Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

# Лабораторна робота № 8

з дисципліни «Технології розроблення програмного забезпечення»

Тема: «Патерни проектування.»

«Менеджер завдань»

Виконав:
студент групи - ІА-32
Воробйов Кирило Андрійович

Перевірив:
Мягкий Михайло Юрійович

Київ 2025

**Тема:** Патерни проектування.

**Мета:** Вивчити структуру шаблонів «Composite», «Flyweight» (Пристосуванець), «Interpreter», «Visitor» та навчитися застосовувати їх в реалізації програмної системи.

**Тема проєкту:** Менеджер завдань (To Do Manager)

**Патерни:** Strategy, Command, Observer, Template Method, Composite, Client-Server

**Опис:** Додаток повинен мати можливість створювати, редагувати та видаляти завдання, забезпечувати гнучку фільтрацію та сортування (Strategy), та обробляти дії користувача (Command). Система має підтримувати ієрархічну структуру (Composite), забезпечувати автоматичне оновлення візуальних компонентів при зміні даних (Observer), а також надавати функціонал генерації звітів та експорту даних, використовуючи послідовний алгоритм (Template Method). Зберігання та синхронізація завдань реалізується через клієнт-серверний зв'язок (Client-Server).

## Зміст

## Теоретичні відомості

### Шаблон «Composite»

**Призначення**: Шаблон використовується для складання об'єктів в деревоподібну структуру для подання ієрархій типу «частина цілого». Даний шаблон дозволяє уніфіковано обробляти як поодинокі об'єкти, так і об'єкти з вкладеністю [6].

Простим прикладом може служити складання компонентів всередині звичайної форми. Форма може містити дочірні елементи (поля для введення тексту, цифр, написи, малюнки тощо); дочірні елементи можуть в свою чергу містити інші елементи. Наприклад, при виконанні операції розтягування форми необхідно, щоб вся ієрархія розтягнулася відповідним чином. В такому випадку форма розглядається як композитний об'єкт і операція розтягування застосовується до всіх дочірніх елементів рекурсивно.

Даний шаблон зручно використовувати при необхідності подання та обробки ієрархій об'єктів. Крім того, патерн «Composite» (Компонувальник) краще використовувати, коли ви представляєте структуру даних у вигляді дерева.

**Проблема**: Ви розробляєте систему керування проєктами. Кожен проєкт складається із наборів функцій, кожна функція з userstory, а кожна userstory в свою чергу із задач по її реалізації. Ви реалізуєте функціонал відображення оціночної вартості робіт по кожній із функцій, а також відображення чи всі userstory були оцінені. Це потрібно бізнес-аналітики, щоб розуміти, що всі userstory розробниками були розглянуті і оцінені, а також обговорити необхідність реалізації того чи іншого функціоналу на основі попередньої оцінки.

**Рішення**: Крашим підходом в даній ситуації буде використання патерну Компоновщик. Класи що представляють функції, userstory, задачі будуть наслідуватися від одного інтерфейсу ITask, функції (Feature) та Userstory будуть складними об'єктами і міститимуть колекції об'єктів ITask, а задачі (Task) будуть представляли кінцеві об'єкти без дочірніх елементів.

Для розрахунку оціночної вартосі робіт, в ITask інтефейс додаємо метод GetEstimatedPoints(). В класах-компоновщиках методи GetEstimatedPoints() реалізовуємо як обхід всіх дочірніх елементів та сумування результатів відповідей GetEstimatedPoints().

Таким чином, візуальні форми будуть працювати з колекцією елементів ITask і їм не потрібно буде знати конкретні типи дочірніх класів з якими вони працюють. В результаті логіка візуальних форм виходить достатньо простою і вона не буде містити бізнес-логіки розрахунку загальної оцінки по проєктам, а просто викликає метод GetEstimatedPoints() не замислюючись містить цей об'єкт дочірні об'єкти чи ні.

**Переваги та недоліки**:

+ Спрощує представлення деревоподібної структури.

+ Додає гнучкості в роботі з складними об’єктами та рекурсивними операціями.

+ Дозволяє додавати та видаляти об’єкти в ієрархії без впливу на клієнтський код.

- Потрібні додаткові зусилля для початкового впровадження.

- Вимагає гарно спроєктованого загального інтерфейсу.

**Шаблон «Flyweight»**

Призначення: Шаблон використовується для зменшення кількості об'єктів в додатку шляхом поділу цих об'єктів між ділянками додатку. Flyweight являє собою поділюваний об'єкт.

Дуже важливою є концепція «внутрішнього» і «зовнішнього» станів [6].

Внутрішній стан відображає дані, характерні саме поділюваному об'єкту (наприклад, код букви); зовнішній стан несе інформацію про його застосування в додатку (наприклад, рядок і стовпчик). Внутрішній стан зберігається в самому поділюваному об'єкті, зовнішній – в об'єктах додатку (контексту використання поділюваного об'єкта).

