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

Лабораторна робота №4

із дисципліни «*Технології розроблення програмного забезпечення*»

Тема: «Вступ до паттернів проектування»

Виконала:
Студент групи ІА-31
Самелюк А.С.

Перевірив:
Мягкий М.Ю.

Київ-2025

Тема: Вступ до паттернів проектування.

Тема проєкту: Онлайн-щоденник (strategy, adapter, observer, facade, composite, client-server). Веб-застосунок дозволяє користувачу створювати, редагувати та організовувати особисті записи з можливістю пошуку, сортування та додавання тегів, а також отримання email-нагадувань.

Мета: Вивчити структуру шаблонів «Singleton», «Iterator», «Proxy», «State», «Strategy» та навчитися застосовувати їх в реалізації програмної системи.

Посилання на репозиторій з проєктом:
https://github.com/Sameliuk/OnlineDiary_trpz/tree/main

Хід роботи

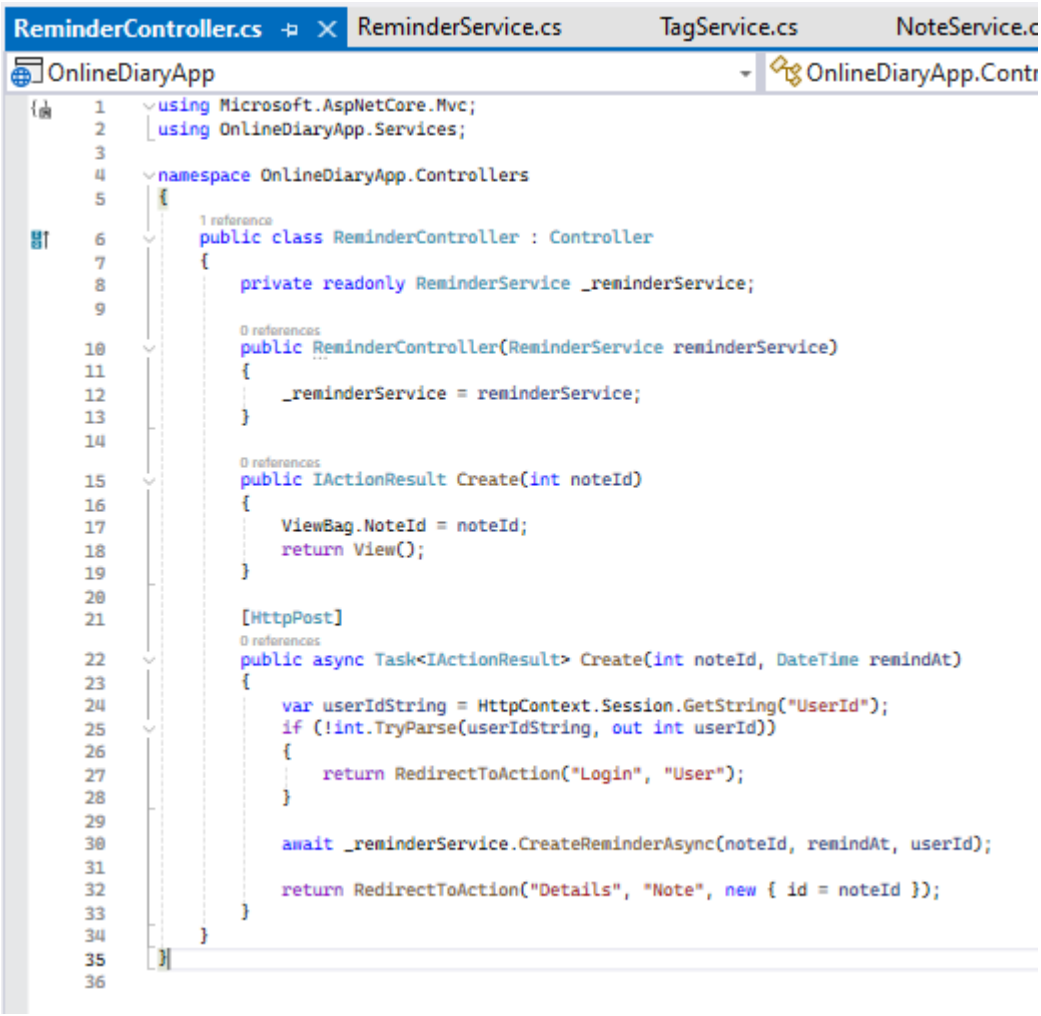
1. Ознайомитись з короткими теоретичними відомостями.
2. Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.

