**Міністерство освіти та науки України**

**Державний вищий навчальний заклад “Київський національний економічний університет ім. Вадима Гетьмана”**

**Кафедра інформаційних систем в економіці**

**Дисципліна “Адміністрування та програмування баз даних в штучному інтелекті”**

**ЗВІТ**

**з лабораторної роботи №2**

**«С# Essential»**

**Підготував:**

студент 5 курсу гр. ІШІ-501

спеціальності «8і09»

Донєв Данило Романович

**Прийняла:**

доц. Добролюбова М.В.

***ВІДПОВІДІ НА ПИТАННЯ***

**1. Що таке клас, об’єкт, екземпляр класу? Що таке властивість та конструктор? Які бувають види властивостей та конструкторів?**

**Клас** — це спеціальна конструкція, яка використовується для групування пов'язаних змінних та функцій. При цьому, згідно з термінологією ООП, глобальні змінні класу (члени-змінні) називаються полями даних (також властивостями або атрибутами), а члени-функції називають методами класу. На основі одного класу можна створити безліч об'єктів, що відрізнятимуться один від одного своїм станом (значеннями полів).

**Обʼєкти** містять в собі статичні поля та всі методи. **Екземпляри** містять нестатичні поля.

**Властивість** - це конструкція мови С#, яка заміняє собою використання звичайних методів доступу. Властивість складається з імені, типу та тіла. В тілі задаються методи доступу, через використання ключових слів set та get.

Бувають властивості, які реалізуються автоматично, та які реалізуються користувачем.

Властивості, які реалізуються автоматично, - це більш лаконічна форма властивостей, їх використовують, коли в методах доступу get та set не потрібна додаткова логіка. При створенні властивостей, які реалізуються автоматично, компілятор створить з закрите анонімне резервне поле, яке буде доступне за допомогою методів get та set властивості.

**Конструктор** класу - спеціальний метод, який викликається під час побудови класу.

Бувають Конструктори за замовчуванням та Конструктор користувача. Задача конструктора за замовчуванням - ініціалізація полів значеннями за

замовчуванням. Задача конструктора користувача - ініціалізація полів зумовленими користувачем значеннями.

**2. Що таке ООП? Описати основні парадигми OOП. Що таке Cast, Upcast, Downcast? Що таке модифікатори доступу та де їх використовують?**

Об'є́ктно-орієнто́ване програмува́ння — одна з парадигм програмування, яка розглядає програму як множину «об'єктів», що взаємодіють між собою. Основні парадигми:

* Інкапсуляція - об’єкт вміщує не тільки дані, але і правила їх обробки, оформлені в вигляді виконуваних фрагментів (методів). А також про те, що доступ до стану об'єкта напряму заборонено, і ззовні з ним можна взаємодіяти виключно через заданий інтерфейс (відкриті поля та методи), що дозволяє знизити зв'язність. Таким чином контролюються звернення до полів класів та їхня правильна ініціалізація, усуваються можливі помилки пов'язані з неправильним викликом методу. Оскільки користувачі працюють лише через відкриті елементи класів, то розробники класу можуть як завгодно змінювати всі закриті елементи і, навіть, перейменовувати та видаляти їх, не турбуючись, що десь хтось їх використовує у своїх програмах.
* Наслідування - дає класу можливість використовувати програмний код іншого (базового) класу, доповнюючи його своїми власними деталями реалізації. Іншими словами, під час наслідування відбувається отримання нового (похідного) класу, який містить програмний код базового класу з зазначенням власних особливостей використання. Наслідування належить до типу is-a відношень між класами. При успадкуванні створюється спеціалізована версія вже існуючого класу.
* Поліморфізм - використання єдиного інтерфейсу для різнотипних сутностей або у використанні однакового символу для маніпуляцій над даними різного типу
* Ревикористання коду - це використання існуючого програмного забезпечення або знань про програмне забезпечення для створення нового програмного забезпечення відповідно до принципів повторного використання.
* Абстракція - процес виділення загальних характеристик і функціональності об'єктів або системи, ігноруючи деталі реалізації.
* Обмін повідомленнями

UpCast - приведення екземпляра похідного класу до базового типу.