Здається, ніби для кожної літери існує окремий об'єкт. Насправді фізично об'єкт всього один, існує лише безліч посилань на нього. (рисунку 8.4)

Даний шаблон дуже добре застосовувати у випадках, коли використовується безліч однакових об'єктів (наприклад, графічних примітивів).

**Переваги та недоліки**:

+ Заощаджує оперативну пам’ять.

- Витрачає процесорний час на пошук/обчислення контексту.
- Ускладнює код програми внаслідок введення безлічі додаткових класів.

**Шаблон «Interpreter»**

**Призначення**: Даний шаблон використовується для подання граматики і інтерпретатора для вибраної мови (наприклад, скриптової) [6]. Граматика мови представлена термінальними і нетермінальними символами, кожен з яких інтерпретується в контексті використання. Клієнт передає контекст і сформовану пропозицію в використовувану мову в термінах абстрактного синтаксичного дерева (деревоподібна структура, яка однозначно визначає ієрархію виклику підвиразів), кожен вираз інтерпретується окремо з використанням контексту. У разі наявності дочірніх виразів, батьківський вираз інтерпретує спочатку дочірні (рекурсивно), а потім обчислює результат власної операції.

Шаблон зручно використовувати в разі невеликої граматики (інакше розростеться кількість використовуваних класів) і відносно простого контексту (без взаємних залежностей і т.п.).

Даний шаблон визначає базовий каркас інтерпретатора, який за допомогою рекурсії повертає результат обчислення пропозиції на основі результатів окремих елементів.

При використанні даного шаблону дуже легко реалізовується і розширюється граматика, а також додаються нові способи інтерпретації виразів

Проблема: Стоїть задача пошуку рядків по зразку, як така, що часто зустрічається. Або ж загалом є якась задача, яка дуже часто змінюється.

**Рішення**: Може бути вирішена шляхом створення інтерпретатора, який визначає граматику мови. «Клієнт» будує речення у вигляді абстрактного синтаксичного дерева, у вузлах якого знаходяться об’єкти класів «НетермінальнийВираз» і «ТермінальнийВираз» (рекурсивне), потім «Клієнт» ініціалізує контекст і виражає операцію Розібрати(Контекст). На кожному вузлі типу «НетермінальнийВираз» визначається операція Розібрати для кожного підвиразу. Для класу «ТермінальнийВираз» операція Розібрати визначає базу рекурсії. «АбстрактнийВираз» визначає абстрактну операцію Розібрати, загальну для всіх вузлів в абстрактному синтаксичному дереві.

«Контекст» містить інформацію, глобальну по відношенню до інтерпретатора.

**Переваги та недоліки**:

+ Граматику стає легко розширювати та змінювати, реалізації класів, що описують вузли абстрактного синтаксичного дерева схожі (легко кодуються).

+ Можна легко змінювати спосіб обчислення виразів.

- Супроводження граматики с великою кількістю правил є проблематичним.

**Шаблон «Visitor»**

**Призначення**: Шаблон відвідувач дозволяє вказувати операції над елементами без зміни структури конкретних елементів [6]. Таким чином вкрай зручно додавати нові операції, проте дуже важко додавати нові елементи в ієрархію (необхідно додавати відповідні методи для обробки їх відвідувань в кожного відвідувача).

Даний шаблон дозволяє групувати однотипні операції, що застосовуються над різнотипними об'єктами.

**Проблема**: Ви розробляєте онлайн-корзину інтернет магазина. Товари які представлені в магазині є різних типів, наприклад, електроніка, міцні напої, домашня хімія.

Логіка роботи з товарами в корзині є різна, наприклад, розрахунок вартості, формування замовлення.

Ми можемо всі ці методи зробити в товарах, але тоді ми ускладнюємо товари і змішуємо логіку (розрахунок вартості) з даними (товаром та його кількістю).

Якщо ми в подальшому необхідно буде добавити ще логіку розрахунку вартості з врахуванням знижки, то потрібно буде добавити ще цю логіку до товарів. А якщо буде ще сезонна знижка, то ми знову будемо добавляти нову логіку до класів товарів.

**Рішення**: Основна ідея патерна «Відвідувач» це рознести логіку і дані в різні класи та ієрархії. Якщо так зробити для нашої онлайн-корзини, то в класі відвідувача ми маємо функції для обрахунку логіки для кожного типу, а об'єкти в корзині знають свій тип, отримують екземпляр відвідувача і в нього викликають метод відповідно до свого типу. В результаті ми робимо різні відвідувачі для розрахунку вартості:

один для звичайних розрахунків, інший для розрахунку вартості зі знижкою, ще один для розрахунку сезонних знижок.

За рахунок того, що логіка відокремлена від наших товарів в корзині ми можемо реалізовувати за необхідності нові класи відвідувачі, а класи товарів та і корзини, в цілому, змінюватися не будуть.

Якщо розвивати далі, то можна зробити і клас відвідувача, який буде формувати замовлення з товарів в корзині в залежності від продавців і можливих варіантів доставки. Це дозволить формувати, наприклад, не одне замовлення, а два одна з самовивозом з магазину, а інше з доставкою Новою Поштою.