   В даній частині було реалізовану логіку з тегами та нагадуваннями. Було реалізовано відповідні сервіси та контролери для виконання певного функціоналу.

Код 1 – TagService (services)


```cs
using Microsoft.AspNetCore.Mvc;
using OnlineDiaryApp.Services;

namespace OnlineDiaryApp.Controllers
{
    1 reference
    public class TagController : Controller
    {
        private readonly TagService _tagService;

        0 references
        public TagController(TagService tagService)
        {
            _tagService = tagService;
        }

        0 references
        public async Task<IActionResult> Index()
        {
            var userIdString = HttpContext.Session.GetString("UserId");
            if (!int.TryParse(userIdString, out int userId))
                return RedirectToAction("Login", "User");

            var tags = await _tagService.GetAllTagsAsync(userId);
            return View(tags);
        }

        0 references
        public IActionResult Create()
        {
            return View();
        }

        [HttpPost]
        0 references
        public async Task<IActionResult> Create(string name)
        {
            var userIdString = HttpContext.Session.GetString("UserId");
            if (!int.TryParse(userIdString, out int userId))
                return RedirectToAction("Login", "User");

            if (string.IsNullOrWhiteSpace(name))
            {
                ModelState.AddModelError("", "Назва тегу обов'язкова");
                return View();
            }

            await _tagService.CreateTagAsync(name, userId);
            return RedirectToAction("Index");
        }

        0 references
        public async Task<IActionResult> Delete(int id)
        {
            await _tagService.DeleteTagAsync(id);
            return RedirectToAction("Index");
        }

    }
}
```

Код 2 – TagController (controllers)



Код 3 – ReminderController (controllers)

Код 4 – ReminderService (services)

3. Реалізувати один з розглянутих шаблонів за обраною темою.

Для даного проєкту було обрано патерн Strategy (Стратегія). Шаблон реалізований таким чином:

