Лабораторна робота №3 (максимально - 10 балів)

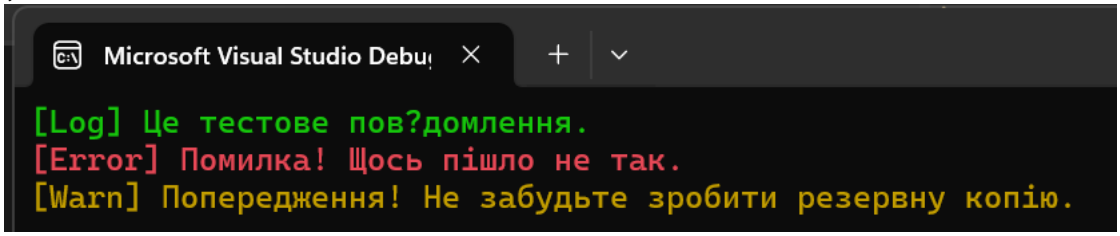
Тема: Структурні шаблони

Мета роботи: навчитися реалізовувати структурні шаблони проєктування Адаптер, Декоратор, Міст, Компонувальник, Проксі, Легковаговик

Завдання 1: Адаптер. 1. Створіть клас Logger, який буде мати методи Log(), Error(), Warn(), які виводять повідомлення в консоль різними кольорами (зеленим, червоним і оранжевим відповідно). 2. Створіть клас FileWriter з методами Write(), WriteLine(). 3. За допомогою шаблону Адаптер створіть файловий логер. 4. Покажіть правильність роботи свого коду запустивши його в головному методі програми

```
using System;

namespace Adapter
{
    // Клас для логування повідомлень у консоль
    public class Logger
    {
        public void Log(string message)
        {
            Console.ForegroundColor = ConsoleColor.Green;
            Console.WriteLine("[Log] " + message);
            Console.ResetColor();
        }

        public void Error(string message)
        {
            Console.ForegroundColor = ConsoleColor.Red;
            Console.WriteLine("[Error] " + message);
            Console.ResetColor();
        }

        public void Warn(string message)
        {
            Console.ForegroundColor = ConsoleColor.DarkYellow;
            Console.WriteLine("[Warn] " + message);
            Console.ResetColor();
        }
    }

    // Клас для запису в файл
    public class FileWriter
    {
        public void Write(string message)
        {
            System.IO.File.AppendAllText("log.txt", message);
        }

        public void WriteLine(string message)
        {
            System.IO.File.AppendAllText("log.txt", message + Environment.NewLine);
        }
    }
```

```
    // Адаптер для забезпечення сумісності з Logger
    public class FileLoggerAdapter : Logger
    {
        private FileWriter fileWriter;

        public FileLoggerAdapter()
        {
            fileWriter = new FileWriter();
        }

        public new void Log(string message)
        {
            fileWriter.WriteLine("[Log] " + message);
            // Виводимо у консоль
            base.Log(message);
        }

        public new void Error(string message)
        {
            fileWriter.WriteLine("[Error] " + message);
            // Виводимо також у консоль
            base.Error(message);
        }

        public new void Warn(string message)
        {
            fileWriter.WriteLine("[Warn] " + message);
            // Виводимо також у консоль
            base.Warn(message);
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            // Створення об'єкту адаптера
            FileLoggerAdapter fileLogger = new FileLoggerAdapter();

            // Тестування методів логування
            fileLogger.Log("Це тестове повідомлення.");
            fileLogger.Error("Помилка! Щось пішло не так.");
            fileLogger.Warn("Попередження! Не забудьте зробити резервну копію.");
        }
    }
}
```