*BaseClass up = new DerivedClass();*

DownCast - приведення екземпляра базового типу до похідного типу.

*Derivedclass down = (Derivedclass) up;*

DownCast неможливий без попереднього UpCast.

Усі поля, способи та інші компоненти класу мають модифікатори доступу .

Модифікатори доступу дозволяють встановити допустиму область видимості для компонентів класу. Тобто модифікатори доступу визначають контекст, у якому можна використовувати цю змінну чи спосіб.

У мові C# застосовуються такі модифікатори доступу:

**private** – закритий або приватний компонент класу чи структури. Приватний компонент доступний лише у межах свого класу чи структури.

**private protected** – компонент класу доступний з будь-якого місця у своєму класі або похідних класах, які визначені в тій же збірці.

**protected** – такий компонент класу доступний з будь-якого місця у своєму класі або похідних класах. При цьому похідні класи можуть розміщуватись в інших зборках.

**internal** – компоненти класу або структури доступні з будь-якого місця коду в тій самій збірці, однак він недоступний для інших програм та збірок.

**protected internal** – поєднує функціонал двох модифікаторів protected та internal. Такий компонент класу доступний з будь-якого місця в поточній збірці та похідних класів, які можуть розташовуватися в інших збірках.

**public** – публічний, загальнодоступний компонент класу чи структури. Такий компонент доступний з будь-якого місця в коді, а також інших програм і збірок.



**3. Що таке абстрактний клас та інтерфейс? В чому між ними різниця?**

Абстрактний клас в обʼєктно-орієнтованому програмуванні - це базовий клас, що не передбачає створення екземплярів через виклик конструктора безпосередньо, але екземпляр абстрактного класу створюється неявно при побудові екземпляра похідного конкретного класу.

Інтерфейс - семантична і синтаксична конструкція в коді програми, яка використовується для специфікування послуг, що надаються класом або

компонентом. Інтерфейс - стереотип, який є аналогом чистого абстрактного класу, в якому заборонена будь-яка реалізація.

Відмінності:

* Інтерфейс описує лише поведінку. Він не має стану. А в абстрактного класу стан є: він описує і те, й інше.
* Абстрактний клас пов'язує між собою та поєднує класи, що мають дуже близький зв'язок. У той самий час, той самий інтерфейс можуть реалізувати класи, які взагалі немає нічого спільного.

**4. Що таке структура? В чому різниця між класом та структурою?**

Структура (класична) - це конструкція мови, що дозволяє утримувати в собі набір полів різних типів. клас - є посилальним типом (reference type), а структури - значущим типом (value type) (див. статтю "Типи даних"). Отже, класи завжди створюються в так званій "купі" (heap), а структури створюються в стеку (stack). Структури, що вказуються в списку параметрів методу, передаються за значенням (тобто копіюються), об'єкти класів - за посиланням.

**5. Що таке делегат? Що таке лямбда вираз та чим він відрізняється від лямбда оператора?**

**Делегат** (delegate) - це різновид обʼєктів, які містять в собі покажчики на методи.

**Лямбда-оператор**

У всіх лямбда-виразах використовується лямбда-оператор =>, який читається як "переходить в" Ліва частина лямбда-оператора визначає параметри введення (якщо такі є), а права частина містить вираз або блок оператора. Лямбда-вираз х => х \* х читається як "х переходить в х, х раз".

**Лямбда-вираз**

Лямбда-вираз - це анонімні методи, яка містить вирази й оператори і може використовуватися для створення делегатів. Лямбда-Оператор - це багатооператорний лямбда-вираз. Лямбда-вираз - це однооператорний лямбда-оператор.

Створення анонімних методів є, по суті, способом передачі блоку коду в якості параметра делегата.

**6. Що таке потік? Який клас необхідно використовувати для створення екземпляра потоку? Який делегат необхідно використовувати для передачі метода в потік?**

**Потік** - це певна незалежна послідовність інструкцій для виконання тієї чи іншої дії в програмі. В одному конкретному потоці виконується одна конкретна послідовність дій.

