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 №6

із дисципліни «*Технології розроблення програмного забезпечення*»

Тема: «Патерни проектування»

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

Перевірив:
Мягкий М.Ю.

Київ-2025

Тема: Патерни 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 «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator» та навчитися застосовувати їх в реалізації програмної системи.

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

https://github.com/Sameliuk/OnlineDiary_trpz/tree/reports

Хід роботи

1. Ознайомитись з короткими теоретичними відомостями.

   Шаблон «Abstract Factory»

   Призначений для створення сімейств пов’язаних об’єктів без зазначення їх конкретних класів.

   Використовується, коли потрібно забезпечити узгодженість елементів, які належать до одного стилю або типу.

   Приклад — створення об’єктів різних стилів у грі (стіни, двері, меблі).

   Переваги: узгодженість об’єктів, легке розширення, відокремлення створення від використання.

   Недоліки: складність коду та труднощі при додаванні нових типів продуктів.

   Шаблон «Factory Method»

   Визначає інтерфейс для створення об’єктів певного типу, дозволяючи підкласам вирішувати, який об’єкт створювати.

   Підходить, коли необхідно розширити систему новими типами без зміни існуючого коду.

Переваги: гнучкість, розширюваність, спрощене тестування.

Недоліки: може призвести до розростання ієрархій класів.

Шаблон «Memento» (Знімок)

Дозволяє зберігати і відновлювати стан об’єкта без порушення інкапсуляції. Стан зберігається у спеціальному об’єкті-знімку, який знає лише вихідний об’єкт.

Переваги: не порушує інкапсуляцію, спрощує реалізацію відкату дій.

Недоліки: збільшує використання пам’яті.

Шаблон «Observer» (Спостерігач)

Визначає залежність типу «один-до-багатьох», коли зміна стану одного об’єкта повідомляє всі підписані об’єкти.

Приклад — підписка на канал або повідомлення про зміну даних.

Переваги: слабке зв’язування об’єктів, динамічне додавання спостерігачів.

Недоліки: не визначається порядок сповіщення.

Шаблон «Decorator» (Декоратор)

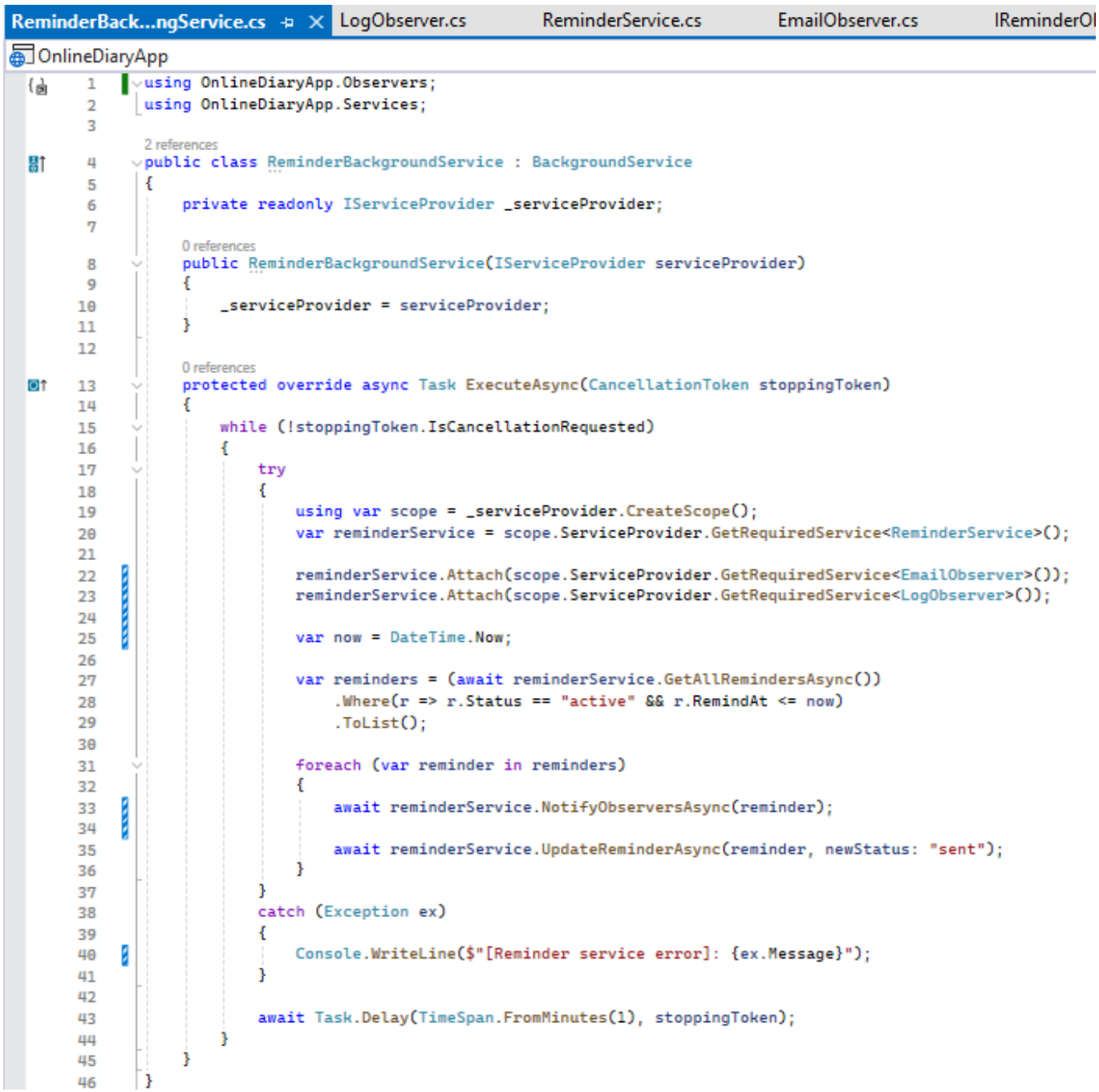
Дозволяє динамічно додавати функціональність об’єктам без зміни їхнього коду. Декоратор «обгортає» базовий об’єкт і додає нову поведінку.

Переваги: більша гнучкість, можливість додавання поведінки під час виконання.

Недоліки: багато дрібних класів, складність при багаторівневому обгортанні.

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

В даній частині було реалізовану логіку з нотатками та відредаговано логіку з нагадуваннями. Було реалізовано відповідні сервіси, контролери та інтерфейси для виконання певного функціоналу.



Код 1 – ReminderBackgroundService (Services)

ReminderBackgroundService - це сервіс, який працює у фоновому режимі та відповідає за перевірку активних нагадувань у задані інтервали часу. Таким чином, ReminderBackgroundService забезпечує автоматичне виконання нагадувань без участі користувача, підтримуючи постійний моніторинг і інтеграцію з іншими модулями системи (ReminderService, EmailService тощо).

