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 № 7

із дисципліни «Технології розроблення програмного забезпечення»

Тема: «Патерни проектування»

Виконав
Студент групи ІА-31: Перевірив:
Трегуб К. В. Мягкий М. Ю.

Київ 2025

Зміст

1. Мета:

Вивчити структуру шаблонів «Mediator», «Facade», «Bridge», «Template method» та навчитися застосовувати їх в реалізації програмної системи

2. Теоретичні відомості:

Mediator (Посередник) — централізує взаємодію між об’єктами, замінюючи прямі зв’язки зверненнями через єдиний медіатор. Добре підходить для інтерфейсів із великою кількістю залежних елементів. Переваги: менша зв’язаність коду, простіше розширення та тестування. Недолік: медіатор може перерости у надто «всезнаючий» об’єкт.

Facade (Фасад) — надає спрощений інтерфейс до складної підсистеми, приховуючи її внутрішню логіку. Застосовується для роботи з великими бібліотеками чи різними протоколами. Переваги: приховує складність, полегшує використання та оновлення системи. Недолік: зменшує гнучкість.

Bridge (Міст) — розділяє абстракцію та реалізацію, дозволяючи розвивати їх окремо. Використовується, коли потрібно підтримувати різні варіанти реалізацій (наприклад, форми в графічному редакторі та різні способи їх рендеру). Переваги: гнучкість та незалежність частин. Недолік: ускладнення структури.

Template Method (Шаблонний метод) — задає загальну структуру алгоритму у базовому класі, а конкретні кроки залишає для підкласів. Ефективний при роботі з різними форматами або варіантами обробки. Переваги: повторне використання коду, чітка структура. Недоліки: жорсткість алгоритму та складність при великій кількості кроків.

3. Хід роботи:

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.

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.

Процес оплати бронювання є складним і вимагає координації багатьох компонентів:

1. Знайти бронювання в БД (BookingRepository).
2. Перевірити валідність переходу статусу (використовуючи патерн State).
3. Вибрати платіжний метод та провести транзакцію (патерн Strategy).
4. Зберегти запис про платіж (PaymentRepository).
5. Сповістити клієнта про успішну оплату (патерн Observer).

Якби вся ця логіка знаходилася в BookingController, він став би перевантаженим ("Fat Controller") і жорстко пов'язаним з усіма цими сервісами.

Рішення: Введено клас PaymentFacade, який надає клієнтському коду (контролеру) простий інтерфейс з одним методом payForBooking(). Фасад бере на себе відповідальність за ініціалізацію та координацію роботи всіх компонентів підсистеми, ізолюючи контролер від складності бізнес-процесу.

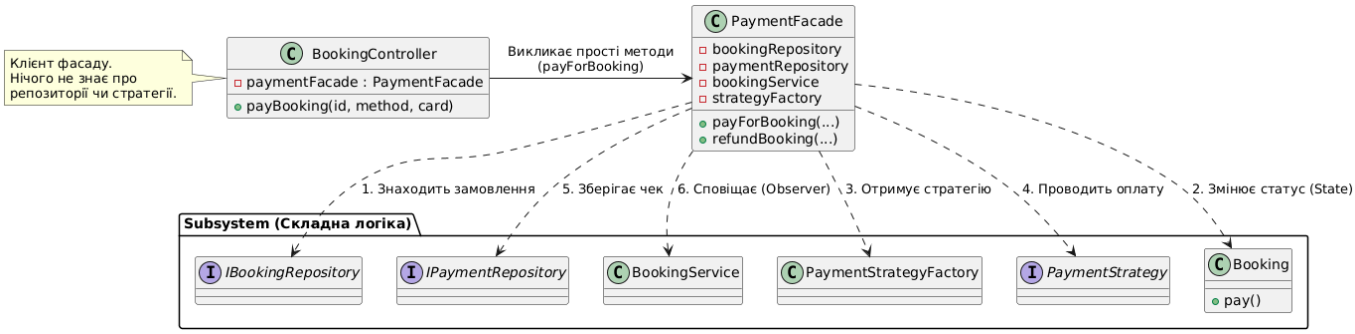


Рисунок 1 - Структура патерну Facade

- BookingController — клієнт фасаду. Він знає лише про існування `PaymentFacade` і викликає його метод для проведення оплати, не вникаючи в деталі транзакції.

