

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 №6
із дисципліни «*Технології розробки програмного забезпечення*»
Тема: «*Патерни проектування*»

Виконав:

Студент групи ІА-34

Ковальчук Станіслав

Перевірив:

асистент кафедри ІСТ

Мягкий Михайло Юрійович

Тема: Патерни проектування.

Мета: Вивчити структуру шаблонів «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator» 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 «Абстрактна фабрика» використовується для створення сімейств об'єктів без вказівки їх конкретних класів. Для цього виноситься загальний інтерфейс фабрики (AbstractFactory) і створюються його реалізації для різних сімейств продуктів. Цей шаблон передусім структурує знання про схожі об'єкти (що називаються сімействами, як класи для доступу до БД) і створює можливість взаємозаміни різних сімейств.

Шаблон «Фабричний метод» визначає інтерфейс для створення об'єктів певного базового типу. Це зручно, коли хочеться додати можливість створення об'єктів не базового типу, а деякого дочірнього. Фабричний метод у такому разі є зачіпкою для впровадження власного конструктора об'єктів.

Шаблон «Memento» використовується для збереження і відновлення стану об'єктів без порушення інкапсуляції. Об'єкт «Memento» служить виключно для збереження змін над початковим об'єктом (Originator).

Шаблон «Observer» визначає залежність «один-до-багатьох» таким чином, що коли один об'єкт змінює власний стан, усі інші об'єкти отримують про це сповіщення і мають можливість змінити власний стан також.

Шаблон «Декоратор» призначений для динамічного додавання функціональних можливостей об'єкту під час роботи програми. Декоратор деяким чином «обертає» (за рахунок агрегації) початковий об'єкт зі збереженням його функцій, проте дозволяє додати додаткові дії. Такий шаблон надає гнучкіший спосіб зміни поведінки об'єкту чим просте спадкоємство, оскільки початкова функціональність зберігається в повному об'ємі. Більше того, таку поведінку можна застосовувати до окремих об'єктів, а не до усієї системи в цілому.

Хід роботи:

1. Повна діаграма класів (без шаблону).

