# Лабораторна робота №4

з навчальної дисципліни: “Проектування програмних систем”

на тему:

“Музей”

Виконав:Лупу Р.Р.  
Студент:301 групи

# 2025

У розглянутій діаграмі класів для предметної області «Музей» були використані наступні архітектурні патерни проєктування:

### 1. **Dependency Inversion**

У системі управління музеєм залежність від абстракції реалізована через інтерфейс ITicket, який визначає спільні методи для всіх видів квитків. Класи StandardTicket, DiscountedTicket, GroupTicket реалізують цей інтерфейс. Завдяки цьому модулі вищого рівня (наприклад, система обліку продажів квитків) не залежать безпосередньо від конкретних реалізацій квитків, а лише від абстракції. Це дозволяє легко додавати нові типи квитків у майбутньому.

### 2. **Стратегія**

Для реалізації системи ціноутворення була застосована стратегія. Різні типи квитків (дитячий, дорослий, пільговий) використовують різні стратегії розрахунку ціни, але вся логіка винесена в окремі класи, які реалізують інтерфейс IPricingStrategy. Це дозволяє динамічно змінювати стратегію під час виконання програми, наприклад, при впровадженні акцій або знижок.

### 3. **Фабрика**

У системі створення квитків використано шаблон Фабрики. TicketFactory відповідає за створення об'єктів класів, які реалізують інтерфейс ITicket. Це дозволяє зручно централізовано створювати об'єкти необхідного типу без жорсткої прив’язки до конкретних реалізацій, спрощуючи підтримку коду.

### 4. **Спостерігач**

Патерн Спостерігач застосовано для реалізації системи сповіщень. Коли змінюється інформація про виставку (наприклад, змінюється дата чи експонати), усі підписані користувачі отримують повідомлення про зміни. Такий підхід дозволяє ефективно підтримувати актуальність даних та зручний для реалізації як у вебінтерфейсі, так і в мобільному додатку.

### 5. **Замісник**

З метою оптимізації продуктивності при відображенні зображень експонатів у системі використовується замісник. Клас ExhibitImageProxy завантажує зображення лише тоді, коли це дійсно потрібно. Це зменшує споживання пам’яті та прискорює час відгуку системи при перегляді списку експонатів.

### 6. **Будівельник**

Створення повноцінного об'єкта виставки (Exhibition) може потребувати поступового додавання різних елементів: експонати, дати, розташування, куратори тощо. Патерн Будівельника дозволяє створювати складні об'єкти поетапно, забезпечуючи при цьому зручність і гнучкість у конфігурації.

### 7. **Команда**

Патерн Команди використано для реалізації історії змін у системі керування виставками. Наприклад, дії на додавання, зміну або видалення виставки інкапсулюються як окремі об’єкти команд. Це дозволяє легко реалізувати скасування дій (undo/redo), а також логувати зміни.

### 8. **Декоратор**

Класи експонатів можуть бути динамічно «обгорнуті» додатковим функціоналом через Декоратор. Наприклад, експонат може бути позначений як “тимчасово недоступний” або “експонат з аудіогідом”. Це реалізовано без зміни базового класу Exhibit, що дозволяє гнучко розширювати функціонал.

### 9. **Одинак**

Для підключення до бази даних використано патерн Одинак. Клас DatabaseConnection гарантує наявність лише одного екземпляра з’єднання з базою протягом усього часу роботи програми. Це забезпечує ефективне використання ресурсів та контроль над підключенням.

### 10. **Посередник**

Усередині системи багато компонентів взаємодіють між собою: модулі обліку квитків, управління виставками, сповіщення користувачів. Щоб уникнути надмірної кількості прямих зв’язків між класами, застосовано патерн Посередник, який інкапсулює логіку взаємодії та дозволяє зменшити зв’язаність компонентів.

**Посилання на репозиторій та скріншот виконаної роботи**

[**https://github.com/R0manD/Software-System-Design**](https://github.com/R0manD/Software-System-Design)