Create.cshtml* | ReminderBackgroungService.cs | LogObserver.cs | Em

OnlineDiaryApp

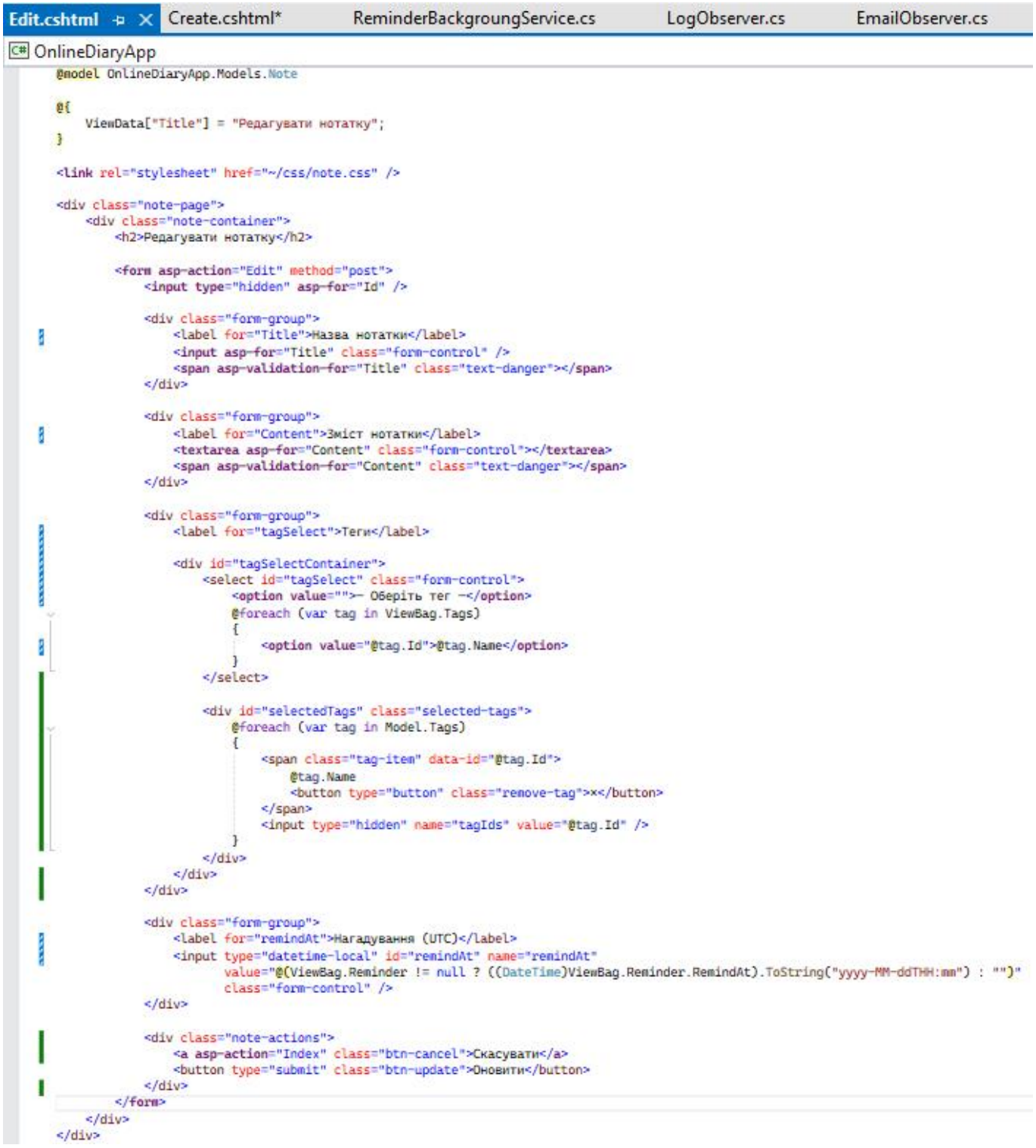
```
@model OnlineDiaryApp.Models.Note
@{
    ViewData["Title"] = "Створити нотатку";
}

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

<div class="note-page">
    <div class="note-container">
        <h2>Створити нотатку</h2>

        <form asp-action="Create" method="post">
            <div class="form-group">
                <label>Заголовок</label>
                <input type="text" name="title" class="form-control" required />
            </div>

            <div class="form-group">
                <label>Текст</label>
                <textarea name="content" class="form-control" required></textarea>
            </div>


            <div class="form-group">
                <label for="tagSelect">Теги</label>

                <div id="tagSelectContainer">
                    <select id="tagSelect" class="form-control">
                        <option value="">– Оберіть тег –</option>
                        @foreach (var tag in ViewBag.Tags)
                        {
                            <option value="@tag.Id">@tag.Name</option>
                        }
                    </select>

                    <div id="selectedTags" class="selected-tags">
                        @if (Model != null && Model.Tags != null)
                        {
                            foreach (var tag in Model.Tags)
                            {
                                <span class="tag-item" data-id="@tag.Id">
                                    @tag.Name
                                    <button type="button" class="remove-tag">×</button>
                                </span>
                                <input type="hidden" name="tagIds" value="@tag.Id" />
                            }
                        }
                    </div>

                </div>
            </div>

            <div class="form-group">
                <label>Нагадування</label>
                <input type="datetime-local" name="reminderDate" class="form-control" />
            </div>

            <div class="note-actions">
                <a asp-action="Index" class="btn-cancel">Скасувати</a>
            <button type="submit" class="btn-save">Зберегти</button>
            </div>
        </form>
    </div>
</div>
```

Код 2.1 – Create (Views.Note)

```
<script>
    document.addEventListener('DOMContentLoaded', function () {
        const tagSelect = document.getElementById('tagSelect');
        const selectedTagsContainer = document.getElementById('selectedTags');

        tagSelect.addEventListener('change', function () {
            const selectedValue = this.value;
            const selectedText = this.options[this.selectedIndex].text;

            if (!selectedValue) return;

            if (selectedTagsContainer.querySelector(`[data-id="${selectedValue}"]`)) {
                this.value = "";
                return;
            }

            const tagEl = document.createElement('span');
            tagEl.className = 'tag-item';
            tagEl.dataset.id = selectedValue;
            tagEl.innerHTML = `${selectedText} <button type="button" class="remove-tag">×</button>`;

            const hiddenInput = document.createElement('input');
            hiddenInput.type = 'hidden';
            hiddenInput.name = 'tagIds';
            hiddenInput.value = selectedValue;

            selectedTagsContainer.appendChild(tagEl);
            selectedTagsContainer.appendChild(hiddenInput);

            this.value = "";
        });

        selectedTagsContainer.addEventListener('click', function (e) {
            if (e.target.classList.contains('remove-tag')) {
                const tagEl = e.target.closest('.tag-item');
                const tagId = tagEl.dataset.id;

                tagEl.remove();
                selectedTagsContainer
                    .querySelector(`input[value="${tagId}"]`)
                    ?.remove();
            }
        });
    });
</script>
```

Код 2.2 – Create (Views.Note)

Create (Views.Note) — це представлення (view), яке відповідає за створення нової нотатки користувачем у застосунку. Таким чином, Create (Views.Note) забезпечує інтерактивне створення нотаток і є стартовою точкою для подальшої роботи з ними — редагування, перегляду чи нагадування.

