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

Лабораторна робота *№7*

із дисципліни ***«Технології розробки програмного забезпечення»***

Тема: ***«Патерни проектування»***

Виконав:

Студент групи ІА-34

Ковальчук Станіслав

Перевірив:

асистент кафедри ІСТ

Мягкий Михайло Юрійович

Київ — 2025

Тема: Патерни проектування.

Мета: Вивчити структуру шаблонів «Mediator», «Facade», «Bridge», «Template

method» та навчитися застосовувати їх в реалізації програмної системи..

Застосунок: Office communicator (strategy, adapter, abstract factory, bridge, composite, client-server)

Мережевий комунікатор для офісу повинен нагадувати функціонал програми Skype з можливостями голосового / відео / конференц-зв'язку, відправки текстових повідомлень і файлів (можливо, оффлайн), веденням організованого списку груп / контактів.

Короткі теоретичні відомості:

Шаблон «Mediator» (посередник) використовується для визначення взаємодії об'єктів за допомогою іншого об'єкта (замість зберігання посилань один на одного). Даний шаблон схожий на шаблон «команда», проте в даному випадку замість зберігання даних про конкретну дію, зберігаються дані про взаємодії між компонентами.

Шаблон «Facade» (фасад) передбачає створення єдиного уніфікованого способу доступу до підсистеми без розкриття внутрішніх деталей підсистеми. Оскільки підсистема може складатися з безлічі класів, а кількість її функцій – не більше десяти, то щоб уникнути створення «спагетікоду» (коли все тісно пов'язано між собою) виділяють один загальний інтерфейс доступу, здатний правильним чином звертатися до внутрішніх деталей.

Шаблон «Bridge» (міст) використовується для поділу інтерфейсу і його реалізації. Це необхідно у випадках, коли може існувати кілька різних абстракцій, над якими можна проводити дії різними способами.

Шаблон «Template Method» (шаблонний метод) дозволяє реалізувати покроково алгоритм в абстрактному класі, але залишити специфіку реалізації підкласам. Даний шаблон дещо нагадує шаблон «Фабричний метод», однак область його використання абсолютно інша – для покрокового визначення

конкретного алгоритму; більш того, даний шаблон не обов'язково створює нові об'єкти – лише визначає послідовність дій.

Хід роботи:

1. Повна діаграма класів (без шаблону).

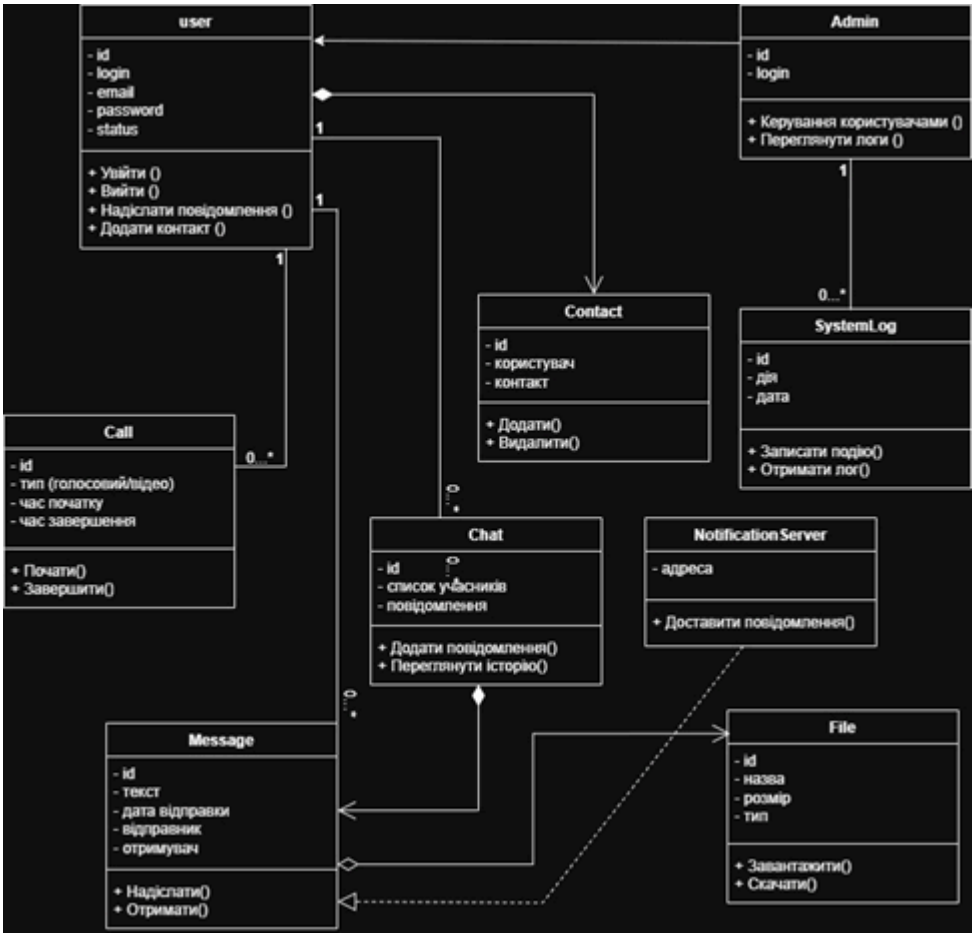


Рис. 7.1 – Діаграма класів системи

2. Діаграма класів з використанням шаблону «Bridge»:

