Лабораторна робота №7

Тема: Поведінкові шаблони

Мета роботи: навчитися реалізовувати структурні шоблони проєктування ***Спостерігач, Стратегія, Ітератор, Команда, Мементо, Стейт, Темплейт, Відвідувач***

Завдання на лабораторну роботу

Завдання 0: Підготовка до виконання завдання

1. Знайти репозиторій з завданням №1 ЛР5
2. Перейти до Завдання 1 🙂

Завдання 1: Створіть HTML своєї мрії 🙂

В межах ЛР №5 Ви почали створювати власний HTML. В цій роботі Ви маєте можливість розширити можливості власної мови розмітки.

В якості ідей для нових фіч Ви можете користуватися своєю багатою фантазією і безмежною уявою, а направити Вас на вірний шлях можуть допомгти наступні приклади можливих фіч:

- за допомогою шаблону Спостерігач реалізуйте можливість додавання EventListener до Ваших HTML елементів;

Лістинг програми:

(EventManager)

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

namespace lab_5.LightHTML.Observer
{
    public enum EventType
    {
        OnClick,
        OnFocus,
    }
    public class EventManager
    {
        public List<KeyValuePair<EventType, EventHandler<NodeEventArgs>>>
listeners = new();
```

```
        public LightNode Node { get; set; }
        public EventManager(LightNode node)
        {
            Node = node;
        }

        public void Subscribe(EventType eventType, EventHandler<NodeEventArgs>
listener)
        {
            listeners.Add(new KeyValuePair<EventType,
EventHandler<NodeEventArgs>>(eventType, listener));
        }
        public void Unsubscribe(EventHandler<NodeEventArgs> listener)
        {
            foreach (var item in listeners)
            {
                if (item.Value == listener)
                {
                    listeners.Remove(item);
                }
            }
        }

        public void Notify(EventType eventType)
        {
            foreach (var item in listeners)
            {
                if (item.Key == eventType)
                {
                    item.Value.Invoke(this, new NodeEventArgs(Node));
                }
            }
        }
    }
}
```

(nodeEventArgs)

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

namespace lab_5.LightHTML.Observer
{
    public class NodeEventArgs : EventArgs
    {
        public LightNode Node { get; private set; }
        public NodeEventArgs(LightNode node)
        {
            Node = node;
        }
    }
}
```

(IEventListener)

```
namespace lab_5.LightHTML.Observer
{
    public interface IEventListener
    {
        public void Update(LightNode node);
    }
}
```

(OnClickListener)

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

namespace lab_5.LightHTML.Observer
{
    public class OnClickListener : IEventListener
    {
        public event EventHandler<NodeEventArgs> Handler;
        public OnClickListener(EventHandler<NodeEventArgs> eventHandler)
        {
            Handler = eventHandler;
        }
        public void Update(LightNode node)
        {
            Handler.Invoke(this, new NodeEventArgs(node));
        }
    }
}
```

(Program.cs)

```
using DesignPatterns.Composite;
using lab_5;
using lab_5.Flyweight;
using lab_5.LightHTML;
using lab_5.LightHTML.Observer;
using lab_5.MarvelHero;
using lab_5.MarvelHero.Logger;
using lab_5.TextReader;
using System.Text.RegularExpressions;

Console.WriteLine("1. Composite. LightHTML");
var ul = new LightElementNode("ul", NodeType.ElementNode, ClosureType.Even, new()
{ "class1", "class2" });
var li0 = new LightTextNode("li", "li0");
var li1 = new LightTextNode("li", "li1");
var li2 = new LightTextNode("li", "li2");
ul.AppendChild(li1);
ul.AppendChild(li2);
ul.InsertBefore(li1, li0);
LightElementNode ulClone = (LightElementNode)ul.Clone();
Console.WriteLine(ulClone.OuterHTML());

li0.Events.Subscribe(EventType.OnClick, (sender, args) =>
{
```

```
    Console.WriteLine($"{args.Node.OuterHTML()} Clicked");
});
li0.Events.Subscribe(EventType.OnClick, (sender, args) =>
{
    Console.WriteLine($"{args.Node.OuterHTML()} Clicked (2nd listener)");
});
li1.Events.Subscribe(EventType.OnFocus, (sender, args) =>
{
    Console.WriteLine($"{args.Node.OuterHTML()} Focused");
});
li0.InvokeClick();
li1.InvokeOnFocus();
```