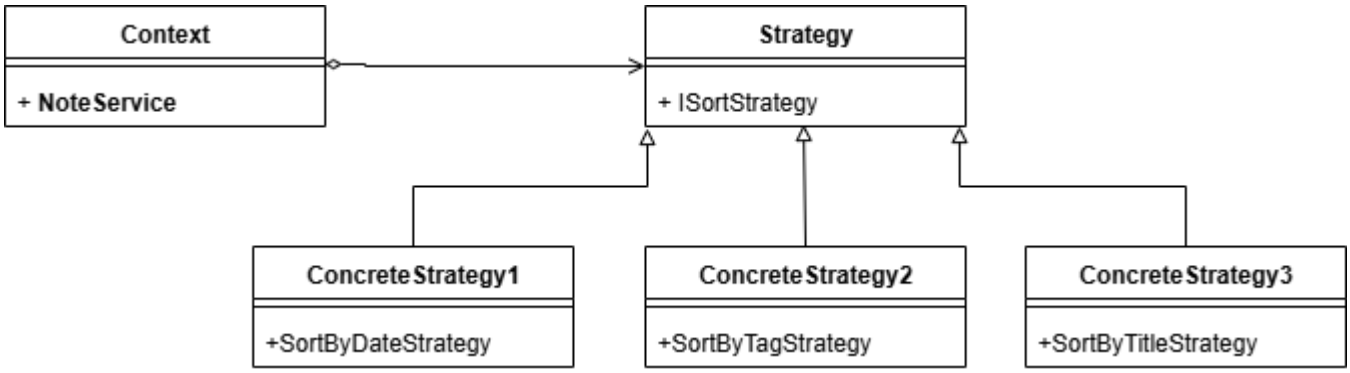


Рисунок 1 - Структура патерну Стратегія

- ISortStrategy – це інтерфейс, який описує метод Sort. Він задає контракт для всіх можливих стратегій сортування.
- SortByDate, SortByTag, SortByTitle – це конкретні реалізації стратегій. Кожен клас реалізує метод Sort по-своєму: один сортує записи за датою, інший – за тегами, ще один – за назвою.
- NoteService – це контекст, який працює з вибраною стратегією. Він не знає деталей реалізації, а просто викликає метод Sort у поточної стратегії.
- Note – модель, яка представляє дані, з якими працюють стратегії (назва, дата, тег).

В результаті:

Мажна легко змінювати алгоритм сортування, не змінюючи код самого NoteOrganizer. Це і є суть Стратегії – інкапсуляція алгоритмів у вигляді окремих класів та можливість динамічно їх змінювати.

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


```
using OnlineDiaryApp.Models;
using OnlineDiaryApp.Repositories.Interfaces;


namespace OnlineDiaryApp.Services
{
    6 references
    public interface ISortStrategy
    {
        5 references
        IEnumerable<Note> Sort(IEnumerable<Note> notes);
    }

    1 reference
    public class SortByDateStrategy : ISortStrategy
    {
        3 references
        public IEnumerable<Note> Sort(IEnumerable<Note> notes) =>
            notes.OrderByDescending(n => n.CreatedAt);
    }

    2 references
    public class SortByTagStrategy : ISortStrategy
    {
        private readonly string _tag;
        1 reference
        public SortByTagStrategy(string tag) => _tag = tag;

        3 references
        public IEnumerable<Note> Sort(IEnumerable<Note> notes) =>
            notes.Where(n => n.Tags.Any(t => t.Name == _tag));
    }

    0 references
    public class SortByTitleStrategy : ISortStrategy
    {
        3 references
        public IEnumerable<Note> Sort(IEnumerable<Note> notes) =>
            notes.OrderBy(n => n.Title);
    }

    6 references
    public class NoteService
    {
        private readonly INoteRepository _noteRepository;
        private readonly ITagRepository _tagRepository;

        0 references
        public NoteService(INoteRepository noteRepository, ITagRepository tagRepository)
        {
            _noteRepository = noteRepository;
            _tagRepository = tagRepository;
        }
        1 reference
        public async Task<IEnumerable<Note>> GetAllNotesByUserAsync(int userId, ISortStrategy? strategy = null)
        {
            var notes = (await _noteRepository.GetAllAsync())
                        .Where(n => n.UserId == userId);

            return strategy != null ? strategy.Sort(notes) : notes;
        }

        1 reference
        public async Task<IEnumerable<Note>> GetAllNotesAsync(ISortStrategy? strategy = null)
        {
            var notes = await _noteRepository.GetAllAsync();
            return strategy != null ? strategy.Sort(notes) : notes;
        }

        2 references
        public async Task<Note?> GetNoteByIdAsync(int id) =>
            await _noteRepository.GetByIdAsync(id);

        1 reference
        public async Task<Note> CreateNoteAsync(string title, string content, int userId, List<int> tagIds)
        {
            var tags = new List<Tag>();
            foreach (var id in tagIds)
            {
                var tag = await _tagRepository.GetByIdAsync(id);
                if (tag != null)
                    tags.Add(tag);
            }

            var utcNow = DateTime.UtcNow;

            var note = new Note
            {
                Title = title,
                Content = content,
                UserId = userId,
                Tags = tags,
                CreatedAt = utcNow
            };

            await _noteRepository.AddAsync(note);
            await _noteRepository.SaveChangesAsync();

            return note;
        }
```