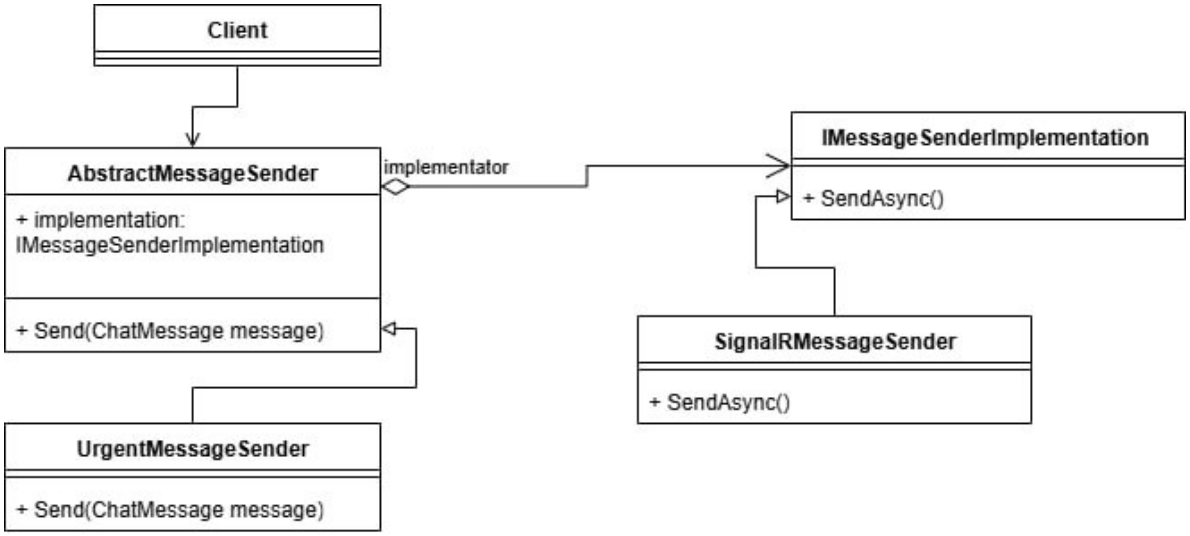


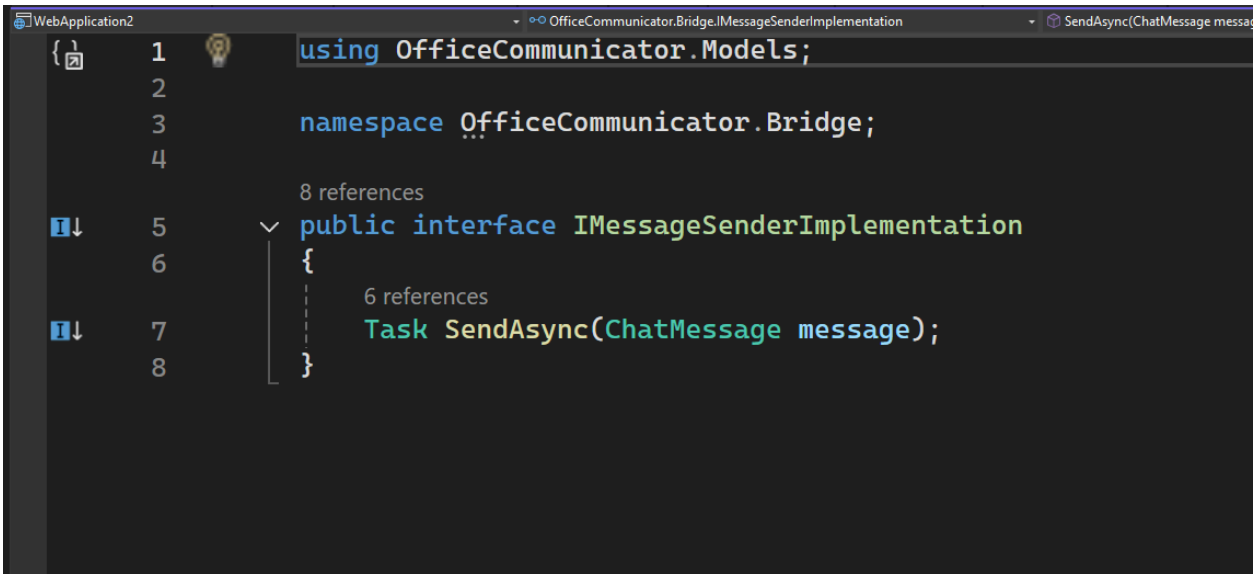
Рис. 7.2 – Діаграма класів системи з використанням шаблону «Bridge»