```
1. Composite. LightHTML
<ul classes="class1 class2 ">
        <li>li0</li>
        <li>li1</li>
        <li>li2</li>
</ul>
<li>li0</li> Clicked
<li>li0</li> Clicked (2nd listener)
<li>li1</li> Focused
Process finished with exit code 0.
```

Результат виконання

додавання EventListener-ів до Ваших HTML-елементів;

- за допомогою шаблону Темплейт додайте методи життєвого циклу (lifecycle hooks) Ваших елементів, наприклад OnCreated, OnInserted, OnRemoved, OnStylesApplied тощо;

Лістинг програми:

(LightElementNode)

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

namespace lab_5.LightHTML
{
    public class LightElementNode : LightNode
    {
        private List<LightNode> children = new List<LightNode>();
        public NodeType NodeType { get; }
        public ClosureType ClosureType { get; }
        public List<string> CssClasses { get; set; }
        public int ChildCount => 0;
```

```
        public LightElementNode(string tag, NodeType nodeType, ClosureType
closureType, List<string> cssClasses)
        {
            Tag = tag;
            NodeType = nodeType;
            ClosureType = closureType;
            CssClasses = cssClasses;
            Lifecycle();
        }
        public override string OuterHTML()
        {
            StringBuilder sb = new StringBuilder($"<{Tag} ");
            sb.Append($"classes=\"");
            foreach (var c in CssClasses)
            {
                sb.Append($"{c} ");
            }
            sb.Append("\"");
            if (ClosureType == ClosureType.Single)
            {
                sb.Append("/>");
            }
            else
            {

                sb.AppendLine(">");
                sb.AppendLine(InnerHTML());
                sb.Append($"</{Tag}>");

            }
            return sb.ToString();
        }
        public override string InnerHTML()
        {

            StringBuilder sb = new StringBuilder();

            foreach (var c in children)
            {
                sb.AppendLine($"\t{c.OuterHTML()}");
            }
            return sb.ToString();
        }
        public void AppendChild(LightNode node)
        {
            children.Add(node);
        }
        public void ReplaceChild(LightNode oldNode, LightNode newNode)
        {
            var index = children.IndexOf(oldNode);
            if (index != -1)
            {
                children[index] = newNode;
            }
        }
        public void RemoveChild(LightNode node)
        {
            var index = children.IndexOf(node);
            if (index != -1)
            {
                children.RemoveAt(index);
            }
        }
```

```
        public void InsertBefore(LightNode node, LightNode newNode)
        {
            var index = children.IndexOf(node);
            if (index != -1)
            {
                children.Insert(index, newNode);
            }
        }
        public override LightNode Clone()
        {
            var clone = new LightElementNode(Tag, NodeType, ClosureType,
new(CssClasses));
            foreach (var c in children)
            {
                clone.children.Add(c.Clone());
            }
            return clone;
        }
    }
}
```

(LightNode)

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

namespace lab_5.LightHTML
{
    public enum NodeType
    {
        TextNode,
        ElementNode
    }
    public enum ClosureType
    {
        Single,
        Even
    }
    abstract public class LightNode
    {
        public string Tag { get; set; }
        abstract public string InnerHTML();
        abstract public string OuterHTML();
        abstract public LightNode Clone();

        public void Lifecycle()
        {
            OnCreated();
            OnInserted();
            OnStylesApplied();
        }

        public virtual void OnCreated()
        {
            Console.WriteLine($"{Tag} node created");
        }
        public virtual void OnInserted()
        {
```