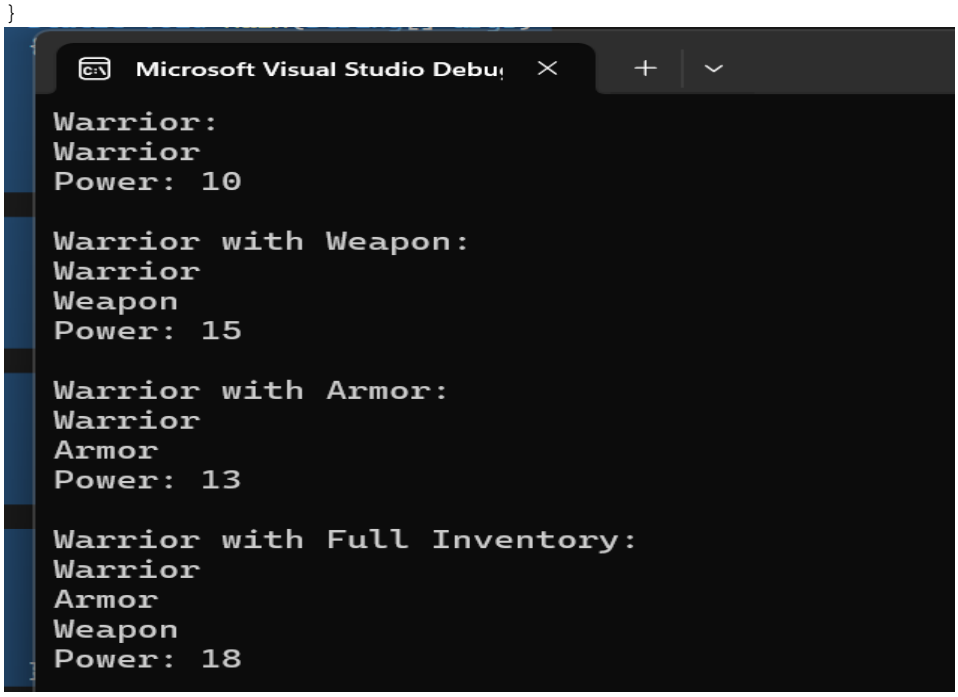


Завдання 2: Декоратор. 1. Ви розробляєте РПГ гру. Створіть класи героїв Warrior, Mage, Palladin. 2. Для героїв створіть інвентар (одяг, зброю, артефакти), який може підходити будь-якому типу героїв, у вигляді декораторів. 3. Важливою вимогою є можливість використання декількох екземплярів інвентаря на герої одночасно. 4. Покажіть правильність роботи свого коду запустивши його в головному методі програми.

```
using System;
using System.Collections.Generic;

namespace Decorator
{
    // Абстрактний клас героя
    public abstract class Hero
    {
        public abstract void Show();

        public abstract int GetPower();
    }
    // Конкретний герой - Воїн
    public class Warrior : Hero
    {
        public override void Show()
        {
            Console.WriteLine("Warrior");
        }

        public override int GetPower()
        {
            return 10;
        }
    }
    // Конкретний герой - Маг
    public class Mage : Hero
    {
        public override void Show()
        {
            Console.WriteLine("Mage");
        }

        public override int GetPower()
        {
            return 8;
        }
    }

    // Конкретний герой - Паладин
    public class Paladin : Hero
    {
        public override void Show()
        {
            Console.WriteLine("Paladin");
        }

        public override int GetPower()
        {
            return 12;
        }
    }

    // Декоратор для інвентарю
    public abstract class InventoryDecorator : Hero
    {
        protected Hero _hero;

        public InventoryDecorator(Hero hero)
        {
            _hero = hero;
        }
```

```
        public override void Show()
        {
            _hero.Show();
        }

        public override int GetPower()
        {
            return _hero.GetPower();
        }
    }
    // Конкретний декоратор - зброя
    public class WeaponDecorator : InventoryDecorator
    {
        public WeaponDecorator(Hero hero) : base(hero) { }

        public override void Show()
        {
            base.Show();
            Console.WriteLine("Weapon");
        }

        public override int GetPower()
        {
            return base.GetPower() + 5; // Покращення сили через зброю
        }
    }
    // Конкретний декоратор - броня
    public class ArmorDecorator : InventoryDecorator
    {
        public ArmorDecorator(Hero hero) : base(hero) { }

        public override void Show()
        {
            base.Show();
            Console.WriteLine("Armor");
        }

        public override int GetPower()
        {
            return base.GetPower() + 3; // Покращення сили через броню
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            // Створення героя Воїна
            Hero warrior = new Warrior();
            Console.WriteLine("Warrior:");
            warrior.Show();
            Console.WriteLine("Power: " + warrior.GetPower());

            // Додавання зброї до героя
            Hero warriorWithWeapon = new WeaponDecorator(warrior);
            Console.WriteLine("\nWarrior with Weapon:");
            warriorWithWeapon.Show();
            Console.WriteLine("Power: " + warriorWithWeapon.GetPower());

            // Додавання броні до героя
```

```
            Hero warriorWithArmor = new ArmorDecorator(warrior);
            Console.WriteLine("\nWarrior with Armor:");
            warriorWithArmor.Show();
            Console.WriteLine("Power: " + warriorWithArmor.GetPower());




            // Додавання інвентарю (броні та зброї) до героя
            Hero warriorWithFullInventory = new WeaponDecorator(new
ArmorDecorator(warrior));
            Console.WriteLine("\nWarrior with Full Inventory:");
            warriorWithFullInventory.Show();
            Console.WriteLine("Power: " + warriorWithFullInventory.GetPower());
        }
    }
}
```