Програмна реалізація

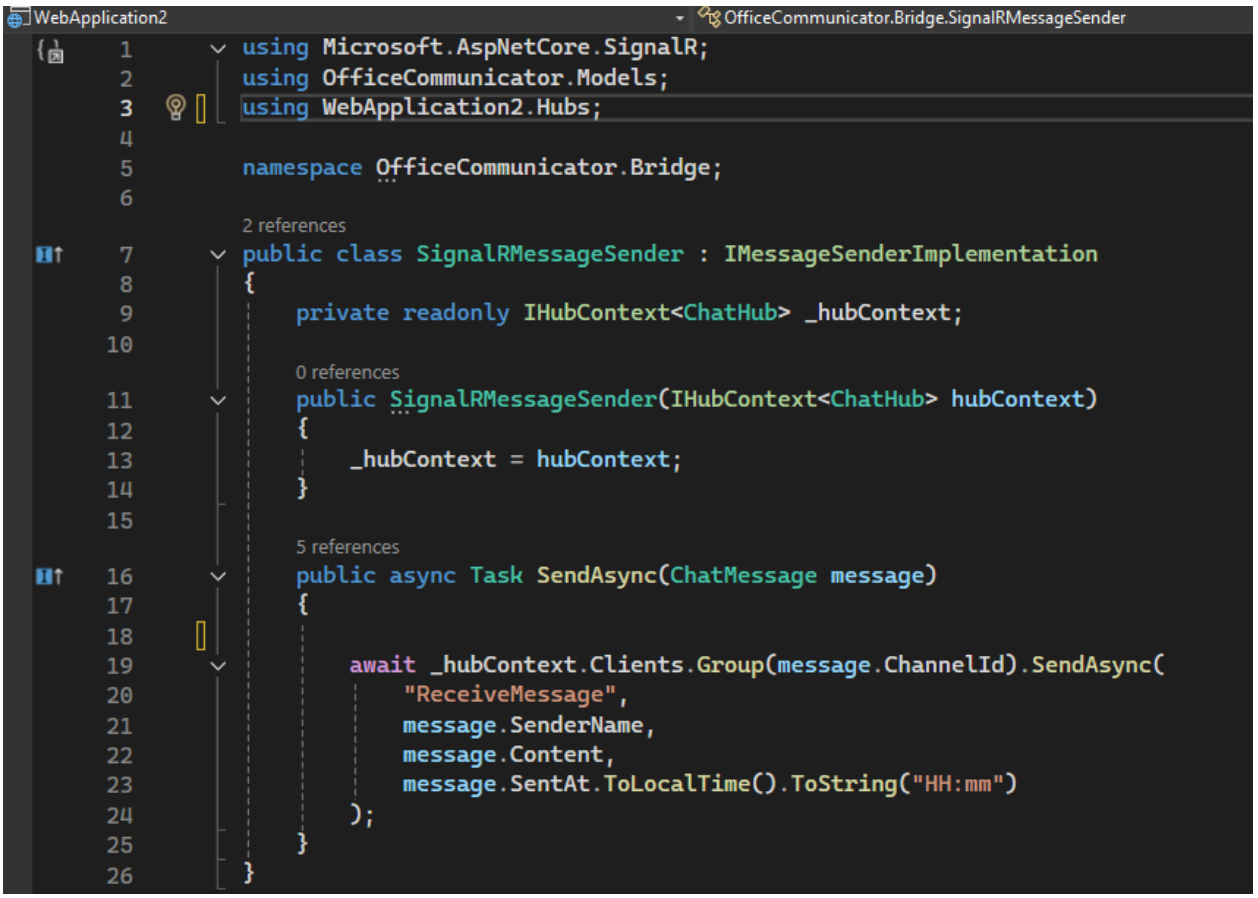
Для вирішення поставленої задачі було реалізовано патерн **Bridge**, який розділяє систему на дві частини:

1. **Реалізація (Implementation):** Відповідає за технічну доставку повідомлення (у нашому випадку — через SignalR).
2. **Абстракція (Abstraction):** Відповідає за логіку повідомлення (форматування, пріоритет).

