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 № 5

із дисципліни «Технології розроблення програмного забезпечення»

Тема: «Патерни проектування»

Виконав
Студент групи ІА-31:
Трегуб К. В.

Перевірив:
Мягкий М. Ю.

Київ 2025

Зміст

1. **Мета:**

Вивчити структуру шаблонів «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator» та навчитися застосовувати їх в реалізації програмної системи.

2. **Теоретичні відомості:**

Abstract Factory – створює групи пов’язаних об’єктів без зазначення конкретних класів. Використовується, коли потрібно забезпечити узгодженість елементів одного стилю чи типу (наприклад, об’єкти різних стилів у грі).

Factory Method – задає спільний інтерфейс для створення об’єктів, дозволяючи підкласам самостійно визначати, який клас інстанціювати. Зручний для розширення системи новими типами.

Memento (Знімок) – дає змогу зберігати та відновлювати стан об’єкта, не порушуючи інкапсуляцію. Стан фіксується в окремому знімку, доступному лише вихідному об’єкту.

Observer (Спостерігач) – організовує зв’язок «один-до-багатьох»: зміна стану одного об’єкта автоматично сповіщає всі підписані. Приклад — система сповіщень.

Decorator (Декоратор) – дозволяє додавати нову функціональність об’єкту під час виконання, «обгортаючи» його без зміни початкового коду.

3. **Хід роботи:**

Тема :

Beauty Salon Booking System (State, Chain of Responsibility, Observer, Facade, Composite, Client-Server) — веб-застосунок для бронювання послуг салону краси, що підтримує управління життєвим циклом запису, проходження валідованого процесу створення бронювання, автоматичне сповіщення користувачів про зміни статусу, централізовану обробку оплати через фасад та роботу як з окремими послугами, так і з пакетними пропозиціями. Система побудована за архітектурою клієнт–сервер з розмежуванням бізнес-логіки, управління станом і відображенням інтерфейсу.

1) Ознайомитись з короткими теоретичними відомостями.

2) Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
3) Реалізувати один з розглянутих шаблонів за обраною темою.
4) Реалізувати не менше 3-х класів відповідно до обраної теми.
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.

У системі бронювання існує потреба надсилати повідомлення клієнту (Email або SMS) при настанні певних подій: успішне створення замовлення, підтвердження адміністратором, скасування або оплата. Жорстке кодування викликів emailService.send() або smsService.send() безпосередньо в методах бізнес-логіки (BookingService) порушує принцип відкритості/закритості (OCP) та робить систему важкою для розширення. Якщо з'явиться новий канал сповіщень (наприклад, Telegram), доведеться змінювати код сервісу. Патерн Observer дозволяє створити механізм підписки, де BookingService виступає як видавець подій (Subject), а сервіси сповіщень (EmailObserver, SmsObserver) — як підписники.

