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 «Adapter», «Builder», «Command», «Chain of responsibility», «Prototype» та навчитися застосовувати їх в реалізації програмної системи.

2. **Теоретичні відомості:**

Adapter: Адаптує інтерфейс одного об’єкта до іншого, дозволяючи несумісним компонентам працювати разом (наприклад, уніфікація інтерфейсів бібліотек принтерів). Спрощує інтеграцію без змін у коді.
Builder: Відокремлює процес створення об’єкта від його представлення, доречний для складних або багатоформних конструкцій (наприклад, відповіді веб-сервера). Забезпечує гнучкий контроль.
Command: Перетворює виклик методу в об’єкт, дозволяючи гнучкі системи команд із відміною, логуванням чи плануванням (наприклад, дії в інтерфейсі). Покращує модульність.
Chain of Responsibility: Передає запити по ланцюжку обробників, поки один не виконає (наприклад, контекстні меню). Зменшує зв’язки та спрощує зміни.
Prototype: Створює об’єкти клонуванням прототипу, корисний для складних об’єктів або динамічних змін (наприклад, редактор рівнів гри). Зменшує ієрархію спадкування.

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.

Для системи реалізовано патерн Chain of Responsibility (Ланцюжок обов'язків) для механізму комплексної валідації створення бронювання. Логіка процесу: Клієнт ініціює створення бронювання => BookingService формує контекст запиту (BookingValidationContext) => передає його в ланцюжок валідаторів. Запит проходить через ланки: ClientExistenceHandler (чи є клієнт?) => MasterExistenceHandler (чи є майстер?) => ServiceExistenceHandler (чи є послуга?) => MasterServiceCompatibilityHandler (чи надає майстер цю послугу?). Якщо будь-яка ланка виявляє помилку, обробка припиняється, і повертається причина відмови. Якщо ланцюжок пройдено успішно — бронювання зберігається.