- **Приклад з життя**: Прикладом може служити написання компілятора. Припустимо, існують різні об'єкти в синтаксисі мови програмування: виклики методів і умовні вирази. Компліятор перед генерацією коду повинен обійти всі вирази (і виклики методів, і умовні вирази) і перевірити безпеку типів, після чого згенерувати відповідний код. Відповідно буде два відвідувачі – для перевірки безпеки типів і для генерації коду. У кожного з них буде по 2 методи – для викликів методів і для умовних операцій. Таким чином при необхідності додавання нових кроків компіляції досить буде визначити нового «відвідувача» і викликати його у відповідний час.

## Хід роботи

**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.

**Діаграма класів патерну Composite**

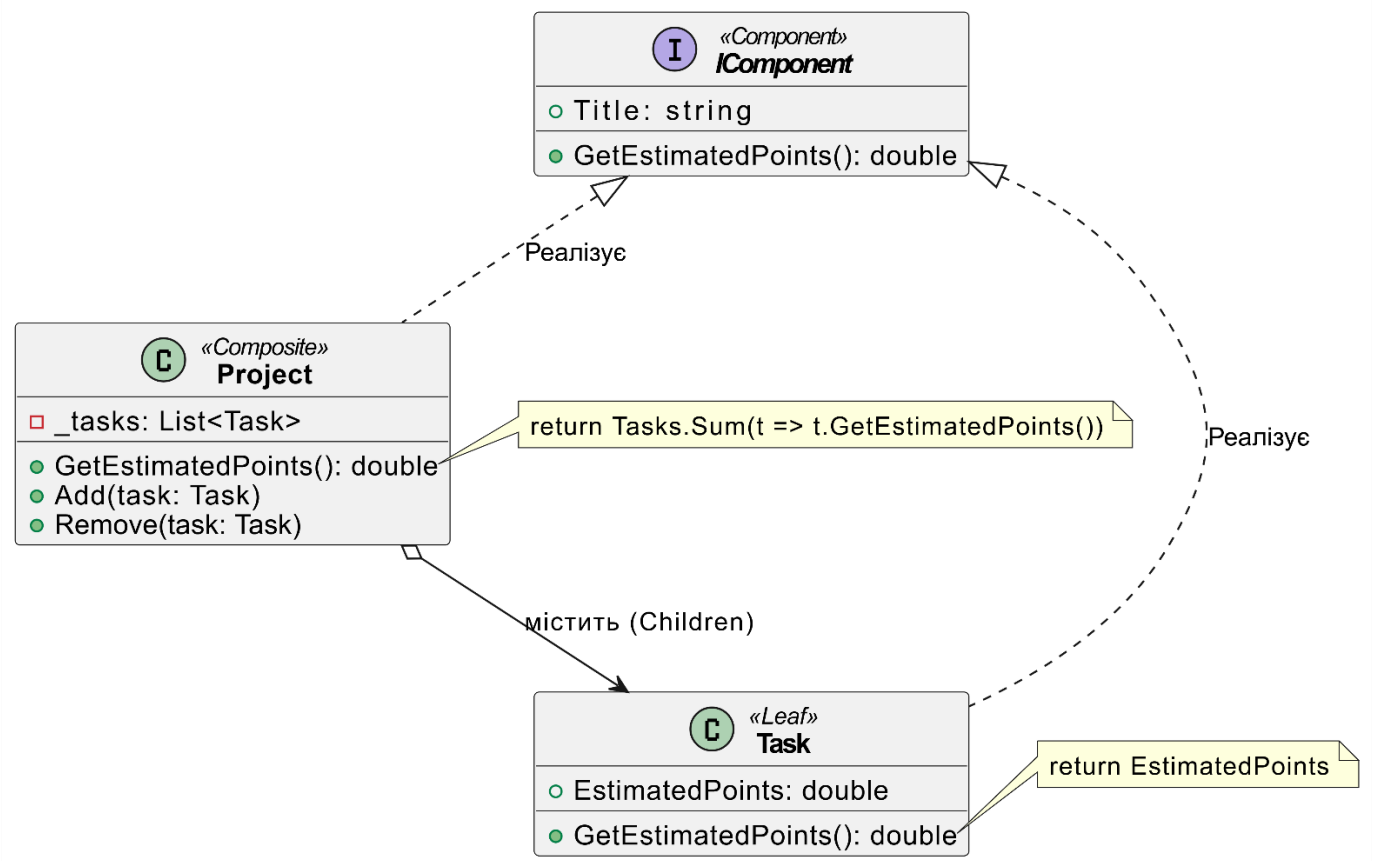



Рис. 1. Діаграма класів патерну Composite

Діаграма демонструє реалізацію патерна Composite для створення ієрархічної структури даних (частина-ціле), забезпечуючи уніфіковану обробку як окремих завдань, так і цілих проєктів.