Сукупність таких потоків, що виконуються в програмі асинхронно, називається багатопотоковістю програми. Створення нового об'єкта **Thread** призводить до створення нового керованого потоку. Клас Thread має конструктори, які приймають делегат ThreadStart або ParameterizedThreadStart. Цей делегат інкапсулює метод, який викликається новим потоком при виклику методу Start. Повторний виклик Start призводить до створення винятку ThreadStateException. Делегат ParameterizedThreadStart надає простий спосіб передати в потік об'єкт з даними при виклику Thread.Start(Object). Див. приклад коду в ParameterizedThreadStart.

**Делегат ParameterizedThreadStart** не є типобезпечним способом передачі даних, оскільки метод Thread.Start(Object) приймає будь-який об'єкт. Замість цього можна інкапсулювати процедуру потоку і дані у допоміжний клас і виконувати процедуру потоку за допомогою делегата ThreadStart.

***ЗАВДАННЯ***

***Загальні завдання***

Покликання на гітхаб з кодом та файлами, які використовуються в роботі - https://github.com/doniev-danylo/kneu\_cs/tree/main/lab2

***Завдання 1***

Використовуючи VisualStudio, створіть проект за шаблоном ConsoleApplication.

Необхідно: Створити клас AbstractHandler.

В тілі класу створіть методи void Open(), void Create(), void Chenge(), void Save().

Створити похідні класи XMLHandler, TXTHandler, DOCHandler від базового класу AbstractHandler. Написати програму, яка буде виконувати визначення документа і для кожного формату повинні бути методи відкриття, створення, редагування, збереження певного формату документу.

КОД ПРОГРАМИ

using System.Xml.Linq;  
  
namespace lab2;  
  
public static class Lab2Task1  
{  
 public static void RunTask1()  
 {  
 TestXml();  
 TestDoc();  
 TestTxt();  
 }  
  
 private static void TestXml()  
 {  
 var xmlExists = HandleDocument("document.xml");  
 xmlExists.Open();  
 xmlExists.Change("to", "Alla");  
 xmlExists.Save();  
  
  
 var xmlDontExist = HandleDocument("document\_new.xml");  
 xmlDontExist.Create();  
  
 xmlDontExist.Change("to", "Ksenia");  
 xmlDontExist.Save();  
 xmlDontExist.Change("from", "Laura");  
 xmlDontExist.Save();  
  
 xmlDontExist.Open();  
 xmlDontExist.Change("to", "Boris");  
 xmlDontExist.Save();  
 }  
  
 private static void TestTxt()  
 {  
 var txtExists = HandleDocument("document.txt");  
 txtExists.Open();  
 txtExists.Change("letter", "message");  
 txtExists.Save();  
  
  
 var txtDontExist = HandleDocument("document\_new.txt");  
 txtDontExist.Create();  
  
 txtDontExist.Change("Laura", "Ksenia");  
 txtDontExist.Save();  
 txtDontExist.Change("Linda", "Laura");  
 txtDontExist.Save();  
  
 txtDontExist.Open();  
 txtDontExist.Change("Laura", "Boris");  
 txtDontExist.Save();  
 }  
  
 private static void TestDoc()  
 {  
 var docExists = HandleDocument("document.doc");  
 docExists.Open();  
 docExists.Change("letter", "message");  
 docExists.Save();  
  
  
 var docDontExist = HandleDocument("document\_new.doc");  
 docDontExist.Create();  
  
 docDontExist.Change("Laura", "Ksenia");  
 docDontExist.Save();  
 docDontExist.Change("Linda", "Laura");  
 docDontExist.Save();  
  
 docDontExist.Open();  
 docDontExist.Change("Laura", "Boris");  
 docDontExist.Save();  
 }  
  
 private static AbstractHandler HandleDocument(string fileName)  
 {  
 AbstractHandler handler;  
 if (fileName.EndsWith(".xml"))  
 {  
 handler = new XmlHandler(fileName);  
 }  
 else if (fileName.EndsWith(".txt"))  
 {  
 handler = new TxtHandler(fileName);  
 }  
 else if (fileName.EndsWith(".doc"))  
 {  
 handler = new DocHandler(fileName);  
 }  
 else  
 {  
 throw new ArgumentException("Unsupported file format");  
 }  
  
 return handler;  
 }  
}  
  
public abstract class AbstractHandler  
{  
 protected readonly string FilePath;  
 protected bool IsFileExists;  
 protected bool IsFileOpened;  
  