- PaymentFacade — основний клас патерну. Він агрегує посилання на всі необхідні сервіси та репозиторії (`Repositories`, `Service`, `Factory`). Його методи (`payForBooking`, `refundBooking`) реалізують сценарії взаємодії цих компонентів у правильному порядку.
- Subsystem (Підсистема) — набір класів, які виконують реальну роботу. Це робота з базою даних, логіка станів (`Booking.pay()`), вибір алгоритму оплати (`Strategy`) та розсилка повідомлень (`Observer`). Фасад приховує їх від зовнішнього світу, зменшуючи зв'язність системи (coupling)

```
@Service
public class PaymentFacade {
    private final IBookingRepository bookingRepository;
    private final IPaymentRepository paymentRepository;
    private final BookingService bookingService;
    private final PaymentStrategyFactory strategyFactory;
```

```
@Transactional
public Booking payForBooking(UUID bookingId, String paymentMethod, String cardNumber) {
    Booking booking = bookingRepository.findById(bookingId)
            .orElseThrow(() -> new RuntimeException(message: "Бронювання не знайдено."));

    booking.pay();

    PaymentStrategy strategy = strategyFactory.getStrategy(paymentMethod);

    if (!strategy.processPayment(booking.getTotalPrice(), cardNumber)) {
        throw new RuntimeException("Зовнішній платіж " + strategy.getId() + " не вдалося викон
    }

    Payment payment = new Payment();
    payment.setBooking(booking);
    payment.setAmount(booking.getTotalPrice());
    payment.setPaymentMethod(strategy.getId());
    payment.setPaymentStatus(paymentStatus: "PAID");
    payment.setPaymentDate(LocalDateTime.now());
    payment.setCardNumber(cardNumber);
    paymentRepository.save(payment);

    Booking savedBooking = bookingRepository.save(booking);
    bookingService.notifyPaymentObservers(savedBooking);
    return savedBooking;
}
```

Рисунок 2 - PaymentFacade.java з методом @Transactional, що приховує всю складність

```java
@Controller
@RequestMapping("/web/bookings")
public class BookingWebController {

    private final BookingService bookingService;
    private final PaymentFacade paymentFacade;
    private final IServiceRepository serviceRepository;
    private final IMasterRepository masterRepository;

@PostMapping("/{id}/pay")
public String payBooking(
        @PathVariable UUID id,
        @RequestParam(defaultValue = "CARD") String paymentMethod,
        @RequestParam String cardNumber,
        HttpServletRequest request) {
    paymentFacade.payForBooking(id, paymentMethod, cardNumber);
    String referer = request.getHeader(name: "Referer");
    return "redirect:" + (referer != null ? referer : "/auth/home");
}
```