1. Component (Компонент) — IComponent

- Роль: Виступає як спільний інтерфейс (абстракція) для всіх об'єктів у структурі — як простих, так і складених. Він оголошує ключову операцію GetEstimatedPoints(), яку повинні реалізувати всі елементи ієрархії. Це дозволяє клієнту взаємодіяти з будь-яким елементом однаково.

2. Leaf (Листок) — Task

- Роль: Представляє кінцевий елемент дерева, який не має підлеглих компонентів.

- Реалізація: У методі GetEstimatedPoints() він виконує реальну роботу — просто повертає власне значення оцінки складності (EstimatedPoints), введене користувачем.

3. Composite (Композит) — Project

- Роль: Представляє складний компонент (контейнер), який містить колекцію дочірніх елементів (Tasks).

- Реалізація: У методі GetEstimatedPoints() він не має власного значення, а делегує виконання своїм дітям: проходить по списку завдань, викликає у кожного з них GetEstimatedPoints() і підсумовує результати.

4. Аргументація Гнучкості

- Ця архітектура дозволяє клієнтському коду (MainForm) працювати з деревом об'єктів через загальний інтерфейс, не перевіряючи типи конкретних класів. Це спрощує додавання нових рівнів вкладеності (наприклад, "Підпроєкт") у майбутньому без зміни логіки підрахунку статистики.

## Код програми

### IComponent.cs

```
namespace ToDo.Components
{
    public interface IComponent
    {
        string Title { get; }

        // Головний метод патерна: отримати оцінку (для завдання - власна, для проєкту - сума)
        double GetEstimatedPoints();
    }
}
```

### Task.cs

```
using System;
using System.Collections.Generic;
using ToDo.Enums;
using ToDo.Components; // Додано

namespace ToDo.Models
{
    public class Task : IComponent // Реалізуємо інтерфейс
    {
        public int TaskId { get; set; }
        public string Title { get; set; } = string.Empty;
        public string Description { get; set; } = string.Empty;
        public double EstimatedPoints { get; set; } // Додано (Оцінка складності)
        public DateTime DueDate { get; set; }
        public PriorityEnum Priority { get; set; }
        public StatusEnum Status { get; set; }

        public int ProjectId { get; set; }
        public virtual Project? Project { get; set; }

        public virtual ICollection<Tag> Tags { get; set; } = new List<Tag>();
```

```
        // Реалізація методу Composite: просто повертаємо своє значення
        public double GetEstimatedPoints()
        {
            return EstimatedPoints;
        }
    }
}
```

**MainForm**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;
using ToDo.Models;
using ToDo.Repositories;
using ToDo.Enums;
using ToDo.Strategies;
using ToDo.Services;
using ToDo.Commands;
using ToDo.Exporters;
using Task = ToDo.Models.Task;

namespace ToDo.UI
{
    public partial class MainForm : Form
    {
        private readonly TaskService _taskService;
        private readonly IRepository<Project> _projectRepository;
        private readonly IRepository<User> _userRepository;
        private readonly IEnumerable<ITaskSortStrategy> _sortStrategies;

        // Експортери (ЛР 7)
        private readonly CsvTaskExporter _csvExporter;
        private readonly HtmlTaskExporter _htmlExporter;

        public MainForm(
            TaskService taskService,
            IRepository<Project> projectRepository,
            IRepository<User> userRepository,
            IEnumerable<ITaskSortStrategy> sortStrategies,
```

```
    CsvTaskExporter csvExporter,
    HtmlTaskExporter htmlExporter)
{
    InitializeComponent();

    _taskService = taskService;
    _projectRepository = projectRepository;
    _userRepository = userRepository;
    _sortStrategies = sortStrategies;

    _csvExporter = csvExporter;
    _htmlExporter = htmlExporter;

    _taskService.TasksChanged += OnTasksChangedHandler;

    InitializeSortComboBox();
    LoadTasks();
}

private void OnTasksChangedHandler()
{
    if (this.InvokeRequired)
    {
        this.Invoke(new Action(LoadTasks));
    }
    else
    {
        LoadTasks();
    }
}

private void InitializeSortComboBox()
{
    foreach (var strategy in _sortStrategies)
    {
        cmbSortStrategy.Items.Add(strategy);
    }
    cmbSortStrategy.SelectedIndex = 0;
    cmbSortStrategy.SelectedIndexChanged += (sender, e) =>
```

```
LoadTasks();
        }

        private void LoadTasks()
        {
            tasksListBox.Items.Clear();

            if (cmbSortStrategy.SelectedItem is not ITaskSortStrategy
selectedStrategy)
                return;

            var tasks = _taskService.GetAllTasks();
            var sortedTasks = selectedStrategy.Sort(tasks);

            foreach (var task in sortedTasks)
            {
                tasksListBox.Items.Add($"[{task.Status}] {task.Title} (Пріоритет:
{task.Priority})");
            }
        }

        private void btnAddTask_Click(object sender, EventArgs e)
        {
            var defaultUser = _userRepository.GetAll().FirstOrDefault();
            if (defaultUser == null)
            {
                MessageBox.Show("Критична помилка: Не знайдено
користувача.", "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }
            var defaultProject = _projectRepository.GetAll().FirstOrDefault(p =>
p.UserId == defaultUser.UserId);
            if (defaultProject == null)
            {
                defaultProject = new Project
                {
                    Title = "Загальний Проєкт",
                    UserId = defaultUser.UserId,
                    CreationDate = DateTime.Now
```

