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 *№5*

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 «Adapter», «Builder», «Command», «Chain of responsibility», «Prototype» 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:

1. Шаблон «Adapter» (Адаптер)

Призначення: Адаптація інтерфейсу одного об'єкта до вимог іншого, забезпечуючи сумісність між несумісними класами. Деталізація: Уявімо ситуацію з використанням зовнішніх бібліотек для роботи з периферійними пристроями, наприклад, принтерами чи сканерами. Попри ідентичне призначення, кожна бібліотека може мати унікальний набір методів та назв. Для уникнення жорсткої прив’язки до конкретної бібліотеки доцільно розробити уніфікований інтерфейс, який включатиме стандартні операції: сканування, асинхронну обробку, двостороннє або потокове сканування. Реалізація відповідних «адаптерів» дозволить програмі взаємодіяти з будь-якою бібліотекою через єдиний шлюз, не змінюючи основну логіку застосунку при заміні вендора обладнання.

2. Шаблон «Builder» (Будівельник)

Призначення: Відокремлення процесу конструювання складного об'єкта від його представлення, що дозволяє використовувати той самий процес для створення різних представлень. Деталізація: Цей шаблон є незамінним, коли створення об'єкта передбачає виконання багатоетапної та складної послідовності кроків. Наприклад, формування об’єкта WebPage як відповіді

сервера включає обробку заголовків, метаданих, тіла документа та підключених ресурсів. Також «Будівельник» ефективно використовується в алгоритмах конвертації даних: один і той самий алгоритм обходу тексту може генерувати результат у різних форматах (HTML, RTF або Plain Text) залежно від обраного «будівельника».

3. Шаблон «Command» (Команда)

Призначення: Перетворення запиту на виконання певної дії в окремий об'єкт. Деталізація: Завдяки цьому патерну звичайний виклик методу інкапсулюється в клас, перетворюючи дії на повноправні об’єкти системи. Сама по собі «Команда» не виконує обробку, а лише перенаправляє запит безпосередньому отримувачу (Receiver). Головна перевага полягає в тому, що такі об’єкти можуть зберігати стан, що дозволяє легко реалізувати:

- Функцію скасування (Undo/Redo): Зберігання історії виконаних команд.
- Логування операцій: Запис дій для аудиту чи відновлення.
- Черги завдань: Відкладене виконання операцій.

4. Шаблон «Chain of Responsibility» (Ланцюжок відповідальності)

Призначення: Організація передачі запитів по ланцюжку потенційних обробників до моменту, поки один з них не опрацює запит. Деталізація: Цей механізм нагадує бюрократичний процес у великій компанії: документ проходить шлях від розробника до менеджера, потім до начальника відділу і, нарешті, до директора для фінального підпису. У програмному забезпеченні це дозволяє уникнути жорсткої прив’язки відправника до отримувача. У контексті браузера це ідеально підходить для обробки HTTP-кодів: запит проходить через ланцюжок фільтрів, де кожен елемент перевіряє свій статус (наприклад, чи це помилка 404, чи 502) і приймає рішення про обробку або передачу далі.

5. Шаблон «Prototype» (Прототип)

Призначення: Створення нових об'єктів шляхом копіювання вже існуючого екземпляра (шаблону), а не через створення з нуля. Деталізація:

Шаблон використовується, коли процес ініціалізації об'єкта є ресурсозатратним або коли система повинна бути незалежною від способу створення її продуктів. Для цього в базовому класі визначається спеціальний метод «клонувати» (clone). Замість того, щоб щоразу налаштовувати складний об’єкт за «кресленням» або «ескізом», програма просто робить дублікат існуючого об’єкта-прототипу, що значно прискорює роботу системи.

Хід роботи:

1. Повна діаграма класів (без шаблону).