 protected AbstractHandler(string filePath)  
 {  
 var projectDirectory = Directory.GetParent(Environment.CurrentDirectory).Parent?.Parent?.FullName;  
 var filePathFromRoot = Path.Combine(projectDirectory, filePath);  
 Console.WriteLine(filePath);  
  
 if (File.Exists(filePath))  
 {  
 FilePath = filePath;  
 IsFileExists = true;  
 }  
 else if (File.Exists(filePathFromRoot))  
 {  
 FilePath = filePathFromRoot;  
 IsFileExists = true;  
 }  
 else  
 {  
 FilePath = filePathFromRoot;  
 IsFileExists = false;  
 }  
  
 IsFileOpened = false;  
 }  
  
 protected void FileShouldExist(bool should)  
 {  
 if (IsFileExists != should)  
 {  
 var isShould = should ? "should" : "should not";  
 throw new ArgumentException($"File {isShould} exists");  
 }  
 }  
  
 protected void FileShouldBeOpened(bool should)  
 {  
 if (IsFileOpened != should)  
 {  
 var isShould = should ? "should" : "should not";  
 throw new ArgumentException($"File {should} be opened");  
 }  
 }  
  
 public abstract void Open();  
 public abstract void Create();  
 public abstract void Change(string param1, string param2);  
 public abstract void Save();  
}  
  
public class XmlHandler : AbstractHandler  
{  
 private XDocument \_xml;  
  
 public XmlHandler(string filePath) : base(filePath)  
 {  
 }  
  
 public override void Open()  
 {  
 FileShouldExist(true);  
 \_xml = XDocument.Load(FilePath);  
 Console.WriteLine("Successfully opened the XML file.");  
 IsFileOpened = true;  
 }  
  
 public override void Create()  
 {  
 FileShouldExist(false);  
 \_xml = new XDocument(  
 new XElement("note",  
 new XElement("to", "Alex"),  
 new XElement("from", "Boris")  
 )  
 );  
 Console.WriteLine("Successfully created the XML file.");  
 IsFileOpened = true;  
 }  
  
 public override void Change(string element, string value)  
 {  
 FileShouldBeOpened(true);  
 foreach (XElement el in \_xml.Root.Elements(element))  
 {  
 el.Value = value;  
 }  
  
 Console.WriteLine("Successfully changed the XML file.");  
 }  
  
 public override void Save()  
 {  
 FileShouldBeOpened(true);  
 \_xml.Save(FilePath);  
 IsFileExists = true;  
 Console.WriteLine("Successfully saved the XML file.");  
 }  
}  
  
public class TxtHandler : AbstractHandler  
{  
 private string \_content;  
  
 public TxtHandler(string filePath) : base(filePath)  
 {  
 }  
  
 public override void Open()  
 {  
 FileShouldExist(true);  
 \_content = File.ReadAllText(FilePath);  
 Console.WriteLine("Successfully opened the TXT file.");  
 IsFileOpened = true;  
 }  
  
 public override void Create()  
 {  
 FileShouldExist(false);  
 \_content = "that is the letter from Laura to Linda";  
 Console.WriteLine("Successfully created the TXT file.");  
 IsFileOpened = true;  
 }  
  
 public override void Change(string line, string newValue)  
 {  
 FileShouldBeOpened(true);  
 \_content = \_content.Replace(line, newValue);  
 Console.WriteLine("Successfully changed the TXT file.");  
 }  
  
 public override void Save()  
 {  
 FileShouldBeOpened(true);  
 File.WriteAllText(FilePath, \_content);  
 IsFileExists = true;  
 Console.WriteLine("Successfully saved the TXT file.");  
 }  
}  
  
public class DocHandler : TxtHandler *// 🤫*{  
 public DocHandler(string filePath) : base(filePath)  
 {  
 }  
}

РЕЗУЛЬТАТИ

Консольний вивід демонструє успішне тестування всіх наявних функцій:

/usr/local/share/dotnet/dotnet --fx-version 7.0.11 /Users/donevd/Applications/Rider.app/Contents/lib/ReSharperHost/JetLauncherILc.exe /Launcher::NoSplash /Launcher::Mode:InvokeMethod /Launcher::Target:Assembly /Launcher::AssemblyFile:/Users/donevd/RiderProjects/kneu\_cs/lab2/bin/Debug/net7.0/lab2.dll /Launcher::ClassName:lab2.Lab2Task1 /Launcher::MethodName:RunTask1

document.xml

Successfully opened the XML file.

