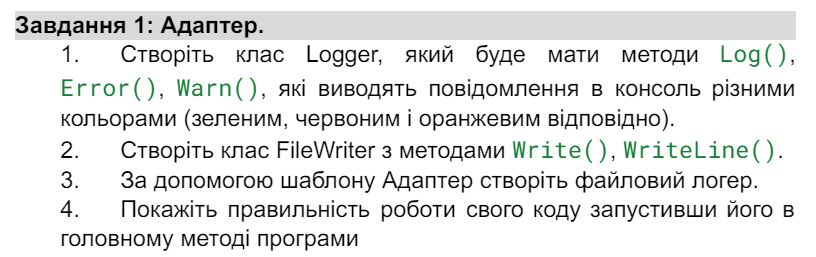
**Лабораторна робота №3**

**Тема: Структурні шаблони**

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

**Хід Роботи**

**Репозиторій:** <https://github.com/Oleksandr-Nagal/KPZ>

****

**Код:**

using System;

public interface ILogger

{

void Log(string message);

void Error(string message);

void Warn(string message);

}

public class Logger : ILogger

{

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.Yellow;

Console.WriteLine($"Warning: {message}");

Console.ResetColor();

}

}

public interface IFileWriter

{

void Write(string text);

void WriteLine(string text);

}

public class FileWriter : IFileWriter

{

public void Write(string text)

{

Console.WriteLine($"Writing to file: {text}");

}

public void WriteLine(string text)

{

Console.WriteLine($"Writing line to file: {text}");

}

}

public class FileLoggerAdapter : ILogger

{

private readonly IFileWriter \_fileWriter;

public FileLoggerAdapter(IFileWriter fileWriter)

{

\_fileWriter = fileWriter;

}

public void Log(string message)

{

\_fileWriter.WriteLine($"Log: {message}");

}

public void Error(string message)

{

\_fileWriter.WriteLine($"Error: {message}");

}

public void Warn(string message)

{

\_fileWriter.WriteLine($"Warning: {message}");

}

}

class Program

{

static void Main(string[] args)

{

ILogger logger = new Logger();

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

logger.Error("This is an error message");

logger.Warn("This is a warning message");

Console.WriteLine();

IFileWriter fileWriter = new FileWriter();

ILogger fileLogger = new FileLoggerAdapter(fileWriter);

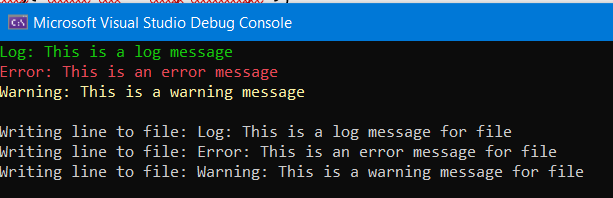
fileLogger.Log("This is a log message for file");

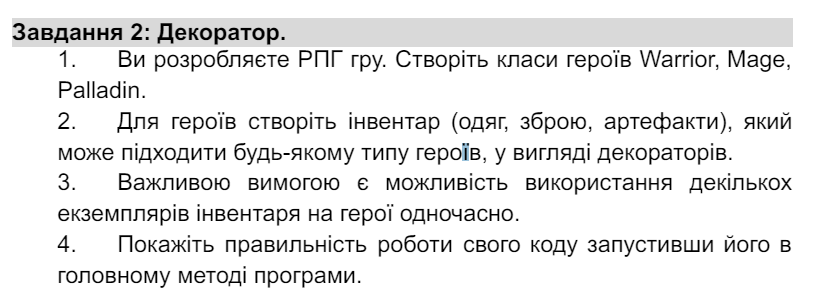
fileLogger.Error("This is an error message for file");

fileLogger.Warn("This is a warning message for file");

}

}

**Результат:** ****

**Код:**

using System;

using System.Collections.Generic;

public interface IInventory

{

void Equip();

}

public abstract class Hero

{

protected string Name;

protected List<IInventory> Inventories = new List<IInventory>();

public Hero(string name)

{

Name = name;