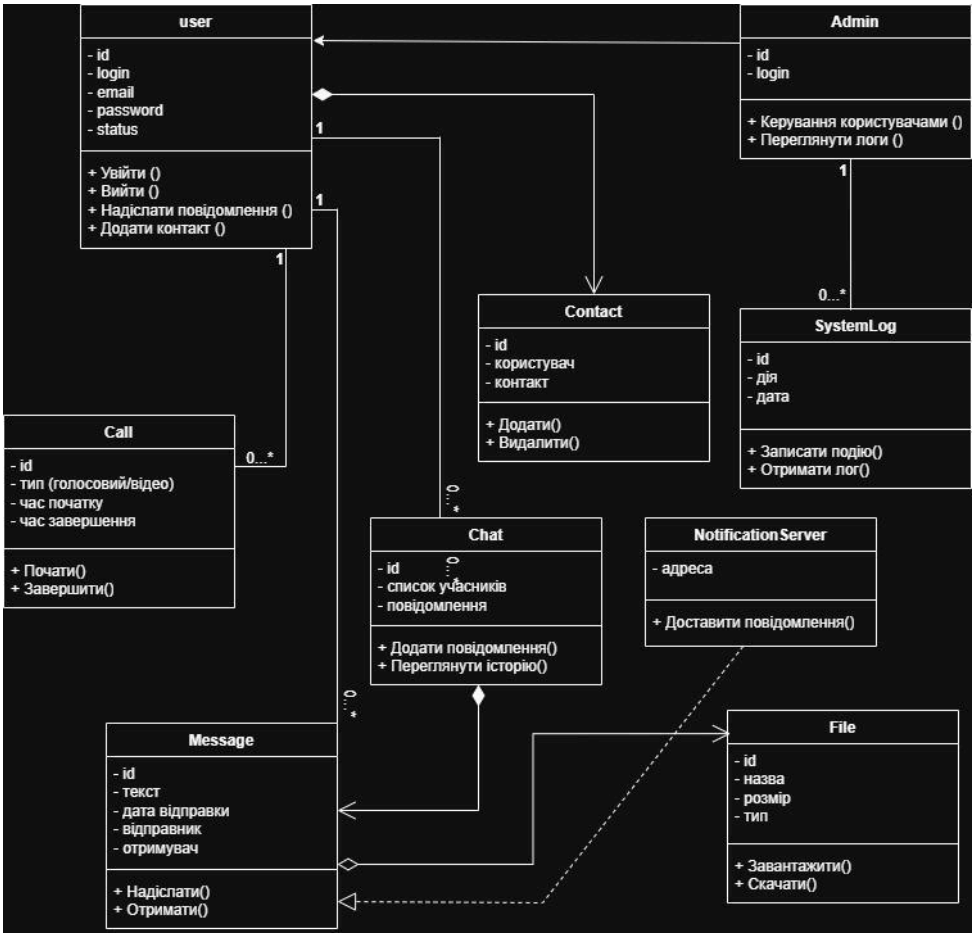


Рис. 5.1 – Діаграма класів системи

2. Діаграма класів з використанням шаблону «Adapter»:

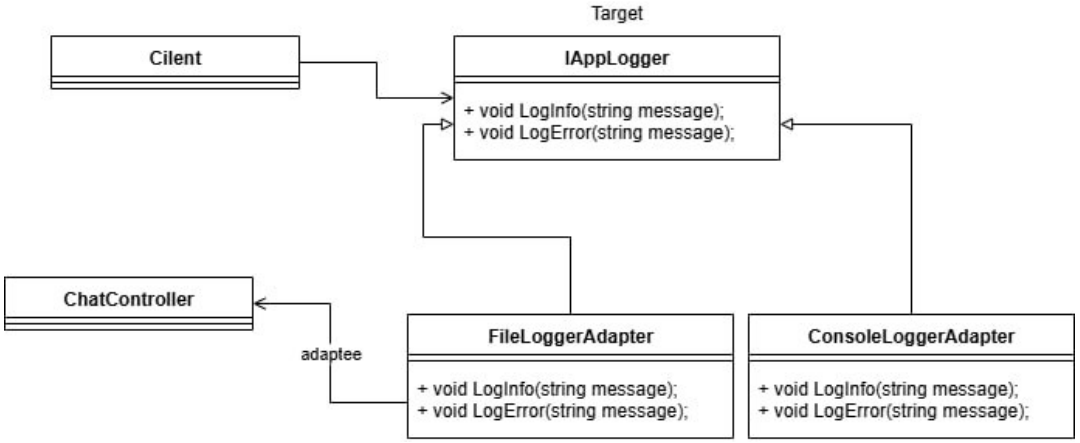


Рис. 5.2 – Діаграма класів системи з використанням шаблону «Adapter»

Я визначаю інтерфейс IAppLogger, який визначає в собі методи методи LogInfo() та LogError(), які потрібні системі.

