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 №7

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 «Mediator», «Facade», «Bridge», «Template method» 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 «Mediator» (Посередник) використовується для централізації взаємодії між об’єктами через окремий об’єкт-посередник. Замість прямого зв’язку об’єкти взаємодіють через посередника, що спрощує структуру системи, зменшує зв’язність між компонентами і полегшує внесення змін. Кожен об’єкт зберігає посилання на медіатор і через нього відпрацьовує логіку взаємодії. Основна перевага – зрозуміліша організація комунікацій та легке розширення системи, недолік – медіатор може стати «God Object» при надмірній складності.

   Шаблон «Facade» (Фасад) створює єдиний спрощений інтерфейс доступу до складної підсистеми. Він приховує внутрішню структуру підсистеми та дозволяє змінювати її реалізацію без впливу на клієнтський код. Фасад полегшує використання підсистеми, зменшує «спагеті код», але знижує гнучкість прямого доступу до внутрішніх компонентів.

Шаблон «Bridge» (Міст) розділяє абстракцію і реалізацію, дозволяючи змінювати їх незалежно. Абстракція делегує виконання методів об’єкту реалізації. Це зменшує кількість перехресних підкласів при комбінації різних абстракцій і реалізацій, підвищує гнучкість та полегшує супровід. Недолік – введення додаткових проміжних рівнів підвищує складність структури.

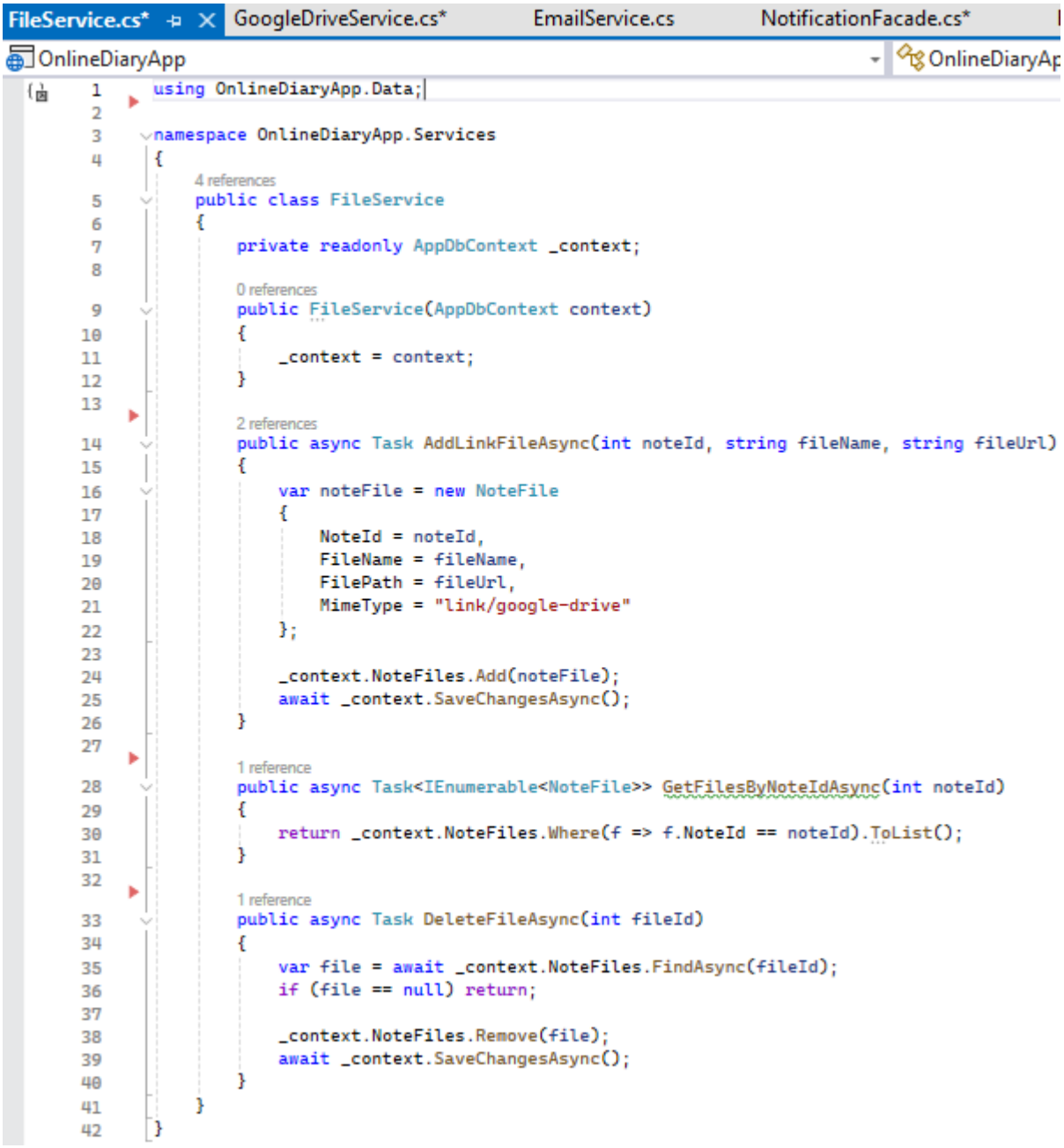
Шаблон «Template Method» (Шаблонний метод) визначає загальний алгоритм у базовому класі, залишаючи частини реалізації підкласам. Це дозволяє повторно використовувати код і спрощує підтримку, оскільки загальна логіка залишається в базовому класі, а специфічні кроки реалізуються у підкласах. Недоліки – обмеженість скелета алгоритму, можливість порушення принципу підстановки Лісков і ускладнення підтримки при великій кількості віртуальних методів.

