(driver =>
                                               vehicleDrivers.Any(vd => vd.VehicleId ==
vehicle.VehicleId && vd.DriverId == driver.DriverId))
                                             select new
                                             {
                                                Brand = vehicle.Brand,
                                                Model = vehicle.Model
                                             };

            Console.WriteLine("List of vehicles not driven by any drivers:");

            foreach (var vehicle in vehiclesNotDrivenByDrivers)
            {
               Console.WriteLine($"Vehicle: {vehicle.Brand} {vehicle.Model}");
            }

            return vehiclesNotDrivenByDrivers;
        }

        // #12. Отримати перелік всіх унікальних власників машин та водіїв,
об'єднавши їх імена
        public IEnumerable<string>
GetUniqueOwnersAndDriversNames(IEnumerable<Owner> owners,
IEnumerable<Driver> drivers)
```

```
    {
        XDocument ownerDoc = XDocument.Load(_owner);
        XDocument driverDoc = XDocument.Load(_driver);

        var ownerNames = from owner in ownerDoc.Descendants("Owner")
                         select $"{owner.Element("FirstName").Value}
{owner.Element("LastName").Value}";

        var driverNames = from driver in driverDoc.Descendants("Driver")
                          select $"{driver.Element("FirstName").Value}
{driver.Element("LastName").Value}";

        var uniqueNames = ownerNames.Concat(driverNames).Distinct();

        Console.WriteLine("Unique owner and driver names:");
        foreach (var name in uniqueNames)
        {
            Console.WriteLine(name);
        }

        return uniqueNames;
    }


    // #13. Отримати перелік власників, що мають декілька машин
    public IEnumerable<Owner>
GetOwnersWithMultipleVehicles(IEnumerable<Owner> owners,
IEnumerable<VehicleOwnership> vehicleOwnerships)
    {
        XDocument ownerDoc = XDocument.Load(_owner);
        XDocument ownershipDoc = XDocument.Load(_vehicleOwnership);

        var ownersWithMultipleVehicles = from ownership in
ownershipDoc.Descendants("VehicleOwnership")
                                         group ownership by
(int)ownership.Element("OwnerId") into ownershipGroup
                                         where ownershipGroup.Count() > 1
```

```
                                join owner in ownerDoc.Descendants("Owner") on
ownershipGroup.Key equals (int)owner.Element("OwnerId")
                                select new Owner
                                {
                                   OwnerId = (int)owner.Element("OwnerId"),
                                   FirstName =
(string)owner.Element("FirstName"),
                                   LastName = (string)owner.Element("LastName"),
                                   DateOfBirth =
(DateTime)owner.Element("DateOfBirth")
                                };

          if (!ownersWithMultipleVehicles.Any())
          {
             Console.WriteLine("No owners with multiple vehicles found.");
          }
          else
          {
             Console.WriteLine("Owners with multiple vehicles:");
             foreach (var owner in ownersWithMultipleVehicles)
             {
                Console.WriteLine($"Owner ID: {owner.OwnerId}, Name:
{owner.FirstName} {owner.LastName}, Date of Birth: {owner.DateOfBirth}");
             }
          }

          return ownersWithMultipleVehicles;
       }

       // #14. Отримати перелік машин, відсортовані за роком випуску
(спадання) з діапазоном років випуску автомобілів
       public IEnumerable<Vehicle>
SortVehiclesByModelYearDescendingWithBoundaries(IEnumerable<Vehicle>
vehicles, int startYear, int endYear)
       {
          XDocument vehicleDoc = XDocument.Load(_vehicle);
```

```
        var sortedVehicles = from vehicle in vehicleDoc.Descendants("Vehicle")
                             let yearOfManufacture =
(int)vehicle.Element("YearOfManufacture")
                             where yearOfManufacture >= startYear &&
yearOfManufacture <= endYear
                             orderby yearOfManufacture descending
                             select new Vehicle
                             {
                                VehicleId = (int)vehicle.Element("VehicleId"),
                                Brand = (string)vehicle.Element("Brand"),
                                Model = (string)vehicle.Element("Model"),
                                YearOfManufacture =
(int)vehicle.Element("YearOfManufacture")
                             };

        if (!sortedVehicles.Any())
        {
            Console.WriteLine($"No vehicles found manufactured between
{startYear} and {endYear}.");
        }
        else
        {
            Console.WriteLine($"Vehicles sorted by model year (descending)
between {startYear} and {endYear}:");
            foreach (var vehicle in sortedVehicles)
            {
                Console.WriteLine($"Vehicle ID: {vehicle.VehicleId}, Brand:
{vehicle.Brand}, Model: {vehicle.Model}, Year of Manufacture:
{vehicle.YearOfManufacture}");
            }
        }

        return sortedVehicles;
    }

    // #15. Отримати перелік водіїв без машин
```

```
        public IEnumerable<Driver>
FindDriversWithoutVehicles(IEnumerable<Driver> drivers,
IEnumerable<VehicleDriver> vehicleDrivers)
        {
            XDocument driverDoc = XDocument.Load(_driver);
            XDocument vehicleDriverDoc = XDocument.Load(_vehicleDriver);

            var driversWithVehicles = from vd in
vehicleDriverDoc.Descendants("VehicleDriver")
                                      select (int)vd.Element("DriverId");

            var driversWithoutVehicles = from driver in
driverDoc.Descendants("Driver")
                                         where
!driversWithVehicles.Contains((int)driver.Element("DriverId"))
                                         select new Driver
                                         {
                                            DriverId = (int)driver.Element("DriverId"),
                                            FirstName = (string)driver.Element("FirstName"),
                                            LastName = (string)driver.Element("LastName"),
                                            Patronymic = (string)driver.Element("Patronymic"),
                                         };

            Console.WriteLine("Drivers without vehicles:");

            foreach (var driver in driversWithoutVehicles)
            {
               Console.WriteLine($"Driver ID: {driver.DriverId}, Name:
{driver.FirstName} {driver.LastName} {driver.Patronymic}");
            }

            return driversWithoutVehicles;
        }

        // #16. Отримати середній вік водіїв
        public double GetAverageAgeOfDrivers(IEnumerable<Driver> drivers)
        {
```

```
            XDocument driverDoc = XDocument.Load(_driver);

            double totalAge = drivers.Sum(driver =>
            {
                var driverElement =
driverDoc.Descendants("Driver").FirstOrDefault(d =>
(int)d.Element("DriverId") == driver.DriverId);
                if (driverElement != null)
                {
                    var dateOfBirthString =
(string)driverElement.Element("DateOfBirth");
                    DateTime dateOfBirth;
                    if (DateTime.TryParseExact(dateOfBirthString, "dd.MM.yyyy",
CultureInfo.InvariantCulture, DateTimeStyles.None, out dateOfBirth))
                    {
                        return (DateTime.Now - dateOfBirth).TotalDays / 365.25;
                    }
                }
                return 0;
            });

            double averageAge = totalAge / drivers.Count();

            Console.WriteLine($"Average age of drivers: {averageAge}");

            return averageAge;
        }

        // #17. Отримати перелік водіїв молодше певного віку
        public IEnumerable<Driver>
GetDriversYoungerThanAge(IEnumerable<Driver> drivers, int maxAge)
        {
            XDocument driverDoc = XDocument.Load(_driver);

            var currentDate = DateTime.Now;
            var maxDateOfBirth = currentDate.AddYears(-maxAge);
```

```
            var driversYoungerThanAge = drivers.Where(driver =>
            {
                var driverElement =
driverDoc.Descendants("Driver").FirstOrDefault(d =>
(int)d.Element("DriverId") == driver.DriverId);
                if (driverElement != null)
                {
                    var dateOfBirthString =
(string)driverElement.Element("DateOfBirth");
                    DateTime dateOfBirth;
                    if (DateTime.TryParseExact(dateOfBirthString, "dd.MM.yyyy",
CultureInfo.InvariantCulture, DateTimeStyles.None, out dateOfBirth))
                    {
                        return dateOfBirth > maxDateOfBirth;
                    }
                }
                return false;
            });