ConsoleLoggerAdapter адаптує функціональність консолі до інтерфейсу IAppLogger.

FileLoggerAdapter адаптує функціональність файлової системи до інтерфейсу IAppLogger.

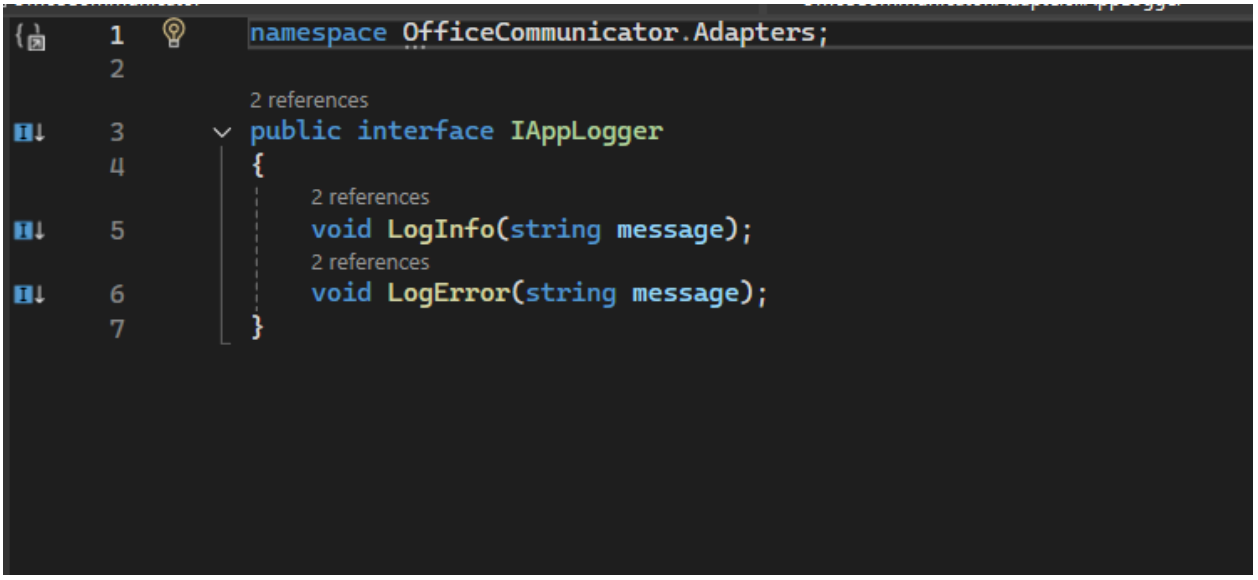
Console/File - це існуючі класи/API, які мають несумісний інтерфейс.

3. Програмна реалізація:

Для забезпечення гнучкості архітектури застосунку та дотримання принципів SOLID було реалізовано ряд патернів проектування. Нижче наведено лістинг коду для патерну Adapter.

Реалізація патерну Adapter: Патерн Adapter було використано для уніфікації інтерфейсу логування подій у системі. Це дозволяє легко перемикатися між виводом логів у консоль та у файл, не змінюючи код бізнес-логіки.

Інтерфейс IAppLogger.cs Цей інтерфейс визначає контракт, який мають виконувати всі адаптери логування.