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

В даній частині було реалізовану логіку з додавання файлів та інтеграція з GoogleDrive. Було реалізовано відповідні сервіси, контролери та інтерфейси для виконання певного функціоналу.

Код 1 – GoogleDriveService (Services)

GoogleDriveService – це сервіс, який відповідає за роботу з Google Drive у додатку. Він дозволяє: завантажувати файли з Google Drive на локальний комп'ютер, видаляти або оновлювати файли з хмарі, керувати файлами користувача без необхідності знати деталі роботи Google API. Тобто основне призначення сервісу – спростити інтеграцію додатку з Google Drive і надати простий спосіб зберігання та отримання файлів у хмарі.



Код 2 – FileService (Services)

FileService – це сервіс для роботи з файлами, прив'язаними до нотаток у додатку. Основне призначення: додавання файлів або посилань на файли: метод AddLinkFileAsync дозволяє зберігати інформацію про файл (ім'я, шлях/URL, тип) у базі даних для конкретної нотатки, отримання файлів: метод GetFilesByNoteIdAsync повертає список усіх файлів, пов'язаних із заданою нотаткою, видалення файлів: метод DeleteFileAsync дозволяє видалити файл з бази даних. Сервіс спрощує роботу з файлами всередині додатку, абстрагуючи деталі роботи з базою даних і забезпечуючи єдиний інтерфейс для взаємодії з файлами.

NoteController.cs | FileService.cs* | GoogleDriveService.cs* | EmailService.cs | Notifica

OnlineDiaryApp

```
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;
        private readonly FileService _fileService;

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

        // GET: /Note/
        0 references
        public async Task<IActionResult> Index(string? sortBy, string? tag)
        {
            var userId = GetUserId();
            if (userId == null) 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.Value, strategy);
            ViewBag.SortBy = sortBy;
            ViewBag.SelectedTag = tag;
            ViewBag.Tags = await _tagService.GetAllTagsAsync(userId.Value);

            return View(notes);
        }

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

            ViewBag.Tags = await _tagService.GetAllTagsAsync(userId);
            return View();
        }

        [HttpPost]
        0 references
        public async Task<IActionResult> Create(
    string title,
    string content,
    List<int>? tagIds,
    DateTime? reminderDate,
    List<string>? GoogleDriveLinks)
        {
            var userId = int.Parse(HttpContext.Session.GetString("UserId") ?? "0");

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

            if (GoogleDriveLinks != null)
            {
                foreach (var link in GoogleDriveLinks)
                {
                    await _fileService.AddLinkFileAsync(note.Id, "Google Drive file", link);
                }
            }


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

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

Код 3.1 – NoteController (Controllers)

```
        // GET: /Note/Edit/5
        0 references
        public async Task<IActionResult> Edit(int id)
        {
            var note = await _noteService.GetNoteByIdAsync(id);
            if (note == null) return NotFound();

            var userId = GetUserId();
            if (userId == null) return RedirectToAction("Login", "User");

            ViewBag.Tags = await _tagService.GetAllTagsAsync(userId.Value);
            ViewBag.Reminder = await _reminderService.GetReminderByNoteIdAsync(id);

            return View(note);
        }

        // POST: /Note/Edit/5
        [HttpPost]
        0 references
        public async Task<IActionResult> Edit(
            int id,
            string title,
            string content,
            List<int>? tagIds,
            DateTime? remindAt,
            List<string>? GoogleDriveLinks,
            List<int>? DeletedFileIds)
        {
            var note = await _noteService.GetNoteByIdAsync(id);
            if (note == null) return NotFound();

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

            // Видалення файлів
            if (DeletedFileIds != null)
            {
                foreach (var fileId in DeletedFileIds)
                    await _fileService.DeleteFileAsync(fileId);
            }

            // Додавання нових файлів
            if (GoogleDriveLinks != null && GoogleDriveLinks.Any())
            {
                foreach (var link in GoogleDriveLinks)
                {
                    await _fileService.AddLinkFileAsync(note.Id, "Google Drive file", link);
                }
            }


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

            return RedirectToAction("Index");
        }

        // GET: /Note/Delete/5
        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");
        }

        // GET: /Note/Details/5
        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);