```
            };
            _projectRepository.Add(defaultProject);
        }

        using (var addTaskForm = new
AddTaskForm(defaultProject.ProjectId))
        {
            if (addTaskForm.ShowDialog() == DialogResult.OK)
            {
                try
                {
                    var newTask = new Task
                    {
                        Title = addTaskForm.TaskTitle!,
                        Description = string.Empty,
                        Priority = addTaskForm.TaskPriority,
                        ProjectId = defaultProject.ProjectId,
                        Status = StatusEnum.New,
                        DueDate = DateTime.Now.AddDays(1),

                        // ЛР 8: Зберігаємо оцінку (бали) з форми
                        EstimatedPoints = addTaskForm.TaskEstimatedPoints
                    };

                    ICommand command = new CreateTaskCommand(_taskService,
newTask);
                    command.Execute();
                }
                catch (Exception ex)
                {
                    MessageBox.Show($"Помилка збереження завдання:
{ex.Message}\n\n" +
                                    (ex.InnerException != null ?
ex.InnerException.Message : ""),
                                    "Помилка БД", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                }
            }
        }
```

```
    }

    // Обробник кнопки "Експорт" (ЛР 7)
    private void btnExport_Click(object sender, EventArgs e)
    {
        using (var saveFileDialog = new SaveFileDialog())
        {
            saveFileDialog.Filter = "CSV файл (*.csv)|*.csv|HTML сторінка
(*.html)|*.html";
            saveFileDialog.Title = "Експорт завдань";
            saveFileDialog.FileName = "tasks_export";

            if (saveFileDialog.ShowDialog() == DialogResult.OK)
            {
                try
                {
                    var tasks = _taskService.GetAllTasks();
                    string filePath = saveFileDialog.FileName;

                    TaskExporter exporter;

                    if (filePath.EndsWith(".csv"))
                    {
                        exporter = _csvExporter;
                    }
                    else
                    {
                        exporter = _htmlExporter;
                    }

                    exporter.Export(filePath, tasks);

                    MessageBox.Show("Дані успішно експортовано!", "Успіх",
MessageBoxButtons.OK, MessageBoxIcon.Information);
                }
                catch (Exception ex)
                {
                    MessageBox.Show($"Помилка експорту: {ex.Message}",
"Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);
```

```
            }
        }
    }
}

// ЛР 8: Метод підрахунку балів (Composite Pattern)
private void btnCalcStats_Click(object sender, EventArgs e)
{
    // Отримуємо поточного користувача
    var defaultUser = _userRepository.GetAll().FirstOrDefault();
    if (defaultUser == null)
    {
        MessageBox.Show("Користувач не знайдений.", "Помилка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
        return;
    }

    var project = _projectRepository.GetAll().FirstOrDefault(p =>
p.UserId == defaultUser.UserId);

    if (project != null)
    {
        // ТУТ ПРАЦЮЄ COMPOSITE:
        // Ми викликаємо метод у Проєкту, а він сам опитує свої
Завдання
        double totalPoints = project.GetEstimatedPoints();

        // Оновлюємо лейбл та показуємо повідомлення
        lblStats.Text = $"Загальна оцінка: {totalPoints}";
        MessageBox.Show($"Загальна складність проєкту
\"{project.Title}\": {totalPoints} балів",
                        "Статистика Проєкту", MessageBoxButtons.OK,
MessageBoxIcon.Information);
    }
    else
    {
        MessageBox.Show("Проєкт не знайдено.", "Інформація",
MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
```