            Console.WriteLine($"Drivers younger than {maxAge}:");

            foreach (var driver in driversYoungerThanAge)
            {
                Console.WriteLine($"Driver: {driver.FirstName} {driver.LastName},
Date of Birth: {driver.DateOfBirth}");
            }

            return driversYoungerThanAge;
        }



        // #18. Отримати список автомобілів, що були зареєстровані у якомусь
році
        public IEnumerable<string>
GetVehiclesRegisteredInYear(IEnumerable<Vehicle> vehicles,
IEnumerable<Registration> registrations, int registrationYear)
        {
```

```
        XDocument registrationDoc = XDocument.Load(_registration);
        XDocument vehicleDoc = XDocument.Load(_vehicle);

        var vehiclesRegisteredInYear = from registration in
registrationDoc.Descendants("Registration")
                                       where
DateTime.ParseExact(registration.Element("RegistrationDate").Value,
"dd.MM.yyyy", CultureInfo.InvariantCulture).Year == registrationYear
                                       join vehicle in vehicleDoc.Descendants("Vehicle")
on (int)registration.Element("VehicleId") equals
(int)vehicle.Element("VehicleId")
                                       select $"{vehicle.Element("Brand").Value}
{vehicle.Element("Model").Value}";

        Console.WriteLine($"Vehicles registered in {registrationYear}:");

        foreach (var vehicle in vehiclesRegisteredInYear)
        {
           Console.WriteLine($"Vehicle: {vehicle}");
        }

        return vehiclesRegisteredInYear;
    }

    // #19. Отримати список водіїв, що народилися у конкретному році
    public IEnumerable<Driver>
GetDriversBornInYear(IEnumerable<Driver> drivers, int birthYear)
    {
        XDocument driverDoc = XDocument.Load(_driver);

        var driversBornInYear = from driver in
driverDoc.Descendants("Driver")
                                let dateOfBirth =
DateTime.ParseExact((string)driver.Element("DateOfBirth"), "dd.MM.yyyy",
CultureInfo.InvariantCulture)
                                where dateOfBirth.Year == birthYear
                                select new Driver
```

```
                        {
                            DriverId = (int)driver.Element("DriverId"),
                            FirstName = (string)driver.Element("FirstName"),
                            LastName = (string)driver.Element("LastName"),
                            Patronymic = (string)driver.Element("Patronymic"),
                            DateOfBirth = dateOfBirth
                        };

            Console.WriteLine($"Drivers born in {birthYear}:");

            foreach (var driver in driversBornInYear)
            {
                Console.WriteLine($"Driver: {driver.FirstName} {driver.LastName},
Date of Birth: {driver.DateOfBirth}");
            }

            return driversBornInYear;
        }

        // #20. Отримати список автомобілів, які належать власнику з
конкретним ID
        public IEnumerable<(string Brand, string Model)>
GetCarsOwnedByOwner(IEnumerable<Vehicle> vehicles,
IEnumerable<Owner> owners, IEnumerable<VehicleOwnership>
vehicleOwnerships, int ownerId)
        {
            XDocument vehicleDoc = XDocument.Load(_vehicle);
            XDocument ownershipDoc = XDocument.Load(_vehicleOwnership);

            var ownerCar = from ownership in
ownershipDoc.Descendants("VehicleOwnership")
                           join vehicle in vehicleDoc.Descendants("Vehicle") on
(int)ownership.Element("VehicleId") equals (int)vehicle.Element("VehicleId")
                           where (int)ownership.Element("OwnerId") == ownerId
                           select (Brand: (string)vehicle.Element("Brand"), Model:
(string)vehicle.Element("Model"));
```

```
            Console.WriteLine($"Cars owned by Owner ID {ownerId}:");

            foreach (var car in ownerCar)
            {
                Console.WriteLine($"Brand: {car.Brand}, Model: {car.Model}");
            }

            return ownerCar;
        }
    }
}
```

XML-файли:

Driver.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<Drivers xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
        <Driver>
                <DriverId>1</DriverId>
                <LastName>Doe</LastName>
                <FirstName>Jane</FirstName>
                <Patronymic>Marie</Patronymic>
                <DateOfBirth>25.08.1993</DateOfBirth>
                <RegistrationAddress>789 Oak St, City,
Country</RegistrationAddress>
                <LicenseNumber>1357924680</LicenseNumber>
        </Driver>
        <Driver>
                <DriverId>2</DriverId>
                <LastName>Williams</LastName>
                <FirstName>Robert</FirstName>
                <Patronymic>David</Patronymic>
                <DateOfBirth>12.04.1988</DateOfBirth>
                <RegistrationAddress>101 Pine St, City,
Country</RegistrationAddress>
```

```
            <LicenseNumber>2468135790</LicenseNumber>
        </Driver>
        <Driver>
            <DriverId>3</DriverId>
            <LastName>Johnson</LastName>
            <FirstName>Emily</FirstName>
            <Patronymic>Anne</Patronymic>
            <DateOfBirth>18.06.1995</DateOfBirth>
            <RegistrationAddress>456 Elm St, City,
Country</RegistrationAddress>
            <LicenseNumber>0987654321</LicenseNumber>
        </Driver>
        <Driver>
            <DriverId>4</DriverId>
            <LastName>Smith</LastName>
            <FirstName>John</FirstName>
            <Patronymic>Michael</Patronymic>
            <DateOfBirth>05.10.1990</DateOfBirth>
            <RegistrationAddress>123 Main St, City,
Country</RegistrationAddress>
            <LicenseNumber>1234567890</LicenseNumber>
        </Driver>
        <Driver>
            <DriverId>5</DriverId>
            <LastName>Brown</LastName>
            <FirstName>Emma</FirstName>
            <Patronymic>Grace</Patronymic>
            <DateOfBirth>30.12.1987</DateOfBirth>
            <RegistrationAddress>222 Cedar St, City,
Country</RegistrationAddress>
            <LicenseNumber>9876543210</LicenseNumber>
        </Driver>
        <Driver>
            <DriverId>6</DriverId>
            <LastName>Wilson</LastName>
            <FirstName>Michael</FirstName>
            <Patronymic>Andrew</Patronymic>
```

```
            <DateOfBirth>15.05.1992</DateOfBirth>
            <RegistrationAddress>333 Maple St, City,
Country</RegistrationAddress>
            <LicenseNumber>0123456789</LicenseNumber>
        </Driver>
        <Driver>
            <DriverId>7</DriverId>
            <LastName>Martinez</LastName>
            <FirstName>Daniel</FirstName>
            <Patronymic>Carlos</Patronymic>
            <DateOfBirth>08.03.1991</DateOfBirth>
            <RegistrationAddress>555 Elm St, City,
Country</RegistrationAddress>
            <LicenseNumber>6789054321</LicenseNumber>
        </Driver>
        <Driver>
            <DriverId>8</DriverId>
            <LastName>Anderson</LastName>
            <FirstName>Jennifer</FirstName>
            <Patronymic>Nicole</Patronymic>
            <DateOfBirth>20.09.1987</DateOfBirth>
            <RegistrationAddress>444 Oak St, City,
Country</RegistrationAddress>
            <LicenseNumber>5432109876</LicenseNumber>
        </Driver>
        <Driver>
            <DriverId>9</DriverId>
            <LastName>Artem</LastName>
            <FirstName>Hrytsenko</FirstName>
            <Patronymic>Volodymyrovych</Patronymic>
            <DateOfBirth>20.09.1987</DateOfBirth>
            <RegistrationAddress>444 Oak St, City,
Country</RegistrationAddress>
            <LicenseNumber>54321098955</LicenseNumber>
        </Driver>
</Drivers>
```