Successfully changed the XML file.

Successfully saved the XML file.

document\_new.xml

Successfully created the XML file.

Successfully changed the XML file.

Successfully saved the XML file.

Successfully changed the XML file.

Successfully saved the XML file.

Successfully opened the XML file.

Successfully changed the XML file.

Successfully saved the XML file.

document.doc

Successfully opened the TXT file.

Successfully changed the TXT file.

Successfully saved the TXT file.

document\_new.doc

Successfully created the TXT file.

Successfully changed the TXT file.

Successfully saved the TXT file.

Successfully changed the TXT file.

Successfully saved the TXT file.

Successfully opened the TXT file.

Successfully changed the TXT file.

Successfully saved the TXT file.

document.txt

Successfully opened the TXT file.

Successfully changed the TXT file.

Successfully saved the TXT file.

document\_new.txt

Successfully created the TXT file.

Successfully changed the TXT file.

Successfully saved the TXT file.

Successfully changed the TXT file.

Successfully saved the TXT file.

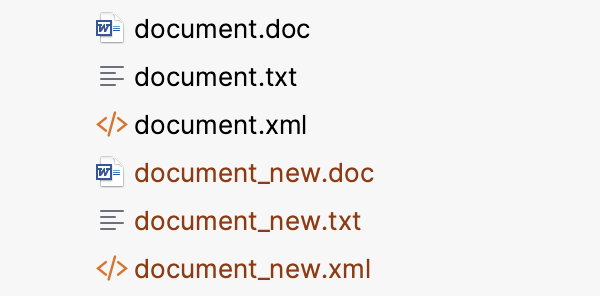
Successfully opened the TXT file.

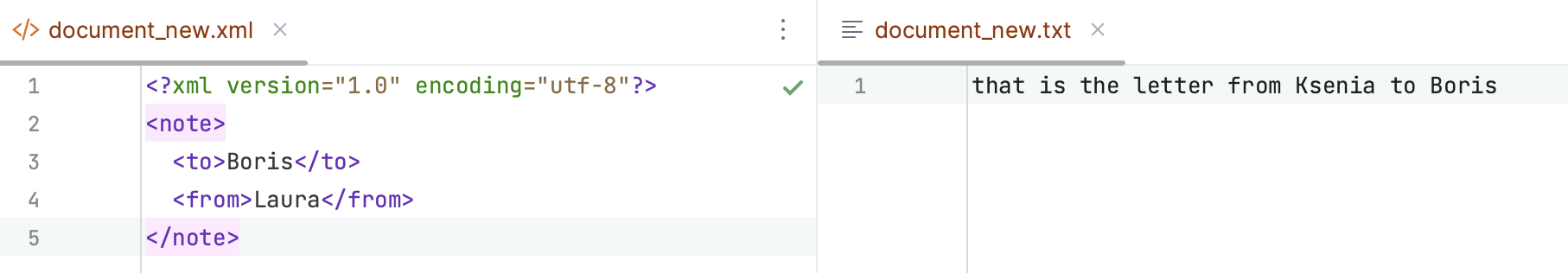
Successfully changed the TXT file.

Successfully saved the TXT file.

Process finished with exit code 0.

Було модифіковано попередньо додані файли та створено і модифіковано нові (виділено червоним).



******

***Завдання 2***

Використовуючи VisualStudio, створіть проект за шаблоном ConsoleApplication.

