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 № 4

із дисципліни «Технології розроблення програмного забезпечення»

Тема: «Вступ до паттернів проектування»

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

Перевірив:
Мягкий М. Ю.

Київ 2025

Зміст

1. **Мета:**

Вивчити структуру шаблонів «Singleton», «Iterator», «Proxy», «State», «Strategy» та навчитися застосовувати їх в реалізації програмної системи.

2. **Теоретичні відомості:**

Singleton: Забезпечує єдиний екземпляр класу з глобальним доступом. Використовується для унікальних ресурсів (наприклад, конфігураційні файли), але вважається антипатерном через глобальний стан.
Iterator: Дозволяє послідовний доступ до елементів колекції без розкриття її внутрішньої структури. Підтримує різні методи обходу (наприклад, у глибину, випадково).
Proxy: Діє як замінник або заглушка для іншого об’єкта, додаючи функціонал (наприклад, ледаче завантаження, контроль доступу). Зменшує кількість запитів до зовнішніх сервісів (наприклад, оптимізація DocuSign).
State: Дозволяє змінювати поведінку об’єкта залежно від його стану (наприклад, типи карток або режими системи). Використовує окремі класи для кожного стану.
Strategy: Уможливлює заміну алгоритмів поведінки об’єкта (наприклад, методи сортування чи маршрути). Відокремлює логіку алгоритмів від контексту для гнучкості.

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.

Логіка процесу: Клієнт створює запит => об'єкт Booking отримує стан PendingState. Адміністратор підтверджує => перехід у ConfirmedState. Клієнт оплачує => перехід у PaidState. Майстер виконує послугу => перехід у CompletedState.

Без використання патерну ця логіка вимагала б громіздких конструкцій if-else або switch у кожному методі бізнес-логіки (наприклад, перевірка if (status == "PAID") перед завершенням). Патерн State дозволив інкапсулювати поведінку кожного статусу в окремий клас, роблячи код об'єкта Booking чистим, а переходи між станами — контрольованими та безпечними.

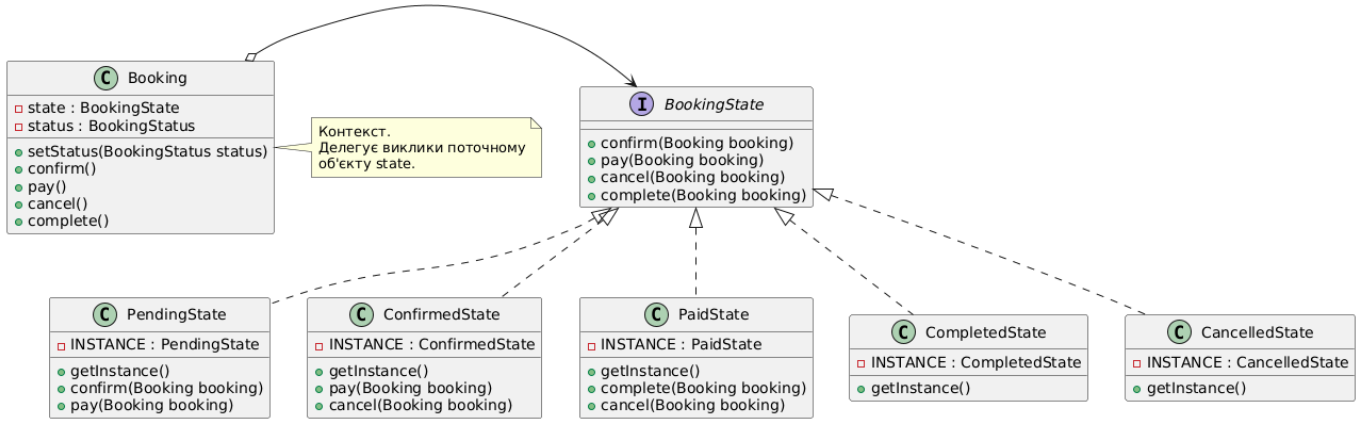


Рисунок 1 - Структура патерну State

Booking – це клас-контекст. Він зберігає посилання на поточний об'єкт стану (state) та делегує йому виконання операцій (підтвердження, оплата, скасування). Коли змінюється статус бронювання, контекст автоматично оновлює об'єкт стану.