            // Підвантажуємо файли для відображення
            note.Files = (await _fileService.GetFilesByNoteIdAsync(note.Id)).ToList();

            return View(note);
        }

        // GET: /Note/Search
        0 references
        public async Task<IActionResult> Search(string keyword)
        {
            var userId = GetUserId();
            if (userId == null) return View(new List<Note>());

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

        // Допоміжний метод для отримання UserId
        3 references
        private int? GetUserId()
        {
            var userIdString = HttpContext.Session.GetString("UserId");
            if (!int.TryParse(userIdString, out int userId)) return null;
            return userId;
        }
    }
}
```

Код 3.2 – 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 Facade. Шаблон реалізований таким чином:

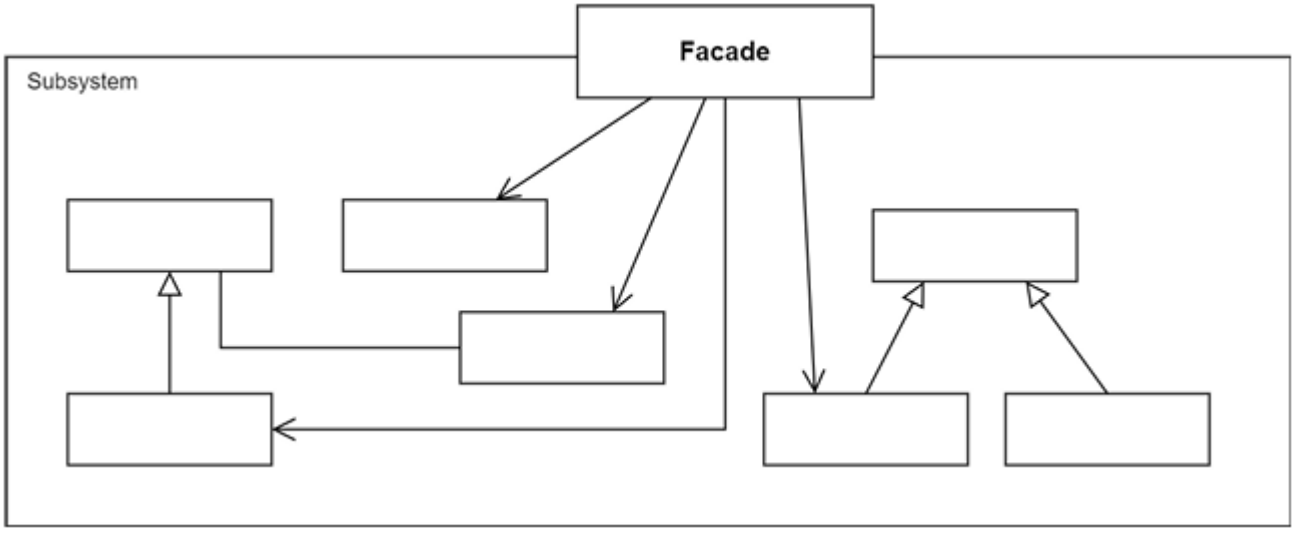


Рис.1 – Структура патерну Фасад

Facade: NotificationFacade — надає спрощений інтерфейс для надсилання повідомлень користувачам, інкапсулюючи роботу з підсистемою, яка відповідає за конкретні типи повідомлень (наразі — EmailService). Завдяки цьому клієнтам (спостерігачам) не потрібно знати, як саме відправляються повідомлення. Також він надає методи: SendEmailNotificationAsync(to, subject, message) — простий метод для відправки email, SendNotificationAsync(to, subject, message, type) —

