



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: .....	3
2. Теоретичні відомості:.....	3
3. Хід роботи:.....	3
4. Висновок .....	11
5. Контрольні питання: .....	11

## 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 заміник 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:** Уможливорює 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, strategy, singleton, proxy, builder, client-server)  
Додаток повинен служити сервером для системи бронювання в салоні краси з можливістю створення, підтвердження, оплати та завершення записів у потоковому режимі; вести облік статусів бронювання (Pending → Confirmed → Paid → Completed); дозволяти клієнтам обирати майстра, послугу та час; підтримувати гнучкі знижки (стратегії оплати) та безпечний доступ до даних через проксі; забезпечувати єдину точку доступу до конфігурації (singleton) та зручне створення складних об'єктів бронювання (builder).

- 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.

Патерн State використано, бо статус бронювання (PENDING, PAID, COMPLETED) повністю змінює поведінку об'єкта Booking. Без State — це були б if/else у кожному методі, дублювання коду та складне тестування. State виносить логіку кожного стану в окремий клас, робить код чистим, розширюваним і відповідальним за один стан. Ініціалізація через ApplicationContext + @PostLoad забезпечує синхронізацію з БД. Результат — гнучка, підтримувальна система.

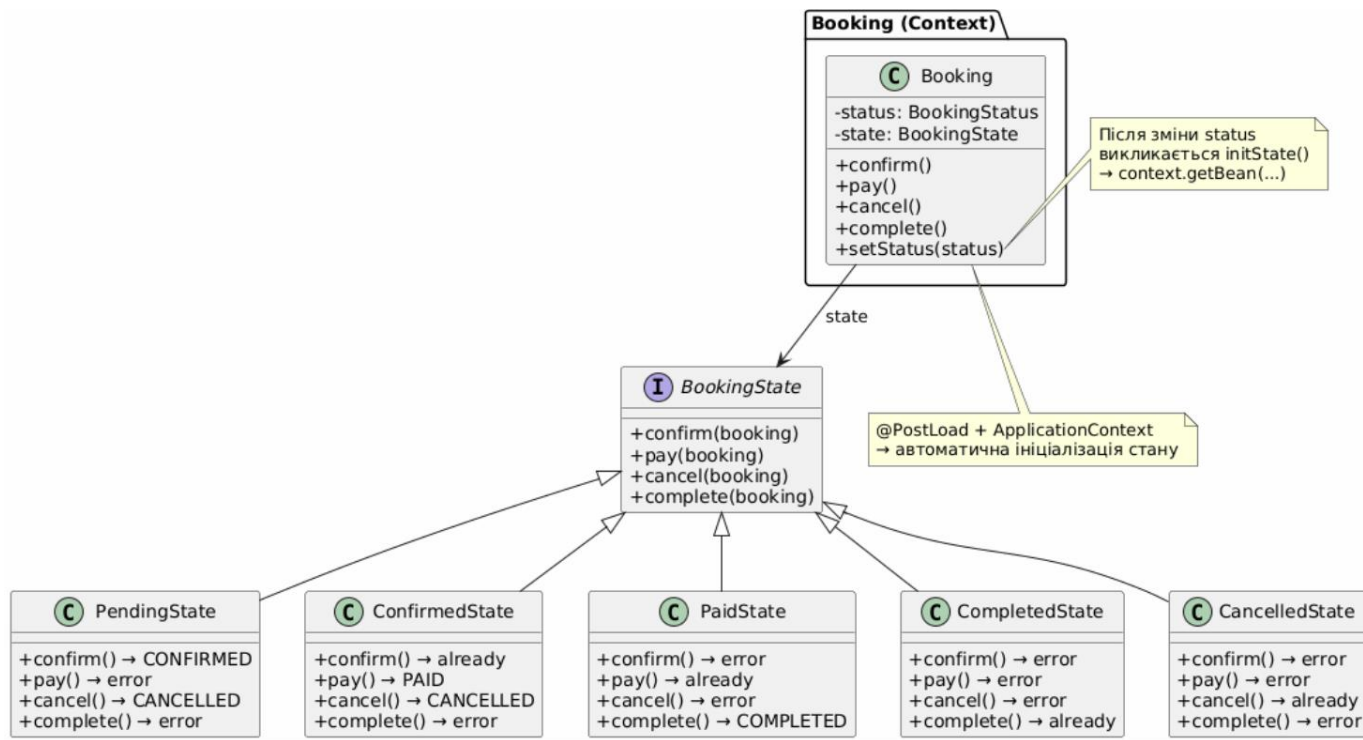


Рисунок 1 - Структура патерну State

Booking — це контекст, який зберігає поточний статус бронювання (status) та посилання на об'єкт стану (state). Він делегує всі дії (confirm(), pay() тощо) поточному стану.