Owner.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<Owners xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Owner>
    <OwnerId>1</OwnerId>
    <LastName>Smith</LastName>
    <FirstName>John</FirstName>
    <Patronymic>Michael</Patronymic>
    <DateOfBirth>15.10.1985</DateOfBirth>
    <RegistrationAddress>123 Main St, City, Country</RegistrationAddress>
    <LicenseNumber>1234567890</LicenseNumber>
  </Owner>
  <Owner>
    <OwnerId>2</OwnerId>
    <LastName>Johnson</LastName>
    <FirstName>Emily</FirstName>
    <Patronymic>Anne</Patronymic>
    <DateOfBirth>20.05.1990</DateOfBirth>
    <RegistrationAddress>456 Elm St, City, Country</RegistrationAddress>
    <LicenseNumber>0987654321</LicenseNumber>
  </Owner>
  <Owner>
    <OwnerId>3</OwnerId>
    <LastName>Williams</LastName>
    <FirstName>David</FirstName>
    <Patronymic>Robert</Patronymic>
    <DateOfBirth>25.08.1983</DateOfBirth>
    <RegistrationAddress>789 Oak St, City, Country</RegistrationAddress>
    <LicenseNumber>1357924680</LicenseNumber>
  </Owner>
  <Owner>
    <OwnerId>4</OwnerId>
    <LastName>Brown</LastName>
    <FirstName>Sarah</FirstName>
    <Patronymic>Elizabeth</Patronymic>
    <DateOfBirth>10.12.1975</DateOfBirth>
```

<RegistrationAddress>101 Pine St, City, Country</RegistrationAddress>
<LicenseNumber>2468135790</LicenseNumber>
</Owner>
<Owner>
<OwnerId>5</OwnerId>
<LastName>Wilson</LastName>
<FirstName>Jessica</FirstName>
<Patronymic>Marie</Patronymic>
<DateOfBirth>05.06.1992</DateOfBirth>
<RegistrationAddress>222 Cedar St, City, Country</RegistrationAddress>
<LicenseNumber>9876543210</LicenseNumber>
</Owner>
<Owner>
<OwnerId>6</OwnerId>
<LastName>Taylor</LastName>
<FirstName>Michael</FirstName>
<Patronymic>Andrew</Patronymic>
<DateOfBirth>15.04.1988</DateOfBirth>
<RegistrationAddress>333 Maple St, City, Country</RegistrationAddress>
<LicenseNumber>0123456789</LicenseNumber>
</Owner>
<Owner>
<OwnerId>7</OwnerId>
<LastName>Anderson</LastName>
<FirstName>Jennifer</FirstName>
<Patronymic>Nicole</Patronymic>
<DateOfBirth>20.09.1987</DateOfBirth>
<RegistrationAddress>444 Oak St, City, Country</RegistrationAddress>
<LicenseNumber>5432109876</LicenseNumber>
</Owner>
<Owner>
<OwnerId>8</OwnerId>
<LastName>Martinez</LastName>
<FirstName>Daniel</FirstName>
<Patronymic>Carlos</Patronymic>
<DateOfBirth>08.03.1991</DateOfBirth>
<RegistrationAddress>555 Elm St, City, Country</RegistrationAddress>

```
    <LicenseNumber>6789054321</LicenseNumber>
  </Owner>
</Owners>
```

Vehicle.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<Vehicles xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
        <Vehicle>
                <VehicleId>1</VehicleId>
                <Brand>Toyota</Brand>
                <Manufacturer>Toyota Motor Corporation</Manufacturer>
                <Model>Camry</Model>
                <BodyType>Sedan</BodyType>
                <YearOfManufacture>2020</YearOfManufacture>
                <ChassisNumber>JT2BF22KX00765432</ChassisNumber>
                <Color>Black</Color>
                <LicensePlate>ABC123</LicensePlate>
                <TechnicalCondition>Good</TechnicalCondition>
        </Vehicle>
        <Vehicle>
                <VehicleId>2</VehicleId>
                <Brand>Honda</Brand>
                <Manufacturer>Honda Motor Co., Ltd.</Manufacturer>
                <Model>Civic</Model>
                <BodyType>Sedan</BodyType>
                <YearOfManufacture>2018</YearOfManufacture>
                <ChassisNumber>2HGFC2F58JH532148</ChassisNumber>
                <Color>Silver</Color>
                <LicensePlate>XYZ789</LicensePlate>
                <TechnicalCondition>Excellent</TechnicalCondition>
        </Vehicle>
        <Vehicle>
                <VehicleId>3</VehicleId>
                <Brand>Ford</Brand>
                <Manufacturer>Ford Motor Company</Manufacturer>
                <Model>F-150</Model>
```

<BodyType>Truck</BodyType>
<YearOfManufacture>2019</YearOfManufacture>
<ChassisNumber>1FTFW1EF8KFA12345</ChassisNumber>
<Color>Blue</Color>
<LicensePlate>DEF456</LicensePlate>
<TechnicalCondition>Good</TechnicalCondition>
</Vehicle>
<Vehicle>
<VehicleId>4</VehicleId>
<Brand>Chevrolet</Brand>
<Manufacturer>General Motors Company</Manufacturer>
<Model>Camaro</Model>
<BodyType>Coupe</BodyType>
<YearOfManufacture>2021</YearOfManufacture>
<ChassisNumber>2G1FA1ED7A9725478</ChassisNumber>
<Color>Red</Color>
<LicensePlate>GHI789</LicensePlate>
<TechnicalCondition>Excellent</TechnicalCondition>
</Vehicle>
<Vehicle>
<VehicleId>5</VehicleId>
<Brand>BMW</Brand>
<Manufacturer>Bayerische Motoren Werke AG</Manufacturer>
<Model>X5</Model>
<BodyType>SUV</BodyType>
<YearOfManufacture>2017</YearOfManufacture>
<ChassisNumber>5UXKR6C56H0U24876</ChassisNumber>
<Color>White</Color>
<LicensePlate>JKL012</LicensePlate>
<TechnicalCondition>Good</TechnicalCondition>
</Vehicle>
<Vehicle>
<VehicleId>6</VehicleId>
<Brand>Mercedes-Benz</Brand>
<Manufacturer>Mercedes-Benz AG</Manufacturer>
<Model>E-Class</Model>
<BodyType>Sedan</BodyType>

<YearOfManufacture>2022</YearOfManufacture>
<ChassisNumber>WDDZF8EB7KA525623</ChassisNumber>
<Color>Gray</Color>
<LicensePlate>MNO345</LicensePlate>
<TechnicalCondition>Excellent</TechnicalCondition>
</Vehicle>
<Vehicle>
<VehicleId>7</VehicleId>
<Brand>Audi</Brand>
<Manufacturer>Audi AG</Manufacturer>
<Model>A4</Model>
<BodyType>Sedan</BodyType>
<YearOfManufacture>2016</YearOfManufacture>
<ChassisNumber>WAUENAF4XHN011235</ChassisNumber>
<Color>Black</Color>
<LicensePlate>PQR678</LicensePlate>
<TechnicalCondition>Good</TechnicalCondition>
</Vehicle>
<Vehicle>
<VehicleId>8</VehicleId>
<Brand>Tesla</Brand>
<Manufacturer>Tesla, Inc.</Manufacturer>
<Model>Model 3</Model>
<BodyType>Electric Sedan</BodyType>
<YearOfManufacture>2023</YearOfManufacture>
<ChassisNumber>5YJ3E1EB5NF105256</ChassisNumber>
<Color>Blue</Color>
<LicensePlate>STU901</LicensePlate>
<TechnicalCondition>Awful</TechnicalCondition>
</Vehicle>
<Vehicle>
<VehicleId>9</VehicleId>
<Brand>Volkswagen</Brand>
<Manufacturer>Volkswagen Group</Manufacturer>
<Model>Golf</Model>
<BodyType>Hatchback</BodyType>
<YearOfManufacture>2015</YearOfManufacture>

```
            <ChassisNumber>WVWZZZ1KZ4U123456</ChassisNumber>
            <Color>Blue</Color>
            <LicensePlate>XYZ789</LicensePlate>
            <TechnicalCondition>Excellent</TechnicalCondition>
        </Vehicle>
</Vehicles>
```