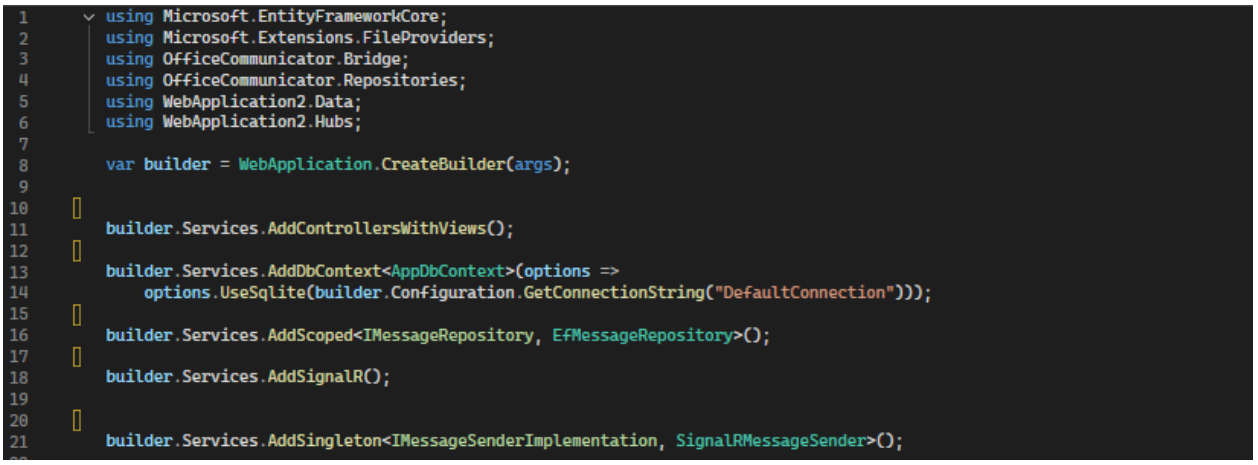
Адаптер ConsoleLoggerAdapter.cs Реалізація логера, що адаптує вивід повідомлень у стандартний потік виводу з кольоровим виділенням помилок.

```
namespace OfficeCommunicator.Adapters;

0 references
public class ConsoleLoggerAdapter : IAppLogger
{
    1 reference
    public void LogInfo(string message)
    {
        Console.ForegroundColor = ConsoleColor.Green;
        Console.WriteLine("[INFO] " + message);
        Console.ResetColor();
    }

    1 reference
    public void LogError(string message)
    {
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("[ERROR] " + message);
        Console.ResetColor();
    }
}
```

Адаптер для файлової системи (FileLoggerAdapter.cs) Реалізація, що адаптує запис подій у текстовий файл.

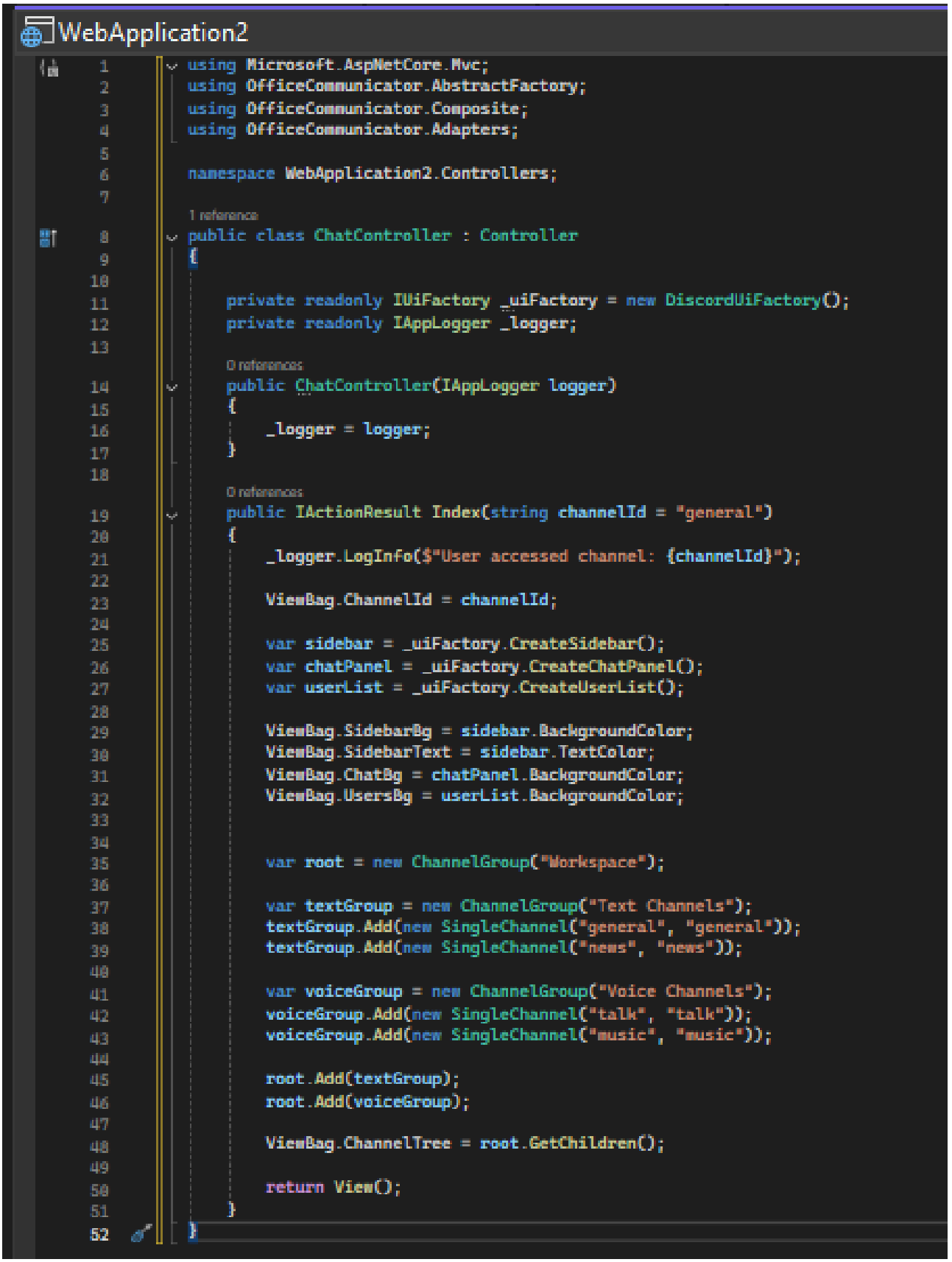
```
OfficeCommunicator                    OfficeCommunicator.Adapters.FileLoggerAdapter
namespace OfficeCommunicator.Adapters;

0 references
public class FileLoggerAdapter : IAppLogger
{
    private readonly string _filePath = "logs.txt";

    1 reference
    public void LogInfo(string message)
    {
        File.AppendAllText(_filePath, $"[INFO] {message}\n");
    }

    1 reference
    public void LogError(string message)
    {
        File.AppendAllText(_filePath, $"[ERROR] {message}\n");
    }
}
```