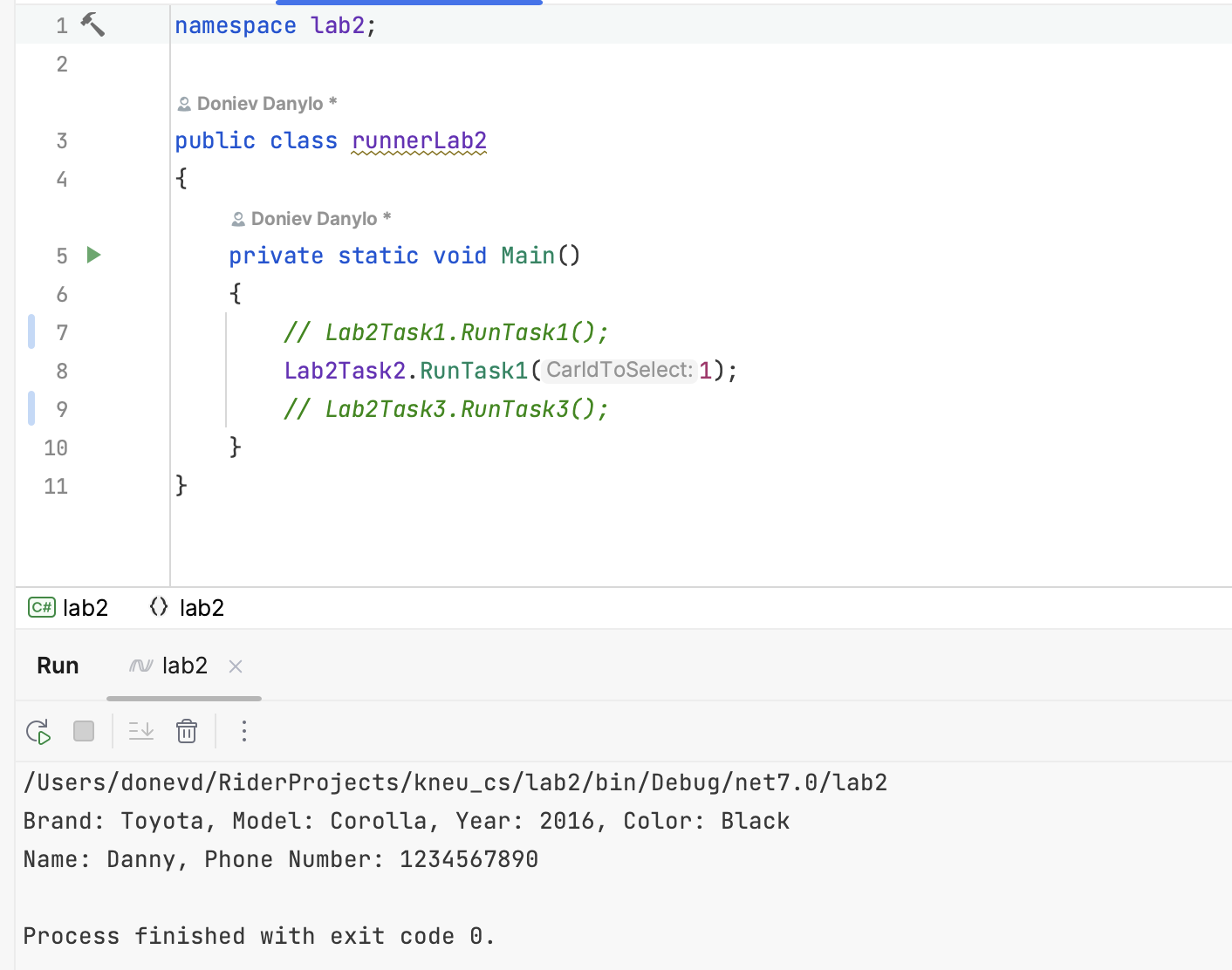
Створіть просту колекцію автомобілів з наступними даними: марка автівки, модель, рік випуску, колір. Створіть іншу колекцію з моделлю автівки, ім’ям покупця та номером його телефону. Використовуючи простий LINQ запит, вивести на екран інформацію про покупця однієї з автівок та повну характеристику придбаної ним моделі.