VehicleDriver.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<VehicleDrivers xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <VehicleDriver>
    <VehicleId>1</VehicleId>
    <DriverId>2</DriverId>
  </VehicleDriver>
  <VehicleDriver>
    <VehicleId>2</VehicleId>
    <DriverId>1</DriverId>
  </VehicleDriver>
  <VehicleDriver>
    <VehicleId>3</VehicleId>
    <DriverId>3</DriverId>
  </VehicleDriver>
  <VehicleDriver>
    <VehicleId>4</VehicleId>
    <DriverId>4</DriverId>
  </VehicleDriver>
  <VehicleDriver>
    <VehicleId>5</VehicleId>
    <DriverId>5</DriverId>
  </VehicleDriver>
  <VehicleDriver>
    <VehicleId>6</VehicleId>
    <DriverId>6</DriverId>
  </VehicleDriver>
  <VehicleDriver>
    <VehicleId>7</VehicleId>
```

```
    <DriverId>7</DriverId>
  </VehicleDriver>
  <VehicleDriver>
    <VehicleId>8</VehicleId>
    <DriverId>8</DriverId>
  </VehicleDriver>
</VehicleDrivers>
```

VehicleOwnership.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<VehicleOwnerships
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <VehicleOwnership>
    <VehicleId>1</VehicleId>
    <OwnerId>1</OwnerId>
  </VehicleOwnership>
  <VehicleOwnership>
    <VehicleId>2</VehicleId>
    <OwnerId>2</OwnerId>
  </VehicleOwnership>
  <VehicleOwnership>
    <VehicleId>3</VehicleId>
    <OwnerId>3</OwnerId>
  </VehicleOwnership>
  <VehicleOwnership>
    <VehicleId>4</VehicleId>
    <OwnerId>4</OwnerId>
  </VehicleOwnership>
  <VehicleOwnership>
    <VehicleId>5</VehicleId>
    <OwnerId>5</OwnerId>
  </VehicleOwnership>
  <VehicleOwnership>
    <VehicleId>6</VehicleId>
    <OwnerId>6</OwnerId>
  </VehicleOwnership>
```

```
  <VehicleOwnership>
    <VehicleId>7</VehicleId>
    <OwnerId>7</OwnerId>
  </VehicleOwnership>
  <VehicleOwnership>
    <VehicleId>8</VehicleId>
    <OwnerId>8</OwnerId>
  </VehicleOwnership>
  <VehicleOwnership>
    <VehicleId>9</VehicleId>
    <OwnerId>8</OwnerId>
  </VehicleOwnership>
</VehicleOwnerships>
```

VehicleRegistrations.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<VehicleRegistrations
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
        <Registration>
                <VehicleId>1</VehicleId>
                <OwnerId>1</OwnerId>
                <RegistrationDate>10.05.2021</RegistrationDate>
                <RegistrationLocation>Department of Motor
Vehicles</RegistrationLocation>
                <IsRegistered>true</IsRegistered>
        </Registration>
        <Registration>
                <VehicleId>2</VehicleId>
                <OwnerId>2</OwnerId>
                <RegistrationDate>20.08.2020</RegistrationDate>
                <RegistrationLocation>Local Registration
Office</RegistrationLocation>
                <IsRegistered>true</IsRegistered>
        </Registration>
        <Registration>
                <VehicleId>3</VehicleId>
```

```
            <OwnerId>3</OwnerId>
            <RegistrationDate>15.03.2022</RegistrationDate>
            <RegistrationLocation>Department of Motor
Vehicles</RegistrationLocation>
            <IsRegistered>true</IsRegistered>
        </Registration>
        <Registration>
            <VehicleId>4</VehicleId>
            <OwnerId>4</OwnerId>
            <RegistrationDate>05.12.2020</RegistrationDate>
            <RegistrationLocation>Local Registration
Office</RegistrationLocation>
            <IsRegistered>true</IsRegistered>
        </Registration>
        <Registration>
            <VehicleId>5</VehicleId>
            <OwnerId>5</OwnerId>
            <RegistrationDate>25.06.2023</RegistrationDate>
            <RegistrationLocation>Department of Motor
Vehicles</RegistrationLocation>
            <IsRegistered>true</IsRegistered>
        </Registration>
        <Registration>
            <VehicleId>6</VehicleId>
            <OwnerId>6</OwnerId>
            <RegistrationDate>18.09.2022</RegistrationDate>
            <RegistrationLocation>Local Registration
Office</RegistrationLocation>
            <IsRegistered>true</IsRegistered>
        </Registration>
        <Registration>
            <VehicleId>7</VehicleId>
            <OwnerId>7</OwnerId>
            <RegistrationDate>30.11.2021</RegistrationDate>
            <RegistrationLocation>Department of Motor
Vehicles</RegistrationLocation>
            <IsRegistered>true</IsRegistered>
```

```
        </Registration>
        <Registration>
                <VehicleId>8</VehicleId>
                <OwnerId>8</OwnerId>
                <RegistrationDate>07.04.2023</RegistrationDate>
                <RegistrationLocation>Local Registration
Office</RegistrationLocation>
                <IsRegistered>true</IsRegistered>
        </Registration>
</VehicleRegistrations>
```

Допоміжні класи для реалізації обробки XML:

DriverXml.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Linq;
using System.Xml.Serialization;
using System.Xml;

namespace laba2.Add
{
   public class DriverXml
   {
      private readonly string _path =
"C:\\Users\\User\\source\\repos\\laba2\\laba2\\XML\\Driver.xml";
      private readonly Data _data;
      public DriverXml(Data data)
      {
         _data = data;
      }
      public void CreateDoc(Driver driver)
      {
         XmlWriterSettings settings = new XmlWriterSettings();
         settings.Indent = true;
```

```
            settings.IndentChars = "\t";
            using (XmlWriter writer = XmlWriter.Create(_path, settings))
            {
                writer.WriteStartDocument();
                writer.WriteStartElement("Drivers");
                writer.WriteStartElement("Driver");
                writer.WriteElementString("DriverId", driver.DriverId.ToString());
                writer.WriteElementString("LastName", driver.LastName);
                writer.WriteElementString("FirstName", driver.FirstName);
                writer.WriteElementString("Patronymic", driver.Patronymic);
                writer.WriteElementString("DateOfBirth",
driver.DateOfBirth.ToString("dd.MM.yyyy"));
                writer.WriteElementString("RegistrationAddress",
driver.RegistrationAddress);
                writer.WriteElementString("LicenseNumber", driver.LicenseNumber);
                writer.WriteEndElement();

                writer.WriteEndElement();
                writer.WriteEndDocument();
                writer.Flush();
            }

            Console.WriteLine("XML created");
        }

        public void AddElem(Driver driver)
        {
            XDocument xdoc = XDocument.Load(_path);
            XElement? root = xdoc.Element("Drivers");

            if (root != null)
            {
                root.Add(new XElement("Driver",
                        new XElement("DriverId", driver.DriverId.ToString()),
                        new XElement("LastName", driver.LastName),
                        new XElement("FirstName", driver.FirstName),
                        new XElement("Patronymic", driver.Patronymic),
```

```
                    new XElement("DateOfBirth",
driver.DateOfBirth.ToString("dd.MM.yyyy")),
                    new XElement("RegistrationAddress",
driver.RegistrationAddress),
                    new XElement("LicenseNumber", driver.LicenseNumber)));

            xdoc.Save(_path);
        }

        Console.WriteLine("XML element added");
    }

    public void ReturnAll()
    {
        XmlDocument xDoc = new XmlDocument();
        xDoc.Load(_path);

        XmlElement? xRoot = xDoc.DocumentElement;
        if (xRoot != null)
        {
            foreach (XmlElement xnode in xRoot)
            {
                foreach (XmlNode childnode in xnode.ChildNodes)
                {
                    Console.WriteLine($"{childnode.Name}:
{childnode.InnerText}");
                }
                Console.WriteLine();
            }
        }
    }

    public void ReadXml()
    {
        XmlRootAttribute xRoot = new XmlRootAttribute();
        xRoot.ElementName = "Drivers";
        xRoot.IsNullable = true;
```

```
        XmlSerializer formatter = new XmlSerializer(typeof(List<Driver>),
xRoot);

        List<Driver> existingDrivers;

        using (System.IO.FileStream fs = new System.IO.FileStream(_path,
System.IO.FileMode.Open))
        {
            existingDrivers = (List<Driver>)formatter.Deserialize(fs);
        }

        foreach (var driver in existingDrivers)
        {
            Console.WriteLine($"DriverId: {driver.DriverId}");
            Console.WriteLine($"LastName: {driver.LastName}");
            Console.WriteLine($"FirstName: {driver.FirstName}");
            Console.WriteLine($"Patronymic: {driver.Patronymic}");
            Console.WriteLine($"DateOfBirth: {driver.DateOfBirth}");
            Console.WriteLine($"RegistrationAddress:
{driver.RegistrationAddress}");
            Console.WriteLine($"LicenseNumber: {driver.LicenseNumber}");
            Console.WriteLine();
        }
    }
  }
}
```