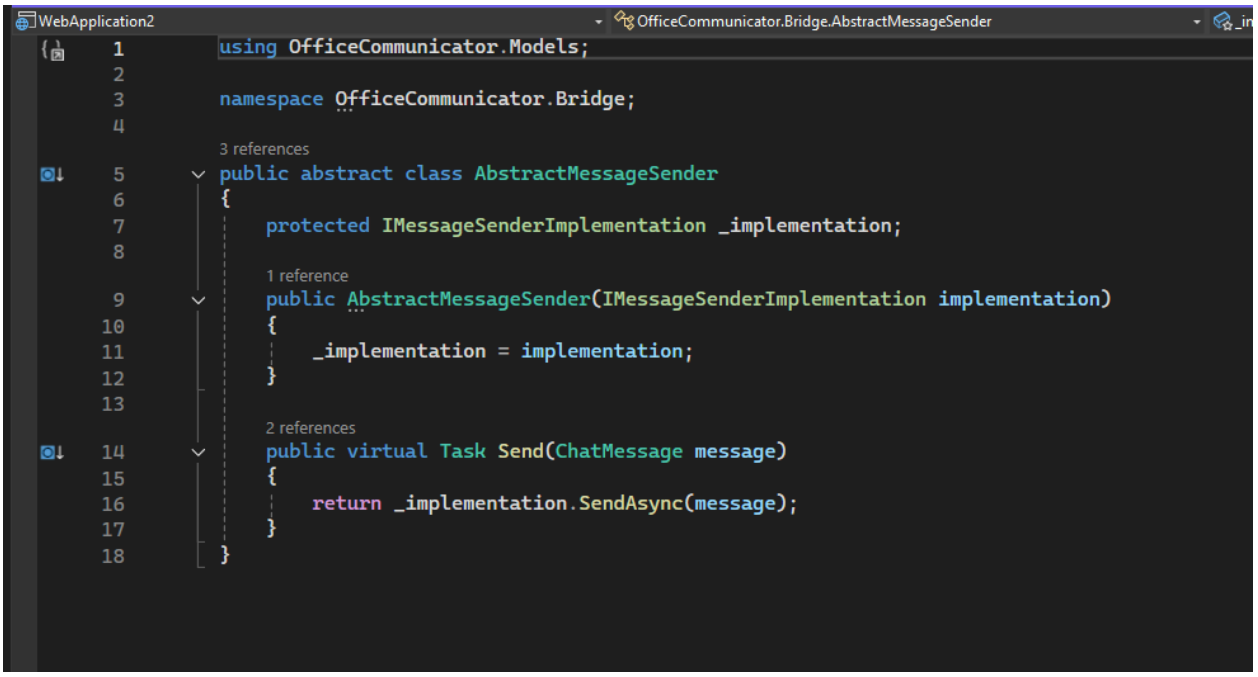
Інтерфейс реалізації (IMessageSenderImplementation.cs) Визначає контракт, який мають виконувати всі "транспортні" механізми.



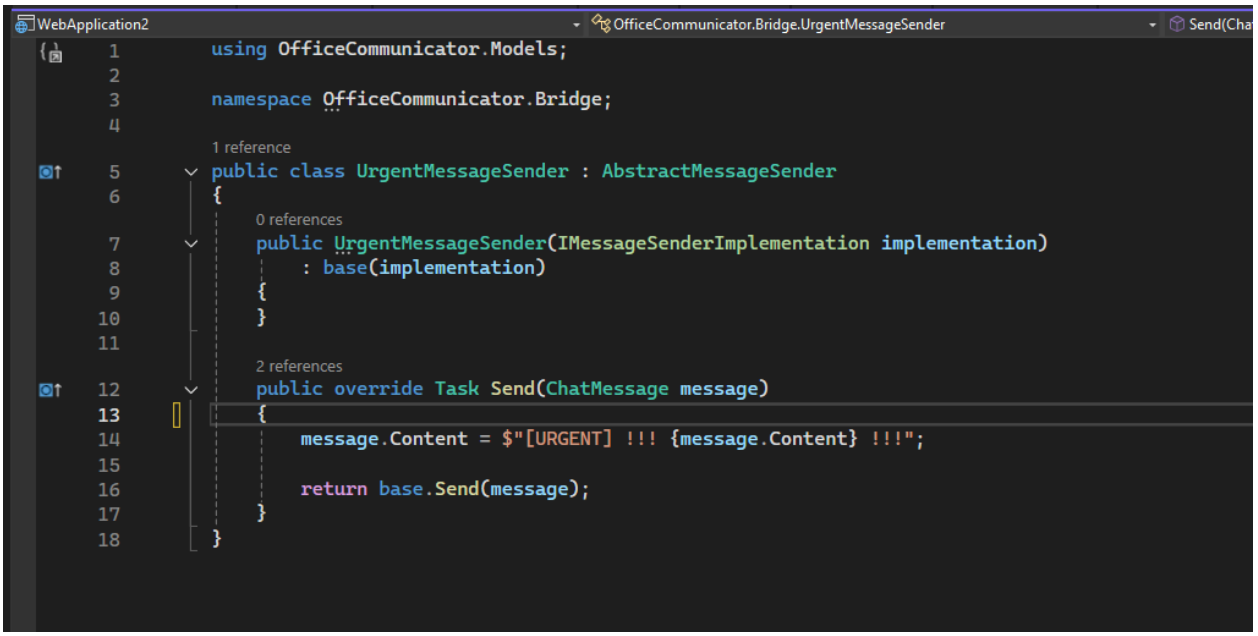
Конкретна реалізація SignalR (SignalRMessageSender.cs) Клас, який безпосередньо взаємодіє з бібліотекою SignalR для відправки повідомлень клієнтам у реальному часі.



