Міністерство освіти і науки України
Національний технічний університет
України
“Київський політехнічний інститут імені Ігоря
Сікорського” Факультет інформатики та обчислювальної
техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №7

з дисципліни «Технології розроблення програмного забезпечення»

Тема: «Патерни проектування»

Виконав:
студент групи ІА-з31
Курбатов Кіріл Андрійович

Перевірив:
Амонс Олександр Анатолійович

Київ 2025

Мета: Вивчити структуру шаблонів «Mediator», «Facade», «Bridge», «Template method» та навчитися застосовувати їх в реалізації програмної системи.

Завдання

• Ознайомитись з короткими теоретичними відомостями.

• Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.

• Реалізувати один з розглянутих шаблонів за обраною темою.

• Реалізувати не менше 3-х класів відповідно до обраної теми.

• Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму класів, яка представляє використання шаблону в реалізації системи, навести фрагменти коду по реалізації цього шаблону.

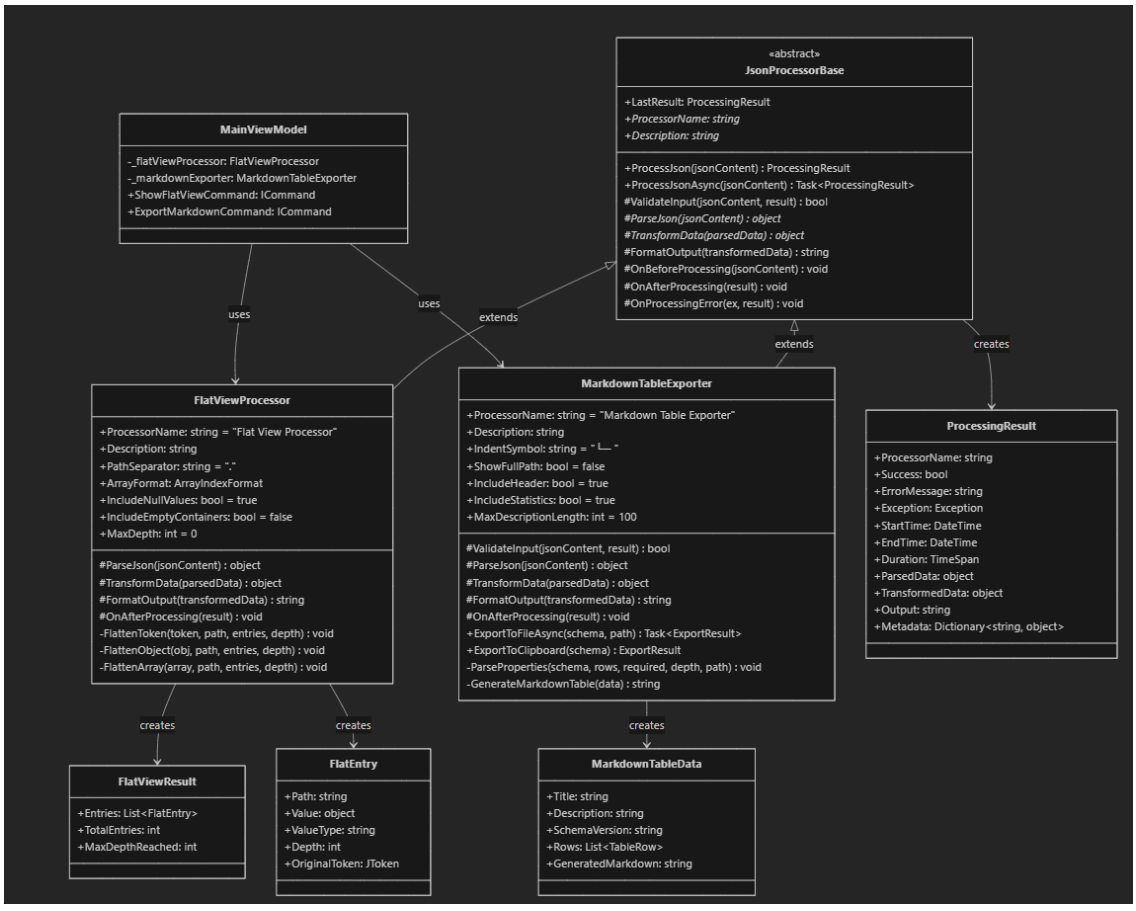

Рис. 1 — Діаграма класів патерну Template Method для обробки JSON

Реалізація патерну Template Method

```
using Newtonsoft.Json.Linq;

namespace JsonTool.Core.TemplateMethod;

public abstract class JsonProcessorBase
{
    public ProcessingResult? LastResult { get; protected set; }
    public abstract string ProcessorName { get; }
    public abstract string Description { get; }

    public ProcessingResult ProcessJson(string jsonContent)
    {
        var result = new ProcessingResult
        {
            ProcessorName = ProcessorName,
            StartTime = DateTime.Now
        };

        try
        {
            OnBeforeProcessing(jsonContent);

            if (!ValidateInput(jsonContent, result))
            {
                result.Success = false;
                result.EndTime = DateTime.Now;
                LastResult = result;
                return result;
            }

            var parsedData = ParseJson(jsonContent);
            result.ParsedData = parsedData;

            var transformedData = TransformData(parsedData);
            result.TransformedData = transformedData;

            var output = FormatOutput(transformedData);
            result.Output = output;

            OnAfterProcessing(result);

            result.Success = true;
        }
        catch (Exception ex)
        {
            result.Success = false;
            result.ErrorMessage = ex.Message;
            result.Exception = ex;

            OnProcessingError(ex, result);
        }
        finally
        {
            result.EndTime = DateTime.Now;
            LastResult = result;
        }

        return result;
    }
```

```
    public async Task<ProcessingResult> ProcessJsonAsync(string jsonContent)
    {
        return await Task.Run(() => ProcessJson(jsonContent));
    }

    protected virtual bool ValidateInput(string jsonContent, ProcessingResult result)
    {
        if (string.IsNullOrWhiteSpace(jsonContent))
        {
            result.ErrorMessage = "JSON content is empty or whitespace";
            return false;
        }

        try
        {
            JToken.Parse(jsonContent);
            return true;
        }
        catch (Exception ex)
        {
            result.ErrorMessage = $"Invalid JSON syntax: {ex.Message}";
            return false;
        }
    }

    protected virtual string FormatOutput(object transformedData)
    {
        if (transformedData is JToken token)
        {
            return token.ToString(Newtonsoft.Json.Formatting.Indented);
        }
        return transformedData?.ToString() ?? string.Empty;
    }

    protected virtual void OnBeforeProcessing(string jsonContent) { }
    protected virtual void OnAfterProcessing(ProcessingResult result) { }
    protected virtual void OnProcessingError(Exception ex, ProcessingResult result) { }

    protected abstract object ParseJson(string jsonContent);
    protected abstract object TransformData(object parsedData);
}

public class ProcessingResult
{
    public string ProcessorName { get; set; } = string.Empty;
    public bool Success { get; set; }
    public string? ErrorMessage { get; set; }
    public Exception? Exception { get; set; }
    public DateTime StartTime { get; set; }
    public DateTime EndTime { get; set; }
    public TimeSpan Duration => EndTime - StartTime;
    public object? ParsedData { get; set; }
    public object? TransformedData { get; set; }
    public string Output { get; set; } = string.Empty;
    public Dictionary<string, object> Metadata { get; } = new();
}
```