```
            Console.WriteLine($"{Tag} node inserted");
        }
        public virtual void OnStylesApplied()
        {
            Console.WriteLine($"{Tag} node styles applied");
        }
    }
}
```

(LightTextNode)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Cryptography.X509Certificates;
using System.Text;
using System.Threading.Tasks;

namespace lab_5.LightHTML
{
    public class LightTextNode : LightNode
    {
        public string? Text { get; set; }
        public LightTextNode(string tag, string text)
        {
            Tag = tag;
            Text = text;
            Lifecycle();
        }
        public LightTextNode(string tag)
        {
            Tag = tag;
        }

        public override string InnerHTML()
        {
            return Text;
        }

        public override string OuterHTML()

        {
            return $"<{Tag}>{Text}</{Tag}>";
        }

        public override LightNode Clone()
        {
            return new LightTextNode(Tag, Text);
        }
        public override void OnCreated()
        {
            Console.WriteLine("Text node created");
        }
        public override void OnInserted()
        {
            Console.WriteLine("Text node inserted");
        }
    }
}
```

(Program.cs)

```
using lab_5.LightHTML;


Console.WriteLine("1. Composite. LightHTML");
var ul = new LightElementNode("ul", NodeType.ElementNode, ClosureType.Even, new()
{ "class1", "class2" });
var li0 = new LightTextNode("li", "li0");

ul.AppendChild(li0);
Console.WriteLine(ul.OuterHTML());
```

```
ul node created
ul node inserted
ul node styles applied
Text node created
Text node inserted
li node styles applied
<ul classes="class1 class2 ">
        <li>li0</li>

</ul>

Process finished with exit code 0.
```

Результат виконання

- за допомогою Ітератору (і можливо Стратегії) додайте можливість перебору всього HTML документу в глибину і в ширину;

Лістинг програми:

(BreadthFirstIterator)

```
namespace lab_5.LightHTML.Iterator
{
    public class BreadthFirstIterator : NodeIterator
    {
        private Queue<LightNode> _queue;
        private LightNode? _current;
        public BreadthFirstIterator(List<LightNode> collection)
        {
            _queue = new Queue<LightNode>();
            foreach (var node in collection)
            {
                _queue.Enqueue(node);
```

```
            }
        }
        public override LightNode? CurrentNode() => _current;

        public override void Dispose()
        {
            _queue.Clear();
            _current = null;
        }

        public override bool MoveNext()
        {
            if (_queue.Count == 0)
            {
                _current = null;
                return false;
            }

            _current = _queue.Dequeue();
            foreach (var childNode in _current.Children)
            {
                _queue.Enqueue(childNode);
            }
            return true;
        }

        public override void Reset()
        {
            _queue.Clear();
            _current = null;
        }
    }
}
```

(DepthFirstIterator)

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Globalization;
using System.Linq;
using System.Security;
using System.Text;
using System.Threading.Tasks;

namespace lab_5.LightHTML.Iterator
{
    public class DepthFirstIterator : NodeIterator
    {
        private Stack<LightNode> _stack;
        private LightNode? _current;
        public DepthFirstIterator(List<LightNode> collection)
        {

            _stack = new Stack<LightNode>();
            foreach (LightNode node in collection.Reverse<LightNode>())
            {
                _stack.Push(node);
            }
        }

        public override LightNode? CurrentNode() => _current;
```

```
        public override void Dispose()
        {
            _stack.Clear();
            _current = null;
        }

        public override bool MoveNext()
        {

            if (_stack.Count == 0)
            {
                _current = null;
                return false;
            }

            _current = _stack.Pop();
            foreach (var childNode in _current.Children)
            {
                _stack.Push(childNode);
            }
            return true;
        }

        public override void Reset()
        {
            _stack.Clear();
            _current = null;
        }
    }
}
```

(IteratorAggregate)

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

namespace lab_5.LightHTML.Iterator
{
    public abstract class IteratorAggregate : IEnumerable<LightNode>
    {
        public abstract IEnumerator<LightNode> GetEnumerator();

        IEnumerator IEnumerable.GetEnumerator()
        {
            return GetEnumerator();
        }
    }
}
```

(NodeIterator)

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

```
namespace lab_5.LightHTML.Iterator
{
    public abstract class NodeIterator : IEnumerator<LightNode>
    {
        public LightNode? Current => CurrentNode();

        object? IEnumerator.Current => CurrentNode();

        public abstract LightNode? CurrentNode();
        public abstract void Dispose();

        public abstract bool MoveNext();

        public abstract void Reset();
    }
}
```