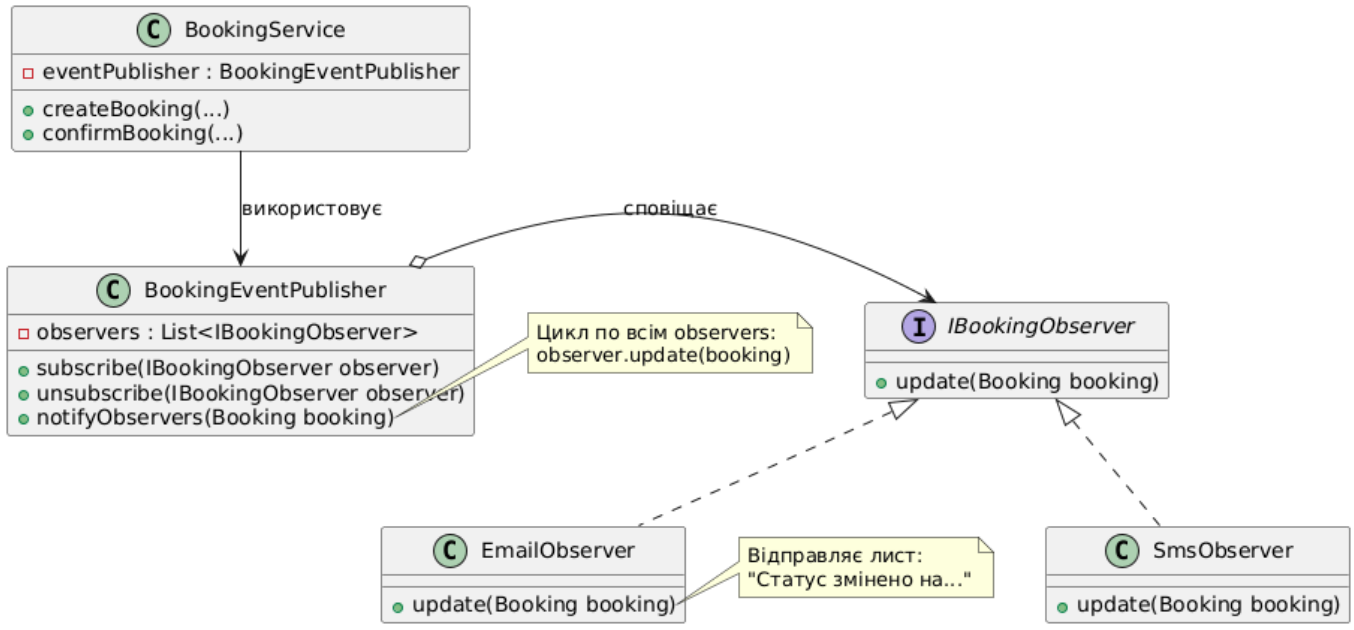


Рисунок 1 - Структура патерну Observer

IBookingObserver – це загальний інтерфейс для всіх підписників. Він визначає єдиний метод update(Booking booking), який викликається при настанні події. Це дозволяє BookingEventPublisher працювати з будь-якими типами спостерігачів однаково.

BookingEventPublisher (Subject) – клас, що керує списком підписників. Він надає методи subscribe() для додавання нових слухачів та notifyObservers() для розсилки повідомлень усім зареєстрованим об'єктам.

EmailObserver / SmsObserver (Concrete Observers) – конкретні реалізації інтерфейсу. EmailObserver формує та відправляє електронний лист, а SmsObserver — SMS-повідомлення. Вони реалізують власну логіку реакції на зміну статусу бронювання.

BookingService (Client) – ініціатор подій. Коли відбувається зміна стану бронювання (створення, оплата тощо), сервіс делегує задачу сповіщення об'єкту BookingEventPublisher, не знаючи, хто саме і як буде обробляти це сповіщення.

```
package com.beautysalon.booking.observer;
import com.beautysalon.booking.entity.Booking;

public interface IBookingObserver {
    void update(Booking booking);
}
```

Рисунок 2 - Інтерфейс спостерігача

```
@Service
public class BookingEventPublisher {

    private final List<IBookingObserver> observers = new ArrayList<>();

    public void subscribe(IBookingObserver observer) {
        System.out.println("BookingEventPublisher: Новий підписник -> " + observer.getClass().getSimpleName());
        observers.add(observer);
    }

    public void unsubscribe(IBookingObserver observer) {
        observers.remove(observer);
    }

    public void notifyObservers(Booking booking) {
        System.out.println("BookingEventPublisher: Повідомляємо " + observers.size() + " спостерігачів про зміну статусу...");
        for (IBookingObserver observer : observers) {
            try {
                observer.update(booking);
            } catch (Exception e) {
                System.err.println("Помилка при повідомленні спостерігача " +
                                   observer.getClass().getSimpleName() + ": " + e.getMessage());
            }
        }
    }
}
```

Рисунок 3 – Клас - видавець

```java
@Configuration
public class ObserverConfig {

    @Autowired
    private BookingEventPublisher publisher;

    @Autowired
    private EmailObserver emailObserver;

    @Autowired
    private SmsObserver smsObserver;
```

Рисунок 4 – Конфігурація (ObserverConfig.java)

