Кам’янець-Подільський національний університет імені Івана Огієнка

КАФЕДРА КОМП’ЮТЕРНИХ НАУК

Навчальна дисципліна «Об’єктно-орієнтоване програмування»

ЛАБОРАТОРНА РОБОТА #0113

Тема:

Патерни проектування у C#

Варіант №3

**Виконала**:  
студентка 1-го курсу  
групи KNms1-B24  
Некрасова Ю.Д.

**Прийняв**:  
Слободянюк О.В.

Кам’янець-Подільський – 2025

1. **Короткі теоретичні відомості**

**Основи патернів проєктування**

Патерни проєктування — це типові рішення для поширених проблем, що виникають під час розробки програмного забезпечення. Вони не є готовим кодом, а радше шаблонами, які можна адаптувати до конкретної ситуації.

Класифікація патернів:

* Креаційні (створюючі) — відповідають за створення об'єктів (наприклад, Singleton, Factory).
* Структурні — визначають, як класи та об'єкти поєднуються (наприклад, Adapter, Composite).
* Поведінкові — описують взаємодію між об'єктами (наприклад, Observer, Strategy).

**Singleton**

Забезпечує існування лише одного екземпляра класу та надає глобальну точку доступу до нього. Використовується для логерів, підключень до бази даних, конфігурацій.

**Factory Method**

Надає інтерфейс для створення об'єктів, але дозволяє підкласам вирішувати, який клас інстанціювати. Використовується, коли потрібно створювати об'єкти без прив'язки до конкретних класів.

**Observer**

Один об'єкт (Subject) повідомляє зареєстровані об'єкти (Observers) про зміну свого стану. Використовується в GUI (кнопки, події), новинних системах, чатах.

**Використання патернів у реальних проєктах**

Singleton − застосовується для сервісів, які повинні бути єдині в системі: налаштування, менеджери кешу.

Factory − широко використовується в системах, де створюється багато об'єктів різних типів, наприклад, в іграх (створення ворогів різних типів).

Observer − використовується у системах сповіщень, чатах, реалізації подій у UI (MVC/MVVM архітектури).

**Приклади реалізації патернів у C#**

Реалізація Singleton:

public class ConfigurationManager

{

private static ConfigurationManager \_instance;

private static readonly object \_lock = new object();

private ConfigurationManager() { }

public static ConfigurationManager Instance

{

get

{

lock (\_lock)

{

if (\_instance == null)

{

\_instance = new ConfigurationManager();

}

return \_instance;

}

}

}

public string GetConfiguration() => "Application Configuration";

}

class Program

{

static void Main()

{

var config = ConfigurationManager.Instance;

Console.WriteLine(config.GetConfiguration());

}

}

Реалізація Factory:

public interface ITransport

{

void Drive();

}

public class Car : ITransport

{

public void Drive() => Console.WriteLine("Driving a car.");

}

public class Motorcycle : ITransport

{

public void Drive() => Console.WriteLine("Riding a motorcycle.");

}

public class Bicycle : ITransport

{

public void Drive() => Console.WriteLine("Riding a bicycle.");

}

public class TransportFactory

{

public ITransport CreateTransport(string type)

{

return type.ToLower() switch

{

"car" => new Car(),

"motorcycle" => new Motorcycle(),

"bicycle" => new Bicycle(),

\_ => throw new ArgumentException("Unknown transport type.")

};

}

}

class Program

{

static void Main()

{

var factory = new TransportFactory();

ITransport car = factory.CreateTransport("car");

car.Drive();

ITransport bicycle = factory.CreateTransport("bicycle");

bicycle.Drive();

}

}

1. **Умова задачі 1 (Варіант 3).** Реалізація Singleton для логування

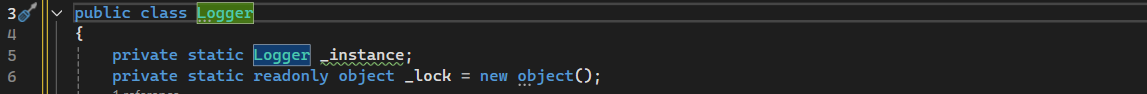
Створіть клас для логування, який реалізує Singleton.

1. **Послідовність виконання роботи**

Бібліотека:

****

Клас Logger із статичним полем \_instance:



Приватний конструктор із властивістю, яка дозволяє отримати єдиний екземпляр Logger:

Зображення, що містить знімок екрана, текст, Мультимедійне програмне забезпечення, програмне забезпечення

Вміст, створений ШІ, може бути неправильним.