Код 5.1 – NoteService (services)

```
86
        1 reference
87      public async Task UpdateNoteAsync(Note note, List<int> tagIds)
88      {
89          var tags = new List<Tag>();
90          foreach (var id in tagIds)
91          {
92              var tag = await _tagRepository.GetByIdAsync(id);
93              if (tag != null)
94                  tags.Add(tag);
95          }
96
97          note.Tags = tags;
98          await _noteRepository.UpdateAsync(note);
99          await _noteRepository.SaveChangesAsync();
100     }
101
        1 reference
102     public async Task DeleteNoteAsync(int id)
103     {
104         await _noteRepository.DeleteAsync(id);
105         await _noteRepository.SaveChangesAsync();
106     }
107
        1 reference
108     public async Task<IEnumerable<Note>> SearchByTitleAsync(string keyword)
109     {
110         var notes = await _noteRepository.GetAllAsync();
111         return notes.Where(n => n.Title.Contains(keyword, StringComparison.OrdinalIgnoreCase));
112     }
113
        0 references
114     public async Task<IEnumerable<Tag>> GetAllTagsAsync(int userId) =>
115 (await _tagRepository.GetAllAsync(userId)).Where(t => t.UserId == userId);
116
117
118 }
119 }
120
```

Код 5.2 – NoteService (services)

Код 6.1 – NoteController (controllers)

```
93
94          [HttpPost]
            0 references
95          public async Task<IActionResult> Edit(int id, string title, string content, List<int>? tagIds, DateTime? reminderDate)
96          {
97              var note = await _noteService.GetNoteByIdAsync(id);
98              if (note == null)
99                  return NotFound();
100
101             note.Title = title;
102             note.Content = content;
103
104             await _noteService.UpdateNoteAsync(note, tagIds ?? new List<int>());
105
106             var existingReminder = await _reminderService.GetReminderByNoteIdAsync(note.Id);
107             if (reminderDate.HasValue)
108             {
109                 if (existingReminder != null)
110                 {
111                     await _reminderService.UpdateReminderAsync(existingReminder, reminderDate.Value);
112                 }
113                 else
114                 {
115                     await _reminderService.CreateReminderAsync(note.Id, reminderDate.Value, note.UserId);
116                 }
117             }
118
119             return RedirectToAction("Index");
120         }
121
            0 references
122         public async Task<IActionResult> Delete(int id)
123         {
124             await _noteService.DeleteNoteAsync(id);
125             return RedirectToAction("Index");
126         }
127
            0 references
128         public async Task<IActionResult> Search(string keyword)
129         {
130             var userIdString = HttpContext.Session.GetString("UserId");
131             if (!int.TryParse(userIdString, out int userId))
132                 return View(new List<Note>());
133
134             var notes = await _noteService.SearchByTitleAsync(keyword);
135             notes = notes.Where(n => n.UserId == userId);
136             return View("Index", notes);
137         }
138     }
139 }
140
```

Код 6.2 – NoteController (controllers)

Index.cshtml | ReminderController.cs | ReminderService.cs | TagService.cs | NoteServi

OnlineDiaryApp

```
@model IEnumerable<OnlineDiaryApp.Models.Note>

<link rel="stylesheet" href="~/css/notes.css" />

<h2 class="notes-header">Мої нотатки</h2>

<a class="btn-create" asp-action="Create">+ Створити нову нотатку</a>

<!-- Панель сортування -->
<div class="sort-panel">
    <form method="get" asp-action="Index">
        <label for="sortBy">Сортувати за:</label>
        <select id="sortBy" name="sortBy" onchange="this.form.submit()">
            <option value="">Без сортування</option>
            <option value="date" selected="@("date" == Context.Request.Query["sortBy"])">Датою</option>
            <option value="tag" selected="@("tag" == Context.Request.Query["sortBy"])">Тегом</option>
            <option value="title" selected="@("title" == Context.Request.Query["sortBy"])">Заголовком</option>
        </select>

        <input type="text" name="tag" placeholder="Введіть тег" value="@Context.Request.Query["tag"]" />
        <button type="submit">🔍 Застосувати</button>
    </form>
</div>

<!-- Список нотаток -->
<div class="notes-grid">
    @foreach (var note in Model)
    {
        <div class="note-card">
            <div class="note-title">@note.Title</div>
            <div class="note-date">@note.CreatedAt.ToString("g")</div>

            <div class="note-tags">
                @foreach (var tag in note.Tags)
                {
                    <span class="note-tag">@tag.Name</span>
                }
            </div>

            <div class="note-actions">
                <a class="btn-edit" asp-action="Edit" asp-route-id="@note.Id">Редагувати</a>
                <a class="btn-delete" asp-action="Delete" asp-route-id="@note.Id">Видалити</a>
            </div>
        </div>
    }
</div>
```