Реєстрація Адаптера в Program.cs У файлі конфігурації програми ми реєструємо залежність IAppLogger. Це дозволяє централізовано керувати тим, куди будуть записуватися логи (у консоль чи файл) для всього застосунку.



```
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.FileProviders;
using OfficeCommunicator.Bridge;
using OfficeCommunicator.Repositories;
using WebApplication2.Data;
using WebApplication2.Hubs;

var builder = WebApplication.CreateBuilder(args);


builder.Services.AddControllersWithViews();

builder.Services.AddDbContext<AppDbContext>(options =>
    options.UseSqlite(builder.Configuration.GetConnectionString("DefaultConnection")));

builder.Services.AddScoped<IMessageRepository, EfMessageRepository>();

builder.Services.AddSignalR();


builder.Services.AddSingleton<IMessageSenderImplementation, SignalRMessageSender>();
```

Приклад використання в коді (ChatController.cs) Контролер отримує екземпляр адаптера через конструктор (Dependency Injection) і використовує його для логування дій користувача. При цьому контролер не знає, куди саме пишуться логи (консоль чи файл), що робить код гнучким.


WebApplication2
using Microsoft.AspNetCore.Mvc;
using OfficeCommunicator.AbstractFactory;
using OfficeCommunicator.Composite;
using OfficeCommunicator.Adapters;
namespace WebApplication2.Controllers;
1 reference
public class ChatController : Controller
{
private readonly IUiFactory _uiFactory = new DiscordUiFactory();
private readonly IAppLogger _logger;
0 references
public ChatController(IAppLogger logger)
{
_logger = logger;
}
0 references
public IActionResult Index(string channelId = "general")
{
_logger.LogInfo($"User accessed channel: {channelId}");
ViewBag.ChannelId = channelId;
var sidebar = _uiFactory.CreateSidebar();
var chatPanel = _uiFactory.CreateChatPanel();
var userList = _uiFactory.CreateUserList();
ViewBag.SidebarBg = sidebar.BackgroundColor;
ViewBag.SidebarText = sidebar.TextColor;
ViewBag.ChatBg = chatPanel.BackgroundColor;
ViewBag.UsersBg = userList.BackgroundColor;
var root = new ChannelGroup("Workspace");
var textGroup = new ChannelGroup("Text Channels");
textGroup.Add(new SingleChannel("general", "general"));
textGroup.Add(new SingleChannel("news", "news"));
var voiceGroup = new ChannelGroup("Voice Channels");
voiceGroup.Add(new SingleChannel("talk", "talk"));
voiceGroup.Add(new SingleChannel("music", "music"));
root.Add(textGroup);
root.Add(voiceGroup);
ViewBag.ChannelTree = root.GetChildren();
return View();
}
}