BookingState — це інтерфейс, який описує методи переходів між станами. Він задає контракт для всіх станів.

PendingState, ConfirmedState, PaidState, CompletedState, CancelledState — це конкретні стани. Кожен знає, які дії дозволені, а які — ні (кидають IllegalStateException).

При зміні status (наприклад, setStatus(CONFIRMED)) викликається initState(), який через ApplicationContext отримує відповідний бін стану (context.getBean(ConfirmedState.class)).

В результаті логіка переходів винесена з if/else у Booking у окремі класи станів. Це і є суть State — об'єкт змінює поведінку залежно від внутрішнього стану, а код залишається чистим і розширюваним.

```

private void initState() {
    if (status == null) status = BookingStatus.PENDING;
    if (context == null) return;

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

public void confirm() { if (state != null) state.confirm(this); }
public void pay() { if (state != null) state.pay(this); }
public void cancel() { if (state != null) state.cancel(this); }
public void complete() { if (state != null) state.complete(this); }

public Booking() {
    this.status = BookingStatus.PENDING;
}

@Autowired
public void setApplicationContext(ApplicationContext context) {
    this.context = context;
    initState();
}

private double totalPrice;

@OneToOne(mappedBy = "booking", cascade = CascadeType.ALL, orphanRemoval = true)
private Payment payment;

@PostLoad
private void postLoad() {
    initState();
}

```

Рисунок 2 – Ключові елементи Booking.java

```

package com.beautysalon.booking.state;

import com.beautysalon.booking.entity.Booking;
import com.beautysalon.booking.entity.BookingStatus;
import org.springframework.stereotype.Service;

@Service
public class PaidState implements BookingState {

    @Override
    public void confirm(Booking booking) {
        throw new IllegalStateException(s: "Вже підтверджено");
    }

    @Override
    public void pay(Booking booking) {
        throw new IllegalStateException(s: "Вже оплачено");
    }

    @Override
    public void cancel(Booking booking) {
        booking.setStatus(BookingStatus.CANCELLED);
        booking.setState(new CancelledState());
    }

    @Override
    public void complete(Booking booking) {
        booking.setStatus(BookingStatus.COMPLETED);
        booking.setState(new CompletedState());
    }
}

```

Рисунок 3 – PaidState.java

```

package com.beautysalon.booking.state;

import com.beautysalon.booking.entity.Booking;
import com.beautysalon.booking.entity.BookingStatus;
import org.springframework.stereotype.Service;

@Service
public class ConfirmedState implements BookingState {
    @Override

    public void pay(Booking booking) {
        booking.setStatus(BookingStatus.PAID);
        booking.setState(new PaidState());
    }

    @Override
    public void cancel(Booking booking) {
        booking.setStatus(BookingStatus.CANCELLED);
        booking.setState(new CancelledState());
    }

    @Override
    public void confirm(Booking booking) { /* вже підтверджено */ }
    @Override
    public void complete(Booking booking) { /* не оплачено */ }
}

```

Рисунок 4 – ConfirmedState.java

```

package com.beautysalon.booking.state;

import com.beautysalon.booking.entity.Booking;
import com.beautysalon.booking.entity.BookingStatus;
import org.springframework.stereotype.Service;

@Service
public class PendingState implements BookingState {

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

    @Override
    public void pay(Booking booking) {
        throw new IllegalStateException(s: "Неможливо оплатити без підтвердження");
    }

    @Override
    public void cancel(Booking booking) {
        booking.setStatus(BookingStatus.CANCELLED);
        booking.setState(new CancelledState());
    }

    @Override
    public void complete(Booking booking) {
        throw new IllegalStateException(s: "Неможливо завершити без оплати");
    }
}