}

public void AddInventory(IInventory inventory)

{

Inventories.Add(inventory);

}

public virtual void ShowInventories()

{

Console.WriteLine($"Inventories for {Name}:");

foreach (var inventory in Inventories)

{

inventory.Equip();

}

}

}

// Клас Warrior

public class Warrior : Hero

{

public Warrior(string name) : base(name)

{

}

public override void ShowInventories()

{

Console.WriteLine("\nWarrior Inventories:");

base.ShowInventories();

}

}

// Клас Mage

public class Mage : Hero

{

public Mage(string name) : base(name)

{

}

public override void ShowInventories()

{

Console.WriteLine("\nMage Inventories:");

base.ShowInventories();

}

}

// Клас Palladin

public class Palladin : Hero

{

public Palladin(string name) : base(name)

{

}

public override void ShowInventories()

{

Console.WriteLine("\nPalladin Inventories:");

base.ShowInventories();

}

}

public abstract class InventoryDecorator : IInventory

{

protected IInventory Inventory;

protected InventoryDecorator(IInventory inventory)

{

Inventory = inventory;

}

public virtual void Equip()

{

Inventory?.Equip();

}

}

public class WeaponDecorator : InventoryDecorator

{

private readonly string \_weaponName;

public WeaponDecorator(IInventory inventory, string weaponName) : base(inventory)

{

\_weaponName = weaponName;

}

public override void Equip()

{

base.Equip();

Console.WriteLine($"Weapon: {\_weaponName}");

}

}

public class ArmorDecorator : InventoryDecorator

{

private readonly string \_armorName;

public ArmorDecorator(IInventory inventory, string armorName) : base(inventory)

{

\_armorName = armorName;

}

public override void Equip()

{

base.Equip();

Console.WriteLine($"Armor: {\_armorName}");

}

}

class Program

{

static void Main(string[] args)

{

Warrior warrior = new Warrior("Warrior1");

Mage mage = new Mage("Mage1");

Palladin palladin = new Palladin("Palladin1");

warrior.AddInventory(new WeaponDecorator(null, "Sword"));

warrior.AddInventory(new ArmorDecorator(null, "Chainmail"));

warrior.AddInventory(new WeaponDecorator(new ArmorDecorator(null, "Leather Armor"), "Axe"));

mage.AddInventory(new WeaponDecorator(null, "Staff"));

mage.AddInventory(new ArmorDecorator(null, "Robe"));

mage.AddInventory(new WeaponDecorator(new ArmorDecorator(null, "Cloth Armor"), "Wand"));

palladin.AddInventory(new WeaponDecorator(null, "Mace"));

palladin.AddInventory(new ArmorDecorator(null, "Plate Armor"));

palladin.AddInventory(new WeaponDecorator(new ArmorDecorator(null, "Plate Armor"), "Shield"));

warrior.ShowInventories();

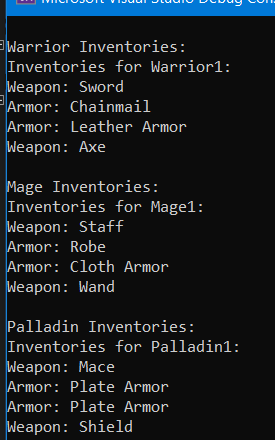
mage.ShowInventories();

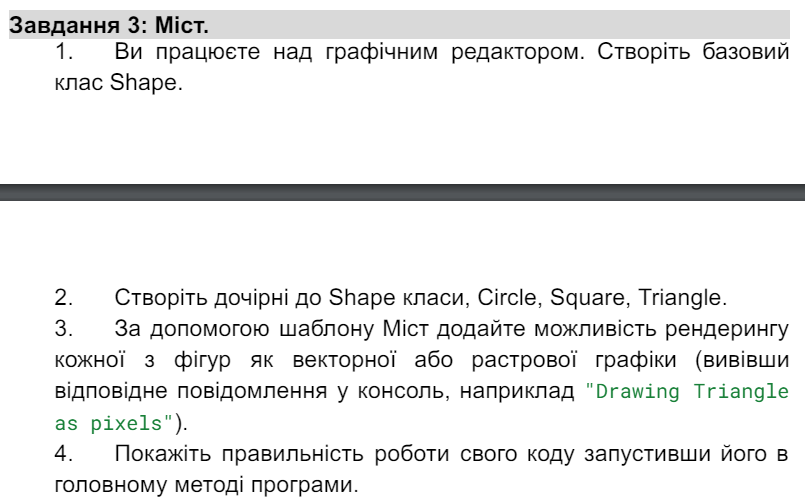
palladin.ShowInventories();