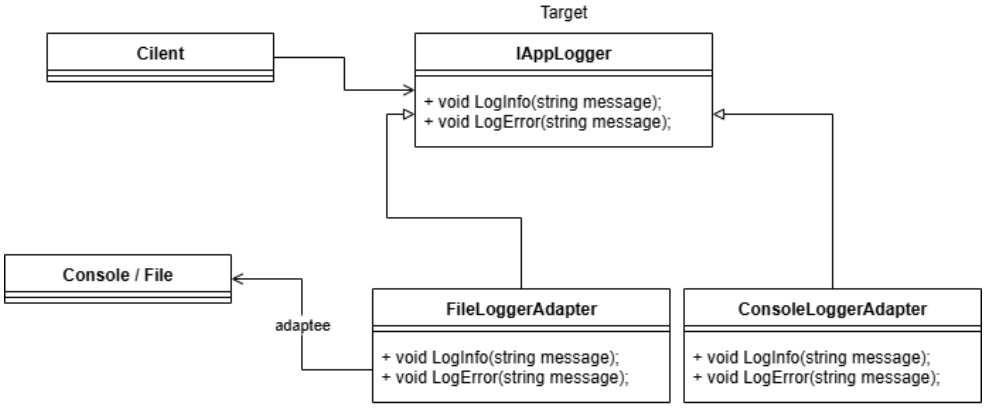
Висновок:Під час виконання лабораторної роботи я опанував концепцію структурних та породжуючих патернів проєктування, вивчив їхню класифікацію, а також проаналізував переваги та недоліки їх застосування.Для практичної реалізації було обрано патерн «Adapter» (Адаптер), який я імплементував у модулі логування подій.Вибір цього шаблону зумовлений його здатністю уніфікувати взаємодію компонентів: він дозволяє легко змінювати механізм запису логів (консоль або файл), відокремлюючи технічну реалізацію від бізнес-логіки. Реалізація включає спільний інтерфейс IAppLogger та конкретні адаптери ConsoleLoggerAdapter і FileLoggerAdapter.Це дозволило дотриматися принципу інверсії залежностей (Dependency Inversion Principle), зробивши систему гнучкою та незалежною від конкретних інструментів виводу інформації.

Відповіді на контрольні питання:

1. Яке призначення шаблону «Адаптер»?

   Шаблон «Адаптер» дозволяє змінити інтерфейс класу таким чином, щоб він відповідав вимогам іншої системи. Це дає змогу використовувати старі компоненти в новій системі без зміни їхнього коду. Адаптер перетворює інтерфейс одного класу в інтерфейс, очікуваний клієнтом.

2. Нарисуйте структуру шаблону «Адаптер».



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

   Адаптер перетворює виклики методів, зроблені клієнтом на інтерфейс цілі, в виклики методів адаптованого класу.

   - **Ціль (Target)** — це інтерфейс або клас, який використовується клієнтом. Це те, що клієнт очікує від адаптованого класу.
   - **Адаптер (Adapter)** — це клас, який реалізує інтерфейс цілі (Target) і забезпечує взаємодію з адаптованим класом.
   - **Адаптований клас (Adaptee)** — це клас, який містить реальну бізнес-логіку, але має інтерфейс, що не підходить для використання в цільовій системі.

4. Яка різниця між реалізацією «Адаптера» на рівні об’єктів та на рівні класів?

   Адаптер на рівні об’єктів — адаптер створюється як окремий об'єкт, який делегує виклики методів до об'єкта адаптованого класу. Кожен адаптер є об'єктом.

Приклад: Клас адаптера створюється для кожного інстансу адаптованого класу, і адаптер передає виклики методів.