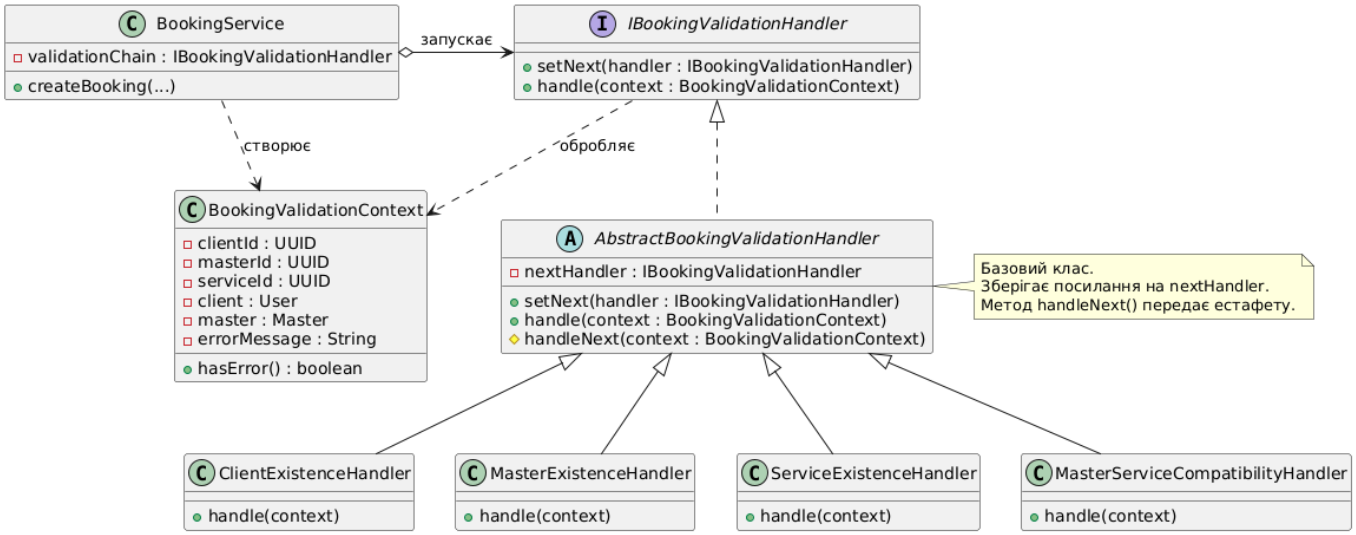


Рисунок 1 - Структура патерну Chain of Responsibility

- BookingService – виступає в ролі Клієнта (ініціатора запиту). Він конструює ланцюжок обробників у своєму конструкторі та запускає виконання методу handle().
- BookingValidationContext – об'єкт-контекст (DTO), який передається між ланками. Він містить вхідні дані (ID) і накопичує результати (знайдені сутності або повідомлення про помилку).
- IBookingValidationHandler – загальний інтерфейс, що визначає метод обробки handle() та метод встановлення наступної ланки setNext().
- AbstractBookingValidationHandler – базовий абстрактний клас, що реалізує механізм передачі запиту далі по ланцюжку (handleNext). Це дозволяє конкретним класам зосередитися лише на своїй логіці перевірки.
- ConcreteHandlers (ClientExistenceHandler, MasterExistenceHandler...) – конкретні реалізації перевірок. Кожен клас відповідає за одну вузьку задачу (Single Responsibility Principle). Якщо перевірка не проходить, хендлер записує помилку в контекст і не викликає наступний елемент.

```java
public abstract class AbstractBookingValidationHandler implements IBookingValidationHandler {

    protected IBookingValidationHandler nextHandler;

    @Override
    public void setNext(IBookingValidationHandler next) {
        this.nextHandler = next;
    }

    protected void handleNext(BookingValidationContext context) {
        if (!context.hasError() && this.nextHandler != null) {
            this.nextHandler.handle(context);
        }
    }

    @Override
    public abstract void handle(BookingValidationContext context);
}
```

Рисунок 2 - Механізм ланцюжка в AbstractBookingValidationHandler.java

```java
@Override
public void handle(BookingValidationContext context) {
    if (context.getMaster() == null || context.getService() == null) {
        context.setErrorMessage(errorMessage: "Помилка ланцюжка: Об'єкти Майстра або Послу
        return;
    }

    if (context.getService().getMaster().getMasterId().equals(context.getMaster().getMaste
        handleNext(context);
    } else {
        context.setErrorMessage(
            "Обраний майстер (" + context.getMaster().getUser().getName() +
            ") не надає послугу '" + context.getService().getName() + "'."
        );
    }
}
```

Рисунок 3 – MasterServiceCompatibilityHandler, ланка для перевірки бізнес-правил

```java
IBookingValidationHandler clientHandler = new ClientExistenceHandler(userRepository);
IBookingValidationHandler masterHandler = new MasterExistenceHandler(masterRepository);
IBookingValidationHandler serviceHandler = new ServiceExistenceHandler(serviceRepository);
IBookingValidationHandler compatibilityHandler = new MasterServiceCompatibilityHandler();

clientHandler.setNext(masterHandler);
masterHandler.setNext(serviceHandler);
serviceHandler.setNext(compatibilityHandler);
this.validationChain = clientHandler;
```

Рисунок 4 – Збірка ланцюжка в BookingService

```
[SUCCESS] Ланцюжок зупинено коректно!
Пов?домлення помилки: "Обраний майстер (Марія Ковач) не надає послугу 'Чоловіча стрижка'."
```

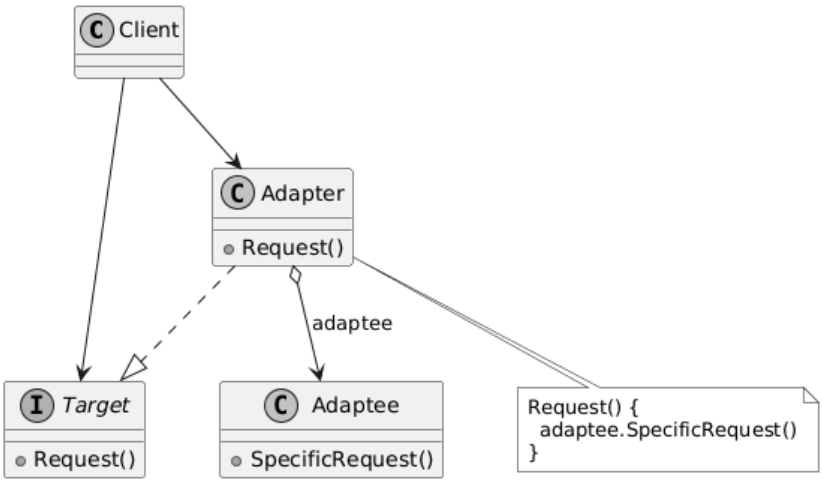
Рисунок 5 – Демонстрація результату

4. Висновок

У ході виконання лабораторної роботи було досліджено та реалізовано низку структурних і поведінкових патернів проєктування, серед яких Adapter, Builder, Chain of Responsibility, Command та Prototype. На прикладі системи бронювання послуг салону краси було практично імплементовано патерн Chain of Responsibility (Ланцюжок обов'язків) для організації послідовної валідації вхідних даних при створенні замовлення. Складна логіка перевірок — від існування клієнта, майстра та послуги до перевірки їх сумісності — була розділена на окремі класи-обробники, які формують гнучкий динамічний ланцюжок. Завдяки цьому вдалося привести метод createBooking до чистого та зрозумілого вигляду, уникнути важких вкладених умов та забезпечити строгий поділ відповідальностей відповідно до принципу SRP. Головною перевагою такого підходу стала можливість легко масштабувати систему: додавання нової бізнес-перевірки потребує лише створення нового обробника без втручання в уже реалізовану логіку.