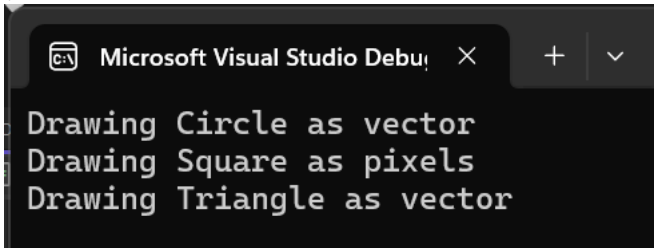


Завдання 3: Міст. 1. Ви працюєте над графічним редактором. Створіть базовий клас Shape. 2. Створіть дочірні до Shape класи, Circle, Square, Triangle. 3. За допомогою шаблону Міст додайте можливість рендерингу кожної з фігур як векторної або растрової графіки (вивівши відповідне повідомлення у консоль, наприклад "Drawing Triangle as pixels"). 4. Покажіть правильність роботи свого коду запустивши його в головному методі програми.

```
using System;

namespace Bridge
{
    // Абстрактний клас фігури
    public abstract class Shape
    {
        // Посилання на об'єкт інтерфейсу, що відповідає за рендеринг
        protected IRenderer renderer;

        // Конструктор, який приймає об'єкт інтерфейсу для рендерингу
        public Shape(IRenderer renderer)
        {
            this.renderer = renderer;
        }
```

```
    // Метод для відображення фігури
    public abstract void Draw();
}

// Конкретна фігура - Коло
public class Circle : Shape
{
    public Circle(IRenderer renderer) : base(renderer) { }

    public override void Draw()
    {
        Console.WriteLine("Drawing Circle " + renderer.Render());
    }
}

// Конкретна фігура - Квадрат
public class Square : Shape
{
    public Square(IRenderer renderer) : base(renderer) { }

    public override void Draw()
    {
        Console.WriteLine("Drawing Square " + renderer.Render());
    }
}

// Конкретна фігура - Трикутник
public class Triangle : Shape
{
    public Triangle(IRenderer renderer) : base(renderer) { }

    public override void Draw()
    {
        Console.WriteLine("Drawing Triangle " + renderer.Render());
    }
}

// Інтерфейс для рендерингу
public interface IRenderer
{
    string Render();
}

// Конкретний клас для векторного рендерингу
public class VectorRenderer : IRenderer
{
    public string Render()
    {
        return "as vector";
    }
}

// Конкретний клас для растрового рендерингу
public class RasterRenderer : IRenderer
{
    public string Render()
    {
        return "as pixels";
```

```
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            // Створення об'єктів для рендерингу
            IRenderer vectorRenderer = new VectorRenderer();
            IRenderer rasterRenderer = new RasterRenderer();




            // Створення фігур та виклик методу Draw() для кожної фігури з різним
рендерером
            Shape circle = new Circle(vectorRenderer);
            circle.Draw();

            Shape square = new Square(rasterRenderer);
            square.Draw();

            Shape triangle = new Triangle(vectorRenderer);
            triangle.Draw();
        }
    }
}
```