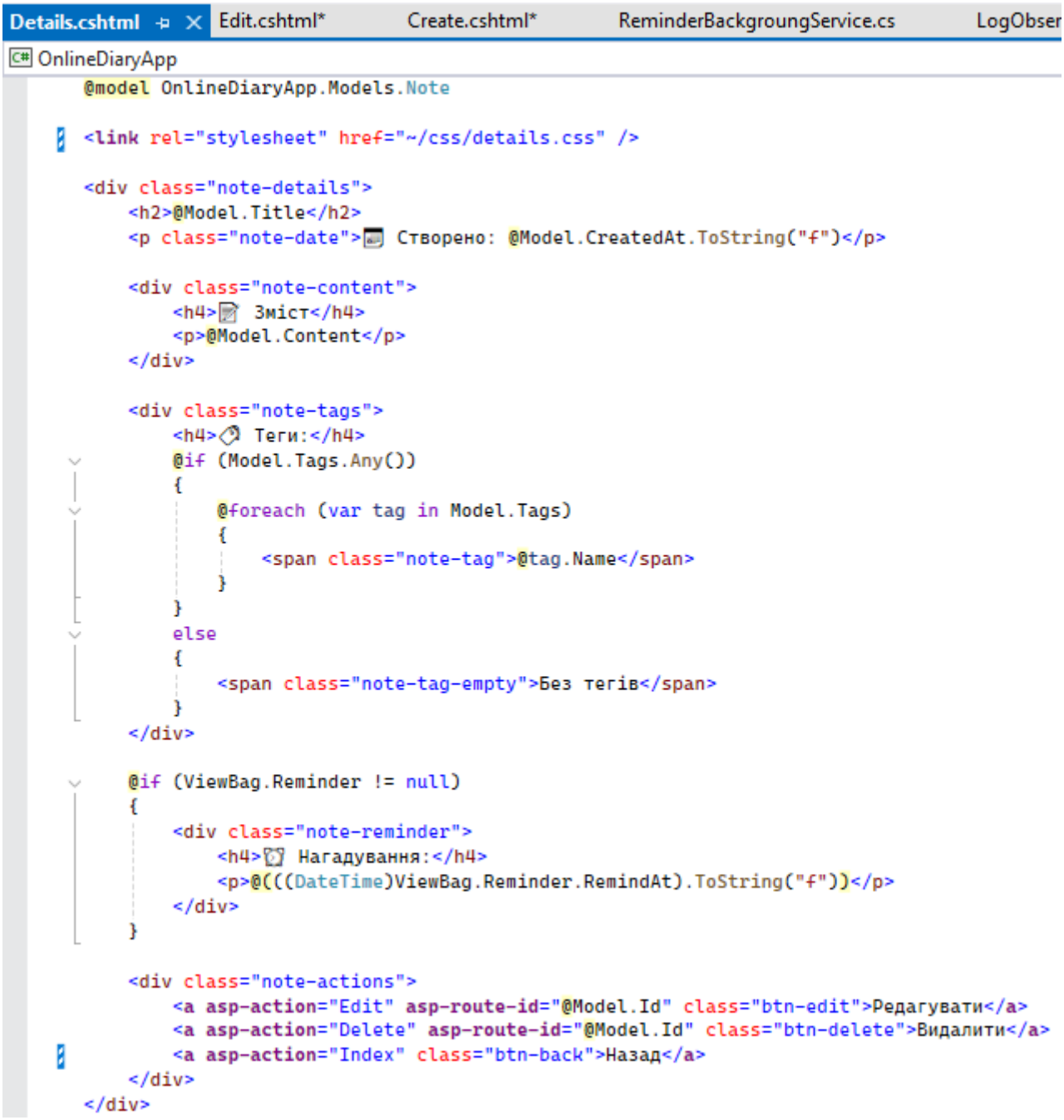


Код 3.1 – Edit (Views.Note)

```
<script>
    document.addEventListener('DOMContentLoaded', function () {
        const tagSelect = document.getElementById('tagSelect');
        const selectedTagsContainer = document.getElementById('selectedTags');

        tagSelect.addEventListener('change', function () {
            const selectedValue = this.value;
            const selectedText = this.options[this.selectedIndex].text;

            if (!selectedValue) return;

            if (selectedTagsContainer.querySelector(`[data-id="${selectedValue}"]`)) {
                this.value = "";
                return;
            }

            const tagEl = document.createElement('span');
            tagEl.className = 'tag-item';
            tagEl.dataset.id = selectedValue;
            tagEl.innerHTML = `${selectedText} <button type="button" class="remove-tag">×</button>`;

            const hiddenInput = document.createElement('input');
            hiddenInput.type = 'hidden';
            hiddenInput.name = 'tagIds';
            hiddenInput.value = selectedValue;

            selectedTagsContainer.appendChild(tagEl);
            selectedTagsContainer.appendChild(hiddenInput);

            this.value = "";
        });

        selectedTagsContainer.addEventListener('click', function (e) {
            if (e.target.classList.contains('remove-tag')) {
                const tagEl = e.target.closest('.tag-item');
                const tagId = tagEl.dataset.id;

                tagEl.remove();
                selectedTagsContainer
                    .querySelector(`input[value="${tagId}"]`)
                    ?.remove();
            }
        });
    });
</script>
```

Код 3.2 – Edit (Views.Note)

Edit (Views.Note) — це представлення (view), яке забезпечує редагування вже створеної нотатки. Отже, Edit (Views.Note) реалізує можливість корекції вмісту та параметрів нотатки, забезпечуючи зручність у підтримці актуальності записів користувача.

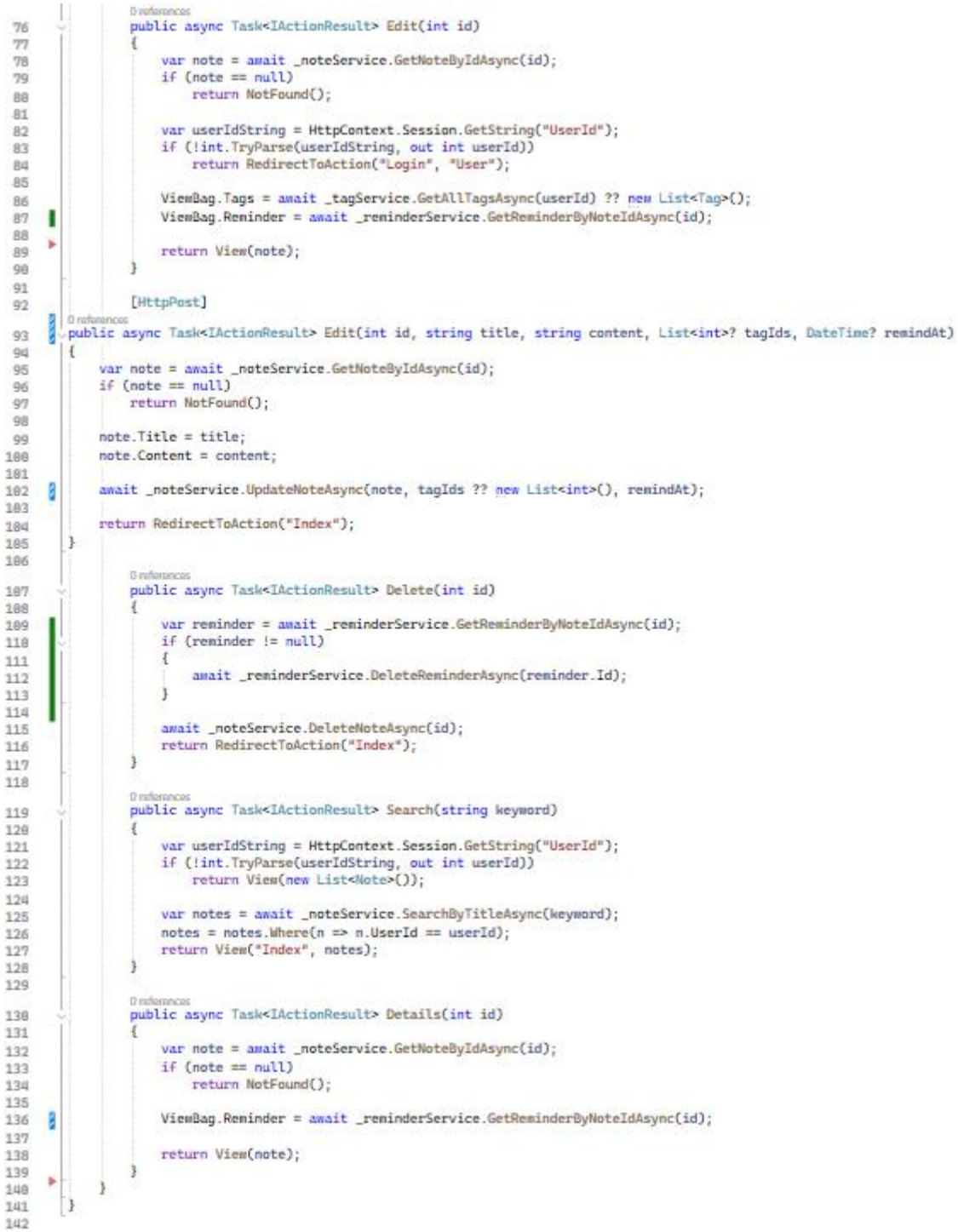


Код 4 – Details (Views.Note)

Details (Views.Note) — це подання, яке призначене для перегляду детальної інформації про конкретну нотатку. Отже, Details (Views.Note) забезпечує читання та ознайомлення з вибраною нотаткою без можливості редагування, що є важливою частиною інтерфейсу для роботи з особистими записами.


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

namespace OnlineDiaryApp.Controllers
{
    1 reference
    public class NoteController : Controller
    {
        private readonly NoteService _noteService;
        private readonly ReminderService _reminderService;
        private readonly TagService _tagService;

        0 references
        public NoteController(NoteService noteService, ReminderService reminderService, TagService tagService)
        {
            _noteService = noteService;
            _reminderService = reminderService;
            _tagService = tagService;
        }

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

            ISortStrategy? strategy = null;
            if (!string.IsNullOrEmpty(sortBy))
            {
                strategy = sortBy.ToLower() switch
                {
                    "date" => new SortByDateStrategy(),
                    "tag" when !string.IsNullOrEmpty(tag) => new SortByTagStrategy(tag),
                    "title" => new SortByTitleStrategy(),
                    _ => null
                };
            }

            var notes = await _noteService.GetAllNotesByUserAsync(userId, strategy);

            ViewBag.SortBy = sortBy;
            ViewBag.SelectedTag = tag;
            ViewBag.Tags = await _tagService.GetAllTagsAsync(userId);

            return View(notes);
        }

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

            var tags = await _tagService.GetAllTagsAsync(userId);
            ViewBag.Tags = tags ?? new List<Tag>();

            return View();
        }

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

            var note = await _noteService.CreateNoteAsync(title, content, userId, tagIds ?? new List<int>());

            if (reminderDate.HasValue)
            {
                await _reminderService.CreateReminderAsync(note.Id, reminderDate.Value, userId);
            }

            return RedirectToAction("Index");
        }
```

Код 5.1 – NoteController (Controllers)



```
        0 references
        public async Task<IActionResult> Edit(int id)
        {
            var note = await _noteService.GetNoteByIdAsync(id);
            if (note == null)
                return NotFound();

            var userIdString = HttpContext.Session.GetString("UserId");
            if (!int.TryParse(userIdString, out int userId))
                return RedirectToAction("Login", "User");