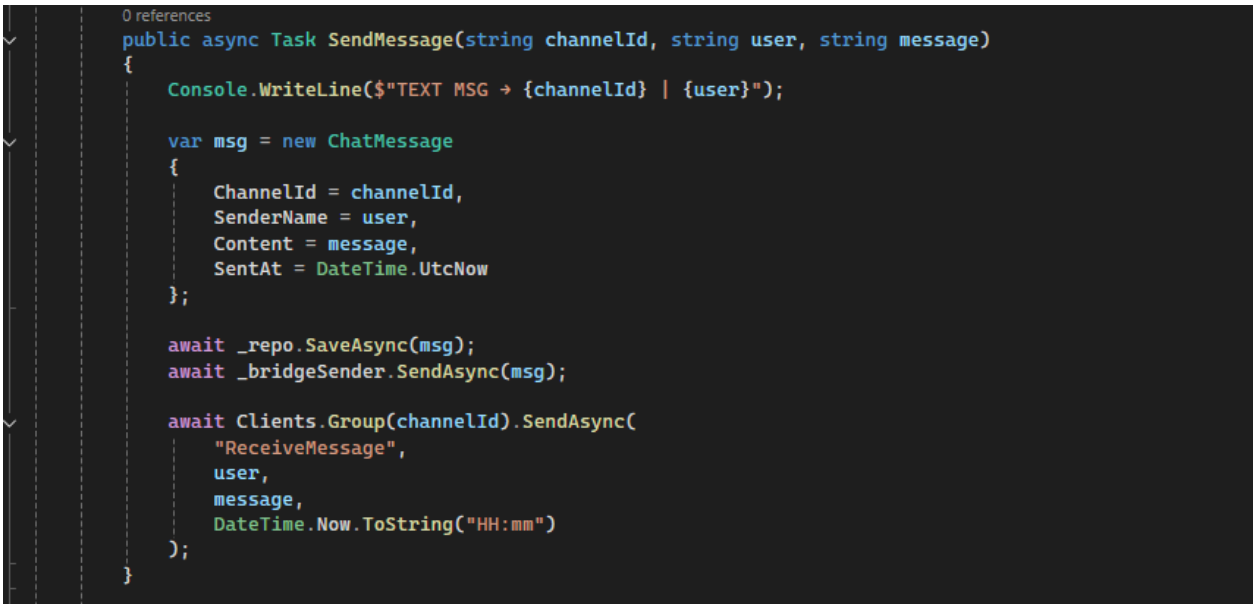
Базова абстракція (AbstractMessageSender.cs) Цей клас містить посилання на об'єкт реалізації (_implementation) і делегує йому роботу.



Уточнена абстракція (UrgentMessageSender.cs) Клас для термінових повідомлень. Він додає специфічну логіку (позначку [URGENT]) перед тим, як передати повідомлення реалізації. При цьому код SignalRMessageSender змінювати не потрібно.



Інтеграція в ChatHub Приклад використання моста всередині класу ChatHub. Хаб виступає клієнтом, який використовує абстракцію для відправки даних.



```
0 references
public async Task SendMessage(string channelId, string user, string message)
{
    Console.WriteLine($"TEXT MSG → {channelId} | {user}");

    var msg = new ChatMessage
    {
        ChannelId = channelId,
        SenderName = user,
        Content = message,
        SentAt = DateTime.UtcNow
    };

    await _repo.SaveAsync(msg);
    await _bridgeSender.SendAsync(msg);

    await Clients.Group(channelId).SendAsync(
        "ReceiveMessage",
        user,
        message,
        DateTime.Now.ToString("HH:mm")
    );
}
```

У класі ChatHub я отримую інтерфейс IMessageSenderImplementation через конструктор. Коли приходить повідомлення, я викликаю метод _bridgeSender.SendAsync(msg), який перенаправляє його на конкретну реалізацію (у нашому випадку — SignalR).