BookingState – це загальний інтерфейс, який визначає контракт для всіх можливих станів. Він гарантує, що кожен стан знає, як реагувати на події confirm(), pay(), cancel() та complete().

ConcreteStates (PendingState, ConfirmedState, PaidState...) – це конкретні реалізації станів. Кожен клас описує унікальну поведінку для конкретного етапу життєвого циклу. Наприклад, PendingState дозволяє перехід у Confirmed, але забороняє перехід у Paid (викидає помилку).

Singleton у станах – кожен конкретний стан реалізовано як Singleton (через метод getInstance()), оскільки об'єкти станів не мають власних полів даних і можуть перевикористовуватися різними бронюваннями для економії пам'яті.

```java
public interface BookingState {

    void confirm(Booking booking);

    void pay(Booking booking);

    void cancel(Booking booking);

    void complete(Booking booking);
}
```

Рисунок 2 – Інтерфейс в BookingState.java

```java
private void initState() {
    if (status == null) {
        status = BookingStatus.PENDING;
    }

    this.state = switch (status) {
        case PENDING   -> PendingState.getInstance();
        case CONFIRMED -> ConfirmedState.getInstance();
        case PAID      -> PaidState.getInstance();
        case COMPLETED -> CompletedState.getInstance();
        case CANCELLED -> CancelledState.getInstance();
    };
}

public void confirm() {
    state.confirm(this);
}

public void pay() {
    state.pay(this);
}

public void cancel() {
    state.cancel(this);
}

public void complete() {
    state.complete(this);
}

public UUID getBookingId() {
    return bookingId;
}

public void setStatus(BookingStatus status) {
    this.status = status;
    initState();
}
```

Рисунок 3 – Booking.java

```java
public class PendingState implements BookingState {

    private static final PendingState INSTANCE = new PendingState();

    private PendingState() {}

    public static PendingState getInstance() {
        return INSTANCE;
    }

    @Override
    public void confirm(Booking booking) {
        booking.setStatus(BookingStatus.CONFIRMED);
    }
```