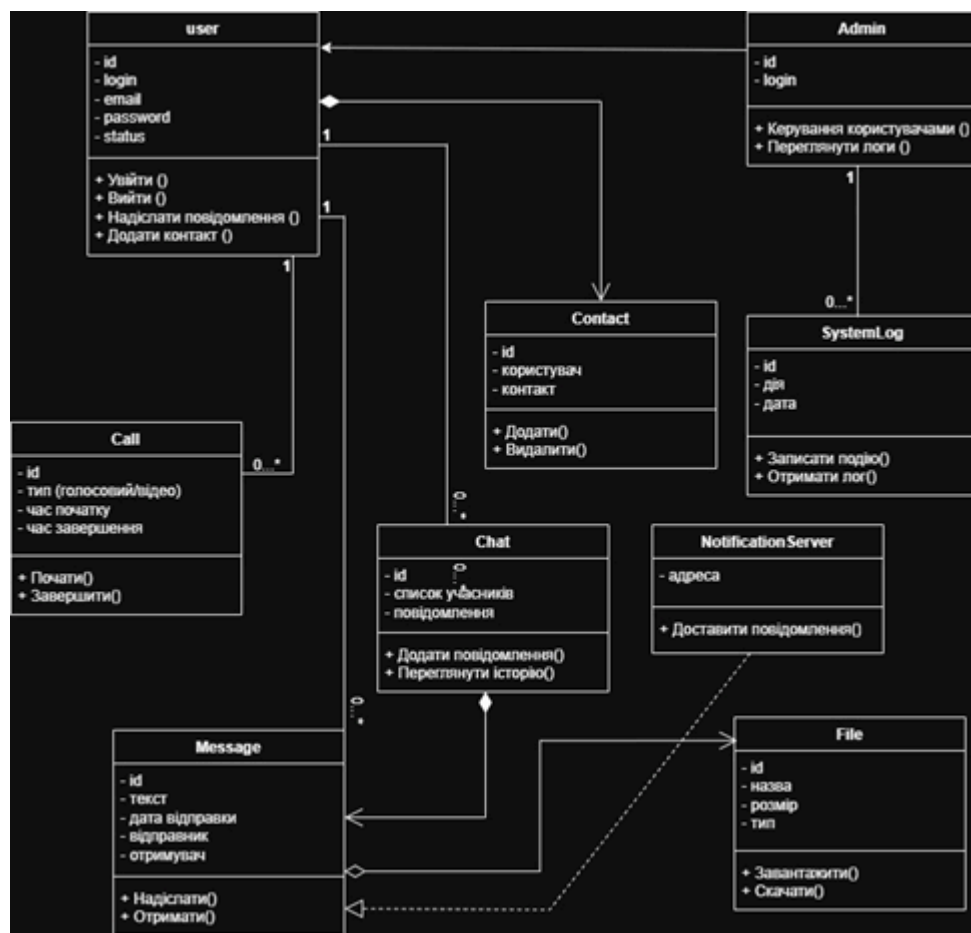


Рис. 6.1 – Діаграма класів системи

2. Діаграма класів з використанням шаблону «Abstract Factory»:

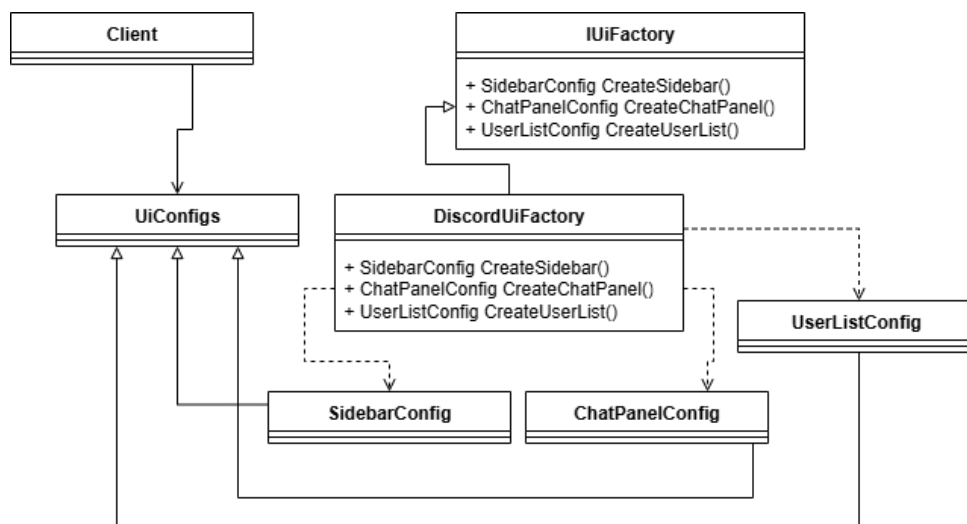


Рис. 6.2 – Діаграма класів системи з використанням шаблону «Abstract Factory»

Я визначаю інтерфейс IUiFactory, який оголошує методи для створення сімейства пов'язаних продуктів.

DiscordUiFactory - конкретна реалізація абстрактної фабрики, а також створює конкретне сімейство продуктів (Discord-тема).

SidebarConfig, ChatPanelConfig, UserListConfig - представляють типи продуктів, які створює фабрика.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ

Для забезпечення гнучкості налаштування інтерфейсу та реалізації системи сповіщень у реальному часі було імплементовано патерни **Abstract Factory** (Абстрактна фабрика) та **Observer** (Спостерігач).

Реалізація патерну Abstract Factory Цей патерн використано для створення системи тем оформлення. Він дозволяє гарантувати, що всі елементи інтерфейсу (сайдбар, панель чату, список користувачів) створюються в єдиному візуальному стилі (наприклад, Discord-style), не прив'язуючи код до конкретних класів віджетів.