5. Контрольні питання:

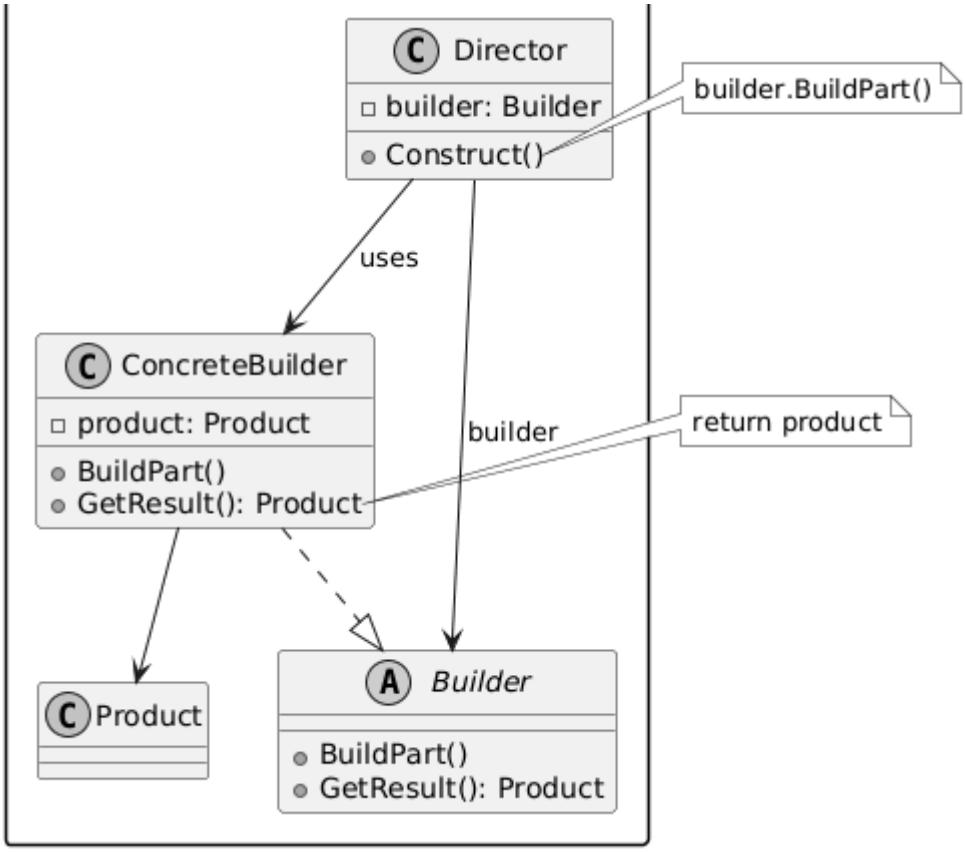
1) Яке призначення шаблону «Адаптер»? Адаптує інтерфейс одного об'єкта до іншого, дозволяючи несумісним компонентам працювати разом (наприклад, уніфікація бібліотек принтерів).
2) Нарисуйте структуру шаблону «Адаптер».



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

Client, Target, Adapter, Adaptee. Client працює з Target через адаптований інтерфейс, Adapter перенаправляє виклики до Adaptee.