Висновок: Під час виконання лабораторної роботи я поглибив свої знання про структурні патерни проєктування, детально вивчив їхню класифікацію та специфіку застосування. Основну увагу було приділено патерну «Bridge» (Міст).Для практичної реалізації в системі "Office Communicator" було розроблено архітектуру, що відокремлює абстракцію від її реалізації:

1. Рівень реалізації (Implementation): Було розроблено інтерфейс IMessageSenderImplementation та його конкретну реалізацію SignalRMessageSender. Це дозволило інкапсулювати технічні деталі взаємодії з бібліотекою SignalR та хабом ChatHub, забезпечивши незалежність транспортного рівня.
2. Рівень абстракції (Abstraction): Було створено базовий клас AbstractMessageSender та його розширення UrgentMessageSender. Це дало змогу реалізувати специфічну бізнес-логіку обробки повідомлень

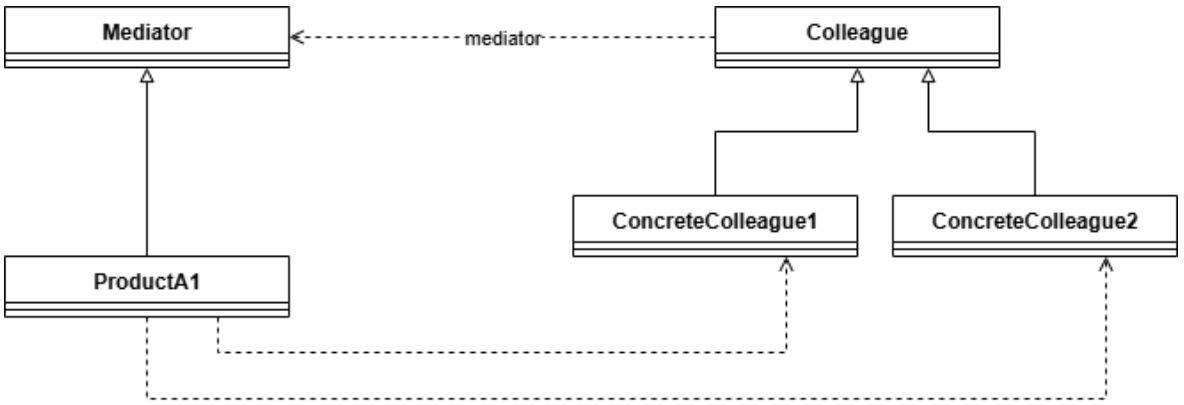
(наприклад, маркування термінових повідомлень) без втручання в код, що відповідає за їх доставку.

Використання патерну «Bridge» дозволило значно підвищити гнучкість системи. Завдяки слабкій зв'язності компонентів, додавання нових каналів комунікації (наприклад, Email або SMS) або нових типів повідомлень у майбутньому не потребуватиме модифікації існуючого коду, що відповідає принципу відкритості/закритості (OCP) та спрощує тестування програмного продукту.

Відповіді на контрольні питання:

1. Яке призначення шаблону «Посередник»?

   Шаблон «Посередник» дозволяє зменшити складність комунікації між об'єктами, централізуючи взаємодію між ними через один об'єкт — посередник. Це дозволяє об'єктам не знати про інші об'єкти та взаємодіяти лише через посередника, що забезпечує більш контрольовану і гнучку архітектуру.
2. Нарисуйте структуру шаблону «Посередник».



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

   Шаблон «Посередник» включає такі класи:

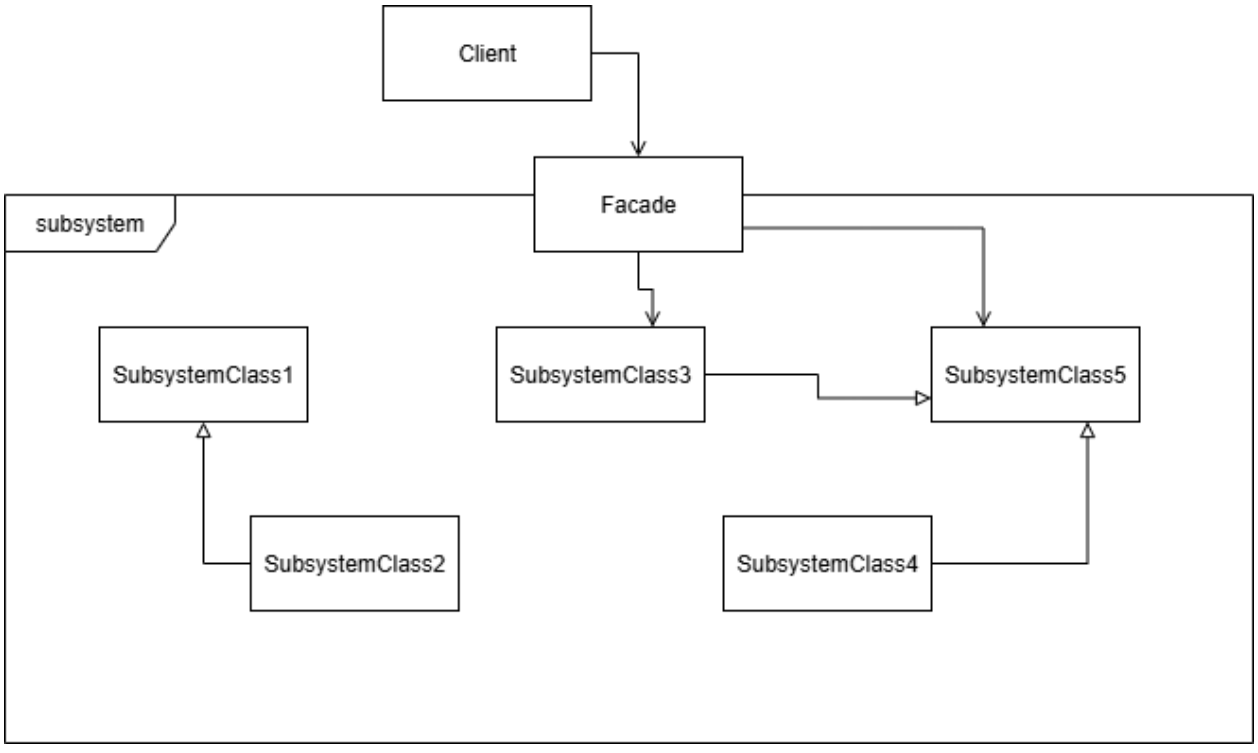
   - **Посередник (Mediator)** — це інтерфейс або абстрактний клас, який визначає методи для взаємодії між компонентами.