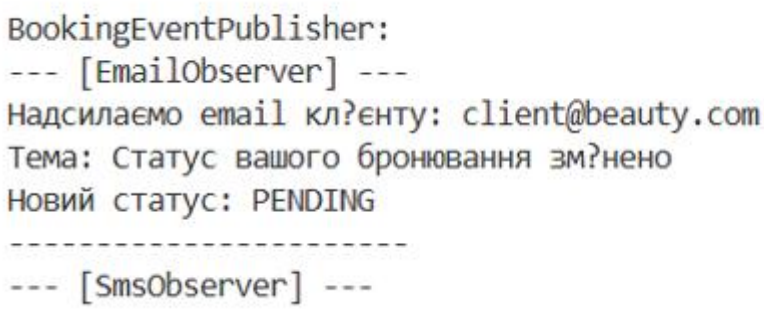


Рисунок 5 – Демонстрація результату

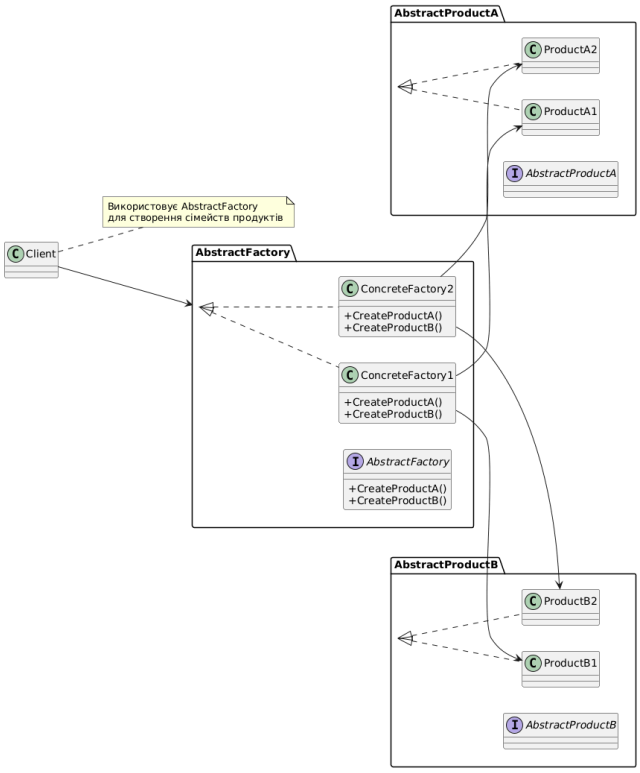
4. Висновок

У ході лабораторної роботи було реалізовано поведінковий патерн Observer, який забезпечив гнучкий механізм асинхронного сповіщення користувачів у системі бронювання послуг. Використання окремого видавця подій (BookingEventPublisher) та інтерфейсу спостерігачів (IBookingObserver) дозволило відокремити бізнес-логіку бронювання від логіки надсилання повідомлень, досягнувши слабкої зв'язності між компонентами. Такий підхід зробив систему розширюваною — додавання нових каналів сповіщень потребує лише створення нового класу-спостерігача. Крім того, сервіс бронювання залишився чистим і не прив'язаним до конкретних способів доставки повідомлень, а підписники можуть динамічно змінюватися під час роботи застосунку, що підвищує гнучкість системи.

5. Контрольні питання:

1. Яке призначення шаблону «Абстрактна фабрика» Шаблон «Абстрактна фабрика» використовується для створення сімейств об'єктів без вказівки їх конкретних класів.