гнучкий метод, який дозволяє додати інші типи повідомлень (SMS, Push) у майбутньому.

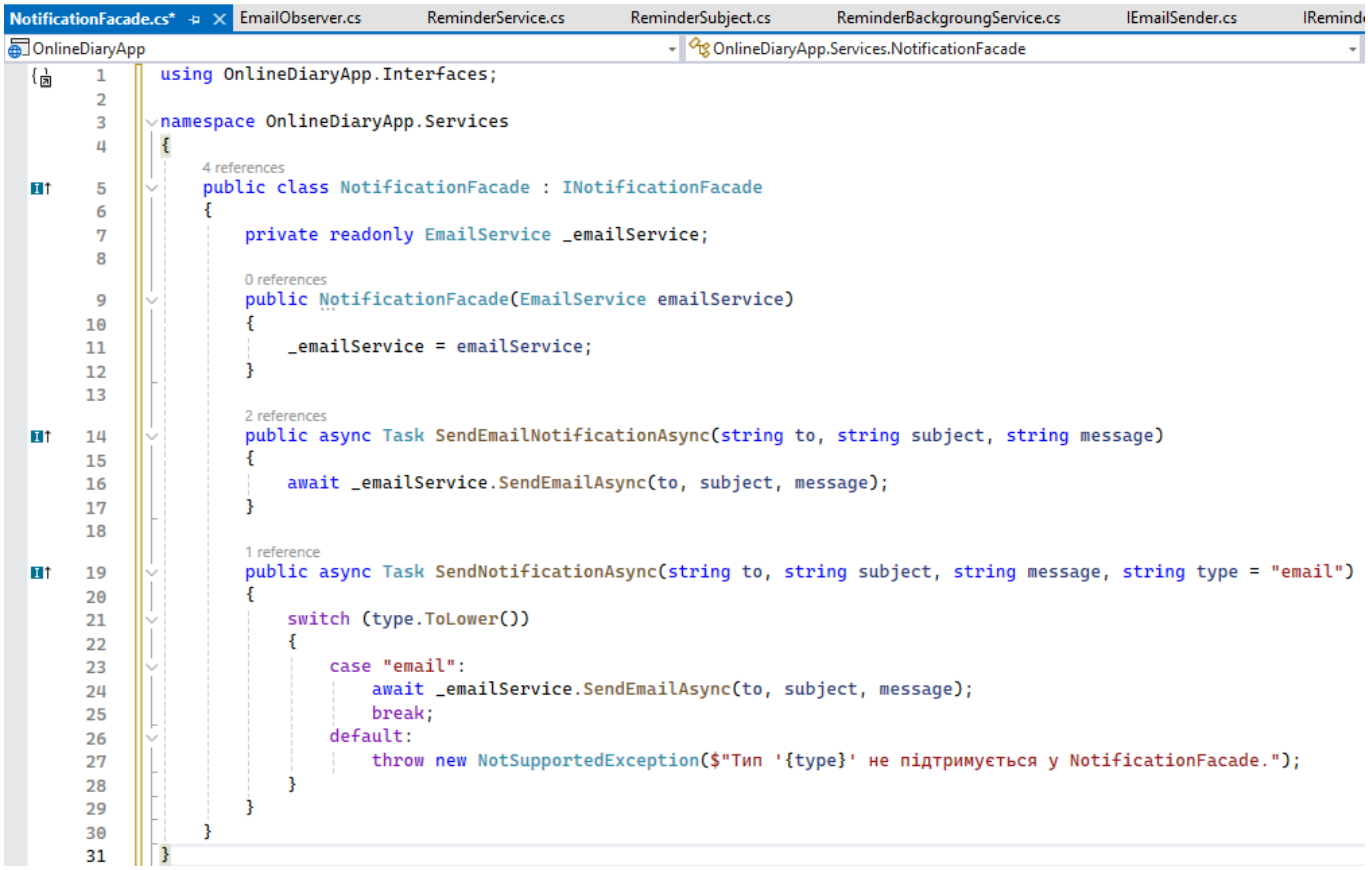
Subsystem: EmailService — це низькорівнева підсистема, яка безпосередньо відповідає за відправлення електронних листів через SMTP за допомогою бібліотек MailKit і MimeKit. Вона знає про технічні деталі: адресу SMTP-сервера, порт, облікові дані та налаштування безпеки. У патерні Фасад цей клас виконує роль конкретного виконавця — він не взаємодіє з клієнтом напряму, а працює лише через NotificationFacade, який координує його використання. Фасад приховує всі деталі конфігурації, тому клієнт може просто сказати: “відправ листа”, не знаючи, як саме це реалізовано.

Client: EmailObserver — це реалізація спостерігача, яка реагує на події нагадувань, надсилаючи користувачу листа. Він використовує інтерфейс IEmailSender, який, у свою чергу, реалізується через EmailService. Таким чином, EmailObserver — це "місток" між бізнес-логікою нагадувань та поштовою підсистемою. Фасад координує цю взаємодію, але не втручається у внутрішню логіку спостерігача.

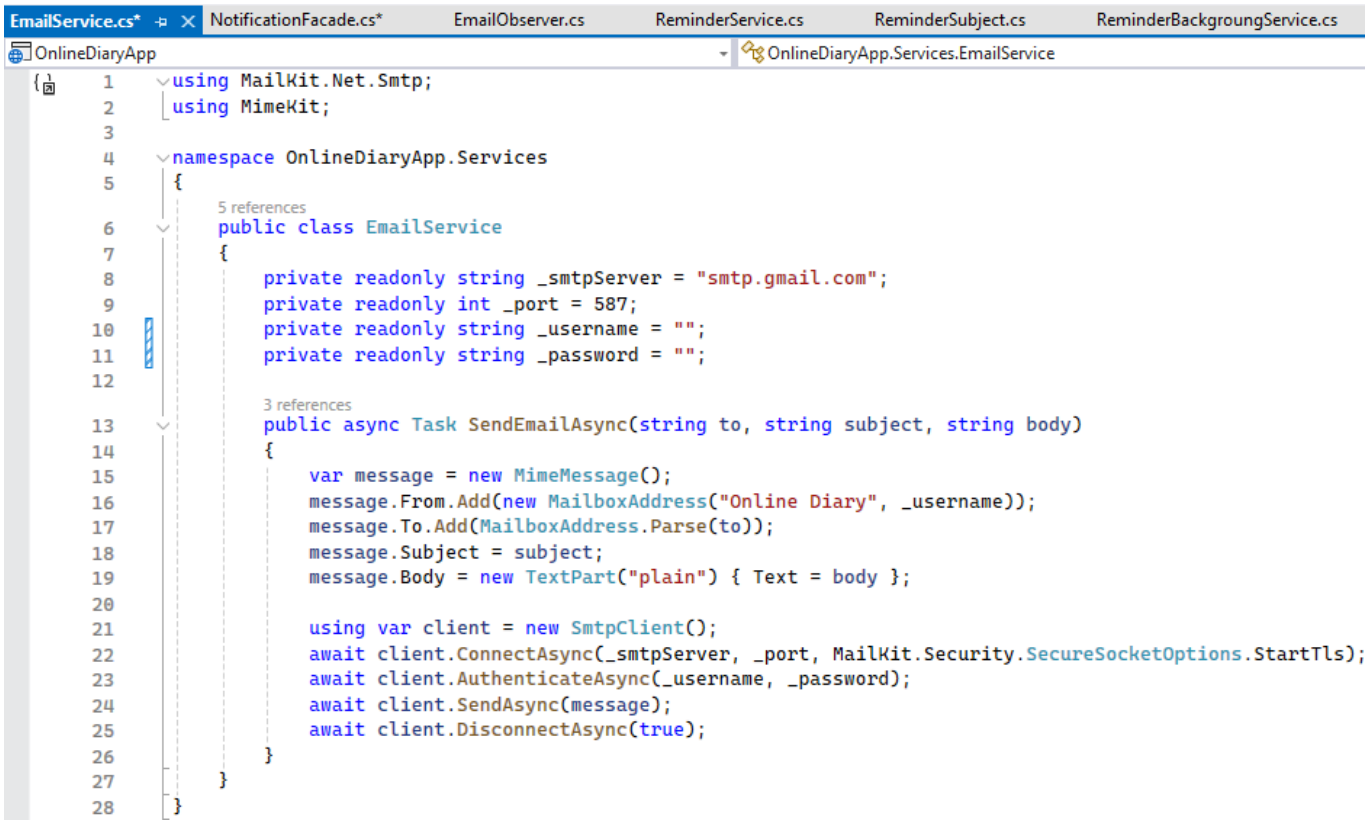