Абстрактна фабрика UI ([IUiFactory.cs](#)):

```

1 namespace OfficeCommunicator.AbstractFactory;
2
3 public interface IUiFactory
4 {
5     SidebarConfig CreateSidebar();
6     ChatPanelConfig CreateChatPanel();
7     UserListConfig CreateUserList();
8 }

```

Конкретна фабрика Discord (DiscordUiFactory.cs) Реалізує створення компонентів у темній кольоровій гамі, характерній для месенджера Discord:

```

1 namespace OfficeCommunicator.AbstractFactory;
2
3 public class DiscordUiFactory : IUiFactory
4 {
5     public SidebarConfig CreateSidebar()
6         => new SidebarConfig("#202225", "#ffffff");
7
8     public ChatPanelConfig CreateChatPanel()
9         => new ChatPanelConfig("#36393f");
10
11     public UserListConfig CreateUserList()
12         => new UserListConfig("#2f3136");
13 }

```

Реалізація патерну Observer Патерн Observer (Спостерігач) реалізовано для механізму підписки на повідомлення. Коли в чаті з'являється нове повідомлення, всі підключені клієнти (спостерігачі) автоматично отримують сповіщення.

Інтерфейс спостерігача ([IChatObserver.cs](#))

```
OfficeCommunicator
1 namespace OfficeCommunicator.Patterns.Observer;
2
3 3 references
4 public interface IChatObserver
5 {
6     1 reference
7     void Update(string message);
8 }
```

Суб'єкт чату (ChatRoomSubject.cs) Клас, що зберігає список підписників та повідомляє їх про зміни.

```
OfficeCommunicator
1 namespace OfficeCommunicator.Patterns.Observer;
2
3 0 references
4 public class ChatRoomSubject
5 {
6     private readonly List<IChatObserver> _observers = new();
7
8     0 references
9     public void Attach(IChatObserver observer) => _observers.Add(observer);
10    0 references
11    public void Detach(IChatObserver observer) => _observers.Remove(observer);
12
13    1 reference
14    public void Notify(string message)
15    {
16        foreach (var observer in _observers)
17        {
18            observer.Update(message);
19        }
20    }
21
22    0 references
23    public void NewMessage(string message)
24    {
25        Notify(message);
26    }
27 }
```

Висновок: Під час виконання лабораторної роботи я поглибив свої знання про породжуючі та поведінкові патерни проєктування, детально вивчив їхню класифікацію, а також проаналізував архітектурні переваги та недоліки їх

застосування у розподілених системах. Для практичної реалізації в системі "Office Communicator" було обрано та імплементовано два ключові патерни: «Abstract Factory» (Абстрактна Фабрика) та «Observer» (Спостерігач).

1. Вибір патерну «Abstract Factory» зумовлений необхідністю створення гнучкої системи налаштування інтерфейсу користувача. Я реалізував це через інтерфейс IUiFactory та конкретну фабрику DiscordUiFactory, що дозволяє створювати сімейства взаємопов'язаних графічних компонентів (панелей, списків, меню) у єдиному візуальному стилі. Такий підхід гарантує цілісність дизайну та дозволяє додавати нові теми оформлення без модифікації основного коду програми, дотримуючись принципу відкритості/закритості (ОСР).
2. Патерн «Observer» було застосовано для реалізації підсистеми сповіщень у реальному часі. Використання цього шаблону (зокрема, через механізми бібліотеки SignalR) дозволило налаштувати асинхронну взаємодію «один-до-багатьох» між сервером та клієнтами. Це забезпечило миттєву доставку повідомлень у чаті без необхідності постійного опитування сервера, що значно підвищило продуктивність системи та покращило користувацький досвід.