Код 7 – Index (Views/Note)

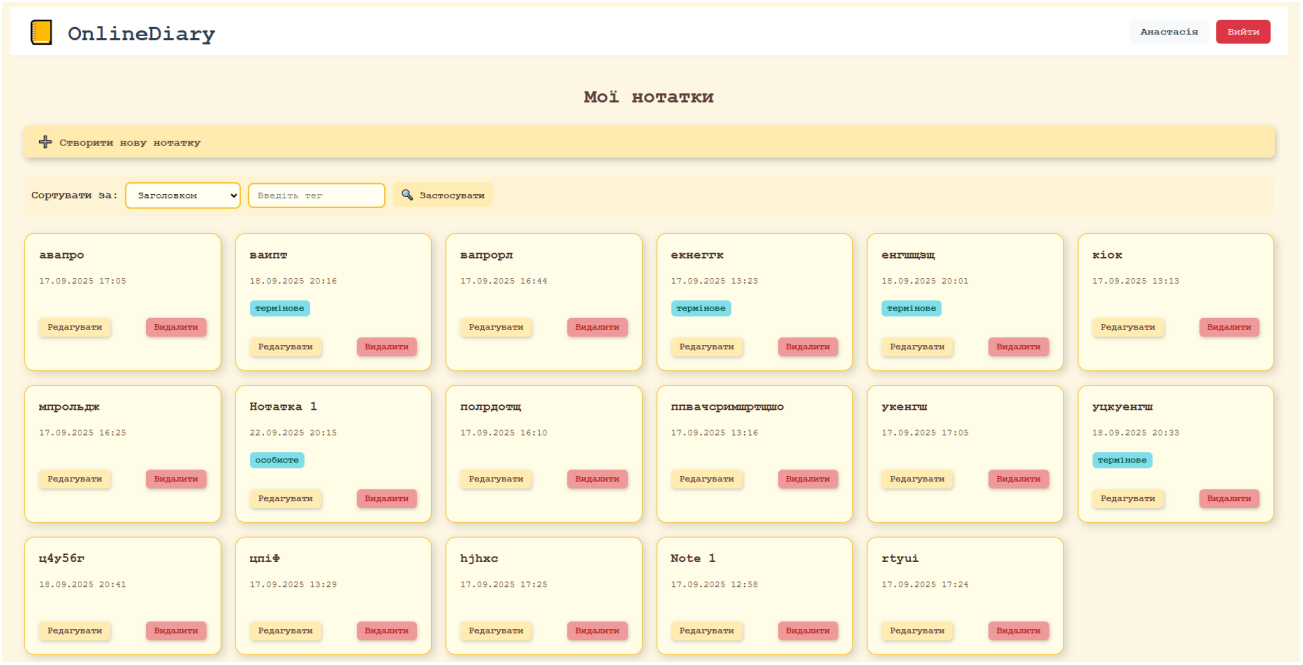


Рисунок 2 – Сортування за заголовком (в алфавітному порядку)