Рисунок 4 – конкретний стан PendingState.java

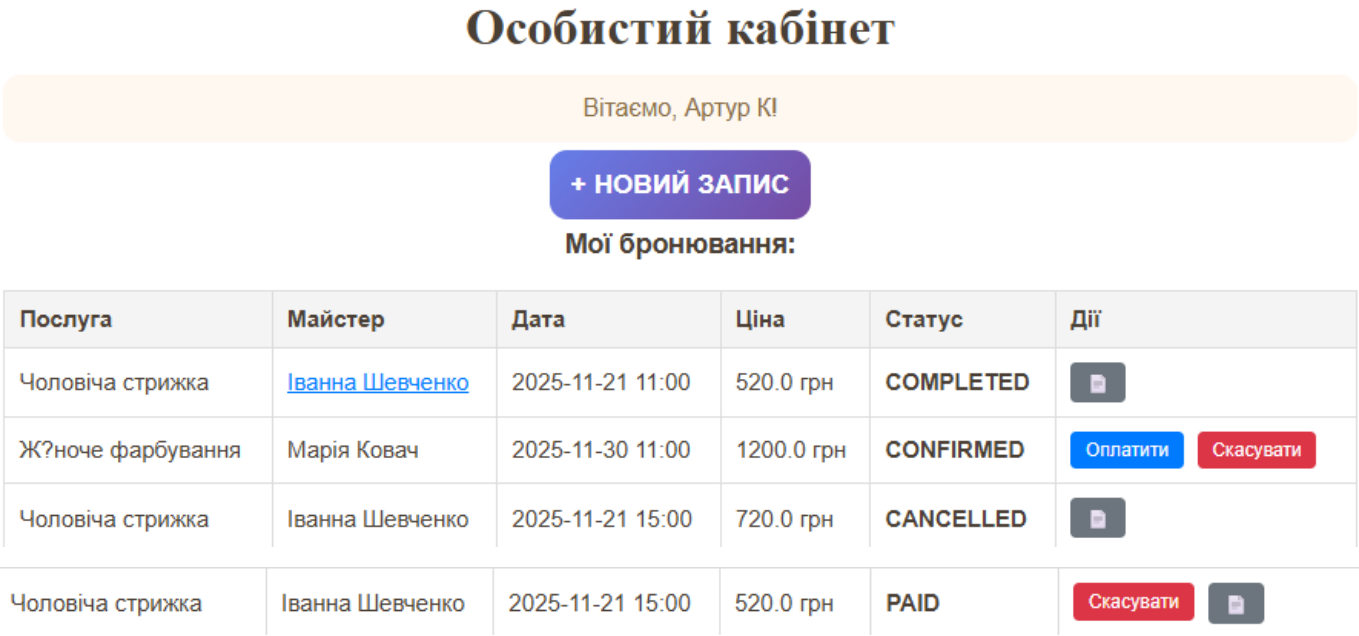


Рисунок 4 – Бронювання в різних станах

4. Висновок

У ході виконання лабораторної роботи було успішно застосовано поведінковий патерн проєктування State (Стан) для моделювання життєвого циклу бронювання в системі салону краси. На прикладі сутності Booking продемонстровано, як зміна внутрішнього стану (PENDING, CONFIRMED, PAID тощо) може впливати на доступну поведінку об’єкта. Логіка переходів і валідації дій була винесена у спеціалізовані класи-стани (PendingState, ConfirmedState), що дозволило позбутися надмірних умовних конструкцій та забезпечити строгий контроль бізнес-правил.

Такий підхід робить систему значно гнучкішою та масштабованішою: додавання нових статусів або зміна правил переходів не потребує модифікації існуючого коду, а лише створення нових станів. Отже, використання патерну State підтвердило свою ефективність для моделювання складних процесів і керування поведінкою об’єктів у веб-застосунках.

5. Контрольні питання:

1. Що таке шаблон проєктування?

Шаблон проєктування — це універсальне рішення для типових проблем у розробці ПЗ, яке описує структуру та взаємодію об’єктів.

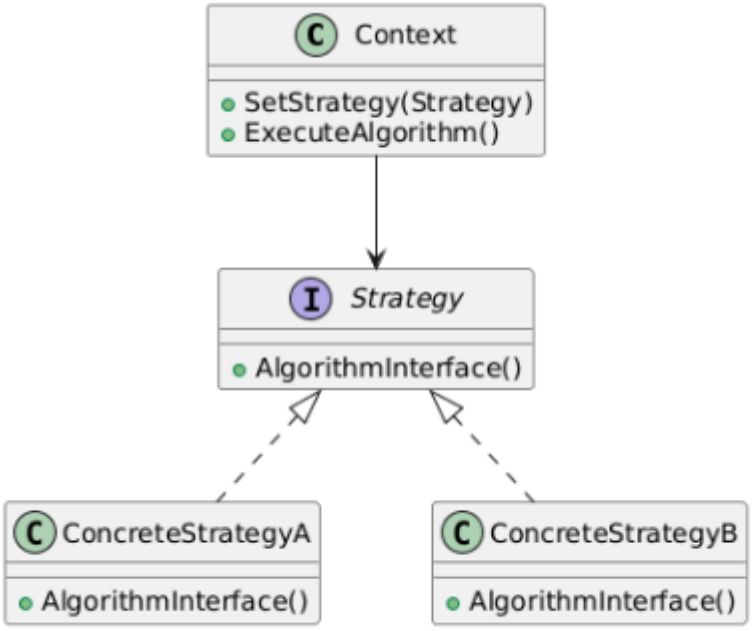
2. Навіщо використовувати шаблони проєктування?

- Спрощують розробку.
- Покращують читабельність і масштабування коду.
- Допомагають уникнути помилок.
- Забезпечують гнучкість і повторне використання.

3. Яке призначення шаблону «Стратегія»?

Шаблон Стратегія дозволяє динамічно вибирати алгоритми, інкапсулюючи їх у взаємозамінні класи, щоб уникнути умовних конструкцій.

6. Нарисуйте структуру шаблону «Стратегія».



5. Які класи входять в шаблон «Стратегія», та яка між ними взаємодія?