```
        }
    }
}
```

**Project.cs**

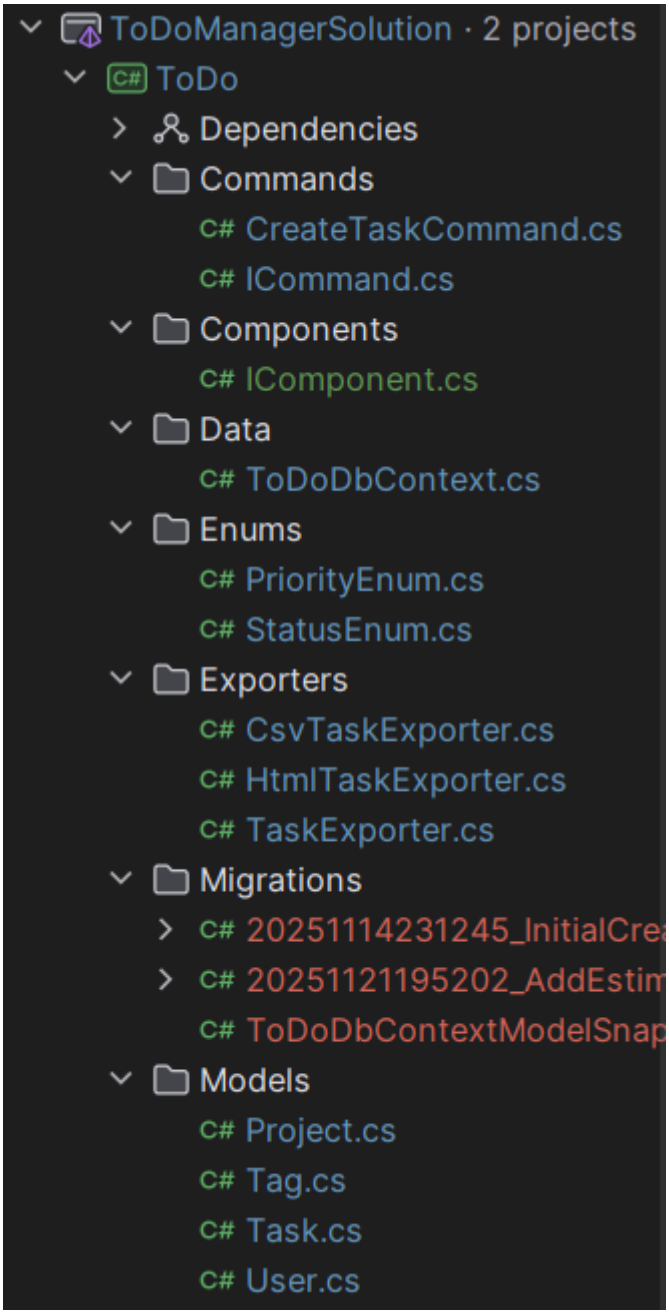
```
using System;
using System.Collections.Generic;
using System.Linq;
using ToDo.Components;

namespace ToDo.Models
{
    public class Project : IComponent // Реалізуємо інтерфейс
    {
        public int ProjectId { get; set; }
        public string Title { get; set; } = string.Empty;
        public DateTime CreationDate { get; set; }

        public int UserId { get; set; }
        public virtual User? User { get; set; }

        public virtual ICollection<Task> Tasks { get; set; } = new List<Task>();

        // Реалізація методу Composite: рекурсивна сума
        public double GetEstimatedPoints()
        {
            // Ми використовуємо поліморфізм: викликаємо GetEstimatedPoints
у кожного Task
            return Tasks.Sum(t => t.GetEstimatedPoints());
        }
    }
}
```

**AddTaskForm.cs**

```
using System;
using System.Windows.Forms;
using ToDo.Enums;
using ToDo.Models;

namespace ToDo.UI
```

```
{
    public partial class AddTaskForm : Form
    {
        public string TaskTitle { get; private set; }
        public PriorityEnum TaskPriority { get; private set; }

        // Нова властивість для балів
        public double TaskEstimatedPoints { get; private set; }

        public int ProjectId { get; private set; }

        public AddTaskForm(int projectId)
        {
            InitializeComponent();
            ProjectId = projectId;
            cmbPriority.DataSource = Enum.GetValues(typeof(PriorityEnum));
        }

        private void btnSave_Click(object sender, EventArgs e)
        {
            if (string.IsNullOrWhiteSpace(txtTaskTitle.Text))
            {
                MessageBox.Show("Назва завдання не може бути порожньою.");
                return;
            }

            TaskTitle = txtTaskTitle.Text;
            TaskPriority = (PriorityEnum)cmbPriority.SelectedItem!;

            // Зчитуємо значення з NumericUpDown
            TaskEstimatedPoints = (double)numPoints.Value;

            this.DialogResult = DialogResult.OK;
            this.Close();
        }

        private void btnCancel_Click(object sender, EventArgs e)
        {
            this.DialogResult = DialogResult.Cancel;
```