}

}

**Результат:**

****

****

**Код:**

using System;

public abstract class Shape

{

protected IRenderer Renderer;

protected Shape(IRenderer renderer)

{

Renderer = renderer;

}

public abstract void Draw();

}

public class Circle : Shape

{

public Circle(IRenderer renderer) : base(renderer)

{

}

public override void Draw()

{

Renderer.RenderCircle();

}

}

public class Square : Shape

{

public Square(IRenderer renderer) : base(renderer)

{

}

public override void Draw()

{

Renderer.RenderSquare();

}

}

public class Triangle : Shape

{

public Triangle(IRenderer renderer) : base(renderer)

{

}

public override void Draw()

{

Renderer.RenderTriangle();

}

}

public interface IRenderer

{

void RenderCircle();

void RenderSquare();

void RenderTriangle();

}

public class RasterRenderer : IRenderer

{

public void RenderCircle()

{

Console.WriteLine("Drawing Circle as pixels");

}

public void RenderSquare()

{

Console.WriteLine("Drawing Square as pixels");

}

public void RenderTriangle()

{

Console.WriteLine("Drawing Triangle as pixels");

}

}

public class VectorRenderer : IRenderer

{

public void RenderCircle()

{

Console.WriteLine("Drawing Circle as vectors");

}

public void RenderSquare()

{

Console.WriteLine("Drawing Square as vectors");

}

public void RenderTriangle()

{

Console.WriteLine("Drawing Triangle as vectors");

}

}

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Rendering shapes in raster format:");

var rasterCircle = new Circle(new RasterRenderer());

rasterCircle.Draw();

var rasterSquare = new Square(new RasterRenderer());

rasterSquare.Draw();

var rasterTriangle = new Triangle(new RasterRenderer());

rasterTriangle.Draw();

Console.WriteLine("\nRendering shapes in vector format:");

var vectorCircle = new Circle(new VectorRenderer());

vectorCircle.Draw();

var vectorSquare = new Square(new VectorRenderer());

vectorSquare.Draw();

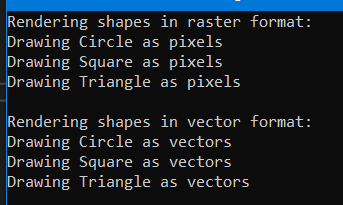
var vectorTriangle = new Triangle(new VectorRenderer());

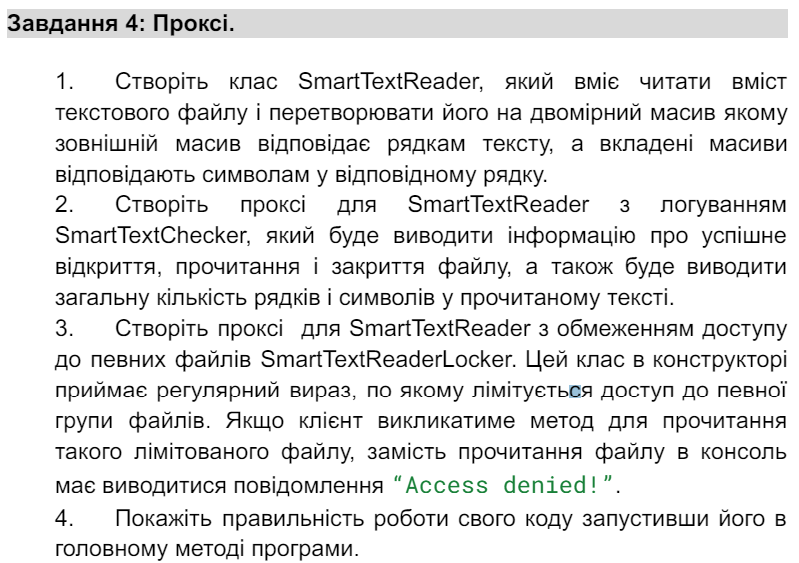
vectorTriangle.Draw();