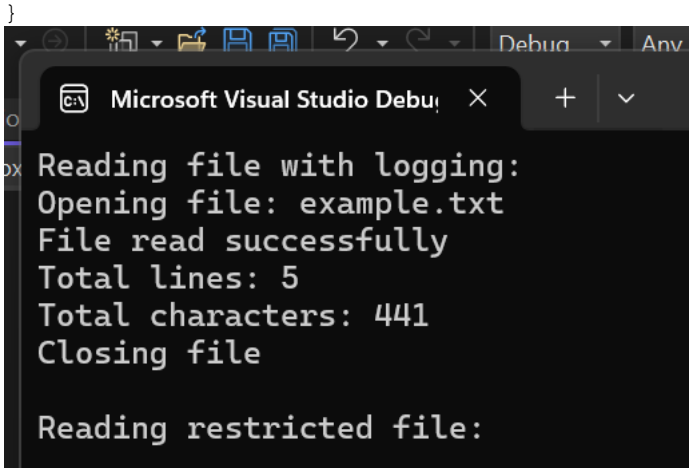


Завдання 4: Проксі. 1. Створіть клас SmartTextReader, який вміє читати вміст текстового файлу і перетворювати його на двомірний масив якому зовнішній масив відповідає рядкам тексту, а вкладені масиви відповідають символам у відповідному рядку. 2. Створіть проксі для SmartTextReader з логуванням SmartTextChecker, який буде виводити інформацію про успішне відкриття, прочитання і закриття файлу, а також буде виводити загальну кількість рядків і символів у прочитаному тексті. 3. Створіть проксі для SmartTextReader з обмеженням доступу до певних файлів SmartTextReaderLocker. Цей клас в конструкторі приймає регулярний вираз, по якому лімітується доступ до певної групи файлів. Якщо клієнт викликатиме метод для прочитання такого лімітованого файлу, замість прочитання файлу в консоль має виводитися повідомлення “Access denied!”. 4. Покажіть правильність роботи свого коду запустивши його в головному методі програми.

```
using System;
using System.IO;
using System.Text.RegularExpressions;

namespace Proxy
{
    // Абстрактний клас читача тексту
    public abstract class TextReader
    {
        public abstract char[][] ReadText(string filePath);
    }

    // Конкретний клас читача тексту
    public class SmartTextReader : TextReader
    {
```

```
    public override char[][] ReadText(string filePath)
    {
        string[] lines = File.ReadAllLines(filePath);
        char[][] textArray = new char[lines.Length][];

        for (int i = 0; i < lines.Length; i++)
        {
            textArray[i] = lines[i].ToCharArray();
        }

        return textArray;
    }
}



// Проксі для SmartTextReader з логуванням
public class SmartTextChecker : TextReader
{
    private TextReader reader;

    public SmartTextChecker(TextReader reader)
    {
        this.reader = reader;
    }

    public override char[][] ReadText(string filePath)
    {
        Console.WriteLine("Opening file: " + filePath);
        char[][] textArray = reader.ReadText(filePath);
        Console.WriteLine("File read successfully");
        Console.WriteLine($"Total lines: {textArray.Length}");
        int totalChars = 0;
        foreach (var line in textArray)
        {
            totalChars += line.Length;
        }
        Console.WriteLine($"Total characters: {totalChars}");
        Console.WriteLine("Closing file");

        return textArray;
    }
}

// Проксі для SmartTextReader з обмеженням доступу до певних файлів
public class SmartTextReaderLocker : TextReader
{
    private TextReader reader;
    private Regex allowedFilesRegex;

    public SmartTextReaderLocker(TextReader reader, string allowedFilesPattern)
    {
        this.reader = reader;
        this.allowedFilesRegex = new Regex(allowedFilesPattern);
    }

    public override char[][] ReadText(string filePath)
    {
        if (!allowedFilesRegex.IsMatch(filePath))
        {
            Console.WriteLine("Access denied!");
            return null;
        }
        else
        {
            return reader.ReadText(filePath);
```

```
            }
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            // Створення об'єкта SmartTextReader
            TextReader smartTextReader = new SmartTextReader();

            // Створення проксі з логуванням
            TextReader smartTextChecker = new SmartTextChecker(smartTextReader);

            // Читання та виведення інформації про текстовий файл з логуванням
            Console.WriteLine("Reading file with logging:");
            smartTextChecker.ReadText("example.txt");



            // Створення проксі з обмеженням доступу до певних файлів
            TextReader smartTextReaderLocker = new
SmartTextReaderLocker(smartTextReader, @".*\.txt");

            // Читання лімітованого файлу
            Console.WriteLine("\nReading restricted file:");
            smartTextReaderLocker.ReadText("restricted.txt");
        }
    }
}
```