OwnerXml.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Linq;
using System.Xml.Serialization;
```

```
using System.Xml;

namespace laba2.Add
{
   public class OwnerXml
   {
      private readonly string _path =
"C:\\Users\\User\\source\\repos\\laba2\\laba2\\XML\\Owner.xml";
      private readonly Data _data;
      public void CreateDoc(Owner owner)
      {
         XmlWriterSettings settings = new XmlWriterSettings();
         settings.Indent = true;
         settings.IndentChars = "\t";
         using (XmlWriter writer = XmlWriter.Create(_path, settings))
         {
            writer.WriteStartDocument();
            writer.WriteStartElement("Owners");
            writer.WriteStartElement("Owner");
            writer.WriteElementString("OwnerId", owner.OwnerId.ToString());
            writer.WriteElementString("LastName", owner.LastName);
            writer.WriteElementString("FirstName", owner.FirstName);
            writer.WriteElementString("Patronymic", owner.Patronymic);
            writer.WriteElementString("DateOfBirth",
owner.DateOfBirth.ToString("dd.MM.yyyy"));
            writer.WriteElementString("RegistrationAddress",
owner.RegistrationAddress);
            writer.WriteElementString("LicenseNumber",
owner.LicenseNumber);
            writer.WriteEndElement();

            writer.WriteEndElement();
            writer.WriteEndDocument();
            writer.Flush();
         }

         Console.WriteLine("XML created");
```

```
    }

    public void AddElem(Owner owner)
    {
        XDocument xdoc = XDocument.Load(_path);
        XElement? root = xdoc.Element("Owners");

        if (root != null)
        {
            root.Add(new XElement("Owner",
                    new XElement("OwnerId", owner.OwnerId.ToString()),
                    new XElement("LastName", owner.LastName),
                    new XElement("FirstName", owner.FirstName),
                    new XElement("Patronymic", owner.Patronymic),
                    new XElement("DateOfBirth",
owner.DateOfBirth.ToString("dd.MM.yyyy")),
                    new XElement("RegistrationAddress",
owner.RegistrationAddress),
                    new XElement("LicenseNumber", owner.LicenseNumber)));

            xdoc.Save(_path);
        }

        Console.WriteLine("XML element added");
    }

    public void ReturnAll()
    {
        XmlDocument xDoc = new XmlDocument();
        xDoc.Load(_path);

        XmlElement? xRoot = xDoc.DocumentElement;
        if (xRoot != null)
        {
            foreach (XmlElement xnode in xRoot)
            {
                foreach (XmlNode childnode in xnode.ChildNodes)
```

```
                {
                    Console.WriteLine($"{childnode.Name}:
{childnode.InnerText}");
                }
                Console.WriteLine();
            }
        }
    }

    public void ReadXml()
    {
        XmlRootAttribute xRoot = new XmlRootAttribute();
        xRoot.ElementName = "Owners";
        xRoot.IsNullable = true;

        XmlSerializer formatter = new XmlSerializer(typeof(List<Owner>),
xRoot);

        List<Owner> existingOwners;

        using (System.IO.FileStream fs = new System.IO.FileStream(_path,
System.IO.FileMode.Open))
        {
            try
            {
                existingOwners = (List<Owner>)formatter.Deserialize(fs);
            }
            catch (InvalidOperationException ex)
            {
                Console.WriteLine("Error deserializing XML: " + ex.Message);
                return;
            }
        }

        foreach (var owner in existingOwners)
        {
            Console.WriteLine($"OwnerId: {owner.OwnerId}");
```

```
            Console.WriteLine($"LastName: {owner.LastName}");
            Console.WriteLine($"FirstName: {owner.FirstName}");
            Console.WriteLine($"Patronymic: {owner.Patronymic}");
            Console.WriteLine($"DateOfBirth: {owner.DateOfBirthString}");
            Console.WriteLine($"RegistrationAddress:
{owner.RegistrationAddress}");
            Console.WriteLine($"LicenseNumber: {owner.LicenseNumber}");
            Console.WriteLine();
        }
    }