Рисунок 3 – Клієнтський код BookingWebController.java

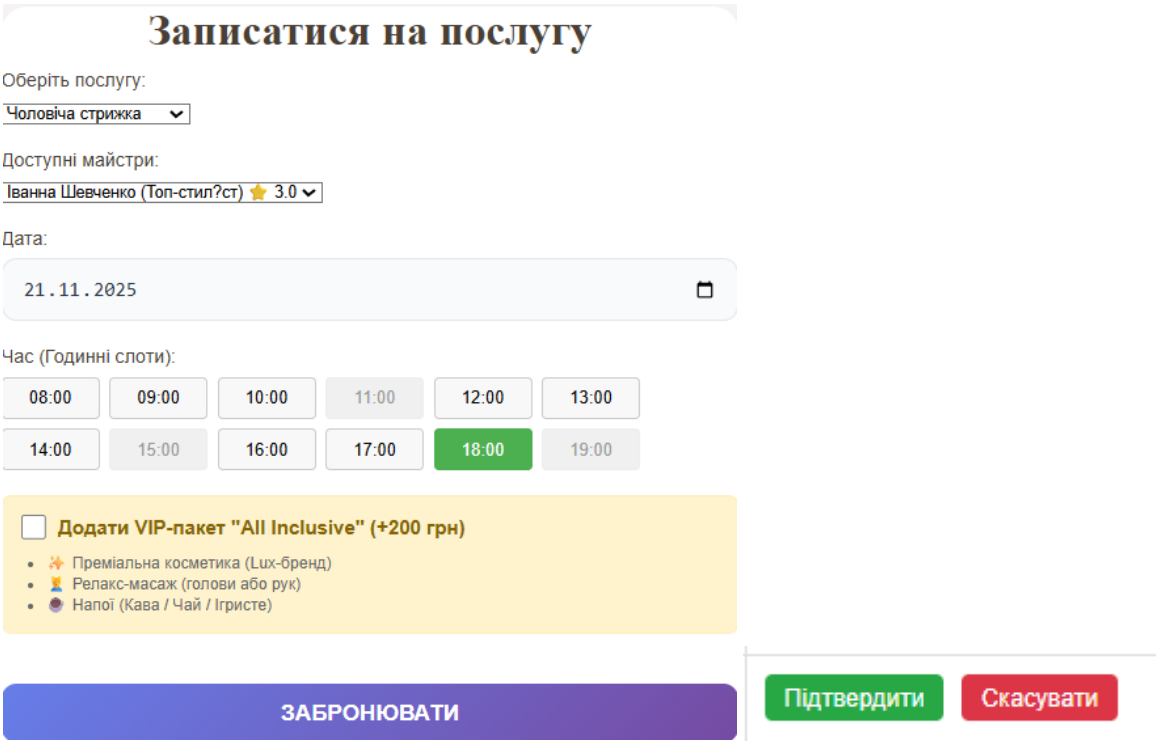


Рисунок 4 – Створюємо та підтверджуємо бронювання

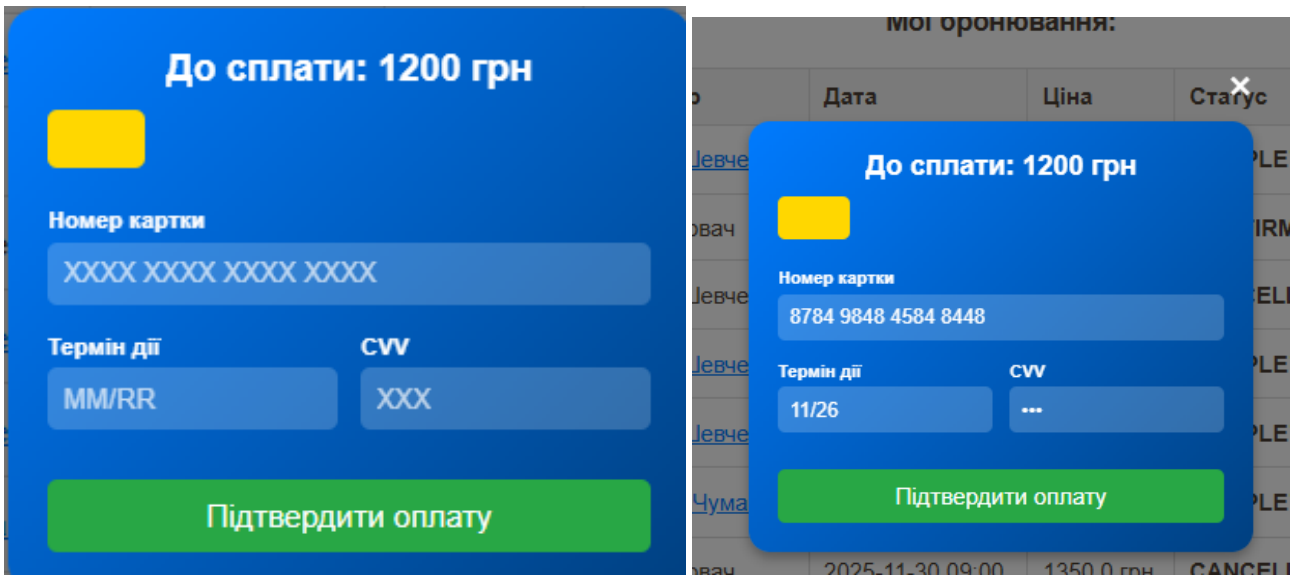


Рисунок 5 – Виклик (booking.pay())