Класи:

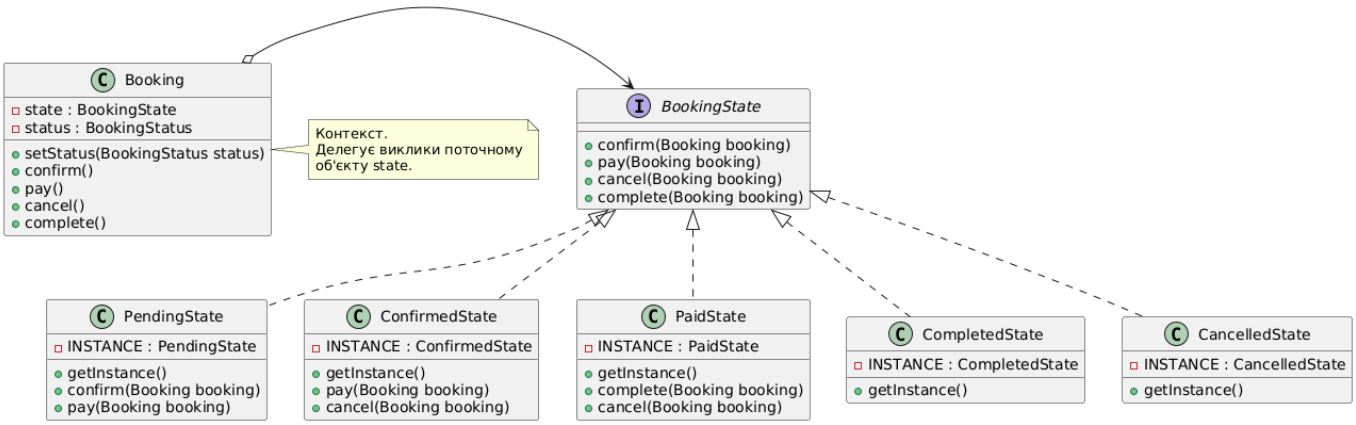
- Strategy: Інтерфейс із методом algorithm().
- ConcreteStrategy: Реалізації алгоритмів.
- Context: Використовує Strategy через setStrategy() і викликає algorithm().

Взаємодія: Контекст делегує виконання алгоритму об’єкту Strategy, який клієнт встановлює динамічно.

6. Яке призначення шаблону «Стан»?

Шаблон «Стан» дозволяє об’єкту змінювати поведінку залежно від стану, інкапсулюючи стани в окремі класи.

7. Нарисуйте структуру шаблону «Стан».



8. Які класи входять в шаблон «Стан», та яка між ними взаємодія?

Класи:

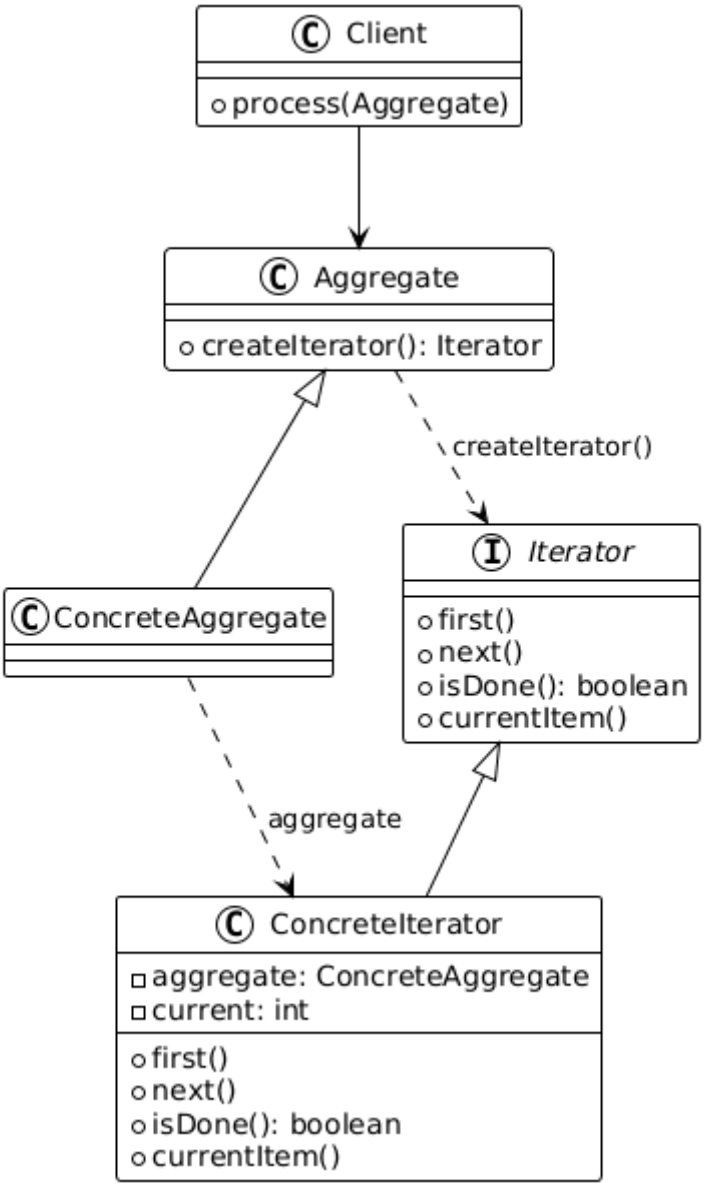
- State: Інтерфейс із методом handle().
- ConcreteState: Реалізації поведінки для стану.
- Context: Зберігає стан, делегує виконання handle().