```
            this.Close();
        }
    }
}
```


ToDoManagerSolution · 2 projects
ToDo
Dependencies
Commands
CreateTaskCommand.cs
ICommand.cs
Components
IComponent.cs
Data
ToDoDbContext.cs
Enums
PriorityEnum.cs
StatusEnum.cs
Exporters
CsvTaskExporter.cs
HtmlTaskExporter.cs
TaskExporter.cs
Migrations
20251114231245_InitialCrea
20251121195202_AddEstim
ToDoDbContextModelSnap
Models
Project.cs
Tag.cs
Task.cs
User.cs

## Висновки

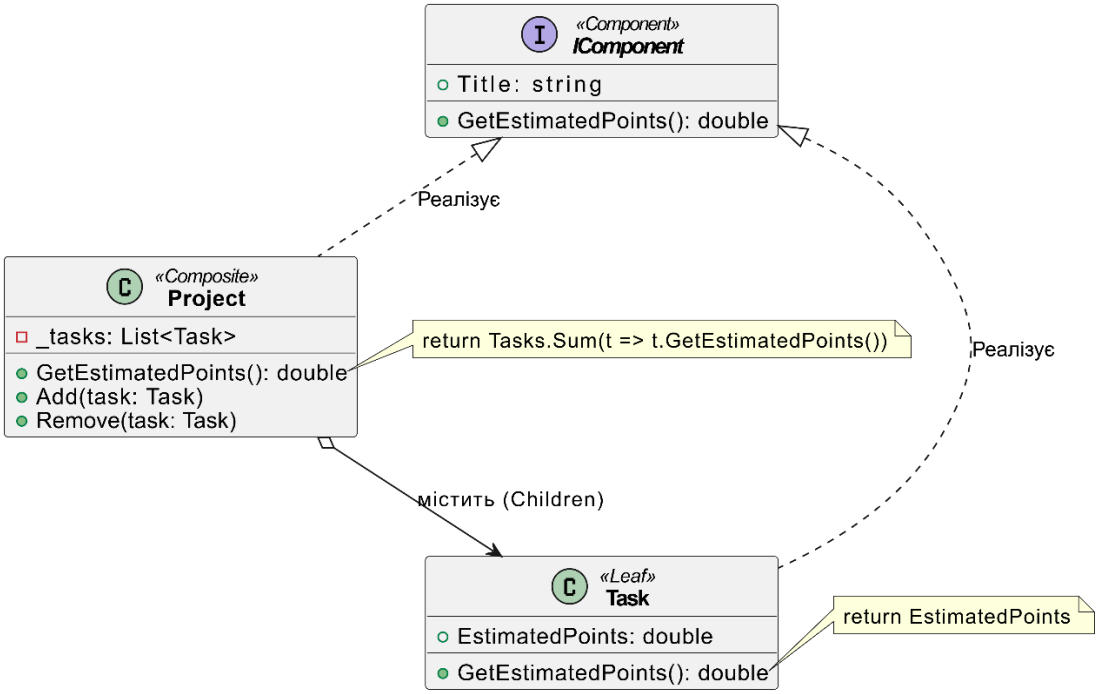
Висновки: Під час виконання лабораторної роботи ми успішно засвоїли принципи реалізації структурного патерну Composite (Компонувальник). Ми створили ієрархічну структуру «частина-ціле» (Project → Task) за допомогою спільного інтерфейсу IComponent. Це дозволило нам уніфіковано працювати з окремими завданнями та цілими проєктами, успішно реалізувавши рекурсивний підрахунок загальної оцінки складності проєкту. Таким чином, ми підтвердили гнучкість системи до роботи зі складними, деревоподібними структурами даних.

## Питання до лабораторної роботи

1. Яке призначення шаблону «Композит»?

Шаблон «Composite» (Компонувальник) використовується для складання об'єктів у деревоподібну структуру для подання ієрархій типу «частина-ціле». Даний шаблон дозволяє клієнтам уніфіковано обробляти як поодинокі об'єкти, так і групи об'єктів (композиції) .

2. Нарисуйте структуру шаблону «Композит».



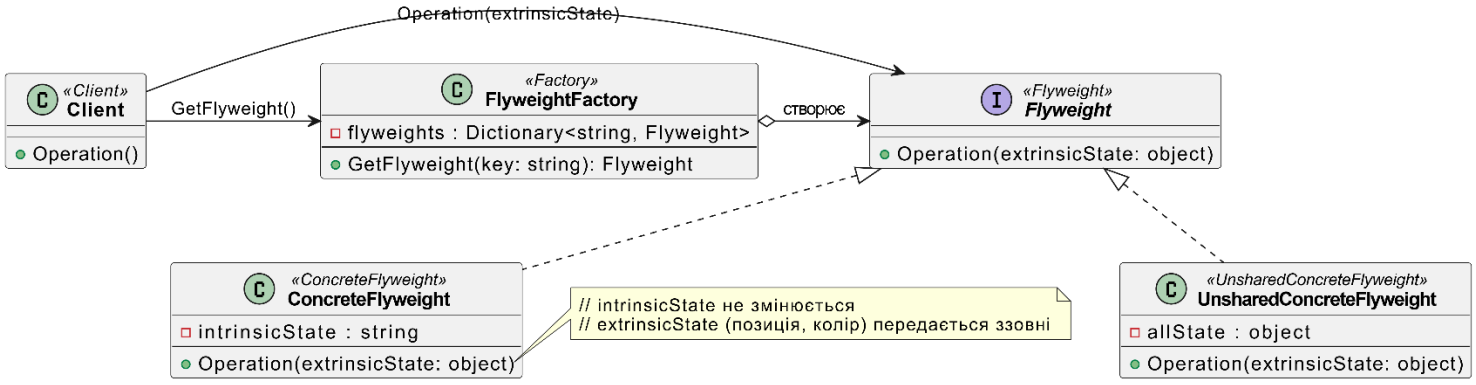
3. Які класи входять в шаблон «Композит», та яка між ними взаємодія?

Входять Component (абстракція для всіх елементів), Leaf (кінцевий елемент без дочірніх вузлів), Composite (елемент, що має дочірні компоненти) та Client. Client маніпулює об'єктами через інтерфейс Component. Composite переадресовує запити своїм дочірнім компонентам і може виконувати додаткові операції до або після переадресації .