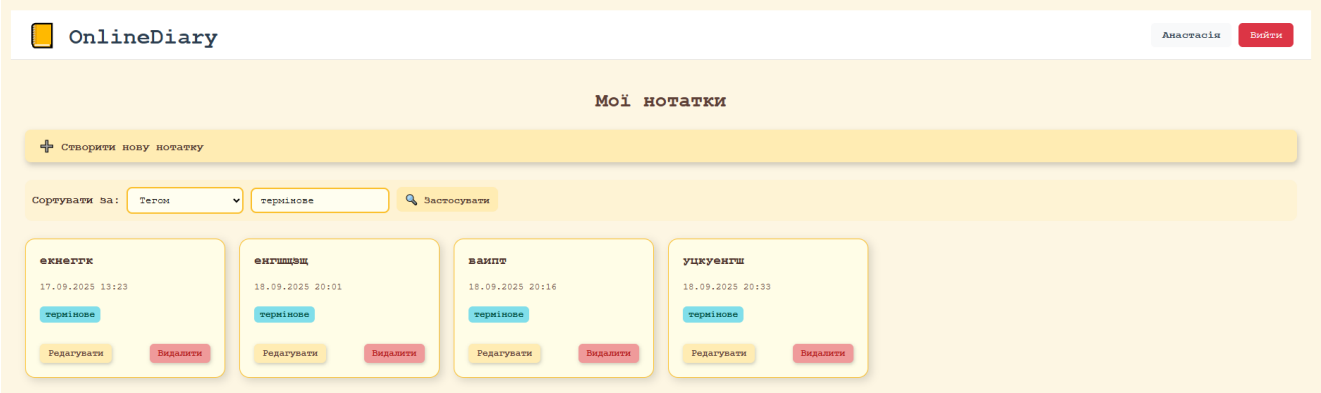


Рисунок 3 – Сортування за тегом

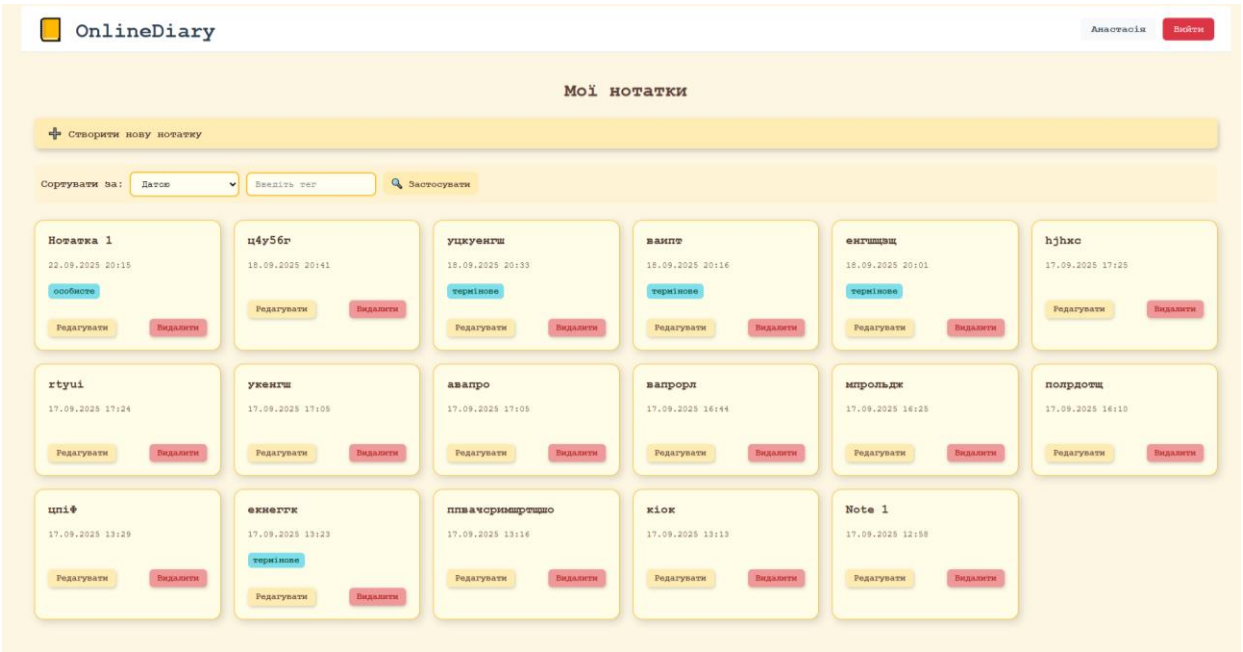


Рисунок 4 – Сортування за датою (від нових до давніх)

5. 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.