            ViewBag.Tags = await _tagService.GetAllTagsAsync(userId) ?? new List<Tag>();
            ViewBag.Reminder = await _reminderService.GetReminderByNoteIdAsync(id);

            return View(note);
        }

        [HttpPost]
0 references
public async Task<IActionResult> Edit(int id, string title, string content, List<int>? tagIds, DateTime? remindAt)
{
    var note = await _noteService.GetNoteByIdAsync(id);
    if (note == null)
        return NotFound();

    note.Title = title;
    note.Content = content;

    await _noteService.UpdateNoteAsync(note, tagIds ?? new List<int>(), remindAt);

    return RedirectToAction("Index");
}

        0 references
        public async Task<IActionResult> Delete(int id)
        {
            var reminder = await _reminderService.GetReminderByNoteIdAsync(id);
            if (reminder != null)
            {
                await _reminderService.DeleteReminderAsync(reminder.Id);
            }

            await _noteService.DeleteNoteAsync(id);
            return RedirectToAction("Index");
        }

        0 references
        public async Task<IActionResult> Search(string keyword)
        {
            var userIdString = HttpContext.Session.GetString("UserId");
            if (!int.TryParse(userIdString, out int userId))
                return View(new List<Note>());

            var notes = await _noteService.SearchByTitleAsync(keyword);
            notes = notes.Where(n => n.UserId == userId);
            return View("Index", notes);
        }

        0 references
        public async Task<IActionResult> Details(int id)
        {
            var note = await _noteService.GetNoteByIdAsync(id);
            if (note == null)
                return NotFound();

            ViewBag.Reminder = await _reminderService.GetReminderByNoteIdAsync(id);