}

}

**Результат:**

****

**Код:** using System;

using System.IO;

using System.Text.RegularExpressions;

public interface ITextReader

{

string[,] ReadTextFile(string filePath);

}

public class SmartTextReader : ITextReader

{

public string[,] ReadTextFile(string filePath)

{

string[] lines = File.ReadAllLines(filePath);

int maxLength = lines.Length > 0 ? lines[0].Length : 0;

string[,] result = new string[lines.Length, maxLength];

for (int i = 0; i < lines.Length; i++)

{

for (int j = 0; j < lines[i].Length; j++)

{

result[i, j] = lines[i][j].ToString();

}

}

return result;

}

}

public class SmartTextChecker : ITextReader

{

private readonly ITextReader \_reader;

public SmartTextChecker(ITextReader reader)

{

\_reader = reader;

}

public string[,] ReadTextFile(string filePath)

{

Console.WriteLine($"Opening file: {filePath}");

string[,] result = \_reader.ReadTextFile(filePath);

Console.WriteLine($"Successfully read file: {filePath}");

Console.WriteLine($"Number of lines: {result.GetLength(0)}");

Console.WriteLine($"Number of characters: {result.Length}");

Console.WriteLine($"Closing file: {filePath}");

return result;

}

}

public class SmartTextReaderLocker : ITextReader

{

private readonly ITextReader \_reader;

private readonly Regex \_regex;

public SmartTextReaderLocker(ITextReader reader, string pattern)

{

\_reader = reader;

\_regex = new Regex(pattern);

}

public string[,] ReadTextFile(string filePath)

{

if (\_regex.IsMatch(filePath))

{

Console.WriteLine("Access denied!");

return null;

}

return \_reader.ReadTextFile(filePath);

}

}

class Program

{

static void Main(string[] args)

{

ITextReader textReader = new SmartTextReader();

ITextReader smartReaderWithLogging = new SmartTextChecker(textReader);

string[,] fileContent = smartReaderWithLogging.ReadTextFile("example.txt");

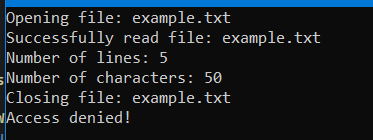
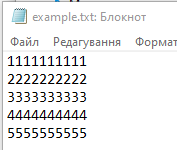
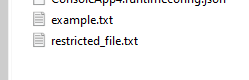
ITextReader smartReaderWithLock = new SmartTextReaderLocker(textReader, "restricted.\*");