Метод Log(), який приймає текстове повідомлення для логування:



Основний клас Program. Отримання єдиного екземпляру Logger:



Реалізація методу Log():



1. **Код програми**

using System;

public class Logger

{

private static Logger \_instance;

private static readonly object \_lock = new object();

private Logger() { }

public static Logger Instance

{

get

{

lock (\_lock)

{

if (\_instance == null)

{

\_instance = new Logger();

}

return \_instance;

}

}

}

public void Log(string message)

{

Console.WriteLine(message);

}

}

class Program

{

static void Main()

{

var logger = Logger.Instance;

logger.Log("Application started.");

logger.Log("This is a log message.");

}

}

1. **Приклад виконання програми**

**Зображення, що містить знімок екрана, програмне забезпечення, Мультимедійне програмне забезпечення, текст

Вміст, створений ШІ, може бути неправильним.**

1. **Умова задачі 2 (Варіант 3).** Використання Factory з додатковими параметрами

Реалізуйте Factory для створення транспортних засобів із зазначенням їх швидкості.

1. **Послідовність виконання роботи**

Бібліотека:



Інтерфейс ITransport з властивістю Speed. Метод Drive() для поїздки:

Зображення, що містить текст, Мультимедійне програмне забезпечення, знімок екрана

Вміст, створений ШІ, може бути неправильним.

Клас Car, що реалізує інтерфейс ITransport, із властивістю Speed:

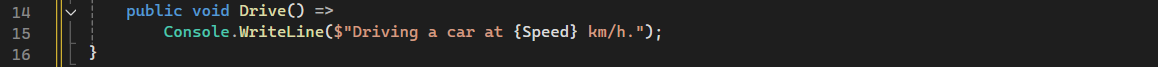
Зображення, що містить текст, Мультимедійне програмне забезпечення, знімок екрана

Вміст, створений ШІ, може бути неправильним.

Конструктор, що приймає значення швидкості:



Реалізація методу Drive():



Клас Motorcycle із аналогічною властивістю Speed та реалізацією методу Drive():

Зображення, що містить текст, знімок екрана, Мультимедійне програмне забезпечення, Графічний редактор

Вміст, створений ШІ, може бути неправильним.

Клас Bicycle із аналогічною властивістю Speed та реалізацією методу Drive():

Зображення, що містить текст, програмне забезпечення, Мультимедійне програмне забезпечення, Графічний редактор

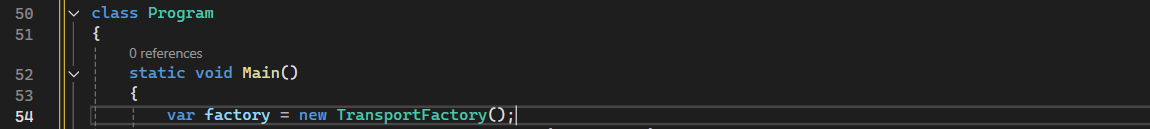
Вміст, створений ШІ, може бути неправильним.

Клас TransportFactory із методом CreateTransport(), що приймає тип транспорту і його швидкість, і створює відповідний об’єкт:

Зображення, що містить текст, знімок екрана

Вміст, створений ШІ, може бути неправильним.

Основний клас Program. Створення об’єкту фабрики:



Створення автомобілю, мотоциклу і велосипеду з різними швидкостями, вивід інформації про них в консоль за допомогою методу Drive():

Зображення, що містить текст, Мультимедійне програмне забезпечення, знімок екрана

Вміст, створений ШІ, може бути неправильним.

1. **Код програми**

using System;

public interface ITransport

{

int Speed { get; set; }

void Drive();

}

public class Car : ITransport

{

public int Speed { get; set;}

public Car(int speed)

{

Speed = speed;

}

public void Drive() =>

Console.WriteLine($"Driving a car at {Speed} km/h.");

}

public class Motorcycle : ITransport

{

public int Speed { get; set; }

public Motorcycle(int speed)

{

Speed = speed;

}

public void Drive() =>

Console.WriteLine($"Riding a motorcycle at {Speed} km/h.");

}

public class Bicycle : ITransport

{

public int Speed { get; set; }

public Bicycle(int speed)

{

Speed = speed;

}

public void Drive() =>

Console.WriteLine($"Riding a bicycle at {Speed} km/h.");

}

public class TransportFactory

{

public ITransport CreateTransport(string type, int speed)

{

return type.ToLower() switch

{

"car" => new Car(speed),

"motorcycle" => new Motorcycle(speed),

"bicycle" => new Bicycle(speed),

\_ => throw new ArgumentException("Unknown transport type.")

};

}

}

class Program

{

static void Main()

{

var factory = new TransportFactory();

ITransport car = factory.CreateTransport("car", 120);

car.Drive();

ITransport bicycle = factory.CreateTransport("bicycle", 25);

bicycle.Drive();

ITransport motorcycle = factory.CreateTransport("motorcycle", 90);

motorcycle.Drive();

}

}

1. **Приклад виконання програми**

Зображення, що містить текст, знімок екрана, програмне забезпечення, Мультимедійне програмне забезпечення

Вміст, створений ШІ, може бути неправильним.