            return View(note);
        }
    }
}
```

Код 5.2 – NoteController (Controllers)

NoteController — це контролер, який відповідає за керування логікою роботи з нотатками у додатку. Його основне призначення — обробляти HTTP-запити, пов’язані зі створенням, переглядом, редагуванням та видаленням нотаток, а також координувати взаємодію між моделлю, представленнями (Views) та сервісами. Отже, NoteController виступає посередником між користувачем і бізнес-логікою,

забезпечуючи правильну обробку дій у системі та передачу даних до відповідних представлень.

3. Реалізувати один з розглянутих шаблонів за обраною темою.

Для даного проєкту було обрано патерн Observer. Шаблон реалізований таким чином:

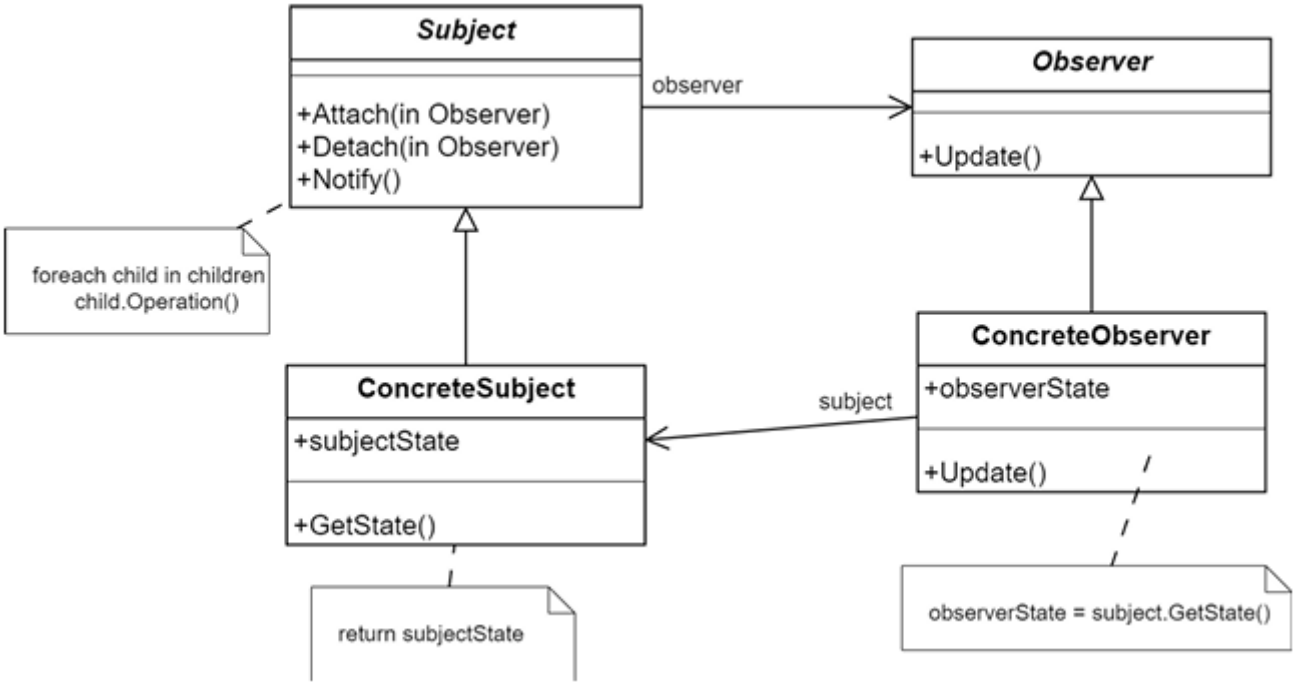


Рис.1 – Структура патерну Спостерігач

Subject: IReminderSubject – це інтерфейс, який визначає контракт для всіх суб'єктів спостереження. Він описує методи: Attach(IReminderObserver observer) – підключає нового спостерігача, Detach(IReminderObserver observer) – відключає спостерігача, NotifyObserversAsync(Reminder reminder, string action) – надсилає сповіщення всім спостерігачам про зміну стану. Завдяки інтерфейсу IReminderSubject можна створювати різні реалізації механізму сповіщення (наприклад, для нагадувань, подій, завдань тощо), не змінюючи код клієнтів, які з ним працюють.

ConcreteSubject: ReminderSubject – це конкретна реалізація інтерфейсу IReminderSubject. Він зберігає список об'єктів-спостерігачів (IReminderObserver) і відповідає за: додавання та видалення спостерігачів, розсилку повідомлень усім підписникам при зміні стану нагадування. Сам ReminderSubject не виконує бізнес-логіку нагадувань — він лише координує сповіщення спостерігачів про події ("created", "updated", "deleted", "time_reached").
Observer: IReminderObserver – це інтерфейс, який задає єдиний метод:
Task OnReminderChangedAsync(Reminder reminder, string action)
Цей метод викликається суб'єктом (ReminderSubject) у момент, коли змінюється стан нагадування. Кожен клас, що реалізує IReminderObserver, сам вирішує, як реагувати на ці зміни (логування, надсилання пошти, оновлення UI тощо).

ConcreteObservers: EmailObserver – реалізація спостерігача, який відправляє електронні листи користувачам, коли створюється, оновлюється або спрацьовує нагадування. Він використовує сервіс IEmailSender для відправлення повідомлень, але не знає, як саме це робиться (через SMTP, API тощо).
LogObserver – реалізація спостерігача, який записує інформаційні повідомлення про події нагадувань у консоль (журнал подій). Обидва спостерігачі підписуються на ті самі події, але реагують по-різному.

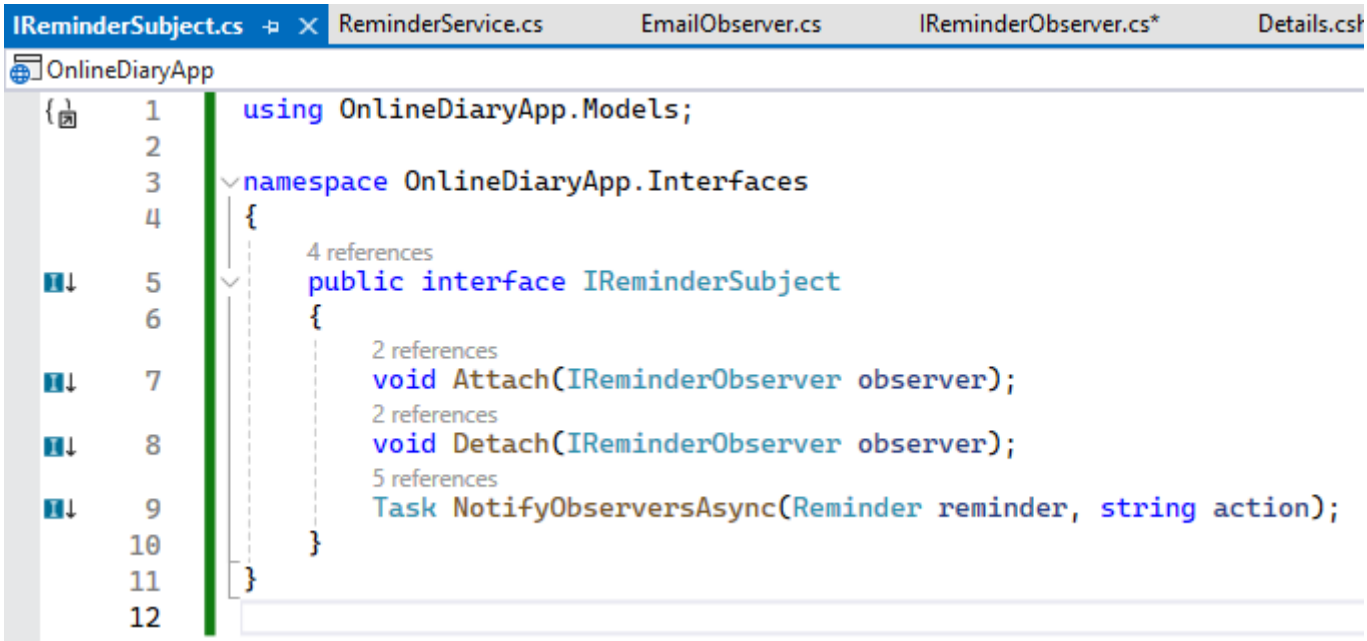
Client: ReminderService – це клієнт, який керує створенням, оновленням і видаленням нагадувань.
Він не реалізує власний механізм сповіщення, а використовує IReminderSubject для інформування зареєстрованих спостерігачів. Завдяки цьому ReminderService не залежить від конкретних дій, що виконуються при сповіщенні (наприклад, відправлення пошти чи логування). Він лише "говорить": "нагадування створено", а спостерігачі самі вирішують, що робити далі.

Завдяки патерну Observer система нагадувань стала гнучкою та розширюваною:

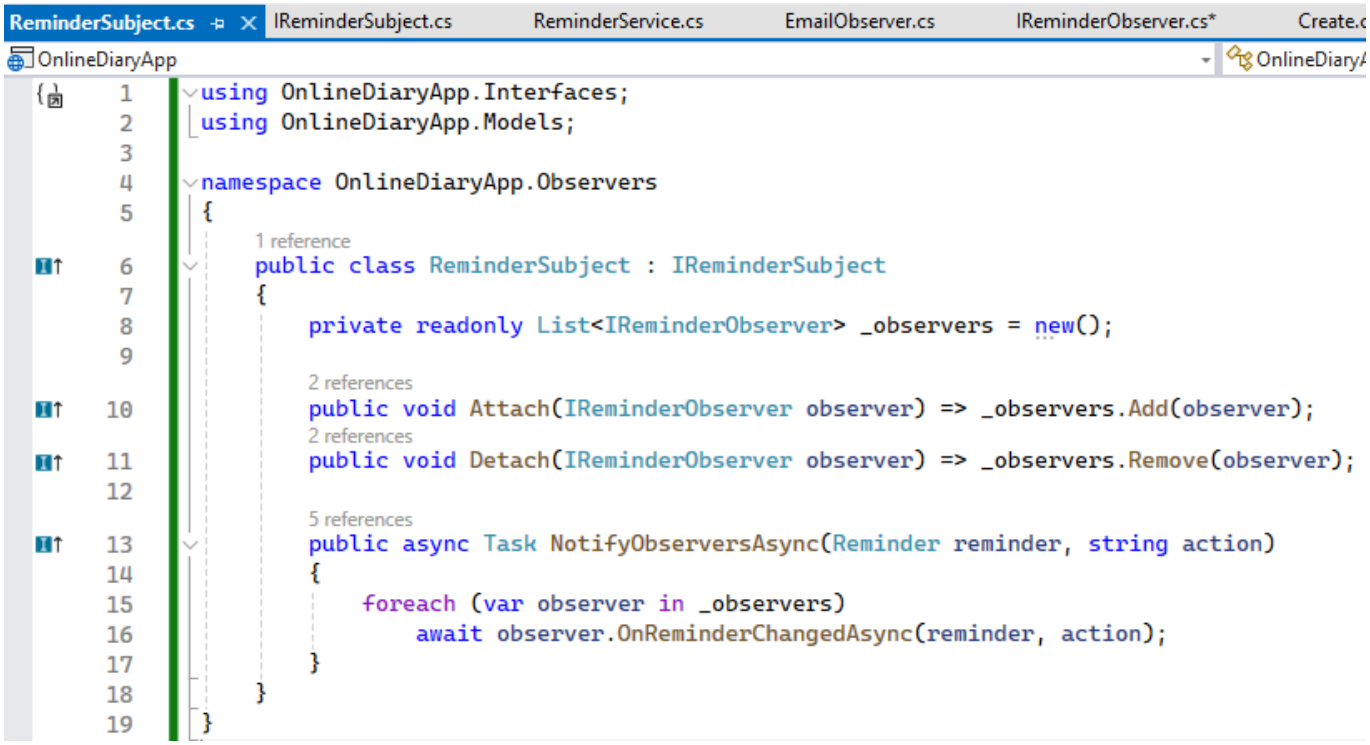
- можна легко додавати нові типи спостерігачів (наприклад, SMS-нагадування або push-повідомлення) без зміни коду ReminderService;
- різні модулі реагують на події незалежно один від одного;
- зберігається слабке зв'язування між компонентами — сервіс нагадувань не залежить від реалізації обробників подій.

Це і є суть патерну "Спостерігач" — встановлення механізму "підписки-сповіщення", де об'єкти автоматично реагують на зміни стану іншого об'єкта без жорсткої залежності від нього.

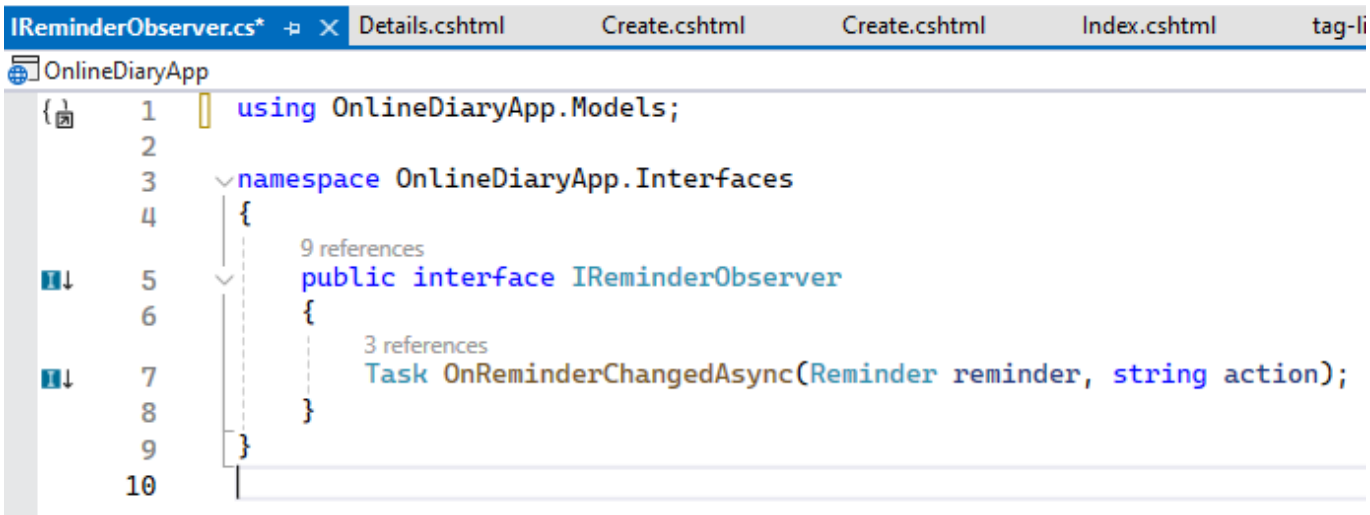
4. Реалізувати не менше 3-х класів відповідно до обраної теми.



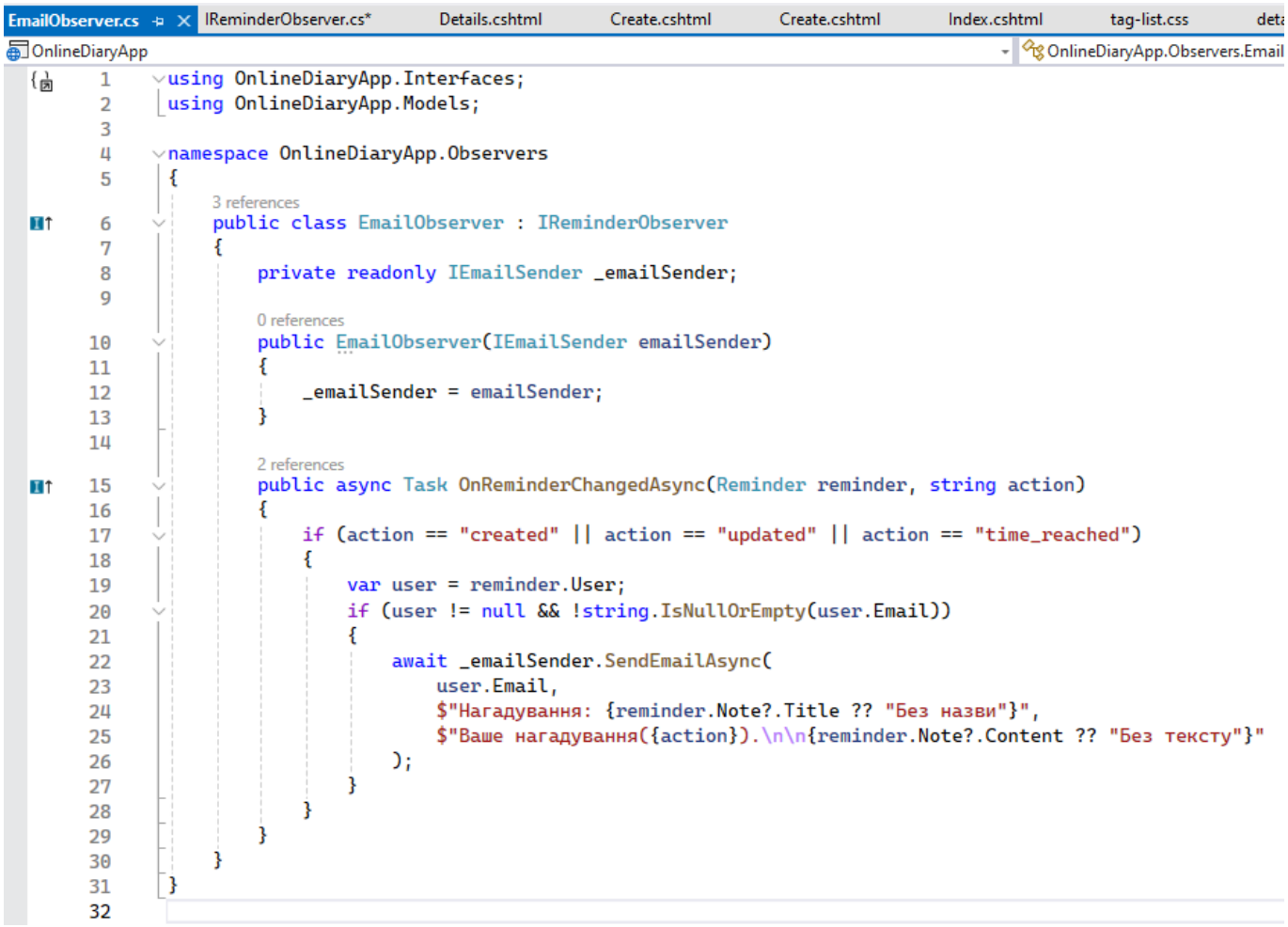
Код 4 – IReminderSubject (Interfaces)



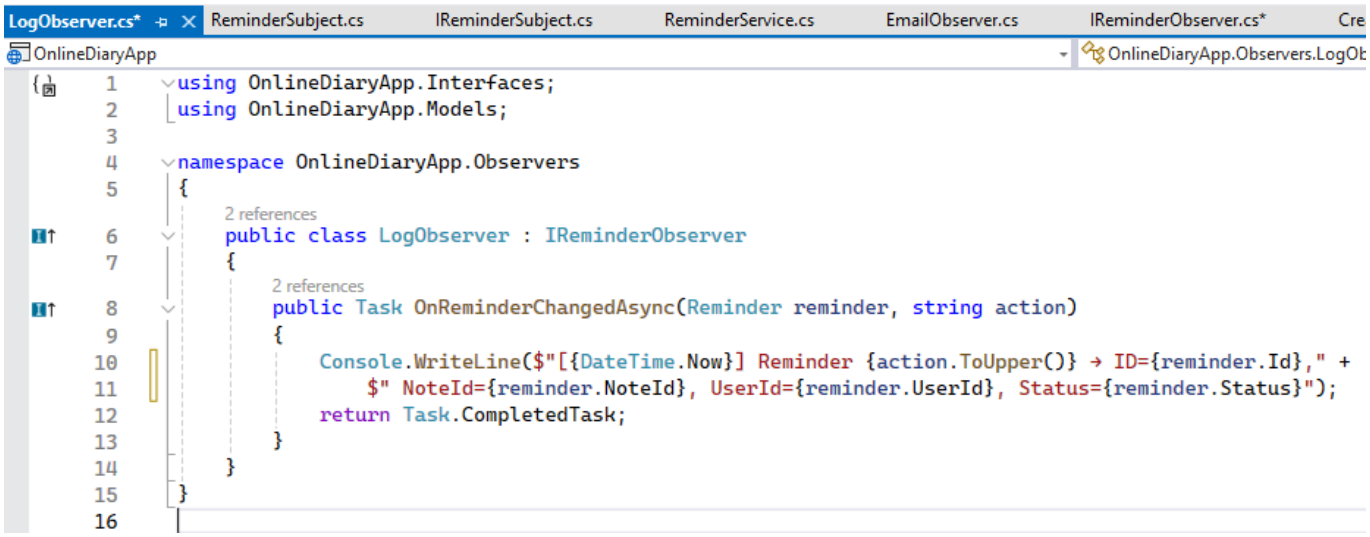
Код 5 – ReminderSubject (Observers)



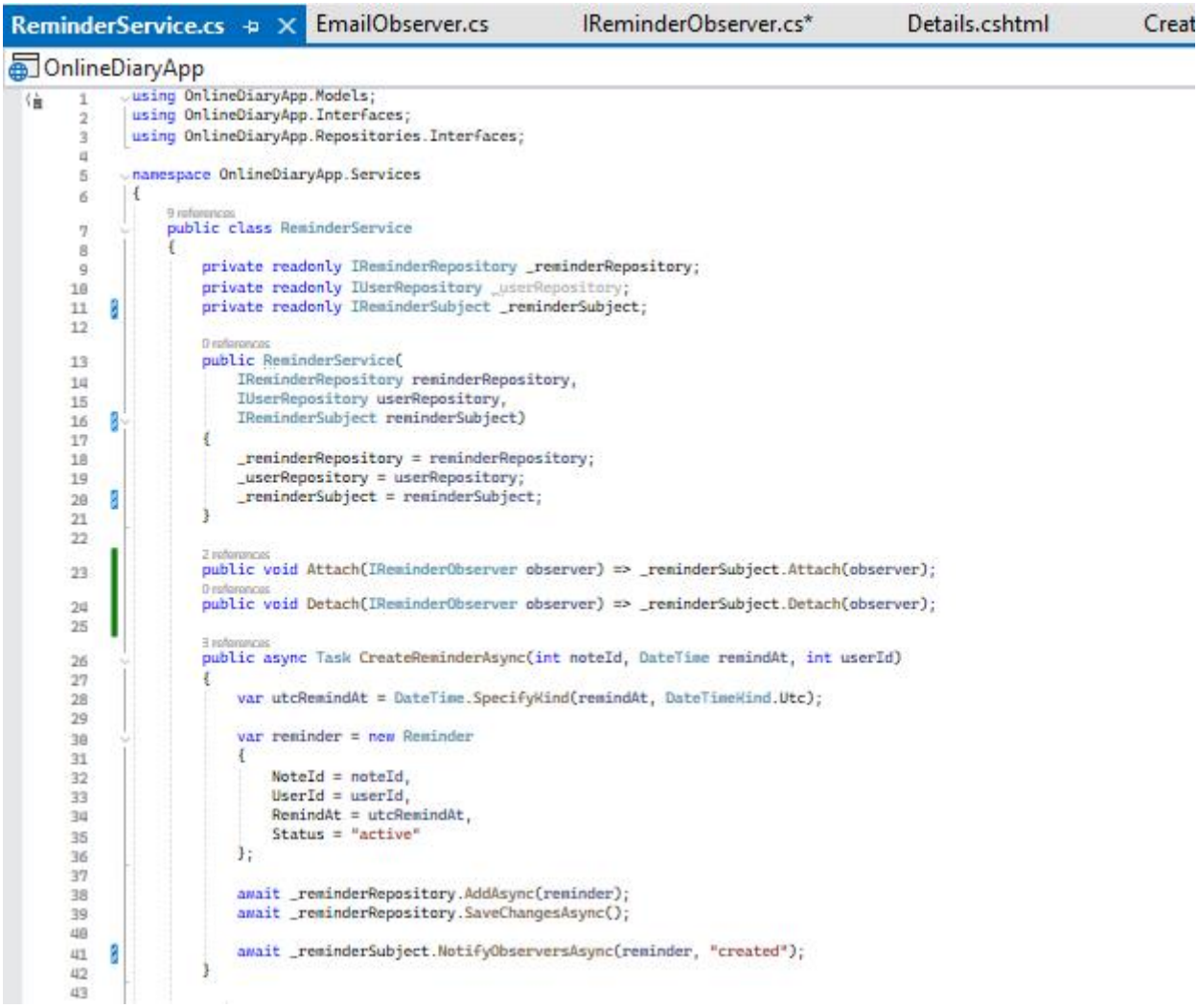
Код 6 – IReminderObserver (Interfaces)



Код 7 – EmailObserver (Observers)



Код 8 – LogObserver (Observers)




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

namespace OnlineDiaryApp.Services
{
    9 references
    public class ReminderService
    {
        private readonly IReminderRepository _reminderRepository;
        private readonly IUserRepository _userRepository;
        private readonly IReminderSubject _reminderSubject;

        0 references
        public ReminderService(
            IReminderRepository reminderRepository,
            IUserRepository userRepository,
            IReminderSubject reminderSubject)
        {
            _reminderRepository = reminderRepository;
            _userRepository = userRepository;
            _reminderSubject = reminderSubject;
        }

        2 references
        public void Attach(IReminderObserver observer) => _reminderSubject.Attach(observer);
        0 references
        public void Detach(IReminderObserver observer) => _reminderSubject.Detach(observer);

        3 references
        public async Task CreateReminderAsync(int noteId, DateTime remindAt, int userId)
        {
            var utcRemindAt = DateTime.SpecifyKind(remindAt, DateTimeKind.Utc);

            var reminder = new Reminder
            {
                NoteId = noteId,
                UserId = userId,
                RemindAt = utcRemindAt,
                Status = "active"
            };

            await _reminderRepository.AddAsync(reminder);
            await _reminderRepository.SaveChangesAsync();

            await _reminderSubject.NotifyObserversAsync(reminder, "created");
        }
```