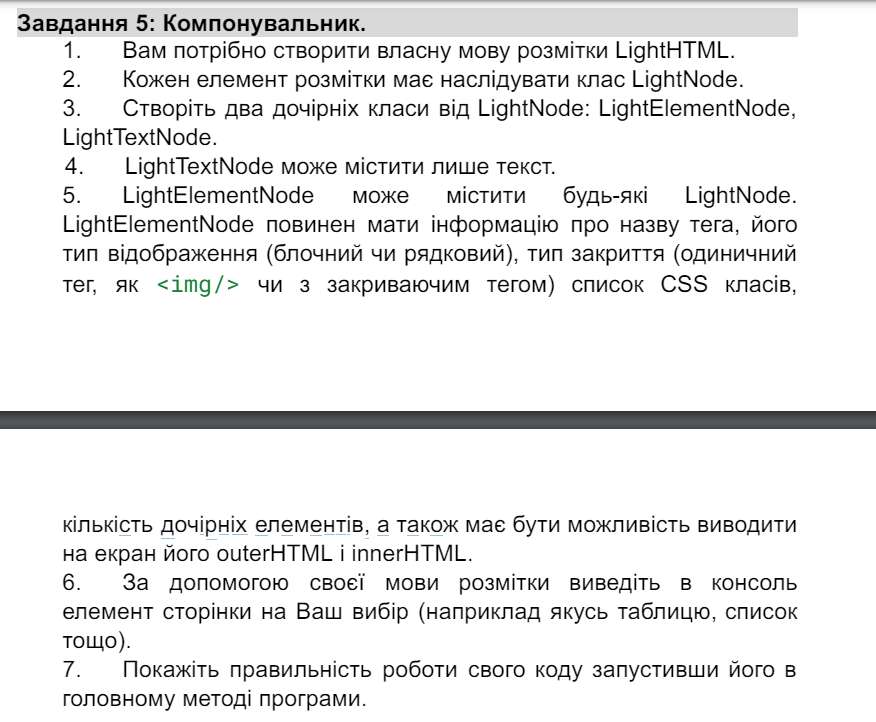
string[,] restrictedContent = smartReaderWithLock.ReadTextFile("restricted\_file.txt");

}

}

**Результат:**

**** **** 

**Код:**

using System;

using System.Collections.Generic;

using System.Text;

public abstract class LightNode

{

public abstract string OuterHTML { get; }

public abstract string InnerHTML { get; }

}

public class LightTextNode : LightNode

{

private readonly string \_text;

public LightTextNode(string text)

{

\_text = text;

}

public override string OuterHTML => \_text;

public override string InnerHTML => \_text;

}

public class LightElementNode : LightNode

{

private readonly string \_tag;

private readonly bool \_blockType;

private readonly bool \_selfClosing;

private readonly List<string> \_classes;

private readonly List<LightNode> \_children;

public LightElementNode(string tag, bool blockType, bool selfClosing, List<string> classes, List<LightNode> children)

{

\_tag = tag;

\_blockType = blockType;

\_selfClosing = selfClosing;

\_classes = classes;

\_children = children;

}

public override string OuterHTML

{

get

{

StringBuilder builder = new StringBuilder();

builder.Append($"<{\_tag}");

if (\_classes.Count > 0)

{

builder.Append(" class=\"");

builder.Append(string.Join(" ", \_classes));

builder.Append("\"");

}

builder.Append(">");

if (!\_selfClosing)

{

foreach (var child in \_children)

{

builder.Append(child.OuterHTML);

}

builder.Append($"</{\_tag}>");

}

return builder.ToString();

}

}

public override string InnerHTML

{

get

{

StringBuilder builder = new StringBuilder();

foreach (var child in \_children)

{

builder.Append(child.OuterHTML);

}

return builder.ToString();

}

}

}

class Program

{

static void Main(string[] args)

{

var title = new LightTextNode("Welcome to My Website");

var paragraph1 = new LightTextNode("This is a simple paragraph.");

var paragraph2 = new LightTextNode("This is another paragraph.");

var listItems = new List<LightNode>

{

new LightTextNode("Item 1"),

new LightTextNode("Item 2"),

new LightTextNode("Item 3")

};

var unorderedList = new LightElementNode("ul", true, false, new List<string>(), listItems);

var bodyChildren = new List<LightNode> { paragraph1, paragraph2, unorderedList };

var body = new LightElementNode("body", true, false, new List<string>(), bodyChildren);

var htmlChildren = new List<LightNode> { title, body };

var html = new LightElementNode("html", true, false, new List<string>(), htmlChildren);

Console.WriteLine("Inner HTML of the page:");

Console.WriteLine(html.InnerHTML);

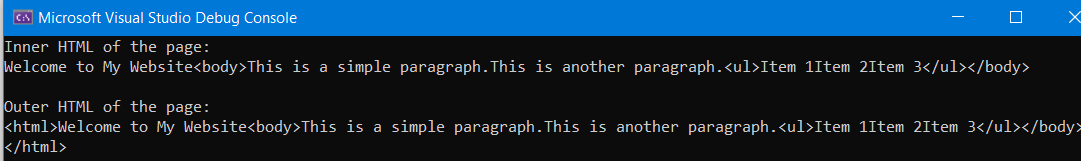
Console.WriteLine();