- **Конкретний посередник (ConcreteMediator)** — це клас, який реалізує логіку взаємодії між конкретними компонентами. Він управляє комунікацією між компонентами та координує їх дії.
- **Колега (Colleague)** — це компоненти або об'єкти, які взаємодіють через посередника. Вони не мають знання про інших колег, лише через посередника.

4. Яке призначення шаблону «Фасад»?

   Шаблон «Фасад» надає спрощений інтерфейс для складної системи, приховуючи її складність. Це дозволяє знизити взаємодію користувача з низькорівневими частинами системи та зробити її більш зручною для використання.

5. Нарисуйте структуру шаблону «Фасад».



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

   Шаблон «Фасад» складається з таких класів:

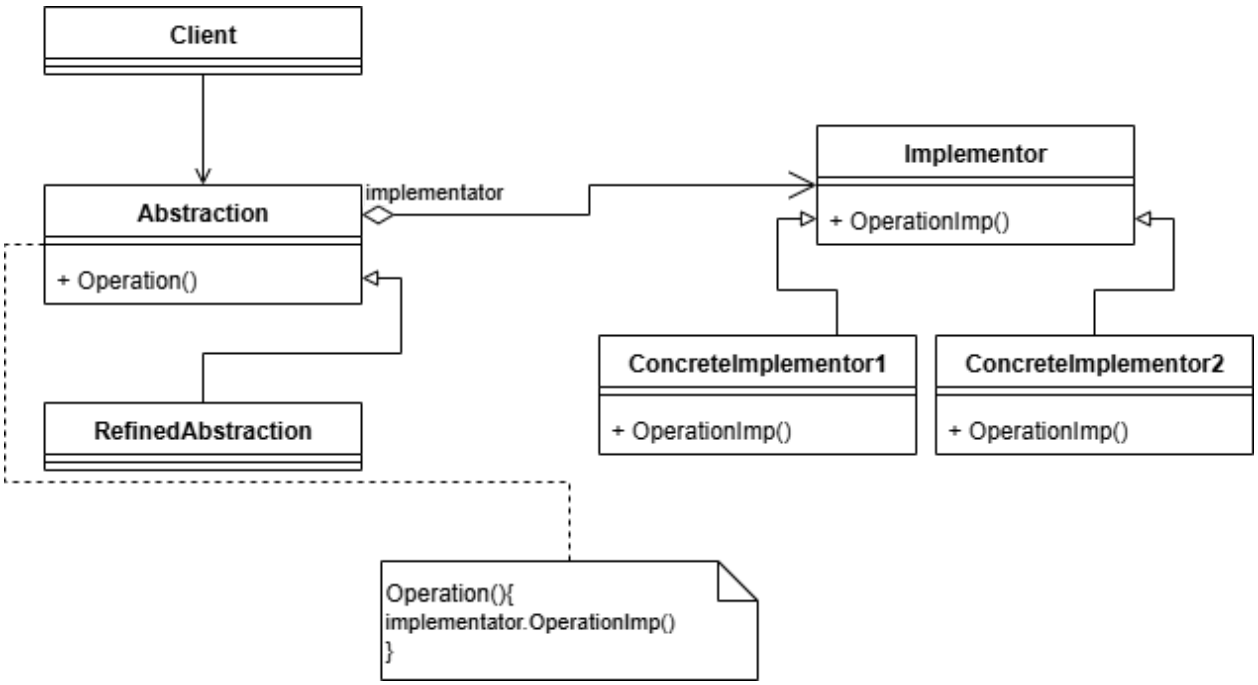
   - **Фасад (Facade)** — це клас, який надає простий інтерфейс для складної системи. Він забезпечує доступ до низькорівневих підсистем та спрощує взаємодію з ними.