  }
}
```

RegistrationXml.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Linq;
using System.Xml.Serialization;
using System.Xml;
using static System.Runtime.InteropServices.JavaScript.JSType;

namespace laba2.Add
{
  public class RegistrationXml
  {
    private readonly string _path =
"C:\\Users\\User\\source\\repos\\laba2\\laba2\\XML\\VehicleRegistrations.xml";
    private readonly Data _data;

    public void CreateDoc(Registration registration)
    {
      XmlWriterSettings settings = new XmlWriterSettings();
      settings.Indent = true;
```

```
            settings.IndentChars = "\t";
            using (XmlWriter writer = XmlWriter.Create(_path, settings))
            {
                writer.WriteStartDocument();
                writer.WriteStartElement("Registrations");
                writer.WriteStartElement("Registration");
                writer.WriteElementString("VehicleId",
registration.VehicleId.ToString());
                writer.WriteElementString("OwnerId",
registration.OwnerId.ToString());
                writer.WriteElementString("RegistrationDate",
registration.RegistrationDate.ToString("dd.MM.yyyy"));
                writer.WriteElementString("RegistrationLocation",
registration.RegistrationLocation);
                writer.WriteElementString("IsRegistered",
registration.IsRegistered.ToString());
                writer.WriteEndElement();

                writer.WriteEndElement();
                writer.WriteEndDocument();
                writer.Flush();
            }

            Console.WriteLine("XML created");
        }

        public void AddElem(Registration registration)
        {
            XDocument xdoc = XDocument.Load(_path);
            XElement? root = xdoc.Element("VehicleRegistrations");

            if (root != null)
            {
                root.Add(new XElement("Registration",
                        new XElement("VehicleId", registration.VehicleId.ToString()),
                        new XElement("OwnerId", registration.OwnerId.ToString()),
```

```
                    new XElement("RegistrationDate",
registration.RegistrationDate.ToString("dd.MM.yyyy")),
                    new XElement("RegistrationLocation",
registration.RegistrationLocation),
                    new XElement("IsRegistered",
registration.IsRegistered.ToString())));

            xdoc.Save(_path);
        }

        Console.WriteLine("XML element added");
    }

    public void ReturnAll()
    {
        XmlDocument xDoc = new XmlDocument();
        xDoc.Load(_path);

        XmlElement? xRoot = xDoc.DocumentElement;
        if (xRoot != null)
        {
            foreach (XmlElement xnode in xRoot)
            {
                foreach (XmlNode childnode in xnode.ChildNodes)
                {
                    Console.WriteLine($"{childnode.Name}:
{childnode.InnerText}");
                }
                Console.WriteLine();
            }
        }
    }

    public void ReadXml()
    {
        XmlRootAttribute xRoot = new XmlRootAttribute();
        xRoot.ElementName = "VehicleRegistrations";
```

```
        xRoot.IsNullable = true;

        XmlSerializer formatter = new
XmlSerializer(typeof(List<Registration>), xRoot);

        List<Registration> existingRegistrations;

        using (System.IO.FileStream fs = new System.IO.FileStream(_path,
System.IO.FileMode.Open))
        {
            existingRegistrations = (List<Registration>)formatter.Deserialize(fs);
        }

        foreach (var registration in existingRegistrations)
        {
            Console.WriteLine($"VehicleId: {registration.VehicleId}");
            Console.WriteLine($"OwnerId: {registration.OwnerId}");
            Console.WriteLine($"RegistrationDate:
{registration.RegistrationDate:yyyy-MM-dd}");
            Console.WriteLine($"RegistrationLocation:
{registration.RegistrationLocation}");
            Console.WriteLine($"IsRegistered: {registration.IsRegistered}");
            Console.WriteLine();
        }
    }
  }
}
```

VehicleDriverXml.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Linq;
using System.Xml.Serialization;
using System.Xml;
```

```
using static System.Runtime.InteropServices.JavaScript.JSType;

namespace laba2.Add
{
   public class VehicleDriverXml
   {
      private readonly string _path =
"C:\\Users\\User\\source\\repos\\laba2\\laba2\\XML\\VehicleDriver.xml";
      private readonly Data _data;

      public void CreateDoc(VehicleDriver driver)
      {
         XmlWriterSettings settings = new XmlWriterSettings();
         settings.Indent = true;
         settings.IndentChars = "\t";
         using (XmlWriter writer = XmlWriter.Create(_path, settings))
         {
            writer.WriteStartDocument();
            writer.WriteStartElement("VehicleDrivers");
            writer.WriteStartElement("VehicleDriver");
            writer.WriteElementString("VehicleId", driver.VehicleId.ToString());
            writer.WriteElementString("DriverId", driver.DriverId.ToString());
            writer.WriteEndElement();

            writer.WriteEndElement();
            writer.WriteEndDocument();
            writer.Flush();
         }

         Console.WriteLine("XML created");
      }

      public void AddElem(VehicleDriver driver)
      {
         XDocument xdoc = XDocument.Load(_path);
         XElement? root = xdoc.Element("VehicleDrivers");
```

```
        if (root != null)
        {
            root.Add(new XElement("VehicleDriver",
                        new XElement("VehicleId", driver.VehicleId.ToString()),
                        new XElement("DriverId", driver.DriverId.ToString())));

            xdoc.Save(_path);
        }

        Console.WriteLine("XML element added");
    }


    public void ReturnAll()
    {
        XmlDocument xDoc = new XmlDocument();
        xDoc.Load(_path);

        XmlElement? xRoot = xDoc.DocumentElement;
        if (xRoot != null)
        {
            foreach (XmlElement xnode in xRoot)
            {
                foreach (XmlNode childnode in xnode.ChildNodes)
                {
                    Console.WriteLine($"{childnode.Name}:
{childnode.InnerText}");
                }
                Console.WriteLine();
            }
        }
    }

    public void ReadXml()
    {
```

```
        XmlSerializer serializer = new
XmlSerializer(typeof(List<VehicleDriver>), new
XmlRootAttribute("VehicleDrivers"));

        List<VehicleDriver> existingVehicleDrivers;

        using (FileStream fs = new FileStream(_path, FileMode.Open))
        {
            existingVehicleDrivers =
(List<VehicleDriver>)serializer.Deserialize(fs);
        }

        foreach (var vehicleDriver in existingVehicleDrivers)
        {
            Console.WriteLine($"VehicleId: {vehicleDriver.VehicleId}");
            Console.WriteLine($"DriverId: {vehicleDriver.DriverId}");
            Console.WriteLine();
        }
    }

  }
}
```

VehicleOwnershipXml.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Linq;
using System.Xml.Serialization;
using System.Xml;

namespace laba2.Add
{
  public class VehicleOwnershipXml
  {
```

```
        private readonly string _path =
"C:\\Users\\User\\source\\repos\\laba2\\laba2\\XML\\VehicleOwnership.xml";
        private readonly Data _data;

        public void CreateDoc(VehicleOwnership ownership)
        {
            XmlWriterSettings settings = new XmlWriterSettings();
            settings.Indent = true;
            settings.IndentChars = "\t";
            using (XmlWriter writer = XmlWriter.Create(_path, settings))
            {
                writer.WriteStartDocument();
                writer.WriteStartElement("VehicleOwnerships");
                writer.WriteStartElement("VehicleOwnership");
                writer.WriteElementString("VehicleId",
ownership.VehicleId.ToString());
                writer.WriteElementString("OwnerId",
ownership.OwnerId.ToString());
                writer.WriteEndElement();

                writer.WriteEndElement();
                writer.WriteEndDocument();
                writer.Flush();
            }

            Console.WriteLine("XML created");
        }

        public void AddElem(VehicleOwnership ownership)
        {
            XDocument xdoc = XDocument.Load(_path);
            XElement? root = xdoc.Element("VehicleOwnerships");

            if (root != null)
            {
                root.Add(new XElement("VehicleOwnership",
                        new XElement("VehicleId", ownership.VehicleId.ToString()),
```

```
                    new XElement("OwnerId", ownership.OwnerId.ToString())));

            xdoc.Save(_path);
        }

        Console.WriteLine("XML element added");
    }

    public void ReturnAll()
    {
        XmlDocument xDoc = new XmlDocument();
        xDoc.Load(_path);

        XmlElement? xRoot = xDoc.DocumentElement;
        if (xRoot != null)
        {
            foreach (XmlElement xnode in xRoot)
            {
                foreach (XmlNode childnode in xnode.ChildNodes)
                {
                    Console.WriteLine($"{childnode.Name}:
{childnode.InnerText}");
                }
                Console.WriteLine();
            }
        }
    }

    public void ReadXml()
    {
        XmlSerializer serializer = new
XmlSerializer(typeof(List<VehicleOwnership>), new
XmlRootAttribute("VehicleOwnerships"));

        List<VehicleOwnership> existingVehicleOwnerships;

        using (FileStream fs = new FileStream(_path, FileMode.Open))
```

```
            {
                existingVehicleOwnerships =
(List<VehicleOwnership>)serializer.Deserialize(fs);
            }

            foreach (var vehicleOwnership in existingVehicleOwnerships)
            {
                Console.WriteLine($"VehicleId: {vehicleOwnership.VehicleId}");
                Console.WriteLine($"OwnerId: {vehicleOwnership.OwnerId}");
                Console.WriteLine();
            }
        }
    }
}
```