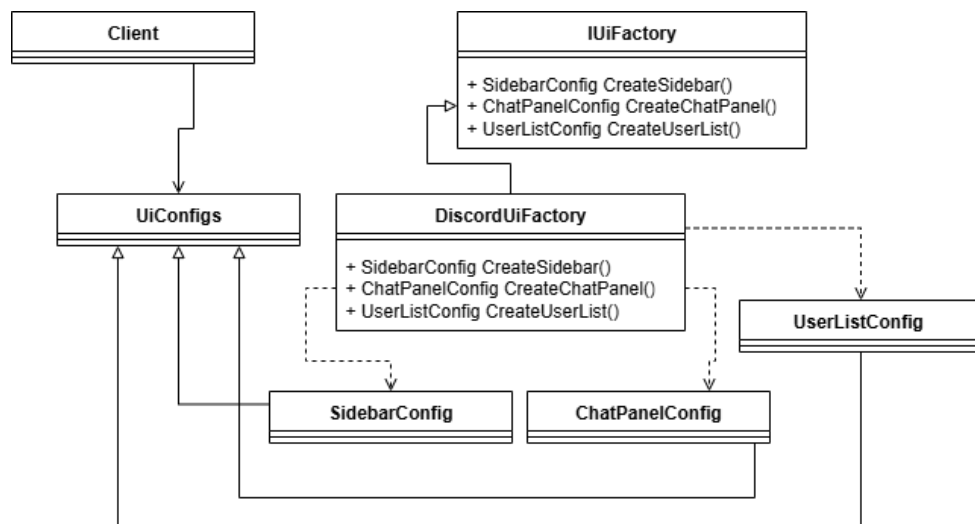
У результаті виконання роботи було створено модульну архітектуру, яка характеризується слабкою зв'язністю компонентів та високою придатністю до масштабування.

Відповіді на контрольні питання:

1. Яке призначення шаблону «Абстрактна фабрика»?

«Абстрактна фабрика» дозволяє створювати сімейства взаємопов'язаних або залежних об'єктів, не вказуючи їх конкретні класи. Це корисно, коли потрібно створювати об'єкти, що повинні бути сумісними між собою (наприклад, різні види інтерфейсів користувача або різні бази даних).

2. Нарисуйте структуру шаблону «Абстрактна фабрика».



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

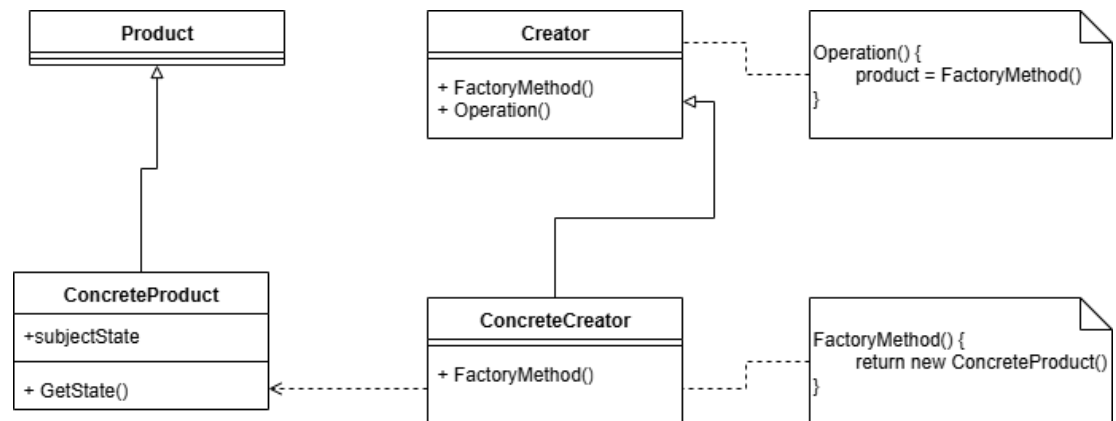
Шаблон «Абстрактна фабрика» складається з таких основних класів:

- Абстрактна фабрика (**AbstractFactory**) — це інтерфейс або абстрактний клас, який визначає методи для створення різних абстрактних продуктів.
- Конкретна фабрика (**ConcreteFactory**) — це клас, який реалізує інтерфейс абстрактної фабрики і створює конкретні об'єкти продуктів.
- Абстрактний продукт (**AbstractProduct**) — це інтерфейс або абстрактний клас для продуктів, що створюються.
- Конкретний продукт (**ConcreteProduct**) — це клас, який реалізує інтерфейс абстрактного продукту і визначає конкретні об'єкти.

4. Яке призначення шаблону «Фабричний метод»?

Шаблон «Фабричний метод» дозволяє визначити інтерфейс для створення об'єктів, але дозволяє підкласам змінювати тип створюваних об'єктів. Це дозволяє делегувати створення об'єкта дочірнім класам, що дає змогу створювати різні варіанти об'єктів без зміни клієнтського коду.