Console.WriteLine("Outer HTML of the page:");

Console.WriteLine(html.OuterHTML);

}

}

**Результат:** ****

**Код:** using System;

using System.Collections.Generic;

using System.Text;

public abstract class LightNode

{

public abstract string OuterHTML { get; }

public abstract string InnerHTML { get; }

}

public class LightTextNode : LightNode

{

private readonly string \_text;

public LightTextNode(string text)

{

\_text = text;

}

public override string OuterHTML => \_text;

public override string InnerHTML => \_text;

}

public class LightElementNode : LightNode

{

private readonly string \_tag;

private readonly bool \_blockType;

private readonly bool \_selfClosing;

private readonly List<string> \_classes;

private readonly List<LightNode> \_children;

public LightElementNode(string tag, bool blockType, bool selfClosing, List<string> classes, List<LightNode> children)

{

\_tag = tag;

\_blockType = blockType;

\_selfClosing = selfClosing;

\_classes = classes;

\_children = children;

}

public override string OuterHTML

{

get

{

StringBuilder builder = new StringBuilder();

builder.Append($"<{\_tag}");

if (\_classes.Count > 0)

{

builder.Append(" class=\"");

builder.Append(string.Join(" ", \_classes));

builder.Append("\"");

}

builder.Append(">");

if (!\_selfClosing)

{

foreach (var child in \_children)

{

builder.Append(child.OuterHTML);

}

builder.Append($"</{\_tag}>");

}

return builder.ToString();

}

}

public override string InnerHTML

{

get

{

StringBuilder builder = new StringBuilder();

foreach (var child in \_children)

{

builder.Append(child.OuterHTML);

}

return builder.ToString();

}

}

}

public class LightWeightNode : LightNode

{

private readonly string \_content;

public LightWeightNode(string content)

{

\_content = content;

}

public override string OuterHTML => \_content;

public override string InnerHTML => \_content;

}

class Program

{

static void Main(string[] args)

{

var h1 = new LightWeightNode("<h1>Chapter 1: Introduction</h1>");

var h2 = new LightWeightNode("<h2>Section 1.1: Overview</h2>");

var p1 = new LightWeightNode("<p>This is the first paragraph of the introduction.</p>");

var p2 = new LightWeightNode("<p>This is the second paragraph of the introduction.</p>");

var blockquote = new LightWeightNode("<blockquote>This is a blockquote.</blockquote>");

Console.WriteLine("Outer HTML:");

Console.WriteLine(h1.OuterHTML);

Console.WriteLine(h2.OuterHTML);

Console.WriteLine(p1.OuterHTML);

Console.WriteLine(p2.OuterHTML);

Console.WriteLine(blockquote.OuterHTML);

Console.WriteLine();

Console.WriteLine("Inner HTML:");

Console.WriteLine(h1.InnerHTML);

Console.WriteLine(h2.InnerHTML);

Console.WriteLine(p1.InnerHTML);

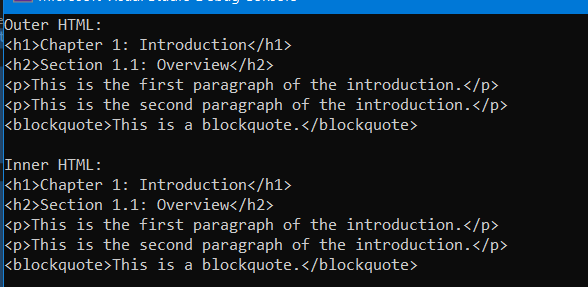
Console.WriteLine(p2.InnerHTML);

Console.WriteLine(blockquote.InnerHTML);

}

}

**Результат:**

****

**Висновок:** У ході виконання лабораторної роботи я навчився реалізовувати структурні шаблони проєктування Адаптер, Декоратор, Міст, Компонувальник, Проксі, Легковаговик.