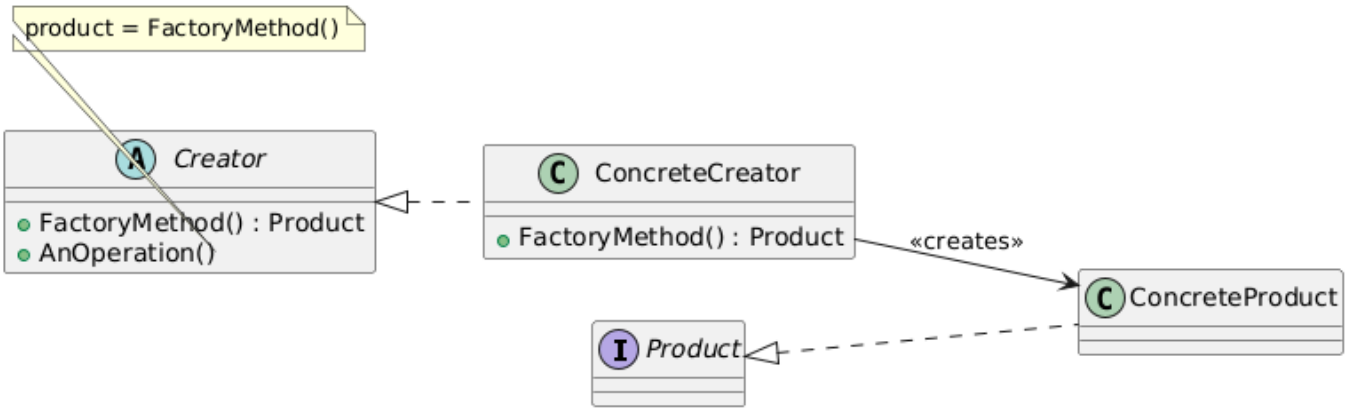
2. Нарисуйте структуру шаблону «Абстрактна фабрика»



3. Які класи входять в шаблон «Абстрактна фабрика», та яка між ними взаємодія? Класи: AbstractFactory, ConcreteFactory, AbstractProductA/B, ConcreteProductA/B, Client. Взаємодія: Client використовує AbstractFactory для створення об'єктів через інтерфейси продуктів; ConcreteFactory реалізує методи створення конкретних продуктів одного сімейства.

4. Яке призначення шаблону «Фабричний метод»? Шаблон «Фабричний метод» визначає інтерфейс для створення об'єктів певного базового типу, залишаючи реалізацію підтипам.

5. Нарисуйте структуру шаблону «Фабричний метод»



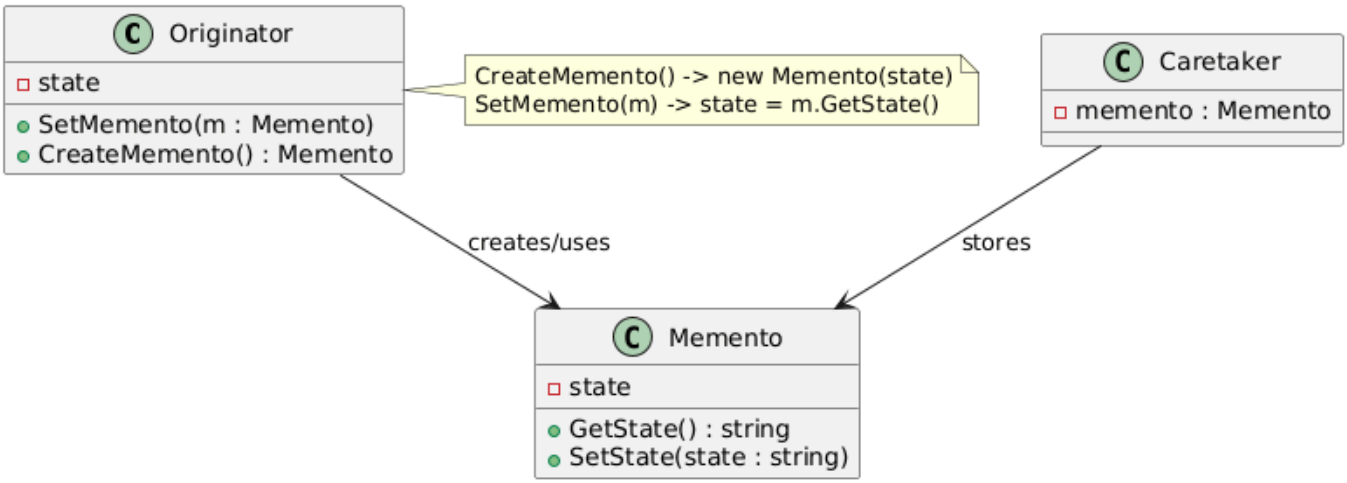
6. Які класи входять в шаблон «Фабричний метод», та яка між ними взаємодія? Класи: Creator, ConcreteCreator, Product, ConcreteProduct. Взаємодія: Creator

викликає FactoryMethod() для створення об'єкта; ConcreteCreator повертає конкретний ConcreteProduct.