```

Рисунок 5 – PendingState.java

```
package com.beautysalon.booking.state;

import com.beautysalon.booking.entity.Booking;
import org.springframework.stereotype.Service;

@Service
public class CancelledState implements BookingState {
    @Override
    public void confirm(Booking booking) {
        throw new IllegalStateException(s: "Бронювання скасовано");
    }

    @Override
    public void pay(Booking booking) {
        throw new IllegalStateException(s: "Бронювання скасовано");
    }

    @Override
    public void cancel(Booking booking) {}

    @Override
    public void complete(Booking booking) {
        throw new IllegalStateException(s: "Бронювання скасовано");
    }
}
```

Рисунок 6 – CancelledState.java

```

package com.beautysalon.booking.state;

import com.beautysalon.booking.entity.Booking;
import org.springframework.stereotype.Service;
@Service
public class CompletedState implements BookingState {
    @Override
    public void complete(Booking booking) { /* вже завершено */ }
    @Override
    public void confirm(Booking booking) { throw new IllegalStateException(s: "Завершено"); }
    @Override
    public void pay(Booking booking) { throw new IllegalStateException(s: "Завершено"); }
    @Override
    public void cancel(Booking booking) { throw new IllegalStateException(s: "Завершено"); }
}

```

Рисунок 7 – CompletedState.java

```

package com.beautysalon.booking;
import com.beautysalon.booking.entity.Booking;
import com.beautysalon.booking.entity.BookingStatus;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import static org.junit.jupiter.api.Assertions.*;

@SpringBootTest
class BookingApplicationTests {

    @Autowired
    private Booking booking;

    @Test
    void testStatePattern() {
        assertEquals(BookingStatus.PENDING, booking.getStatus());
        System.out.println("1. Початок: " + booking.getStatus());

        booking.confirm();
        assertEquals(BookingStatus.CONFIRMED, booking.getStatus());
        System.out.println("2. Після confirm(): " + booking.getStatus());

        booking.pay();
        assertEquals(BookingStatus.PAID, booking.getStatus());
        System.out.println("3. Після pay(): " + booking.getStatus());

        booking.complete();
        assertEquals(BookingStatus.COMPLETED, booking.getStatus());
        System.out.println("4. Після complete(): " + booking.getStatus());

        IllegalStateException exception = assertThrows(
            IllegalStateException.class,
            booking::cancel
        );
        System.out.println("5. Спроба cancel(): " + exception.getMessage());
    }
}

```

Рисунок 8 – Тест BookingApplicationTests.java

1. Початок: PENDING
2. Після confirm(): CONFIRMED
3. Після pay(): PAID
4. Після complete(): COMPLETED
5. Спроба cancel(): Завершено

Рисунок 9 – Результат виконання

#### 4. Висновок

У ході виконання лабораторної роботи було розглянуто базові шаблони проєктування, зокрема State, Strategy, Singleton, Proxy та Builder. На прикладі веб-застосунку «Beauty Salon Booking System» реалізовано шаблон State для управління життєвим циклом бронювання. Логіка переходів між станами (Pending → Confirmed → Paid → Completed) винесена в окремі класи (PendingState, PaidState тощо), що дозволило усунути if/else у класі Booking, зробити код чистим і розширюваним. Ініціалізація стану через ApplicationContext та @PostLoad забезпечує синхронізацію з базою даних. Це підтверджує ефективність патернів проєктування як інструменту для створення гнучкої, підтримувальної та масштабованої архітектури, готової до додавання нових станів і бізнес-правил.