- **Підсистеми (Subsystems)** — це класи, які реалізують реальну функціональність складної системи. Вони мають складніші інтерфейси, але фасад надає користувачу спрощений доступ до їх методів.

7. Яке призначення шаблону «Міст»?

   Шаблон «Bridge» (міст) використовується для поділу інтерфейсу і його реалізації. Це необхідно у випадках, коли може існувати кілька різних абстракцій, над якими можна проводити дії різними способами.

8. Нарисуйте структуру шаблону «Міст».



9. Які класи входять в шаблон «Міст», та яка між ними взаємодія?

   Шаблон «Міст» включає такі класи:

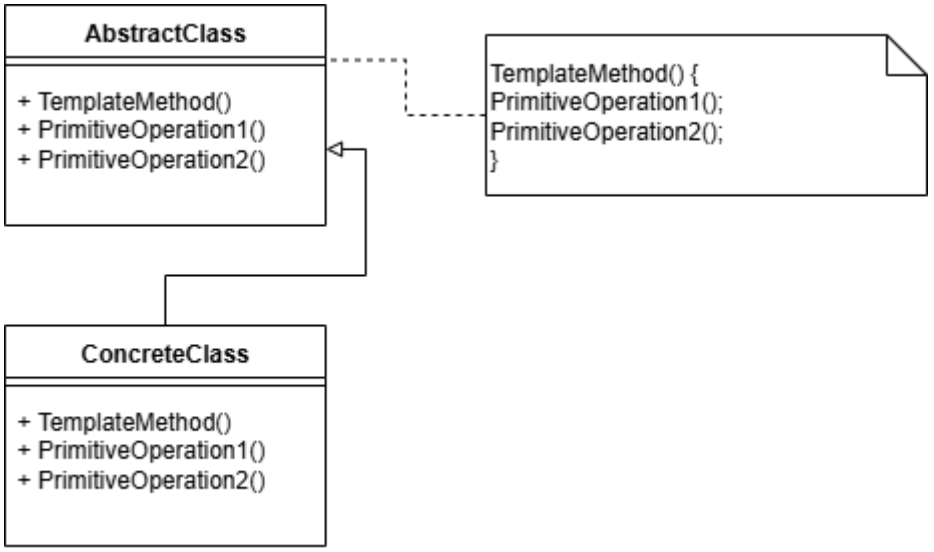
   - **Абстракція (Abstraction)** — це абстрактний клас або інтерфейс, який містить посилання на об'єкт реалізації і визначає основні методи.
   - **Реалізація (Implementor)** — це інтерфейс, який визначає методи, що мають бути реалізовані. Всі конкретні реалізації повинні слідувати цьому інтерфейсу.

- **Конкретна абстракція (RefinedAbstraction)** — це клас, який розширює абстракцію і може додавати додаткові функції або поведінку.
- **Конкретна реалізація (ConcreteImplementor)** — це клас, який реалізує інтерфейс реалізації та визначає конкретну логіку роботи.

10. Яке призначення шаблону «Шаблоний метод»?

    Шаблон «Template Method» (шаблонний метод) дозволяє реалізувати покроково алгоритм в абстрактному класі, але залишити специфіку реалізації підкласам. Даний шаблон дещо нагадує шаблон «Фабричний метод», однак область його використання абсолютно інша – для покрокового визначення конкретного алгоритму; більш того, даний шаблон не обов'язково створює нові об'єкти – лише визначає послідовність дій.

11. Нарисуйте структуру шаблону «Шаблоний метод».



12. Які класи входять в шаблон «Шаблоний метод», та яка між ними взаємодія?

    Шаблон «Шаблонний метод» складається з таких класів:

    - **Абстрактний клас (AbstractClass)** — це клас, який визначає шаблонний метод, що складається з викликів абстрактних і конкретних методів. Він визначає порядок виконання алгоритму.

- **Конкретний клас (ConcreteClass)** — це клас, який реалізує абстрактні методи, визначаючи конкретну реалізацію етапів алгоритму.

13. Чим відрізняється шаблон «Шаблонний метод» від «Фабричного методу»?

    Основна різниця між шаблоном «Шаблонний метод» і «Фабричним методом» полягає в тому, як вони змінюють поведінку:

    - **Шаблонний метод** визначає загальний шаблон алгоритму і дозволяє підкласам змінювати частини цього алгоритму (методи).
    - **Фабричний метод** дозволяє змінювати процес створення об'єктів, надаючи підкласам можливість визначати, які об'єкти створювати, але не змінюючи саму структуру алгоритму.

14. Яку функціональність додає шаблон «Міст»?

    Шаблон «Міст» додає функціональність для розділення абстракцій та їх реалізацій, що дозволяє змінювати одну з частин без впливу на іншу. Це дозволяє змінювати спосіб реалізації без необхідності змінювати абстракцію, а також забезпечує гнучкість при розширенні системи.