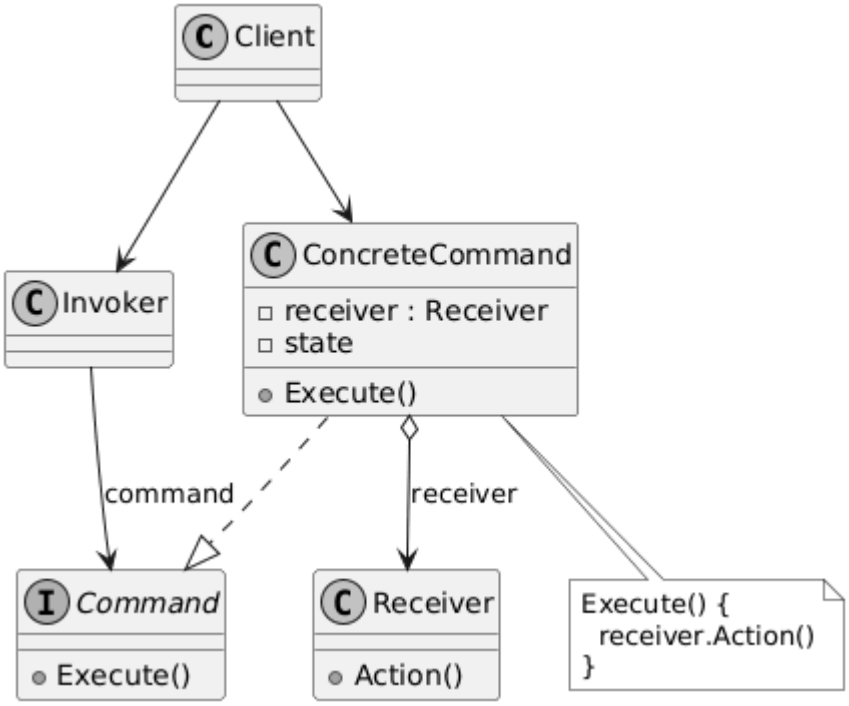
4) Яка різниця між реалізацією «Адаптера» на рівні об’єктів та на рівні класів? Об’єктний адаптер використовує композицію, класовим — успадкування.
5) Яке призначення шаблону «Будівельник»? Відокремлює процес створення об’єкта від його представлення, придатний для складних або багатоформних конструкцій.
6) Нарисуйте структуру шаблону «Будівельник».



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

Director, Builder, ConcreteBuilder, Product. Director керує будівництвом, Builder визначає інтерфейс, ConcreteBuilder реалізує його, Product — результат.

8) У яких випадках варто застосовувати шаблон «Будівельник»? При складному процесі створення або необхідності різних форм об’єкта.
9) Яке призначення шаблону «Команда»? Перетворює виклик методу в об’єкт для гнучкої системи команд із відміною, логуванням чи плануванням.
10) Нарисуйте структуру шаблону «Команда».



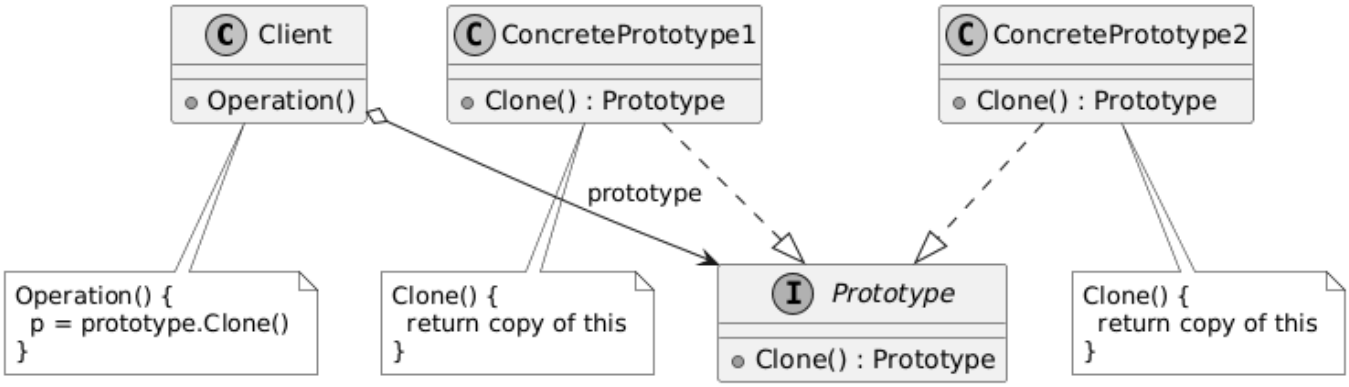
11) Які класи входять в шаблон «Команда», та яка між ними взаємодія?

Client, Invoker, Command, ConcreteCommand, Receiver. Client створює команду, Invoker виконує її, Receiver реалізує дію.

12) Розкажіть як працює шаблон «Команда». Команда інкапсулює запит як об’єкт, який передається Invoker до Receiver для виконання, підтримуючи додаткові функції (скасування, логування).

13) Яке призначення шаблону «Прототип»? Створює об’єкти клонуванням прототипу, зменшуючи ієрархію спадкування.

14) Нарисуйте структуру шаблону «Прототип».



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

Client, Prototype. Client клонувати об’єкт через метод Prototype.

16) Які можна привести приклади використання шаблону «Ланцюжок відповідальності»? Формування контекстного меню в UI, обробка запитів у ієрархії документів.