4. Яке призначення шаблону «Легковаговик»?

Шаблон «Flyweight» (Легковаговик/Пристосуванець) використовується для зменшення кількості об'єктів у додатку шляхом поділу (спільного використання) цих об'єктів між різними ділянками додатку. Це дозволяє заощаджувати пам'ять, коли в системі присутня велика кількість подібних дрібних об'єктів .

5. Нарисуйте структуру шаблону «Легковаговик».




6. Які класи входять в шаблон «Легковаговик», та яка між ними взаємодія?

Головна концепція — поділ станів. Внутрішній стан зберігається в самому об'єкті, що поділяється (ConcreteFlyweight), і не залежить від контексту. Зовнішній стан передається клієнтом (Client) у методи

об'єкта під час виклику. FlyweightFactory відповідає за створення та управління пулом легковаговиків .

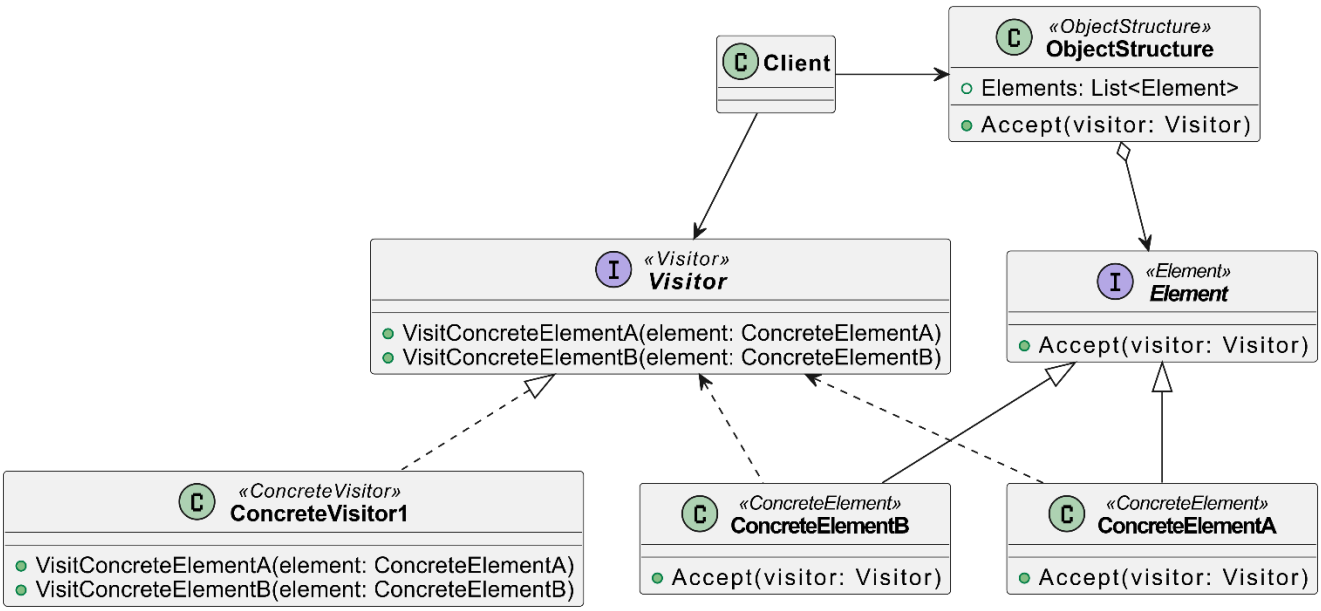
7. Яке призначення шаблону «Інтерпретатор»?

Шаблон «Interpreter» (Інтерпретатор) використовується для подання граматики певної мови та інтерпретатора, який використовує це подання для інтерпретації речень цієї мови. Клієнт будує речення у вигляді абстрактного синтаксичного дерева .

8. Яке призначення шаблону «Відвідувач»?

Шаблон «Visitor» (Відвідувач) дозволяє додавати нові операції над елементами об'єктної структури без зміни класів цих елементів. Це досягається шляхом винесення операції в окремий клас-відвідувач.

9. Нарисуйте структуру шаблону «Відвідувач».




10. Які класи входять в шаблон «Відвідувач», та яка між ними взаємодія?

Входять Visitor (оголошує методи Visit для кожного типу елемента), ConcreteVisitor (реалізує конкретну операцію), Element (оголошує метод Accept), ConcreteElement (реалізує метод Accept, викликаючи відповідний метод відвідувача visitor.VisitConcreteElement(this)). Це дозволяє рознести дані та алгоритми їх обробки