VehicleXml.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Linq;
using System.Xml.Serialization;
using System.Xml;

namespace laba2.Add
{
    public class VehicleXml
    {
        private readonly string _path =
"C:\\Users\\User\\source\\repos\\laba2\\laba2\\XML\\Vehicle.xml";
        private readonly Data _data;

        public void CreateDoc(Vehicle vehicle)
        {
            XmlWriterSettings settings = new XmlWriterSettings();
            settings.Indent = true;
```

```
            settings.IndentChars = "\t";
            using (XmlWriter writer = XmlWriter.Create(_path, settings))
            {
                writer.WriteStartDocument();
                writer.WriteStartElement("Vehicles");
                writer.WriteStartElement("Vehicle");
                writer.WriteElementString("VehicleId", vehicle.VehicleId.ToString());
                writer.WriteElementString("Brand", vehicle.Brand);
                writer.WriteElementString("Manufacturer", vehicle.Manufacturer);
                writer.WriteElementString("Model", vehicle.Model);
                writer.WriteElementString("BodyType", vehicle.BodyType);
                writer.WriteElementString("YearOfManufacture",
vehicle.YearOfManufacture.ToString());
                writer.WriteElementString("ChassisNumber",
vehicle.ChassisNumber);
                writer.WriteElementString("Color", vehicle.Color);
                writer.WriteElementString("LicensePlate", vehicle.LicensePlate);
                writer.WriteElementString("TechnicalCondition",
vehicle.TechnicalCondition);
                writer.WriteEndElement();

                writer.WriteEndElement();
                writer.WriteEndDocument();
                writer.Flush();
            }

            Console.WriteLine("XML created");
        }

        public void addElem(Vehicle vehicle)
        {
            XDocument xdoc = XDocument.Load(_path);
            XElement? root = xdoc.Element("Vehicles");

            if (root != null)
            {
                root.Add(new XElement("Vehicle",
```

```
                    new XElement("VehicleId", vehicle.VehicleId.ToString()),
                    new XElement("Brand", vehicle.Brand),
                    new XElement("Manufacturer", vehicle.Manufacturer),
                    new XElement("Model", vehicle.Model),
                    new XElement("BodyType", vehicle.BodyType),
                    new XElement("YearOfManufacture",
vehicle.YearOfManufacture.ToString()),
                    new XElement("ChassisNumber", vehicle.ChassisNumber),
                    new XElement("Color", vehicle.Color),
                    new XElement("LicensePlate", vehicle.LicensePlate),
                    new XElement("TechnicalCondition",
vehicle.TechnicalCondition)));

            xdoc.Save(_path);
        }

        Console.WriteLine("XML element added");
    }

    public void returnAll()
    {
        XmlDocument xDoc = new XmlDocument();
        xDoc.Load(_path);

        XmlElement? xRoot = xDoc.DocumentElement;
        if (xRoot != null)
        {
            foreach (XmlElement xnode in xRoot)
            {
                foreach (XmlNode childnode in xnode.ChildNodes)
                {
                    Console.WriteLine($"{childnode.Name}:
{childnode.InnerText}");
                }
                Console.WriteLine();
            }
        }
```

```
        }

        public void ReadXml()
        {
            XmlSerializer serializer = new XmlSerializer(typeof(List<Vehicle>),
new XmlRootAttribute("Vehicles"));

            List<Vehicle> existingVehicles;

            using (FileStream fs = new FileStream(_path, FileMode.Open))
            {
                existingVehicles = (List<Vehicle>)serializer.Deserialize(fs);
            }

            foreach (var vehicle in existingVehicles)
            {
                Console.WriteLine($"VehicleId: {vehicle.VehicleId}");
                Console.WriteLine($"Brand: {vehicle.Brand}");
                Console.WriteLine($"Manufacturer: {vehicle.Manufacturer}");
                Console.WriteLine($"Model: {vehicle.Model}");
                Console.WriteLine($"BodyType: {vehicle.BodyType}");
                Console.WriteLine($"YearOfManufacture:
{vehicle.YearOfManufacture}");
                Console.WriteLine($"ChassisNumber: {vehicle.ChassisNumber}");
                Console.WriteLine($"Color: {vehicle.Color}");
                Console.WriteLine($"LicensePlate: {vehicle.LicensePlate}");
                Console.WriteLine($"TechnicalCondition:
{vehicle.TechnicalCondition}");
                Console.WriteLine();
            }
        }

    }
}
```

Основна програма:

```
using laba2.Add;
using System;
using System.Reflection.Metadata;
using System.Xml;
using System.Xml.Linq;

namespace laba2
{
   internal class Program
   {
      static void Main()
      {
         Console.OutputEncoding = System.Text.Encoding.Unicode;
         string answer = "";

         Data dataLists = new Data();
         DriverXml driverXml = new DriverXml(dataLists);
         OwnerXml ownerXml = new OwnerXml();
         RegistrationXml registrationXml = new RegistrationXml();
         VehicleDriverXml vehicleDriverXml = new VehicleDriverXml();
         VehicleOwnershipXml vehicleOwnershipXml = new
VehicleOwnershipXml();
         VehicleXml vehicleXml = new VehicleXml();

         Driver driver;
         Owner owner;
         Registration registration;
         Vehicle vehicle;
         VehicleDriver vehicleDriver;
         VehicleOwnership vehicleOwnership;

         do
         {
            Console.Clear();
            Console.WriteLine("Головне меню \n" +
               "\n1. Створити новий XML-файл Driver.xml" +
```

```
            "\n2. Створити новий XML-файл Owner.xml" +
            "\n3. Створити новий XML-файл Registration.xml" +
            "\n4. Створити новий XML-файл VehicleDriver.xml" +
            "\n5. Створити новий XML-файл VehicleOwnership.xml" +
            "\n6. Створити новий XML-файл Vehicle.xml" +
            "\n7. Додати новий об'єкт Driver до XML документу" +
            "\n8. Додати новий об'єкт Owner до XML документу" +
            "\n9. Додати новий об'єкт Registration до XML документу" +
            "\n10. Додати новий об'єкт VehicleDriver до XML документу" +
            "\n11. Додати новий об'єкт VehicleOwnership до XML
документу" +
            "\n12. Додати новий об'єкт Vehicle до XML документу" +
            "\n13. Вивести список об'єктів Driver з XML файлу" +
            "\n14. Вивести список об'єктів Owner з XML файлу" +
            "\n15. Вивести список об'єктів Registration з XML файлу" +
            "\n16. Вивести список об'єктів VehicleDriver з XML файлу" +
            "\n17. Вивести список об'єктів VehicleOwnership з XML файлу"
+
            "\n18. Вивести список об'єктів Vehicle з XML файлу" +
            "\n19. Запити над даними ");

        int input = Convert.ToInt32(Console.ReadLine());
        switch (input)
        {
            case 1:
                Console.Clear();
                driver = Driver.CreateDriver();
                driverXml.CreateDoc(driver);
                break;
            case 2:
                Console.Clear();
                owner = Owner.CreateOwner();
                ownerXml.CreateDoc(owner);
                break;
            case 3:
                Console.Clear();
                registration = Registration.CreateRegistration();
```

```
                registrationXml.CreateDoc(registration);
                break;
            case 4:
                Console.Clear();
                vehicleDriver = VehicleDriver.CreateVehicleDriver();
                vehicleDriverXml.CreateDoc(vehicleDriver);
                break;
            case 5:
                Console.Clear();
                vehicleOwnership =
VehicleOwnership.CreateVehicleOwnership();
                vehicleOwnershipXml.CreateDoc(vehicleOwnership);
                break;
            case 6:
                Console.Clear();
                vehicle = Vehicle.CreateVehicle();
                vehicleXml.CreateDoc(vehicle);
                break;
            case 7:
                Console.Clear();
                driver = Driver.CreateDriver();
                driverXml.AddElem(driver);
                break;
            case 8:
                Console.Clear();
                owner = Owner.CreateOwner();
                ownerXml.AddElem(owner);
                break;
            case 9:
                Console.Clear();
                registration = Registration.CreateRegistration();
                registrationXml.AddElem(registration);
                break;
            case 10:
                Console.Clear();
                vehicleDriver = VehicleDriver.CreateVehicleDriver();
                vehicleDriverXml.AddElem(vehicleDriver);
```