КОД ПРОГРАМИ

namespace lab2;  
  
public class Lab2Task2  
{  
 public class Car  
 {  
 public int Id { get; set; }  
 public string Brand { get; set; }  
 public string Model { get; set; }  
 public int Year { get; set; }  
 public string Color { get; set; }  
 }  
  
 public class Customer  
 {  
 public string CarModel { get; set; }  
 public int CarId { get; set; }  
 public string Name { get; set; }  
 public string PhoneNumber { get; set; }  
 }  
  
  
 public static void RunTask1(int CarIdToSelect)  
 {  
 List<Car> cars = new List<Car>  
 {  
 new Car {Id = 1, Brand = "Toyota", Model = "Corolla", Year = 2016, Color = "Black" },  
 new Car {Id = 2, Brand = "Mazda", Model = "CX-7", Year = 2016, Color = "Black" },  
 new Car {Id = 3, Brand = "BMW", Model = "M5", Year = 2016, Color = "Black" },  
 new Car {Id = 4, Brand = "BMW", Model = "X6", Year = 2016, Color = "Black" },  
 new Car {Id = 5, Brand = "Ford", Model = "Kuga", Year = 2016, Color = "Black" },  
 new Car {Id = 6, Brand = "VW", Model = "Transit", Year = 2016, Color = "Black" },  
  
 };  
  
 List<Customer> customers = new List<Customer>  
 {  
 new Customer { CarModel = "Corolla", CarId = 1, Name = "Danny", PhoneNumber = "1234567890" },  
 new Customer { CarModel = "BMW", CarId = 3,Name = "Boris", PhoneNumber = "1234567890" },  
 new Customer { CarModel = "Mazda",CarId = 2, Name = "Stepan", PhoneNumber = "1234567890" },  
 new Customer { CarModel = "VW",CarId = 6, Name = "Kelly", PhoneNumber = "1234567890" },  
  
 };  
  
 var carQuery = from car in cars  
 where car.Id == CarIdToSelect  
 select car;  
  
 var customerQuery = from customer in customers  
 where customer.CarId == CarIdToSelect  
 select customer;  
  
 foreach (var car in carQuery)  
 {  
 Console.WriteLine($"Brand: {car.Brand}, Model: {car.Model}, Year: {car.Year}, Color: {car.Color}");  
 }  
  
 foreach (var customer in customerQuery)  
 {  
 Console.WriteLine($"Name: {customer.Name}, Phone Number: {customer.PhoneNumber}");  
 }  
 }  
}

РЕЗУЛЬТАТИ

При передачі в якості параметра id автомобіля = 1, отримуємо інфо про авто і власника



***Індивідуальні завдання***

**Варіант 5**

Скласти опис класу для подання комплексних чисел. Забезпечити виконання операцій додавання, віднімання і множення комплексних чисел.

Написати програму, яка демонструвала б всі розроблені елементи класу.

КОД ПРОГРАМИ

namespace lab2;  
  
public class Lab2Task3  
{  
 public class ComplexNumber  
{  
 public double Real { get; set; }  
 public double Imaginary { get; set; }  
  
 public ComplexNumber(double real, double imaginary)  
 {  
 Real = real;  
 Imaginary = imaginary;  
 }  
  
 public ComplexNumber Add(ComplexNumber other)  
 {  
 return new ComplexNumber(Real + other.Real, Imaginary + other.Imaginary);  
 }  
   
 public ComplexNumber Subtract(ComplexNumber other)  
 {  
 return new ComplexNumber(Real - other.Real, Imaginary - other.Imaginary);  
 }  
   
 public ComplexNumber Multiply(ComplexNumber other)  
 {  
 double real = Real \* other.Real - Imaginary \* other.Imaginary;  
 double imaginary = Real \* other.Imaginary + Imaginary \* other.Real;  
 return new ComplexNumber(real, imaginary);  
 }  
  
 public override string ToString()  
 {  
 return $"{Real} + {Imaginary}i";  
 }  
}  
  
 public static void RunTask3()  
 {  
 ComplexNumber complex1 = new ComplexNumber(3, 2); *// 3 + 2i* ComplexNumber complex2 = new ComplexNumber(1, 7); *// 1 + 7i* Console.WriteLine($"Complex1: {complex1}");  
 Console.WriteLine($"Complex2: {complex2}");  
  
 ComplexNumber result = complex1.Add(complex2);  
 Console.WriteLine($"Add: {result}");  
  
 result = complex1.Subtract(complex2);  
 Console.WriteLine($"Subtract: {result}");  
  
 result = complex1.Multiply(complex2);  
 Console.WriteLine($"Multiply: {result}");  
 }  
}

РЕЗУЛЬТАТИ