Взаємодія: Контекст викликає handle() поточного стану, який може змінити стан через setState().

9. Яке призначення шаблону «Ітератор»?

Шаблон «Ітератор» забезпечує послідовний доступ до елементів колекції, приховуючи її внутрішню структуру.

10. Нарисуйте структуру шаблону «Ітератор».



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

Класи:

- Iterator: Інтерфейс із методами next(), hasNext().
- ConcreteIterator: Реалізація обходу.
- Aggregate: Інтерфейс із createIterator().
- ConcreteAggregate: Створює ConcreteIterator.

Взаємодія: Клієнт отримує ітератор через createIterator() і використовує його для обходу колекції.

12. В чому полягає ідея шаблона «Одинак»?

Шаблон «Одинак» забезпечує єдиний екземпляр класу з глобальним доступом до нього.

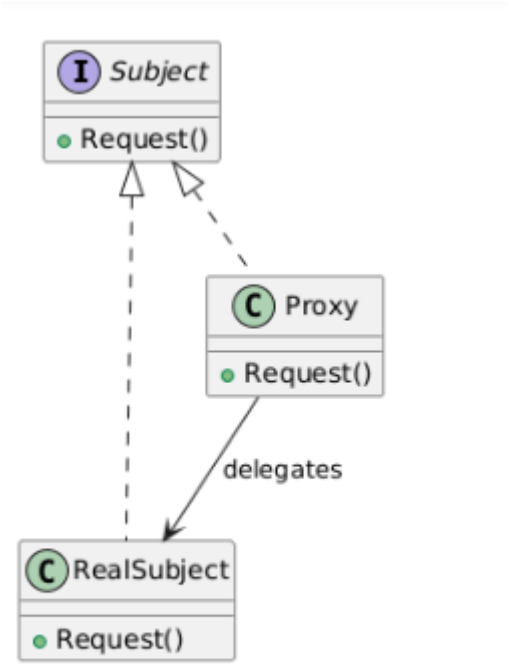
13. Чому шаблон «Одинак» вважають «анти-шаблоном»?

Бо він порушує принципи ООП: ускладнює тестування, створює приховані залежності та глобальний стан

14. Яке призначення шаблону «Проксі»?

Шаблон «Проксі» контролює доступ до об’єкта, додаючи функціонал, як-от ледарська ініціалізація чи перевірка прав.

15. Нарисуйте структуру шаблону «Проксі».



16. Які класи входять в шаблон «Проксі», та яка між ними взаємодія?

Класи:

- Subject: Інтерфейс із методом request().
- RealSubject: Виконує основну роботу.
- Proxy: Контролює доступ до RealSubject.

Взаємодія: Клієнт викликає request() через Proxy, який делегує виклик до RealSubject або додає логіку.