# Лабораторна робота №4

з навчальної дисципліни: “Проектування програмних систем”

на тему:

“Ветеринарна клініка”

# 2025

У ветеринарній клініці система керування включає кілька ключових сутностей: користувачів (адміністраторів, лікарів, власників тварин), записи на прийоми, медичні записи, інвентар, платежі та звіти. Кожен користувач виконує специфічні дії: власники тварин бронюють прийоми через bookAppointment(), лікарі оновлюють медичні записи за допомогою updateMedicalRecord(), а адміністратори керують інвентарем і генерують звіти через manageInventory() і generateReports(). Для всіх користувачів визначено базовий абстрактний клас User, від якого походять Administrator, Doctor і PetOwner. Щоб забезпечити гнучкість у взаємодії, застосовується патерн **Інверсія залежностей (Dependency Inversion)**. Наприклад, метод makePayment() класу PetOwner залежить від абстрактного інтерфейсу IPayment, а не від конкретної реалізації Payment. Це дозволяє перемикатися між методами оплати, такими як Google Pay чи PayPal, без зміни коду PetOwner).

Оплата в системі може здійснюватися різними способами, залежно від вибору користувача, наприклад, через Google Pay або PayPal. Для цього використовується патерн **Стратегія (Strategy)**, який забезпечує динамічний вибір стратегії оплати. Клас Payment з методами processPayment() і validatePayment() реалізує інтерфейс IPayment, дозволяючи методу makePayment() класу PetOwner використовувати потрібну стратегію, наприклад, GooglePayPayment, що забезпечує гнучкість обробки платежів.

Інвентар клініки, керований класом Inventory через методи addItem(), checkStock() і updateStock(), містить медикаменти, які потрібно ефективно відстежувати. Для абстрагування доступу до інвентарних даних застосовується патерн **Репозиторій (Repository)**. Наприклад, InventoryRepository може надавати методи, такі як getByItemId(), для доступу до даних про медикаменти, приховуючи деталі їх зберігання. Це спрощує взаємодію з інвентарем у методі manageInventory() класу Administrator.

Коли лікар оновлює медичний запис через updateMedicalRecord(), додаючи діагноз за допомогою addDiagnosis(), власник тварини має бути сповіщений, щоб переглянути оновлення через viewMedicalHistory(). Аналогічно, бронювання прийому через bookAppointment() класом PetOwner сповіщає лікаря через viewAppointments(). Для таких подій застосовується патерн **Спостерігач (Observer)**. MedicalRecord виступає суб’єктом, сповіщаючи PetOwner, а Appointment сповіщає Doctor, забезпечуючи оновлення в реальному часі.

Медичні записи, що включають діагнози, лікування та вакцинації, зберігаються як послідовність подій у масиві medicalHistory класу Pet. Для цього використовується патерн **Подієве джерело (Event Sourcing)**. Замість зберігання поточного стану MedicalRecord, система записує події, такі як DiagnosisAdded чи TreatmentPrescribed, а метод viewHistory() відтворює запис шляхом повторного відтворення подій, що дозволяє відстежувати історію змін.

Прийоми та медичні записи створюються з різними атрибутами, наприклад, purpose для Appointment чи diagnosis для MedicalRecord. Для спрощення створення об’єктів застосовується патерн **Фабрика (Factory)**. Наприклад, AppointmentFactory створює прийоми для різних цілей (огляд, хірургія) для методу bookAppointment() класу PetOwner. Аналогічно, MedicalRecordFactory генерує записи для різних типів візитів (вакцинація) для createMedicalRecord() класу Doctor.

Автентифікація через клас Authorization з методами processAuthorizationRequest() і validateToken() залежить від зовнішніх служб, які можуть бути ненадійними. Для захисту системи від збоїв застосовується патерн **Автомат відключення (Circuit Breaker)**. Наприклад, AuthorizationCircuitBreaker може зупинити processAuthorizationRequest(), якщо служба автентифікації неодноразово дає збій, повертаючи помилку або кешовані дані, що забезпечує стабільність.

Дії з прийомами, такі як бронювання чи скасування через book() і cancel() класу Appointment, можна інкапсулювати за допомогою патерну **Команда (Command)**. Наприклад, BookAppointmentCommand або CancelAppointmentCommand дозволяють Administrator через manageAppointment() логувати або скасовувати операції, що полегшує відстеження дій.

Медикаменти в інвентарі, керовані класом Inventory, можна повторно використовувати для оптимізації ресурсів. Для цього застосовується патерн **Пул об’єктів (Object Pool)**. InventoryPool керує пулом медикаментів, таких як вакцини, дозволяючи методу addVaccination() класу MedicalRecord повторно використовувати об’єкти без зайвих викликів addItem().

Звіти, що генеруються класом Report через generate(), filterByDate() і filterByType(), потребують складної конфігурації (startDate, endDate, type). Для цього використовується патерн **Будівельник (Builder)**. ReportBuilder поступово налаштовує тип звіту (наприклад, фінансовий) і діапазон дат перед викликом generate(), що спрощує generateReports() класу Administrator.

Для забезпечення єдиного екземпляра служби автентифікації клас Authorization з методами processAuthorizationRequest() і validateToken() використовує патерн **Одинак (Singleton)**. Це гарантує одне з’єднання з зовнішньою службою автентифікації, запобігаючи множинним підключенням, що підтримується потребою в validateToken().

Взаємодії між Administrator (manageAppointment()), Doctor (updateMedicalRecord()) і PetOwner (bookAppointment()) потребують координації. Для цього застосовується патерн **Посередник (Mediator)**. VetClinicMediator централізує комунікацію, наприклад, обробляючи bookAppointment() і сповіщаючи Doctor через viewAppointments(), зменшуючи залежності.

Фільтрація звітів і прийомів через filterByDate() класу Report або перевірка status класу Appointment вимагає бізнес-правил. Для цього використовується патерн **Шаблон специфікації (Specification)**. ReportSpecification визначає фільтри (діапазон дат) для filterByDate(), а AppointmentSpecification перевіряє статус “підтверджено” для viewAppointmentStatus() класу PetOwner.

Інтеграція автентифікації з зовнішніми службами, такими як OAuth, через processAuthorizationRequest() класу Authorization підтримується патерном **Адаптер (Adapter)**. OAuthAdapter адаптує зовнішній API до processAuthorizationRequest(), дозволяючи User використовувати login() із сторонньою автентифікацією.

Оновлення медичних записів через addDiagnosis(), addTreatment() і addVaccination() класу MedicalRecord передбачає послідовну обробку. Для цього застосовується патерн **Конвеєр обробки (Pipeline)**. MedicalRecordPipeline об’єднує addDiagnosis() і addTreatment() для updateMedicalRecord() класу Doctor, забезпечуючи консистентність даних.