ID	Дата/Час	Клієнт	Майстер	Послуга	Ціна	Статус	Дата оплати
912fad77...	2025-12-10 14:00	Іванна Шевченко	Іванна Шевченко (Топ-стил?ст)	Чоловіча стрижка	520.0 грн	PENDING	-
258d7aaf...	2025-11-30 11:00	Артур К	Марія Ковач (Перукар-колорист)	Ж?ноче фарбування	1200.0 грн	PAID	21-11 20:53

Рисунок 6 – Створення запису Payment в БД, отже з’являється дата оплати

```
--- [EmailObserver] ---
Надсилаємо email кл?єнту: client@beauty.com
Тема: Статус вашого бронювання зм?нено
Новий статус: PAID
```

Рисунок 7 – Виклик патерну Observer

4. Висновок

У ході виконання лабораторної роботи було опрацьовано структурні шаблони проєктування, зокрема Mediator, Facade, Bridge та Template Method. На практиці, для підсистеми оплати веб-застосунку, було реалізовано шаблон Facade**.** Створений клас PaymentFacade інкапсулював усю складну внутрішню логіку взаємодії між репозиторіями бази даних, системою станів замовлення, платіжними стратегіями та модулем сповіщень.

Застосування цього підходу дало змогу істотно спростити взаємодію з підсистемою оплати: контролери отримали єдиний, зручний метод для проведення транзакції без необхідності знати її внутрішні кроки. Окрім цього, було досягнуто зниження зв’язності — зміни в алгоритмі обробки платежів тепер

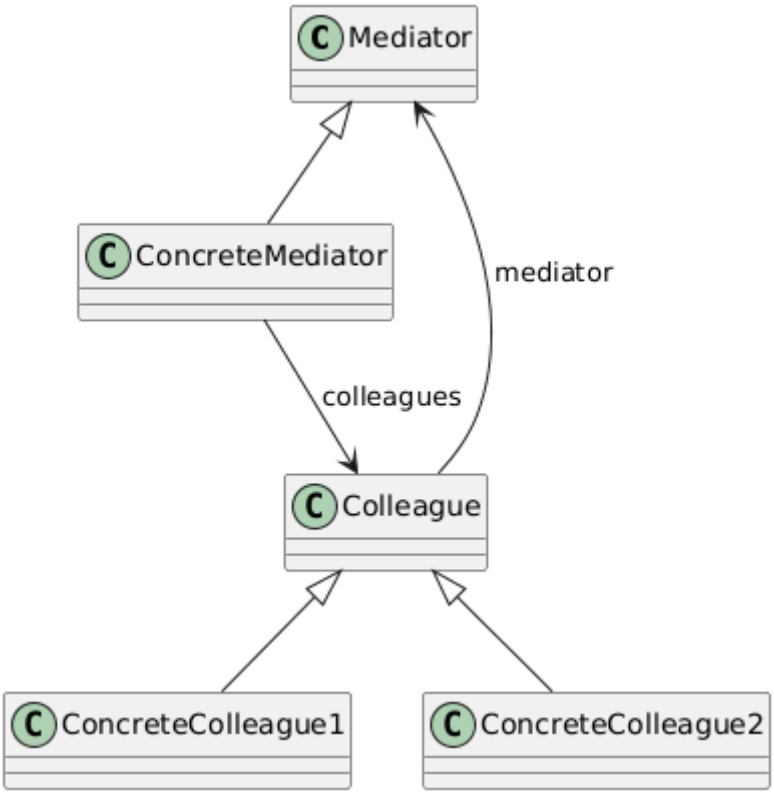
локалізуються лише у фасаді — та централізації бізнес-логіки, що значно полегшує супровід і тестування модулю.