#### 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.

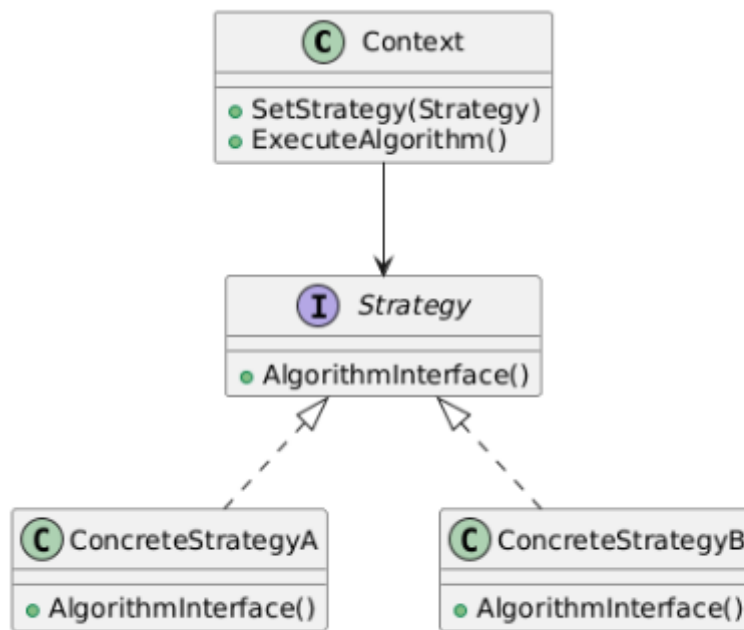
2. Навіщо використовувати шаблони проєктування?

- Спрощують розробку.
- Покращують читабельність і масштабування коду.
- Допмагають 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:

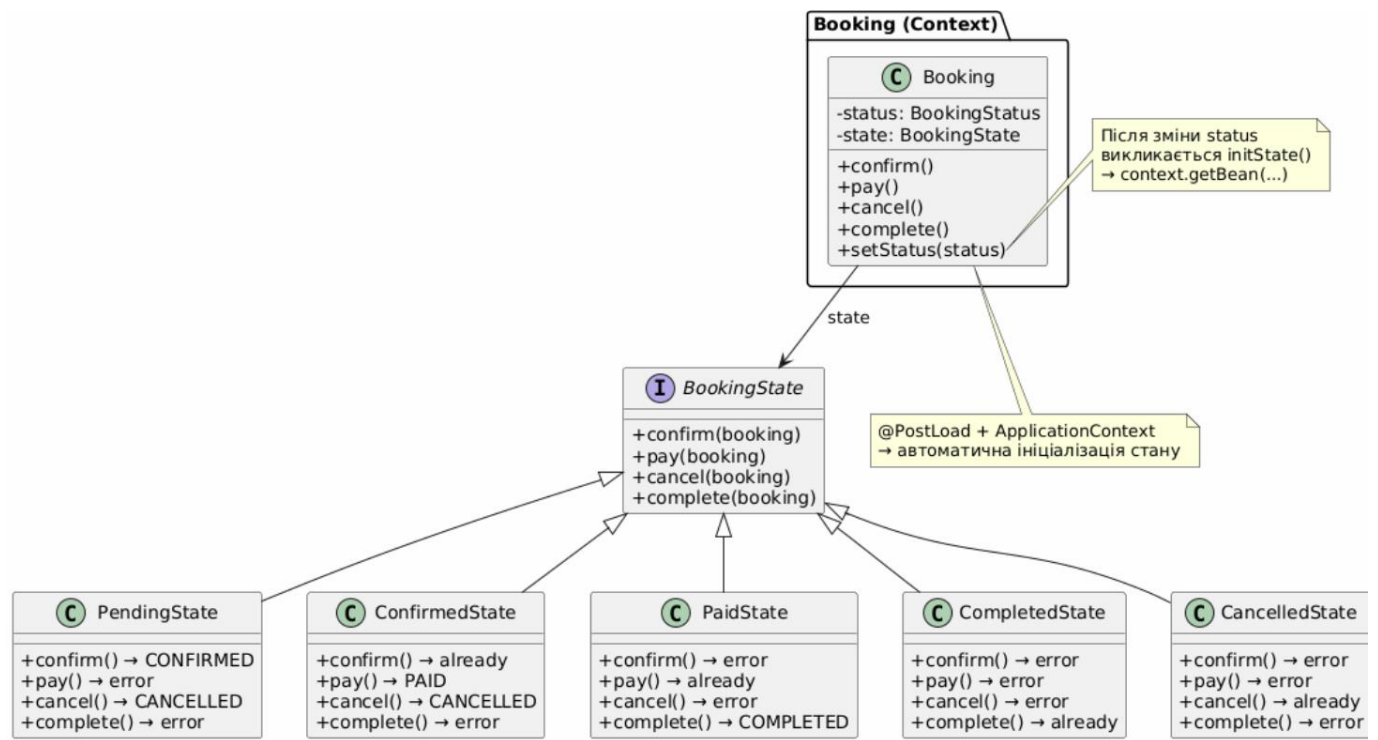
- **Strategy**: Інтерфейс із методом `algorithm()`.
- **ConcreteStrategy**: Реалізації алгоритмів.
- **Context**: Використовує **Strategy** через `setStrategy()` і викликає `algorithm()`.