(NodesCollection)

```
using lab_5.LightHTML.Strategy;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace lab_5.LightHTML.Iterator
{
    public class NodesCollection : IteratorAggregate
    {
        private List<LightNode> _collection = new List<LightNode>();
        public IIteratorStrategy Strategy { get; set; } = new SurfaceStrategy();
        public List<LightNode> GetItems()
        {
            return _collection;
        }
        public void AddItem(LightNode node)
        {
            _collection.Add(node);
        }
        public override IEnumerator<LightNode> GetEnumerator()
        {
            return Strategy.GetEnumerator(_collection);
        }

    }
}
```

(BreadthFirstStrategy)

```
using lab_5.LightHTML.Iterator;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace lab_5.LightHTML.Strategy
{
```

```
    public class BreadthFirstStrategy : IIteratorStrategy
    {
        public IEnumerator<LightNode> GetEnumerator(List<LightNode> collection)
        {
            return new BreadthFirstIterator(collection);
        }
    }
}
```

(DepthFirstStrategy)

```
using lab_5.LightHTML.Iterator;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace lab_5.LightHTML.Strategy
{
    public class DepthFirstStrategy : IIteratorStrategy
    {
        public IEnumerator<LightNode> GetEnumerator(List<LightNode> collection)
        {
            return new DepthFirstIterator(collection);
        }
    }
}
```

(IIteratorStrategy)

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

namespace lab_5.LightHTML.Strategy
{
    public interface IIteratorStrategy
    {
        public IEnumerator<LightNode> GetEnumerator(List<LightNode> collection);
    }
}
```

(SurfaceStrategy)

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

namespace lab_5.LightHTML.Strategy
{
    public class SurfaceStrategy : IIteratorStrategy
    {
        public IEnumerator<LightNode> GetEnumerator(List<LightNode> collection)
        {
            return collection.GetEnumerator();
```

```
        }
    }
}
```

(Program)

```
using lab_5.LightHTML;
using lab_5.LightHTML.Strategy;


//Console.WriteLine("1. Composite. LightHTML");
var ul = new LightElementNode("ul", NodeType.ElementNode, ClosureType.Even, new()
{ "class1", "class2" });
var li0 = new LightElementNode("li", NodeType.ElementNode, ClosureType.Even, new()
{ "class1" });
var span1 = new LightTextNode("span", "Text");
var span2 = new LightTextNode("span", "Text");
var p = new LightTextNode("p", "Paragraph");
var li1 = new LightElementNode("li", NodeType.ElementNode, ClosureType.Even, new()
{ "class1" });
var li2 = new LightElementNode("li", NodeType.ElementNode, ClosureType.Even, new()
{ "class1" });
li0.AppendChild(span1);
li0.AppendChild(span2);
ul.AppendChild(li1);
ul.AppendChild(li2);
li1.AppendChild(p);
ul.InsertBefore(li1, li0);
Console.WriteLine(ul.OuterHTML());
Console.WriteLine("Surface iterator:");
ul.Children.Strategy = new SurfaceStrategy();
printDOM();
Console.WriteLine("Depth iterator:");
ul.Children.Strategy = new DepthFirstStrategy();
printDOM();
Console.WriteLine("Breadth iterator:");
ul.Children.Strategy = new BreadthFirstStrategy();
printDOM();
void printDOM()
{
    foreach (var child in ul.Children)
    {
        Console.WriteLine($"{child.Tag}");
    }

}
```

```
<ul classes="class1 class2 ">
        <li classes="class1 ">
        <span>Text</span>
        <span>Text</span>
</li>
        <li classes="class1 ">
        <p>Paragraph</p>
</li>
        <li classes="class1 "></li>
</ul>
Surface iterator:
li
li
li
Depth iterator:
li
span
span
li
p
li
Breadth iterator:
li
li
li
span
span
p
```

Результат виконання

- додайте новий елемент Image, який за допомогою стратегії в залежності від переданого href буде завантажувати картинку або з файлової системи або з мережі.