5. Контрольні питання:

1. Яке призначення шаблону «Посередник»?

Централізує взаємодію між об’єктами через єдиний об’єкт-посередник, усуваючи прямі зв’язки між ними. Зменшує зв’язність, спрощує логіку взаємодії.

2. Нарисуйте структуру шаблону «Посередник».



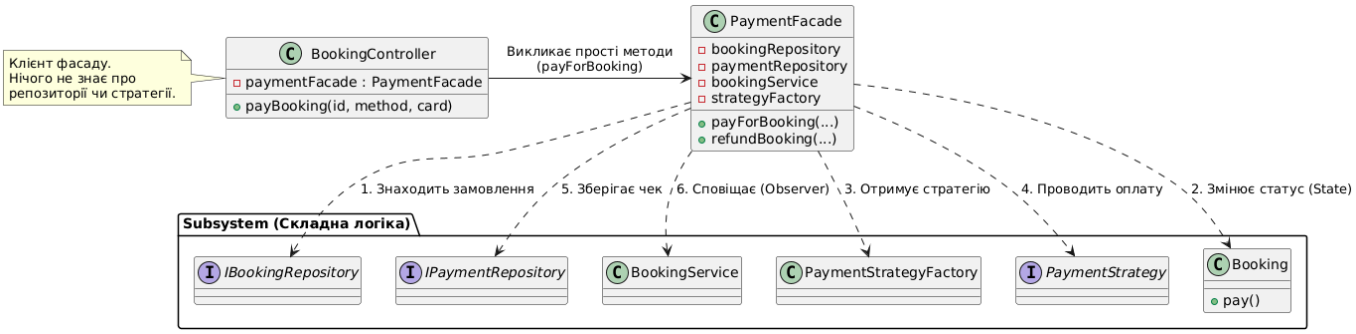
3. Які класи входять в шаблон «Посередник», та яка між ними взаємодія?

- Mediator — інтерфейс посередника
- ConcreteMediator — реалізація, координує взаємодію
- Colleague — базовий клас компонентів
- ConcreteColleague1, ConcreteColleague2 — конкретні компоненти

4. Яке призначення шаблону «Фасад»?

Надає спрощений уніфікований інтерфейс до складної підсистеми, приховуючи її внутрішню складність.

5. Нарисуйте структуру шаблону «Фасад».



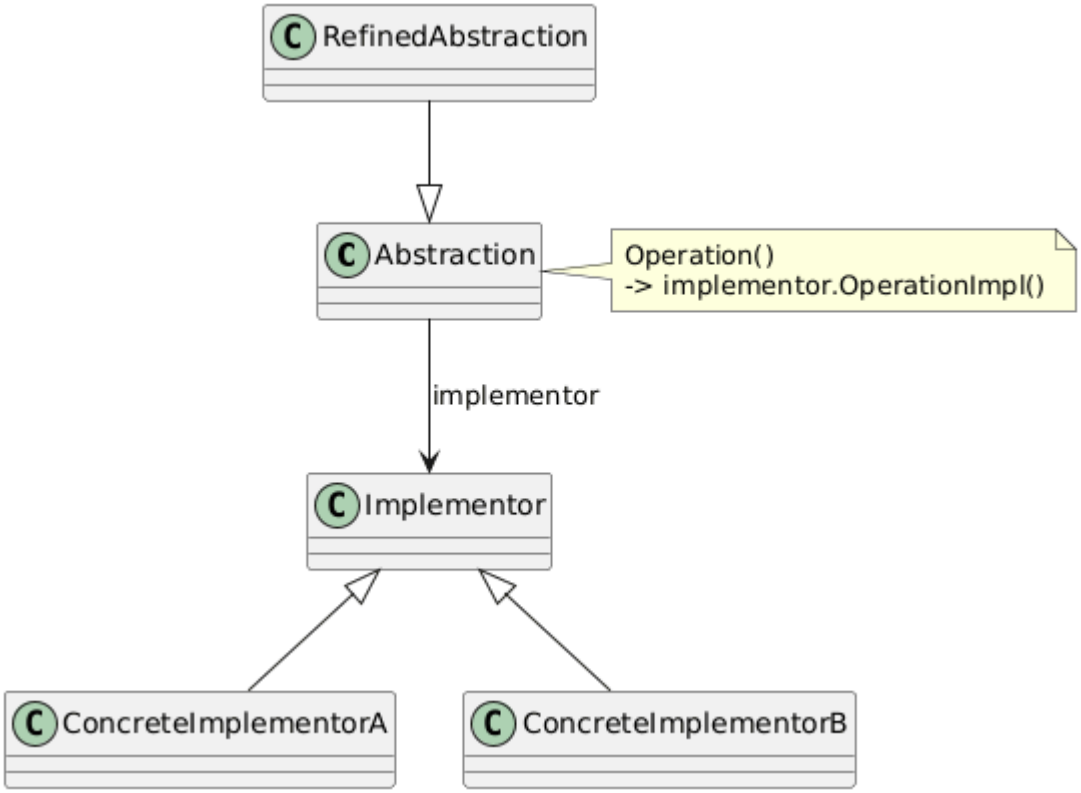
6. Які класи входять в шаблон «Фасад», та яка між ними взаємодія?