Результат

Завдяки застосуванню патерну Фасад система стала: простішою у використанні — клієнту достатньо одного виклику, замість кількох складних кроків; гнучкою та розширюваною — можна змінювати реалізацію будь-якої підсистеми (наприклад, перейти з SMTP на API) без зміни коду клієнтів; з низьким зв’язуванням — фасад ізолює клієнтів від деталей реалізації підсистем; більш надійною — координує виклики у правильному порядку, зменшуючи ризик помилок.

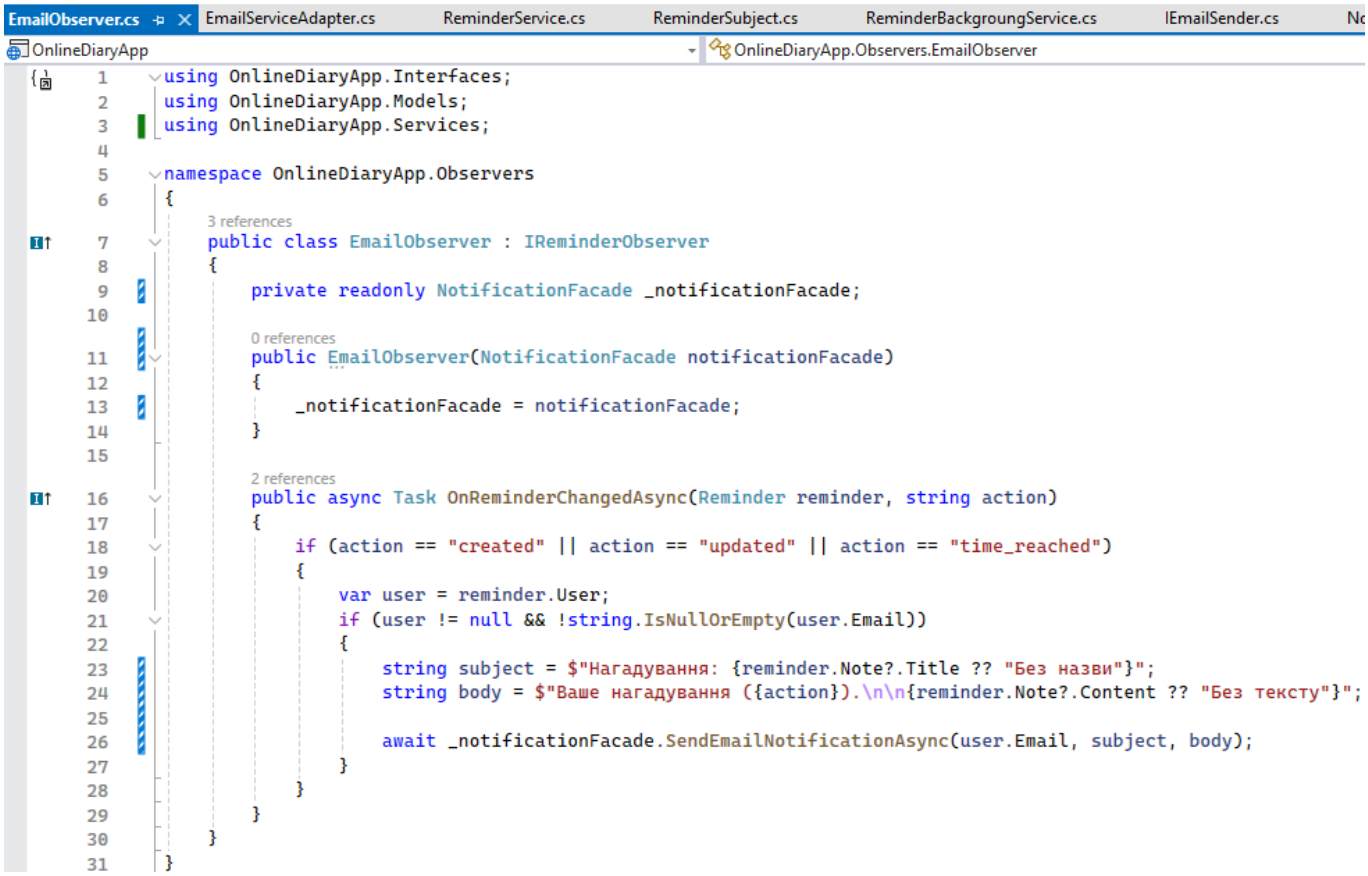
4. Реалізувати не менше 3-х класів відповідно до обраної теми.



Код 4 – NotificationFacade (Services)



Код 5 – EmailService (Services)



Код 6 – EmailObserver (Observers)

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 патерн Фасад для системи надсилання сповіщень у додатку онлайн-щоденника. Основна мета патерну полягала у спрощенні взаємодії клієнтів із підсистемою відправки повідомлень, що дозволяє працювати лише з одним об’єктом NotificationFacade замість безпосередньої роботи з EmailService та іншими сервісами. Це забезпечує відокремлення реалізації від використання: клієнти не залежать від деталей роботи з SMTP чи інших каналів надсилання, а у разі змін у сервісі або додавання нових типів сповіщень код клієнтів