Код 9.1 – ReminderService (Services)

```
2 references
public async Task UpdateReminderAsync(Reminder reminder, DateTime? newRemindAt = null, string? newStatus = null)
{
    if (newRemindAt.HasValue)
        reminder.RemindAt = DateTime.SpecifyKind(newRemindAt.Value, DateTimeKind.Utc);

    if (!string.IsNullOrEmpty(newStatus))
        reminder.Status = newStatus;

    await _reminderRepository.UpdateAsync(reminder);
    await _reminderRepository.SaveChangesAsync();

    await _reminderSubject.NotifyObserversAsync(reminder, "updated");
}

1 reference
public async Task<IEnumerable<Reminder>> GetAllRemindersAsync() =>
    await _reminderRepository.GetAllAsync();

4 references
public async Task<Reminder?> GetReminderByNoteIdAsync(int noteId) =>
    await _reminderRepository.GetByNoteIdAsync(noteId);

2 references
public async Task DeleteReminderAsync(int reminderId)
{
    var reminder = await _reminderRepository.GetByIdAsync(reminderId);
    if (reminder != null)
    {
        await _reminderRepository.DeleteAsync(reminder.Id);
        await _reminderRepository.SaveChangesAsync();

        await _reminderSubject.NotifyObserversAsync(reminder, "deleted");
    }
}

1 reference
public async Task NotifyObserversAsync(Reminder reminder)
{
    await _reminderSubject.NotifyObserversAsync(reminder, "time_reached");
}
}
}
```