```
                    break;
                case 11:
                    Console.Clear();
                    vehicleOwnership =
VehicleOwnership.CreateVehicleOwnership();
                    vehicleOwnershipXml.AddElem(vehicleOwnership);
                    break;
                case 12:
                    Console.Clear();
                    vehicle = Vehicle.CreateVehicle();
                    vehicleXml.addElem(vehicle);
                    break;
                case 13:
                    Console.Clear();
                    DisplayDataChoice("driver", () => driverXml.ReturnAll(), () =>
driverXml.ReadXml());
                    break;
                case 14:
                    Console.Clear();
                    DisplayDataChoice("owner", () => ownerXml.ReturnAll(), () =>
ownerXml.ReadXml());
                    break;
                case 15:
                    Console.Clear();
                    DisplayDataChoice("registration", () =>
registrationXml.ReturnAll(), () => registrationXml.ReadXml());
                    break;
                case 16:
                    Console.Clear();
                    DisplayDataChoice("vehicleDriver", () =>
vehicleDriverXml.ReturnAll(), () => vehicleDriverXml.ReadXml());
                    break;
                case 17:
                    Console.Clear();
                    DisplayDataChoice("vehicleOwnership", () =>
vehicleOwnershipXml.ReturnAll(), () => vehicleOwnershipXml.ReadXml());
                    break;
```

```
                case 18:
                    Console.Clear();
                    DisplayDataChoice("vehicle", () => vehicleXml.returnAll(), ()
=> vehicleXml.ReadXml());
                    break;
            }

            Queries queries = new Queries();
            if (input == 19)
            {
                Console.Clear();
                Console.WriteLine("Оберіть запит який хочете виконати: \n" +
                    "\n1. Вивести усі машини з типом кузову SUV" +
                    "\n2. Вивести усі машини, що мають тип кузову SUV та
відповідно їх власників" +
                    "\n3. Вивести перелік машин, що мають відмінні технічні
показники" +
                    "\n4. Вивести перелік власників, вік яких входить у діапазон
від 10 до 40 років" +
                    "\n5. Вивести водія, що має найдовше ім'я" +
                    "\n6. Вивести машину, що має найстаріший рік випуску
(найбільш старенька)" +
                    "\n7. Вивести відсортований перелік водіїв за датою
народження в порядку зростання (діапазон з 1990 до 2000 року
народження)" +
                    "\n8. Вивести усі машини, що не є зареєстрованими" +
                    "\n9. Вивести перелік машин, випущених до 2016 року" +
                    "\n10. Вивести машини, зареєстрованих в локації \"Local
Registration Office\"" +
                    "\n11. Вивести машини, що не мають водія" +
                    "\n12. Вивести перелік всіх унікальних власників машин та
водіїв, об'єднавши їх імена" +
                    "\n13. Вивести перелік власників, що мають декілька машин"
+
                    "\n14. Вивести перелік машин, відсортовані за роком
випуску (спадання, діапазон 2000-2017)" +
                    "\n15. Вивести перелік водіїв без машин" +
```

```
                "\n16. Вивести середній вік водіїв" +
                "\n17. Вивести перелік водіїв молодше 30" +
                "\n18. Вивести список автомобілів, що були зареєстровані у
2023 році" +
                "\n19. Вивести список водіїв, що народилися у 1990 році" +
                "\n20. Вивести список автомобілів, які належать власнику з
ID: 1");

            int inputQuery = Convert.ToInt32(Console.ReadLine());
            switch (inputQuery)
            {
                case 1:
                    Console.Clear();
                    queries.GetSUVVehicles();
                    break;
                case 2:
                    Console.Clear();
                    queries.GetAllSUVsWithOwners();
                    break;
                case 3:
                    Console.Clear();
                    queries.GetMachinesInExcellentCondition();
                    break;
                case 4:
                    Console.Clear();
                    queries.GetOwnersWithinAgeRange(10, 40);
                    break;
                case 5:
                    Console.Clear();
                    queries.GetDriverWithLongestName(dataLists.Drivers);
                    break;
                case 6:
                    Console.Clear();

queries.GetVehicleWithEarliestManufactureYear(dataLists.Vehicles);
                    break;
                case 7:
```

```
                    Console.Clear();

queries.SortDriversByDateOfBirthAscendingWithBoundaries(dataLists.Drivers,
1990, 2000);
                    break;
                case 8:
                    Console.Clear();
                    queries.GetUnregisteredVehicles(dataLists.Vehicles,
dataLists.Registrations);
                    break;
                case 9:
                    Console.Clear();
                    queries.GetMachinesReleasedBeforeYear(dataLists.Vehicles,
2016);
                    break;
                case 10:
                    Console.Clear();

queries.GetVehiclesRegisteredInLocalOffice(dataLists.Vehicles,
dataLists.Registrations);
                    break;
                case 11:
                    Console.Clear();
                    queries.GetVehiclesNotDrivenByDrivers(dataLists.Vehicles,
dataLists.Drivers, dataLists.VehicleDrivers);
                    break;
                case 12:
                    Console.Clear();

queries.GetUniqueOwnersAndDriversNames(dataLists.Owners,
dataLists.Drivers);
                    break;
                case 13:
                    Console.Clear();
                    queries.GetOwnersWithMultipleVehicles(dataLists.Owners,
dataLists.VehicleOwnerships);
                    break;
```

```
                case 14:
                    Console.Clear();

queries.SortVehiclesByModelYearDescendingWithBoundaries(dataLists.Vehicle
s, 2000, 2017);
                    break;
                case 15:
                    Console.Clear();
                    queries.FindDriversWithoutVehicles(dataLists.Drivers,
dataLists.VehicleDrivers);
                    break;
                case 16:
                    Console.Clear();
                    queries.GetAverageAgeOfDrivers(dataLists.Drivers);
                    break;
                case 17:
                    Console.Clear();
                    queries.GetDriversYoungerThanAge(dataLists.Drivers, 30);
                    break;
                case 18:
                    Console.Clear();
                    queries.GetVehiclesRegisteredInYear(dataLists.Vehicles,
dataLists.Registrations, 2023);
                    break;
                case 19:
                    Console.Clear();
                    queries.GetDriversBornInYear(dataLists.Drivers, 1990);
                    break;
                case 20:
                    Console.Clear();
                    queries.GetCarsOwnedByOwner(dataLists.Vehicles,
dataLists.Owners, dataLists.VehicleOwnerships, 1);
                    break;
            }
        }
        Console.WriteLine("Натисніть Enter для продовження");
        answer = Console.ReadLine();
```

```
        }
        while (answer == "+");

    }

    public static void DisplayDataChoice(string dataType, Action
usingXmlDocument, Action usingXmlSerializer)
    {
        Console.Clear();
        Console.WriteLine($"Оберіть спосіб відображення {dataType}
даних:");
        Console.WriteLine("1. Використовуючи XmlDocument");
        Console.WriteLine("2. Використовуючи XmlSerializer");
        Console.Write("Ваш вибір: ");
        string choice = Console.ReadLine();

        switch (choice)
        {
            case "1":
                usingXmlDocument(); // XmlDocument
                break;
            case "2":
                usingXmlSerializer(); // XmlSerializer
                break;
            default:
                Console.WriteLine("Invalid choice. Please enter 1 or 2.");
                break;
        }
    }
  }
}
```