залишатиметься незмінним. Застосування фасаду дозволяє легко додавати нові канали сповіщень, підвищує тестованість і надійність системи, а також сприяє слабкому зв’язуванню між ReminderService та

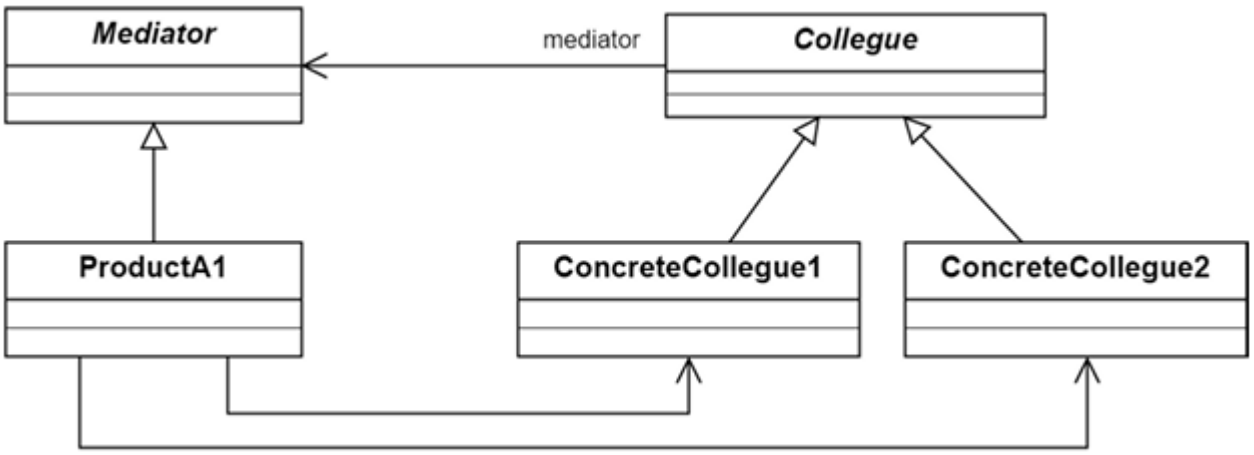
механізмом надсилання повідомлень. У результаті було досягнуто гнучкої і зрозумілої архітектури, що дозволяє централізовано керувати сповіщеннями та забезпечує розширюваність системи без необхідності модифікації існуючих класів.

Відповіді на контрольні питання:

1. Призначення шаблону «Посередник»

   Шаблон «Посередник» (Mediator) призначений для централізації взаємодії між об'єктами. Замість того щоб об'єкти безпосередньо посилали повідомлення один одному, вони взаємодіють через посередника, що зменшує зв'язування між компонентами та спрощує підтримку і розширення системи.

2. Структура шаблону «Посередник»



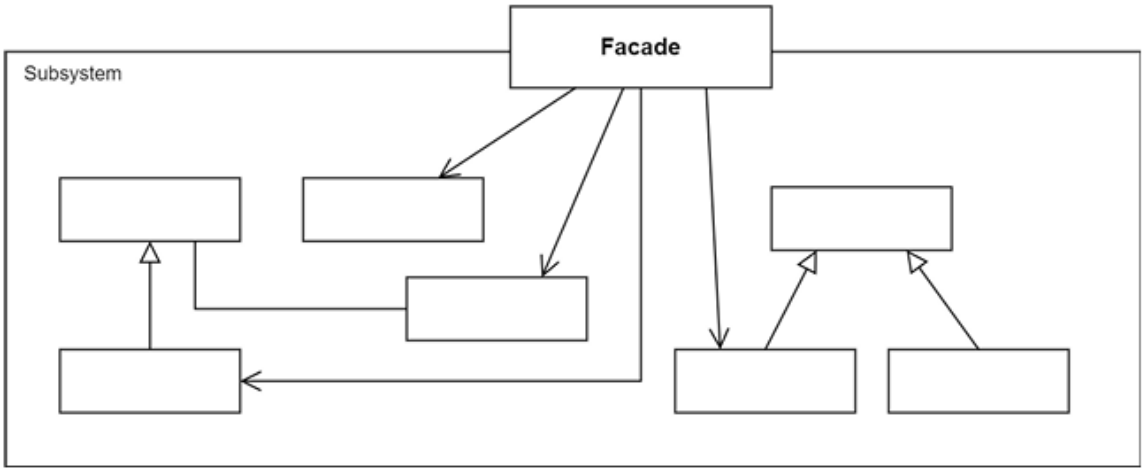
3. Класи та взаємодія шаблону «Посередник»

- Mediator (Інтерфейс Посередника) – описує методи для взаємодії колег.
- ConcreteMediator (Конкретний Посередник) – реалізує логіку взаємодії між колегами.
- Colleague (Колега) – об'єкти, які посилають повідомлення через посередника.
- Взаємодія: колеги не звертаються один до одного напряму, а передають дані через посередника, який координує комунікацію.