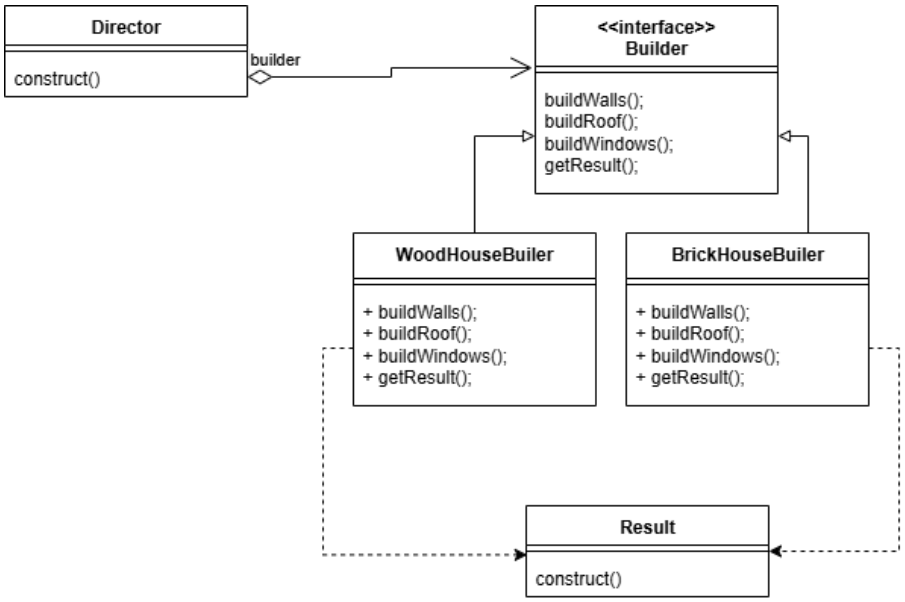
Адаптер на рівні класів — адаптер є статичним класом або статичними методами, які використовуються для перетворення інтерфейсу на рівні класу.

Приклад: Це підхід, де адаптер реалізується через статичні методи або використовує шаблон «Фабрика» для створення об'єктів, які підходять під цільовий інтерфейс.

5. Яке призначення шаблону «Будівельник»?

   Шаблон «Будівельник» використовується для створення складних об'єктів, що складаються з кількох частин. Цей шаблон дозволяє розділити процес побудови об'єкта на окремі кроки, забезпечуючи таким чином гнучкість у створенні складних об'єктів. Це корисно, коли об'єкт має багато можливих варіацій, і не хочеться створювати кожну комбінацію вручну.

6. Нарисуйте структуру шаблону «Будівельник».



7. Які класи входять в шаблон «Будівельник», та яка між ними взаємодія?

- **Будівельник (Builder)** — це абстракція, яка визначає загальний інтерфейс для створення різних частин об'єкта.
- **Конкретний будівельник (ConcreteBuilder)** — це клас, який реалізує інтерфейс будівельника і визначає, як будувати конкретні частини об'єкта.
- **Продукт (Product)** — це складний об'єкт, який будується.
- **Директор (Director)** — це клас, який інструктує будівельника, як створювати об'єкт. Він визначає порядок етапів будівництва.

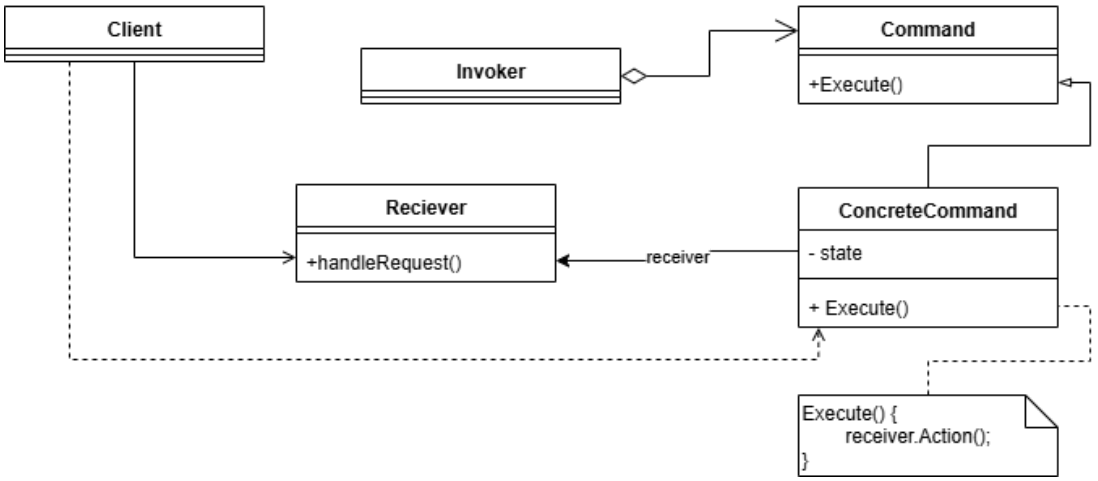
8. У яких випадках варто застосовувати шаблон «Будівельник»?