- Facade — єдиний інтерфейс
- SubsystemA, SubsystemB, SubsystemC — класи підсистеми

7. Яке призначення шаблону «Міст»?

Розділяє абстракцію та її реалізацію, дозволяючи змінювати їх незалежно (дві ієрархії).

8. Нарисуйте структуру шаблону «Міст».



9. Які класи входять в шаблон «Міст», та яка між ними взаємодія?

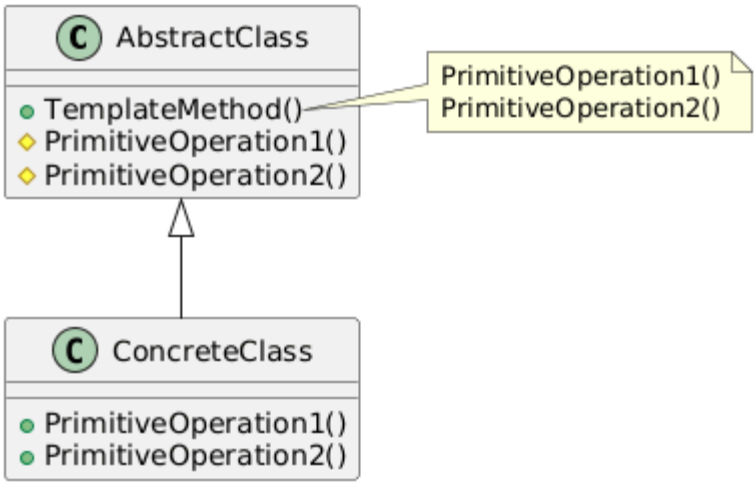
- Abstraction — абстракція
- RefinedAbstraction — розширена абстракція

- Implementor — інтерфейс реалізації
- ConcreteImplementorA/B — конкретні реалізації

10. Яке призначення шаблону «Шаблонний метод»?

Визначає скелет алгоритму в базовому класі, залишаючи окремі кроки для перевизначення в похідних.

11. Нарисуйте структуру шаблону «Шаблонний метод».



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

- AbstractClass — містить TemplateMethod() і абстрактні методи
- ConcreteClass — реалізує абстрактні методи

13. Чим відрізняється шаблон «Шаблонний метод» від «Фабричного методу»?

Шаблонний метод» визначає скелет алгоритму, дозволяючи підкласам перевизначати окремі кроки. Фабричний метод» визначає інтерфейс створення об’єкта, залишаючи підкласам вибір конкретного класу.

14. Яку функціональність додає шаблон «Міст»?

Дозволяє незалежно розвивати абстракцію та реалізацію.

- Додає гнучкість
- Усуває експоненційне зростання підкласів
- Підтримує принцип розділення відповідальностей