4. Призначення шаблону «Фасад»

   Шаблон «Фасад» (Facade) спрощує роботу з комплексною системою або підсистемою, надаючи єдиний інтерфейс для взаємодії. Клієнти працюють через фасад, не залежачи від складної внутрішньої структури системи.

5. Структура шаблону «Фасад»



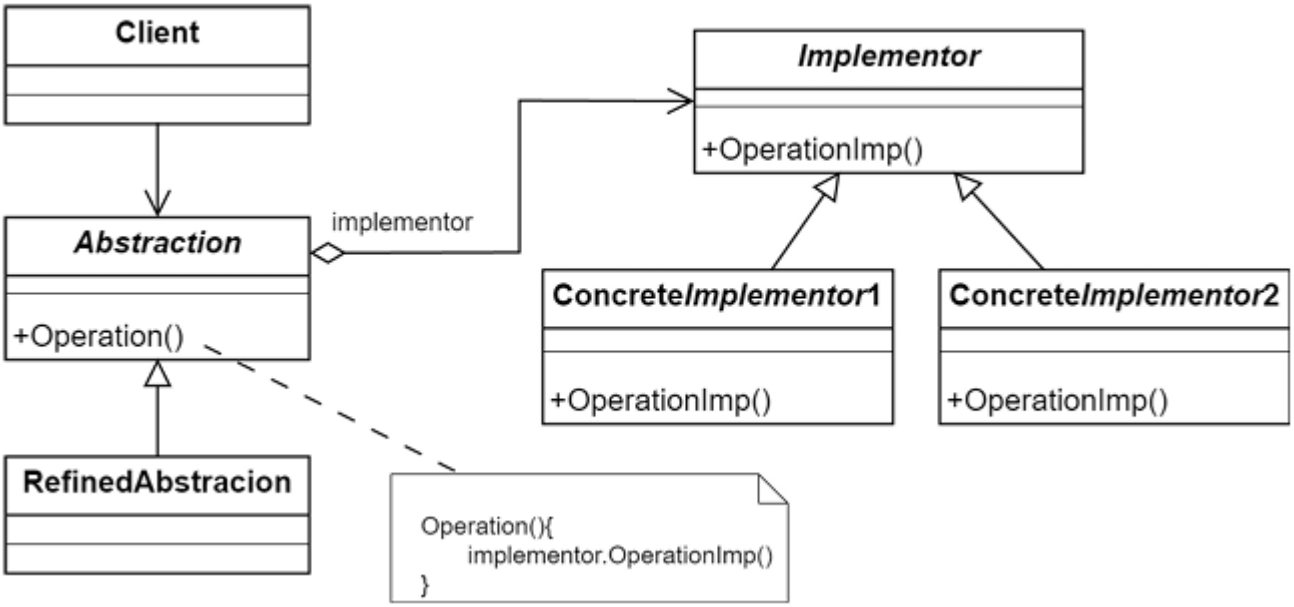
6. Класи та взаємодія шаблону «Фасад»

- Facade (Фасад) – надає спрощений інтерфейс для клієнтів.
- Subsystem (Підсистема) – клас або група класів, що реалізують бізнес-логіку.
- Взаємодія: клієнт викликає методи фасаду → фасад делегує запити підсистемам.

7. Призначення шаблону «Міст»

   Шаблон «Міст» (Bridge) розділяє абстракцію і реалізацію, щоб вони могли змінюватися незалежно. Це дозволяє підключати різні реалізації до одного інтерфейсу без зміни клієнтського коду.

8. Структура шаблону «Міст»



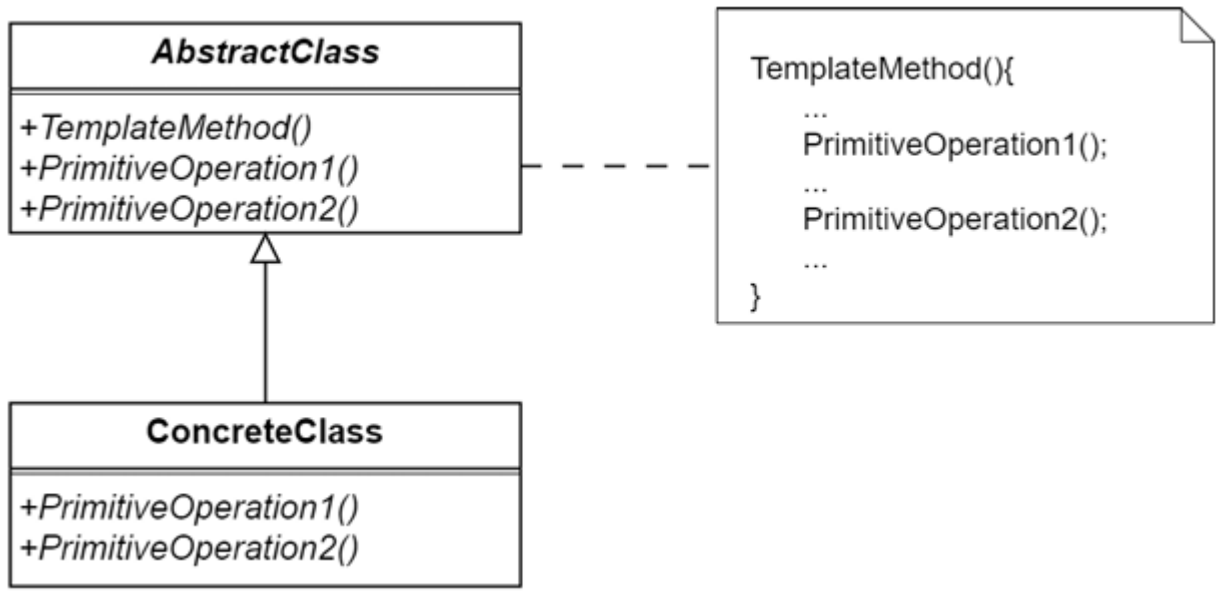
9. Класи та взаємодія шаблону «Міст»

- Abstraction (Абстракція) – визначає високорівневий інтерфейс для клієнта.
- RefinedAbstraction (Уточнена абстракція) – розширює абстракцію.
- Implementor (Інтерфейс реалізації) – описує низькорівневі методи.
- ConcreteImplementor (Конкретна реалізація) – реалізує методи Implementor.