5. Нарисуйте структуру шаблону «Фабричний метод».



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

До шаблону проєктування «Фабричний метод» входять такі класи:

- Продукт (Product) - Інтерфейс або абстрактний клас, який оголошує тип об'єктів, що виробляються фабричним методом. Усі конкретні продукти повинні реалізовувати цей інтерфейс.
- Конкретний продукт (Concrete Product) - конкретна реалізація інтерфейсу Продукту.
- Творець (Creator) - Абстрактний клас або інтерфейс, який оголошує фабричний метод — метод, що повертає об'єкт типу Продукт.
- Конкретний творець (Concrete Creator) - Підклас класу Творця, який перевизначає (реалізує) фабричний метод для створення та повернення конкретного екземпляра Конкретного продукту.

7. Чим відрізняється шаблон «Абстрактна фабрика» від «Фабричний метод»?

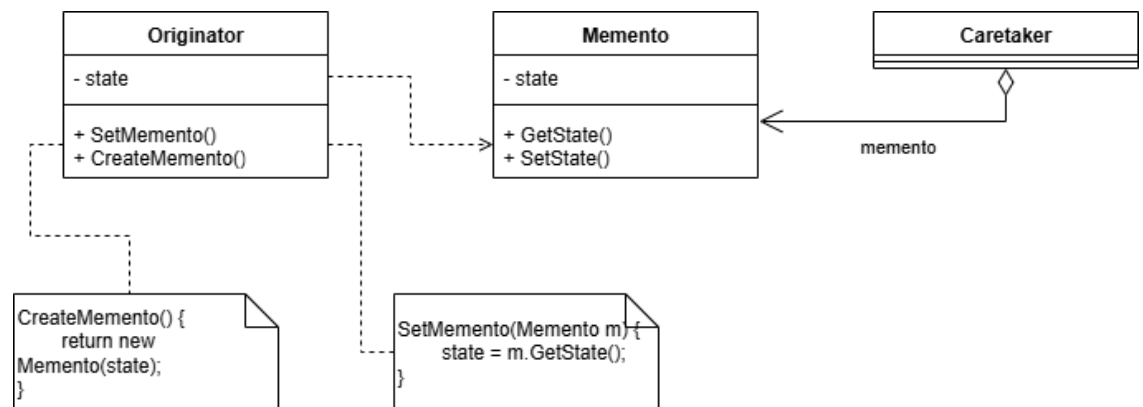
Основні відмінності:

- **Шаблон «Абстрактна фабрика»** створює сімейства взаємопов'язаних продуктів (наприклад, різні інтерфейси користувача для різних операційних систем). У цьому шаблоні фабрика створює різні типи продуктів, що мають спільний інтерфейс.
- **Шаблон «Фабричний метод»** дозволяє створювати конкретний об'єкт певного типу, залишаючи підкласам реалізацію створення. Він більше фокусується на створенні одного типу об'єкта.

8. Яке призначення шаблону «Знімок»?

Шаблон «Знімок» (Memento) дозволяє зберігати та відновлювати внутрішній стан об'єкта без порушення інкапсуляції. Це дозволяє зберігати знімки (сторінки) об'єкта в певний момент часу і відновлювати їх у майбутньому.

9. Нарисуйте структуру шаблону «Знімок».



10. Які класи входять в шаблон «Знімок», та яка між ними взаємодія?