   Це доречно у випадках, коли об'єкт має складний процес створення (наприклад, Webсторінка як елемент повної відповіді web- сервера) або коли об'єкт повинен мати декілька різних форм створення (наприклад, при конвертації тексту з формату у формат).

9. Яке призначення шаблону «Команда»?

   Шаблон "command" (команда) перетворить звичайний виклик методу в клас. Таким чином дії в системі стають повноправними об'єктами.

10. Нарисуйте структуру шаблону «Команда».



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

- **Команда (Command)** — це абстракція, яка визначає інтерфейс для виконання запиту.

- **Конкретна команда (ConcreteCommand)** — це конкретна реалізація команди, яка виконує певну дію.
- **Приймач (Receiver)** — це об'єкт, який фактично виконує дію.
- **Ініціатор (Invoker)** — це клас, який зберігає команду і викликає її для виконання.
- **Клієнт (Client)** — створює конкретні команди та налаштовує ініціатора.

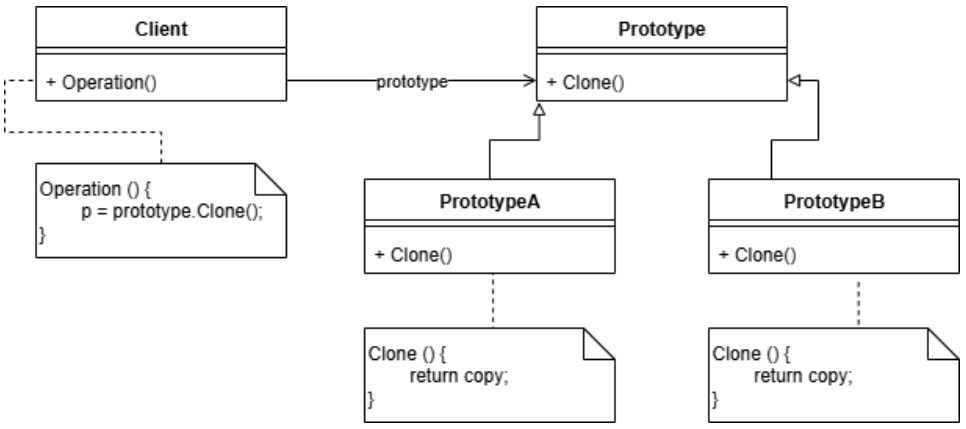
12. Розкажіть як працює шаблон «Команда».

    Шаблон «Команда» дозволяє інкапсулювати запит як об'єкт, що включає всі необхідні параметри для виконання дії, і передати його до виконавця (отримувача). Це дозволяє розділити ініціатора запиту (відправника) і виконавця (отримувача), що забезпечує більшу гнучкість у реалізації команд.

13. Яке призначення шаблону «Прототип»?

    Шаблон «Prototype» (Прототип) використовується для створення об'єктів за «шаблоном» (чи «кресленням», «ескізом») шляхом копіювання шаблонного об'єкту, який називається прототипом.

14. Нарисуйте структуру шаблону «Прототип».



15. Які класи входять в шаблон «Прототип», та яка між ними взаємодія?

- **Прототип (Prototype)** — це абстрактний клас або інтерфейс, який надає методи для копіювання об'єктів.

- **Конкретний прототип (ConcretePrototype)** — це клас, який реалізує метод копіювання та визначає конкретну реалізацію об'єкта.

16. Які можна привести приклади використання шаблону «Ланцюжок відповідальності»?
    - Обробка HTTP-запитів в веб-сервері, де різні обробники відповідають за різні аспекти запиту (наприклад, авторизація, логування, маршрутизація).
    - Обробка винятків у програмі, коли кожен обробник може вирішити, чи обробляти виняток, чи передати його далі.
    - Обробка елементів інтерфейсу користувача в складних формах, де кожен обробник перевіряє введення та приймає рішення щодо подальшої