- Взаємодія: абстракція делегує виконання методів конкретним реалізаціям, не знаючи деталей.

10. Призначення шаблону «Шаблонний метод»

Шаблон «Шаблонний метод» (Template Method) визначає скелет алгоритму в методі, залишаючи реалізацію деяких кроків підкласам. Це дозволяє уникнути дублювання коду та забезпечує гнучкість.

11. Структура шаблону «Шаблонний метод»



12.Класи та взаємодія шаблону «Шаблонний метод»

- AbstractClass (Абстрактний клас) – містить шаблонний метод і деякі реалізації кроків алгоритму.
- ConcreteClass (Конкретний клас) – реалізує конкретні кроки алгоритму.
- Взаємодія: клієнт викликає шаблонний метод → він визначає послідовність кроків → підкласи реалізують конкретні кроки.

13.Відмінність «Шаблонного методу» від «Фабричного методу»

- «Шаблонний метод» визначає алгоритм з деякими змінними кроками в підкласах.
- «Фабричний метод» створює об’єкти через інтерфейс або абстрактний клас, делегуючи створення підкласам. Головне у фабричному методі – інкапсуляція створення об’єктів, а не послідовності кроків.

14.Функціональність, яку додає шаблон «Міст»

Шаблон «Міст» дозволяє змінювати абстракцію і реалізацію незалежно, підключати нові реалізації без зміни клієнтського коду, зменшує зв’язування між класами та підвищує гнучкість і розширюваність системи.