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:

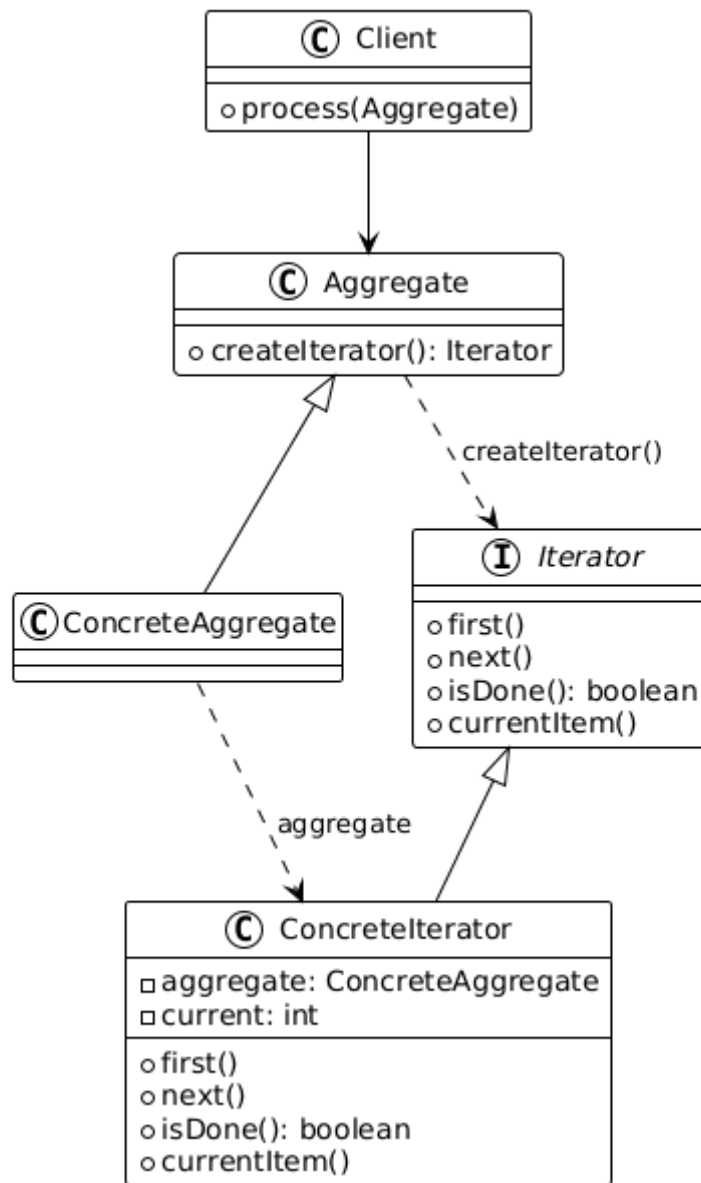
- State: Інтерфейс із методом handle().
- ConcreteState: Реалізації поведінки для стану.
- Context: Зберігає стан, делегує виконання handle().

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 шаблону «Одинак»?

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

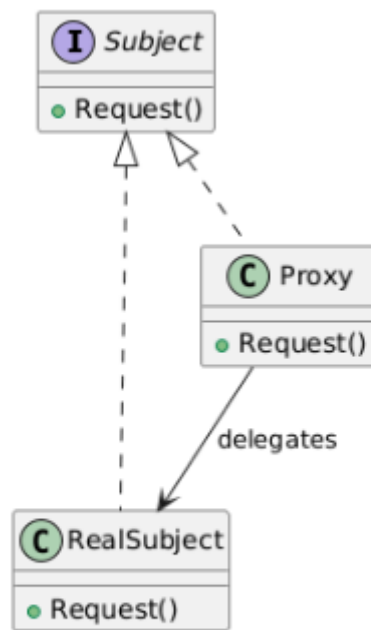
13. Чому шаблон «Одинак» вважають «анти-шаблоном»?

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.