Код 9.2 – ReminderService (Services)

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 лабораторної роботи було реалізовано систему нагадувань із використанням шаблону проєктування Observer (Спостерігач). Вибір саме цього патерну виявився найбільш вдалим, оскільки він забезпечує гнучкість, масштабованість і слабке зв’язування між компонентами системи. Завдяки використанню патерну Observer система стала легко розширюваною: можна без змін у коді основного сервісу (ReminderService) додавати нові типи сповіщень — наприклад, електронну пошту, SMS або push-повідомлення; кожен спостерігач реагує на події незалежно, що підвищує модульність і стабільність системи; забезпечується чітке розділення відповідальностей між компонентами — ReminderService лише повідомляє про подію, не знаючи, хто і як її обробляє.

Таким чином, використання шаблону “Спостерігач” дозволило створити зручну, незалежну та розширювану архітектуру, у якій додавання нових функцій не вимагає змін у вже реалізованих модулях, що відповідає принципам SOLID і робить систему більш підтримуваною в майбутньому.

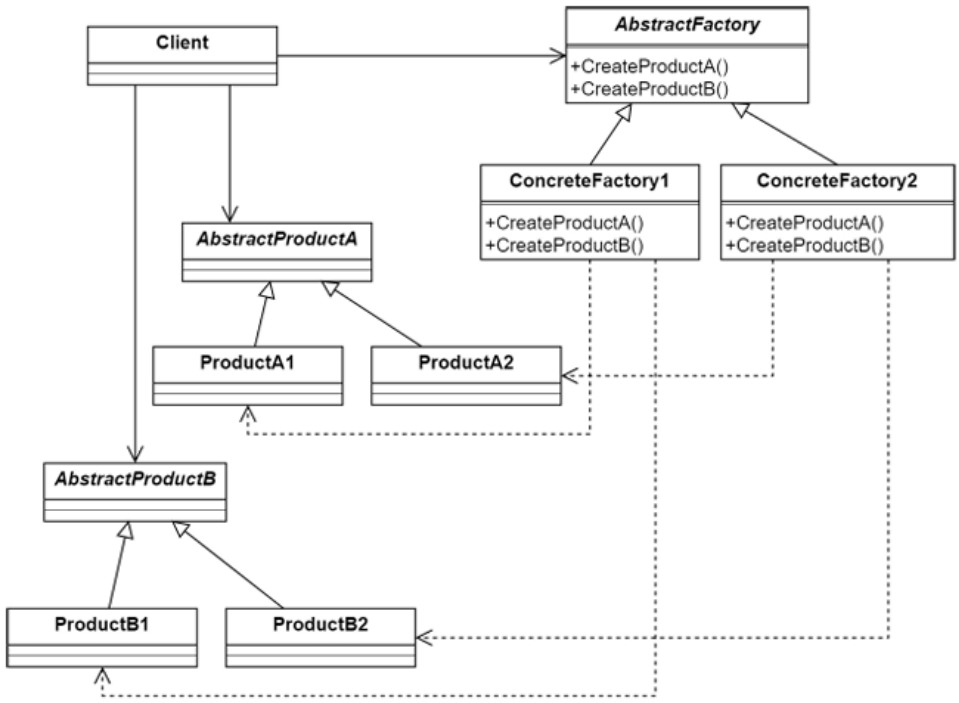
Відповіді на контрольні питання:

1. Призначення шаблону «Абстрактна фабрика»

Шаблон «Абстрактна фабрика» (Abstract Factory) призначений для створення сімейств пов’язаних об’єктів без прив’язки до конкретних класів.

Він дозволяє клієнту працювати лише з інтерфейсами, не знаючи, які саме об’єкти створюються.

2. Структура шаблону «Абстрактна фабрика»



3. Класи та їх взаємодія в «Абстрактній фабриці»

AbstractFactory – оголошує інтерфейс для створення абстрактних продуктів.

ConcreteFactory – реалізує методи створення конкретних продуктів.

AbstractProduct – описує інтерфейс продуктів.

ConcreteProduct – реалізує конкретний продукт.