Шаблон «Знімок» включає такі класи:

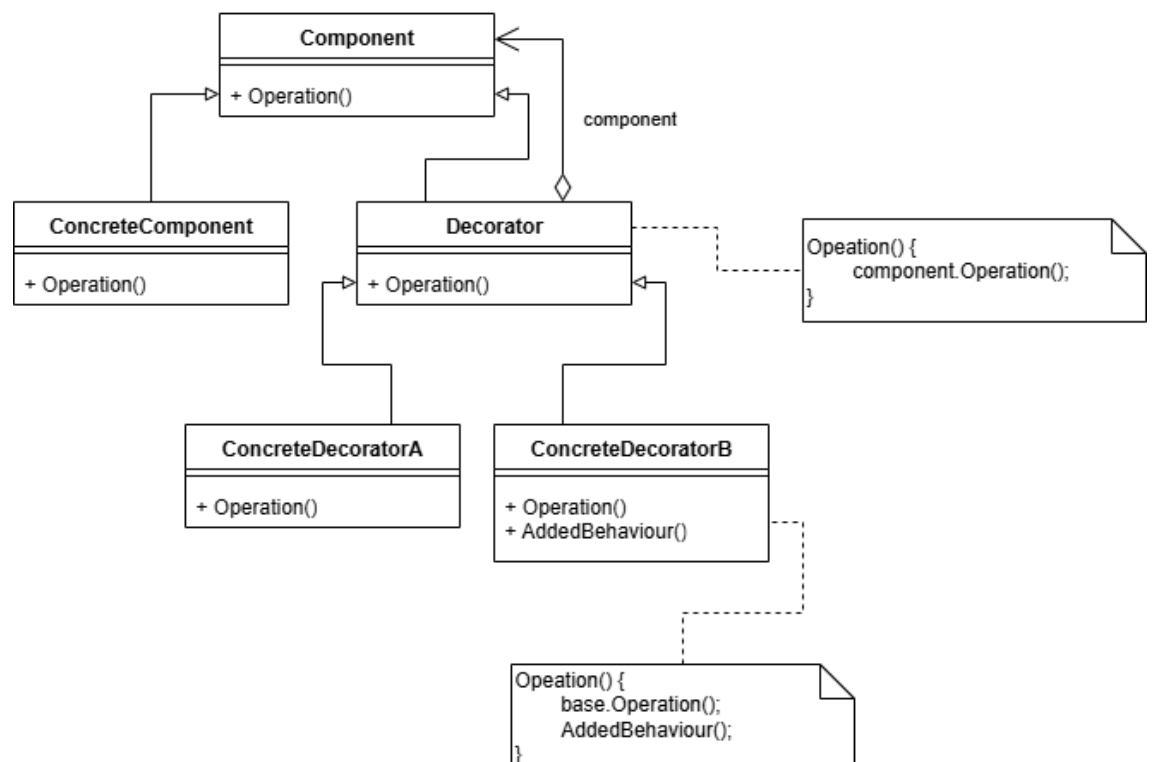
- **Знімок (Memento)** — це клас, який зберігає стан об'єкта. Зазвичай він є незмінним після створення.
- **Поціновувач (Caretaker)** — це клас, який зберігає знімки та може їх відновлювати. Він не змінює стан знімка, лише зберігає або передає його.

- **Оригінал (Originator)** — це клас, чий стан зберігається у знімках. Оригінал може створювати нові знімки та відновлювати свій стан з них.

11. Яке призначення шаблону «Декоратор»?

Шаблон «Декоратор» дозволяє динамічно додавати нові функціональні можливості об'єкту, не змінюючи його структуру. Це корисно, коли потрібно додати нову поведінку без модифікації існуючого коду.

12. Нарисуйте структуру шаблону «Декоратор».



13. Які класи входять в шаблон «Декоратор», та яка між ними взаємодія?

Шаблон «Декоратор» складається з таких класів:

- **Компонент (Component)** — це абстракція для об'єктів, до яких можна додавати поведінку.
- **Конкретний компонент (ConcreteComponent)** — це клас, який реалізує базову поведінку об'єкта.

- Декоратор (Decorator) — це клас, який також реалізує інтерфейс компонента і містить посилання на об'єкт компонента. Декоратор може додавати нову поведінку до цього об'єкта.
- Конкретний декоратор (ConcreteDecorator) — це клас, який додає конкретну нову поведінку до об'єкта, використовуючи його через декоратор.

14. Які є обмеження використання шаблону «декоратор»?

Шаблон «Декоратор» ідеально підходить для змін, які можна додавати динамічно без зміни базового коду, але його використання потребує обережності, якщо об'єктів та їх комбінацій стає занадто багато.