Завдання 5: Компонувальник. 1. Вам потрібно створити власну мову розмітки LightHTML. 2. Кожен елемент розмітки має наслідувати клас LightNode. 3. Створіть два дочірніх класи від LightNode: LightElementNode, LightTextNode. 4. LightTextNode може містити лише текст. 5. LightElementNode може містити будь-які LightNode. LightElementNode повинен мати інформацію про назву тега, його тип відображення (блочний чи рядковий), тип закриття (одиничний тег, як **Ошибка! Не указано имя файла.** чи з закриваючим тегом) список CSS класів, кількість дочірніх елементів, а також має бути можливість виводити на екран його outerHTML і innerHTML. 6. За допомогою своєї мови розмітки виведіть в консоль елемент сторінки на Ваш вибір (наприклад якусь таблицю, список тощо). 7. Покажіть правильність роботи свого коду запустивши його в головному методі програми.

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

namespace Composer
{
    // Абстрактний клас вузла розмітки
    public abstract class LightNode
    {
        public abstract string OuterHTML { get; }
```

```
        public abstract string InnerHTML { get; }
    }

    // Клас для текстового вузла розмітки
    public class LightTextNode : LightNode
    {
        private string text;

        public LightTextNode(string text)
        {
            this.text = text;
        }

        public override string OuterHTML => text;
        public override string InnerHTML => text;
    }




    // Перелік типів відображення елементу
    public enum DisplayType
    {
        Block,
        Inline
    }

    // Перелік типів закриття елементу
    public enum ClosingType
    {
        SingleTag,
        ClosingTag
    }

    // Клас для елемента розмітки
    public class LightElementNode : LightNode
    {
        public string TagName { get; }
        public DisplayType Display { get; }
        public ClosingType Closing { get; }
        public List<string> Classes { get; }
        public List<LightNode> Children { get; }

        public LightElementNode(string tagName, DisplayType display, ClosingType
closing, List<string> classes, List<LightNode> children)
        {
            TagName = tagName;
            Display = display;
            Closing = closing;
            Classes = classes;
            Children = children;
        }

        public override string OuterHTML
        {
            get
            {
                StringBuilder sb = new StringBuilder();
                sb.Append($"<{TagName}");

                foreach (var cls in Classes)
                {
                    sb.Append($" class=\"{cls}\"");
                }
```

```
            if (Closing == ClosingType.SingleTag)
            {
                sb.Append(" />");
            }
            else
            {
                sb.Append(">");

                foreach (var child in Children)
                {
                    sb.Append(child.OuterHTML);
                }

                sb.Append($"</{TagName}>");
            }

            return sb.ToString();
        }
    }




    public override string InnerHTML
    {
        get
        {
            StringBuilder sb = new StringBuilder();

            foreach (var child in Children)
            {
                sb.Append(child.OuterHTML);
            }

            return sb.ToString();
        }
    }
}

class Program
{
    static void Main(string[] args)
    {
        // Створення елементів розмітки
        var paragraph = new LightElementNode("p", DisplayType.Block,
ClosingType.ClosingTag, new List<string> { "paragraph" }, new List<LightNode>
        {
            new LightTextNode("Це перший абзац."),
            new LightTextNode("Це другий абзац.")
        });

        var listItems = new List<LightNode>
        {
            new LightTextNode("Пункт 1"),
            new LightTextNode("Пункт 2"),
            new LightTextNode("Пункт 3")
        };

        var unorderedList = new LightElementNode("ul", DisplayType.Block,
ClosingType.ClosingTag, new List<string> { "list" }, listItems);

        // Виведення елементів розмітки
        Console.WriteLine("Зовнішній HTML:");
        Console.WriteLine(paragraph.OuterHTML);
```

```
            Console.WriteLine("\nВнутрішній HTML:");
            Console.WriteLine(paragraph.InnerHTML);

            Console.WriteLine("\nСписок:");
            Console.WriteLine(unorderedList.OuterHTML);
        }
    }
}
```