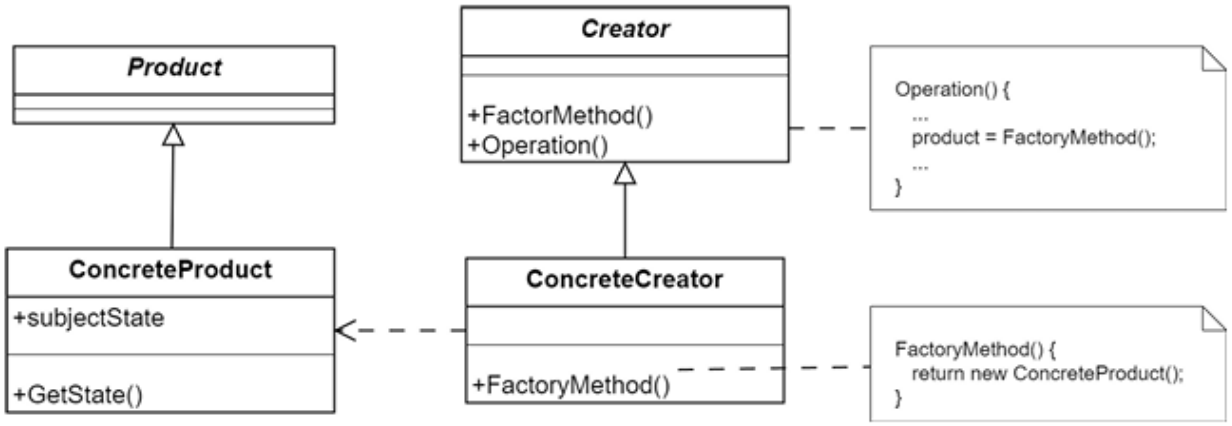
Client – працює лише з інтерфейсами AbstractFactory та AbstractProduct, не знаючи про їх реалізацію.

4. Призначення шаблону «Фабричний метод»

Шаблон **«Фабричний метод» (Factory Method)** визначає інтерфейс для створення об’єкта, але дозволяє підкласам вирішувати, який саме клас створювати.

Він делегує створення об’єктів підкласам.

5. Структура шаблону «Фабричний метод»



6. Класи та взаємодія у «Фабричному методі»

Product – спільний інтерфейс для всіх продуктів.

ConcreteProduct – конкретна реалізація продукту.

Creator – оголошує фабричний метод FactoryMethod().

ConcreteCreator – перевизначає FactoryMethod() для створення конкретного продукту.

Client – використовує об’єкт через базовий інтерфейс Product.

7. Відмінність між «Абстрактною фабрикою» та «Фабричним методом»

«Фабричний метод» визначає інтерфейс для створення одного типу об’єкта, але дозволяє підкласам вирішувати, який саме клас створювати. Тобто він

зосереджується на створенні окремого продукту, надаючи механізм розширення без зміни існуючого коду. Основна ідея полягає в тому, щоб делегувати створення об’єкта підкласам.

«Абстрактна фабрика», своєю чергою, оперує на вищому рівні — вона створює цілі сімейства пов’язаних об’єктів, які повинні використовуватися разом. Наприклад, фабрика може створювати набір інтерфейсних елементів (кнопки, меню, поля вводу), що належать до певного стилю (Windows, macOS тощо).

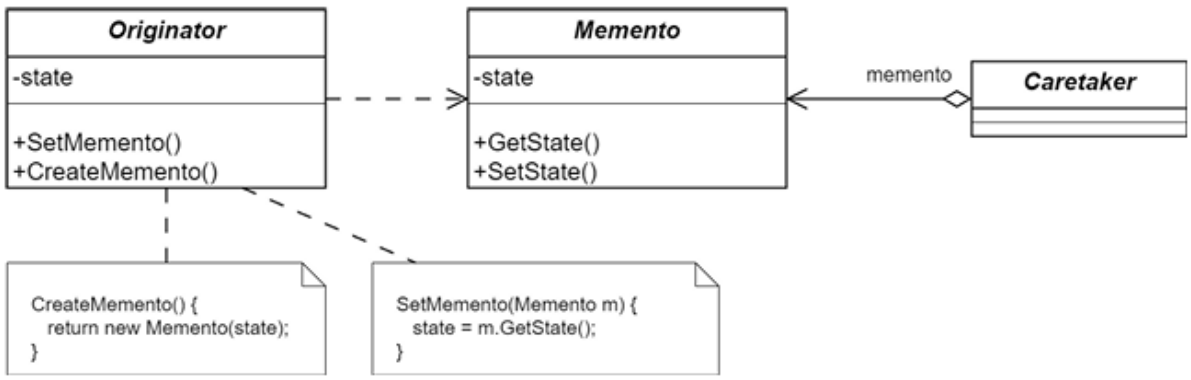
Отже, «Фабричний метод» вирішує, як створити один об’єкт, а «Абстрактна фабрика» — як створити набір сумісних **об’єктів**. Часто «Абстрактна фабрика» реалізується через кілька фабричних методів усередині себе.

8. Призначення шаблону «Знімок» (Memento)

Шаблон «Знімок» використовується для збереження і відновлення попереднього стану об’єкта без порушення інкапсуляції.

Він дозволяє “відкотити” об’єкт до попереднього стану (наприклад, у функції “Undo”).

9. Структура шаблону «Знімок»



10. Класи та взаємодія у «Знімку»

Originator – об’єкт, стан якого потрібно зберегти. Створює Memento і відновлює стан із нього.

Memento – зберігає внутрішній стан Originator.

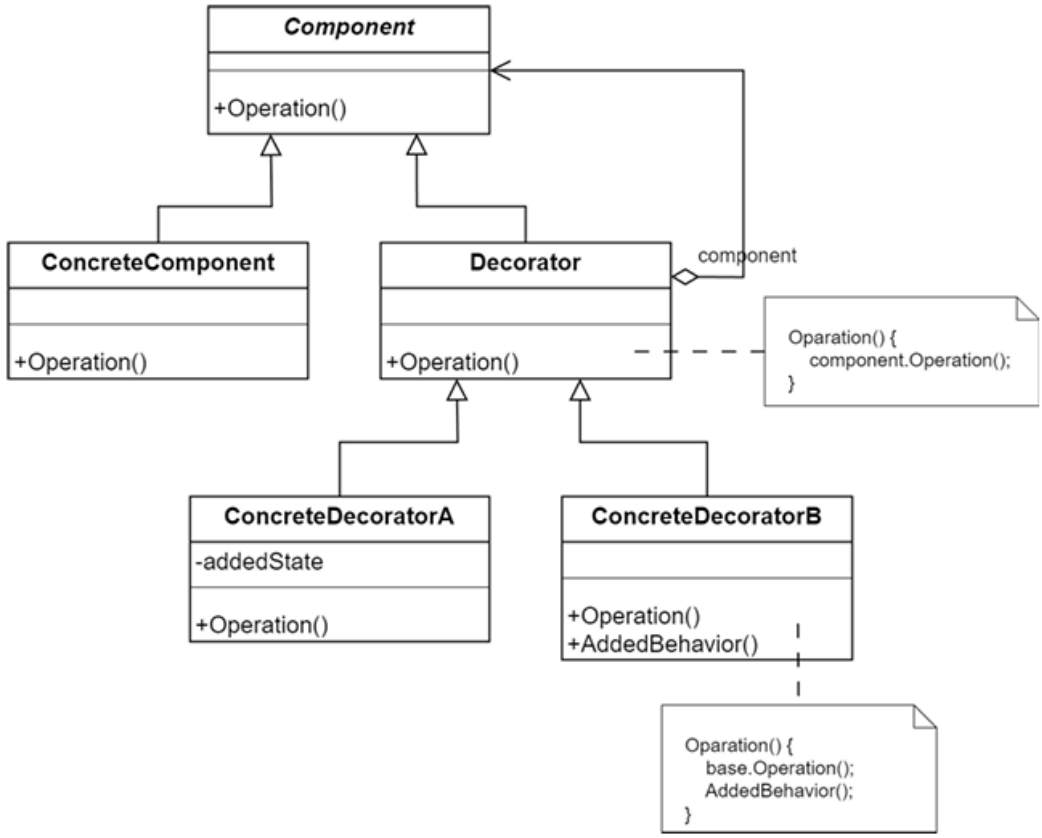
Caretaker – відповідає за збереження та відновлення Memento, але не змінює його вміст.

11. Призначення шаблону «Декоратор»

Шаблон «Декоратор» (Decorator) дозволяє динамічно додавати нову поведінку або функціональність об’єктам без зміни їх коду.

Він “обгортає” об’єкт у додатковий клас, який розширює його можливості.

12. Структура шаблону «Декоратор»



13. Класи та взаємодія у «Декораторі»

Component – базовий інтерфейс або клас.

ConcreteComponent – основний об’єкт, який може бути “обгорнутий”.

Decorator – базовий клас, який реалізує інтерфейс Component і містить посилання на нього.

ConcreteDecorator – додає нову поведінку до об’єкта перед або після виклику базових методів.

14. Обмеження використання шаблону «Декоратор»

- Велика кількість маленьких об’єктів ускладнює відлагодження.
- Може бути важко зрозуміти послідовність викликів при вкладених декораторах.
- Не підходить, коли потрібно змінювати поведінку всіх об’єктів класу одразу (краще використати наслідування).