7. Чим відрізняється шаблон «Абстрактна фабрика» від «Фабричний метод»? «Абстрактна фабрика» створює сімейства пов’язаних об'єктів через набір методів; «Фабричний метод» створює один об'єкт через один віртуальний метод, реалізований у підкласах.

8. Яке призначення шаблону «Знімок»? Шаблон «Знімок» використовується для збереження і відновлення стану об'єктів без порушення інкапсуляції.

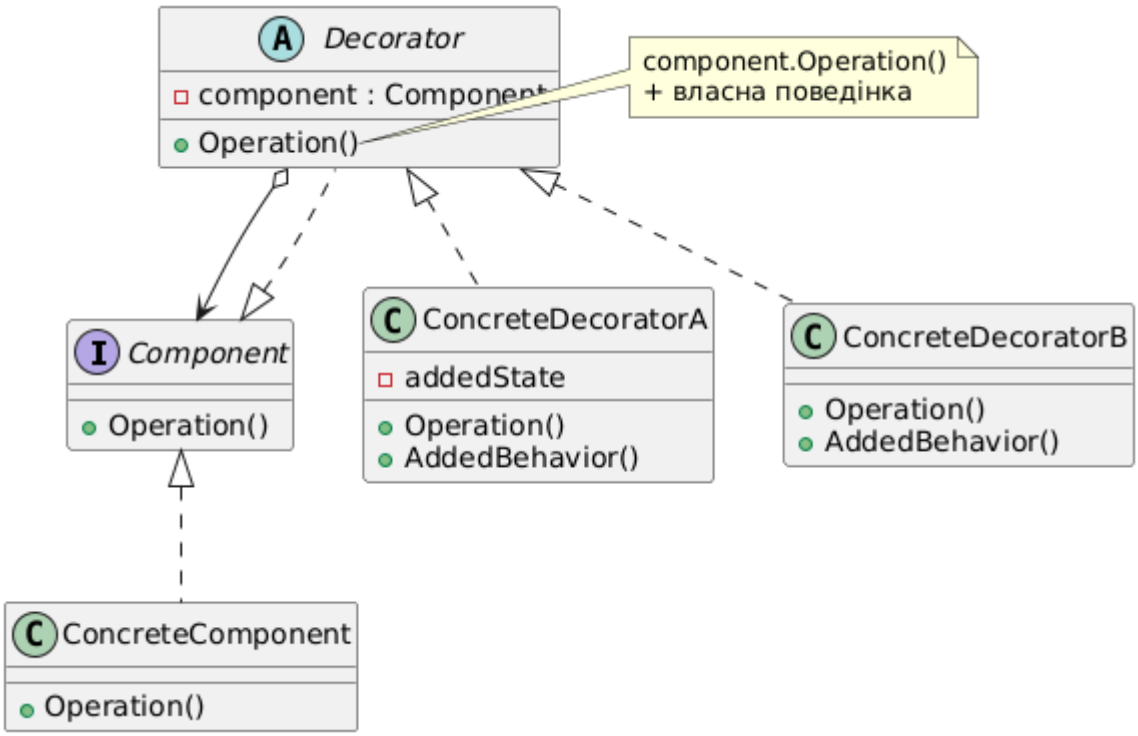
9. Нарисуйте структуру шаблону «Знімок»



10. Які класи входять в шаблон «Знімок», та яка між ними взаємодія? Класи: Originator, Memento, Caretaker. Взаємодія: Originator створює/відновлює Memento; Caretaker зберігає Memento, не маючи доступу до його вмісту.

11. Яке призначення шаблону «Декоратор»? Шаблон «Декоратор» призначений для динамічного додавання функціональних можливостей об'єкту під час роботи програми.

12. Нарисуйте структуру шаблону «Декоратор»



13. Які класи входять в шаблон «Декоратор», та яка між ними взаємодія? Класи: Component, ConcreteComponent, Decorator, ConcreteDecorator. Взаємодія: Decorator обгортає Component, викликає його Operation() і додає власну поведінку.

14. Які є обмеження використання шаблону «Декоратор»? Велика кількість крихітних класів; важко конфігурувати об’єкти, загорнуті в декілька обгорток одночасно.