Висновки: У ході виконання лабораторної роботи було розглянуто базові шаблони проєктування, зокрема Singleton, Iterator, Proxy, State та Strategy. Отримані знання дали можливість зрозуміти, яку роль відіграють патерни у побудові гнучких і масштабованих програмних систем. На прикладі веб-застосунку «Онлайн-щоденник» було реалізовано шаблон Strategy для організації гнучкого механізму сортування нотаток за різними критеріями (датою, тегами тощо). Це дало змогу розділити бізнес-логіку і алгоритми сортування, а також легко додавати нові стратегії без змін у вже написаному коді. Таким чином, робота дозволила закріпити теоретичні знання про шаблони проєктування та на практиці переконатися в їх корисності для створення якісного програмного забезпечення.

Відповіді на контрольні питання:

1. Що таке шаблон проєктування?
   Шаблон проєктування — це типове рішення поширеної задачі проєктування програмного забезпечення, перевірене практикою.

2. Навіщо використовувати шаблони проєктування?
   Вони роблять код зрозумілішим, зручнішим для повторного використання, легшим у підтримці та розширенні.

3. Яке призначення шаблону «Стратегія»?
   Дозволяє визначати сімейство алгоритмів, інкапсулювати кожен із них та робити їх взаємозамінними під час виконання.

4. Нарисуйте структуру шаблону «Стратегія».
(Тут можна вставити діаграму, як у тебе вище: Context → Strategy → ConcreteStrategy1/2).

5. Які класи входять в шаблон «Стратегія», та яка між ними взаємодія?

- Context – клас, який використовує стратегію.
- Strategy – інтерфейс для всіх стратегій.
- ConcreteStrategy – конкретні реалізації алгоритмів.

Context тримає посилання на Strategy і виконує метод вибраної реалізації.

6. Яке призначення шаблону «Стан»?

Дозволяє об’єкту змінювати свою поведінку залежно від внутрішнього стану, ніби він змінює свій клас.

7. Нарисуйте структуру шаблону «Стан».

(Діаграма: Context → State → ConcreteStateA / ConcreteStateB).

8. Які класи входять в шаблон «Стан», та яка між ними взаємодія?

- Context – клас, який зберігає поточний стан.
- State – інтерфейс станів.
- ConcreteState – реалізації різних станів.

Context делегує виконання поточному стану, який може змінити його на інший.

8. Яке призначення шаблону «Ітератор»?

Дозволяє послідовно перебирати елементи колекції, не розкриваючи її внутрішньої структури.

10. Нарисуйте структуру шаблону «Ітератор».

(Діаграма: Iterator ↔ ConcreteIterator ↔ Aggregate ↔ ConcreteAggregate).

11. Які класи входять в шаблон «Ітератор», та яка між ними взаємодія?

- Iterator – інтерфейс для доступу до елементів.
- ConcreteIterator – конкретна реалізація.

- Aggregate – інтерфейс колекції.
- ConcreteAggregate – конкретна колекція, яка створює ітератор.

12. В чому полягає ідея шаблона «Одинак»?

Забезпечити існування лише одного екземпляра класу та надати глобальну точку доступу до нього.

13. Чому шаблон «Одинак» вважають «анти-шаблоном»?

Бо він порушує принципи ООП: ускладнює тестування, створює приховані залежності та глобальний стан.

14. Яке призначення шаблону «Проксі»?

Контролює доступ до іншого об’єкта, виступаючи його замісником (наприклад, ліниве завантаження, кешування, контроль доступу).

15. Нарисуйте структуру шаблону «Проксі».

(Діаграма: Client → Proxy → RealSubject ← Interface).

16. Які класи входять в шаблон «Проксі», та яка між ними взаємодія?

- Subject – спільний інтерфейс.
- RealSubject – реальний об’єкт.
- Proxy – замісник, який містить посилання на RealSubject і контролює звернення клієнта.