Лістинг програми:

(LightElementNode)

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

namespace lab_5.LightHTML
{
    public class LightElementNode : LightNode
    {
        private List<LightNode> children = new List<LightNode>();
        public NodeType NodeType { get; }
        public ClosureType ClosureType { get; }
        public List<string> CssClasses { get; set; }
        public int ChildCount => 0;
        public LightElementNode(string tag, NodeType nodeType, ClosureType
closureType, List<string> cssClasses)
        {
            Tag = tag;
            NodeType = nodeType;
            ClosureType = closureType;
            CssClasses = cssClasses;
        }
        public override string OuterHTML()
        {
            StringBuilder sb = new StringBuilder($"<{Tag} ");
            sb.Append($"classes=\"");
            foreach (var c in CssClasses)
            {
                sb.Append($"{c} ");
            }
            sb.Append("\"");
            if (ClosureType == ClosureType.Single)
            {
                sb.Append("/>");
            }
            else
            {

                sb.AppendLine(">");
                sb.AppendLine(InnerHTML());
                sb.Append($"</{Tag}>");

            }
            return sb.ToString();
        }
        public override string InnerHTML()
        {

            StringBuilder sb = new StringBuilder();

            foreach (var c in children)
            {
                sb.AppendLine($"\t{c.OuterHTML()}");
            }
            return sb.ToString();
        }
        public void AppendChild(LightNode node)
        {
            children.Add(node);
        }
        public void ReplaceChild(LightNode oldNode, LightNode newNode)
        {
```

```
            var index = children.IndexOf(oldNode);
            if (index != -1)
            {
                children[index] = newNode;
            }
        }
        public void RemoveChild(LightNode node)
        {
            var index = children.IndexOf(node);
            if (index != -1)
            {
                children.RemoveAt(index);
            }
        }
        public void InsertBefore(LightNode node, LightNode newNode)
        {
            var index = children.IndexOf(node);
            if (index != -1)
            {
                children.Insert(index, newNode);
            }
        }
        public override LightNode Clone()
        {
            var clone = new LightElementNode(Tag, NodeType, ClosureType,
new(CssClasses));
            foreach (var c in children)
            {
                clone.children.Add(c.Clone());
            }
            return clone;
        }
    }
}
```

(LightNode)

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

namespace lab_5.LightHTML
{
    public enum NodeType
    {
        TextNode,
        ElementNode
    }
    public enum ClosureType
    {
        Single,
        Even
    }
    abstract public class LightNode
    {
        public string Tag { get; set; }
        abstract public string InnerHTML();
        abstract public string OuterHTML();
        abstract public LightNode Clone();
    }
}
```

(LightTextNode)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Cryptography.X509Certificates;
using System.Text;
using System.Threading.Tasks;

namespace lab_5.LightHTML
{
    public class LightTextNode : LightNode
    {
        public string? Text { get; set; }
        public LightTextNode(string tag, string text)
        {
            Tag = tag;
            Text = text;
        }
        public LightTextNode(string tag)
        {
            Tag = tag;
        }

        public override string InnerHTML()
        {
            return Text;
        }

        public override string OuterHTML()

        {
            return $"<{Tag}>{Text}</{Tag}>";
        }

        public override LightNode Clone()
        {
            return new LightTextNode(Tag, Text);
        }
    }
}
```

(Program)

```
using lab_5.LightHTML.Strategy;


internal class Program
{
    private static async Task Main(string[] args)
    {
        var img = new ImageNode("https://i0.wp.com/ourgenerationmusic.com/wp-
content/uploads/2023/01/img_9375.jpg");
        await img.LoadImage();
        Console.WriteLine(img.OuterHTML());

    }
}
```

```
Loaded bytes: 11865
<img href="https://i0.wp.com/ourgenerationmusic.com/wp-content/uploads/2023/01/img_9375.
jpg"/>

Process finished with exit code 0.
```

Результат виконання

Висновок: в ході виконання лабораторної роботи було розглянуто основні можливості реалізовувати структурні шоблони проєктування Спостерігач, Стратегія, Ітератор, Команда, Мементо, Стейт, Темплейт, Відвідувач