ЛАБОРАТОРНА РОБОТА № 5

Платформа трекінгу машин із США

Мета: набуття практичних навичок щодо побудови та використання діаграм класів з урахуванням використання патернів проектування.

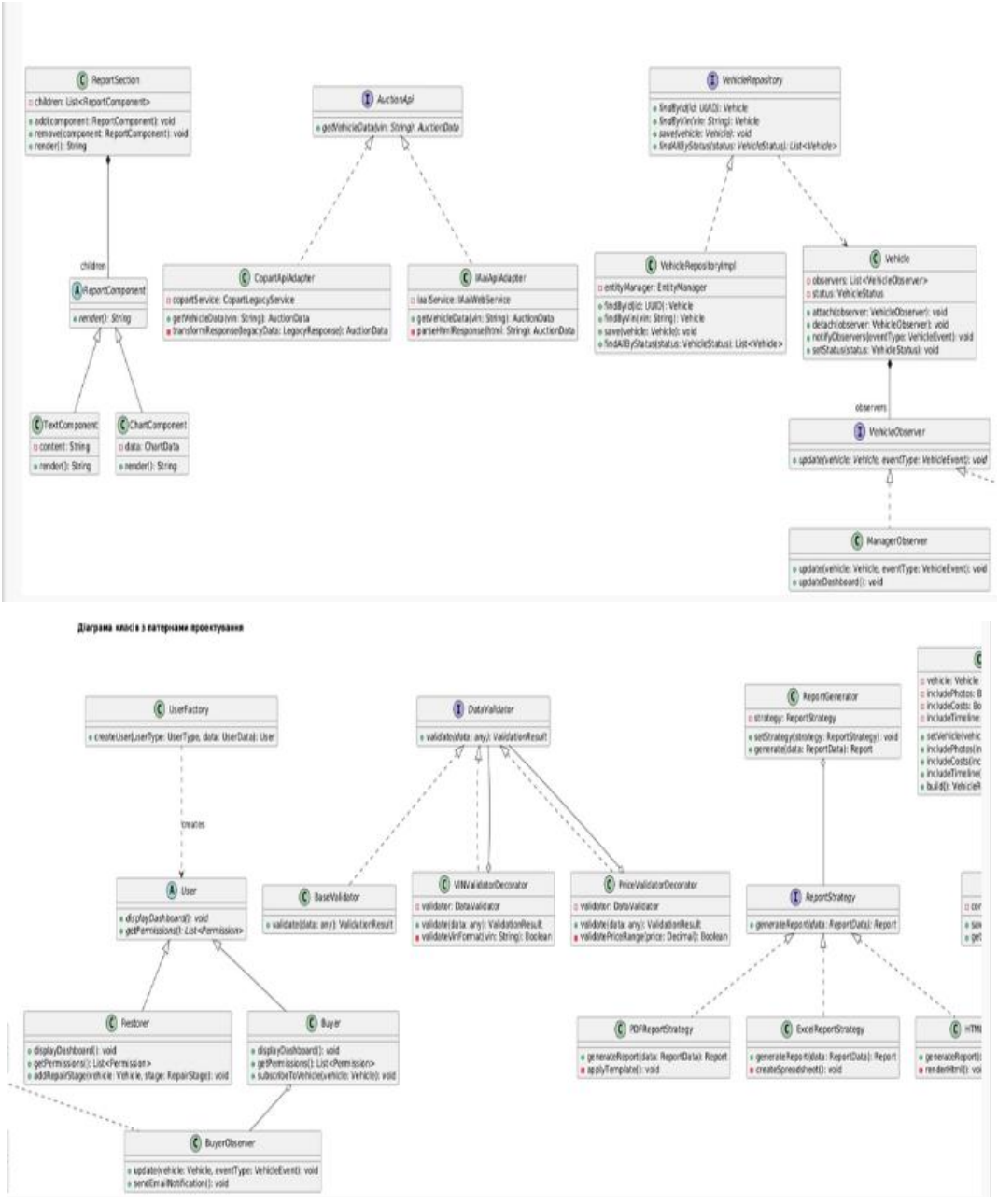

Рис. 1-2. Діаграма класів з урахуванням патернів проектування

					ДУ «Житомирська політехніка».25.121.0.000 – Лр5		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Віцінський Є.В.			Звіт з лабораторної роботи	Літ.	Арк. 1 / Аркушів 4
Перевір.		Левківський В.Л.					
Керівник						ФІКТ Гр. ІПЗ-22-2	
Н. контр.							
Зав. каф.							

Factory Pattern використовується для створення об'єктів користувачів різних типів. Клас UserFactory інкапсулює логіку створення конкретних екземплярів користувачів (Buyer, Restorer, Manager тощо) на основі переданого типу. Це дозволяє відокремити процес створення об'єктів від основної логіки програми, спрощує додавання нових типів користувачів і забезпечує єдину точку для модифікації процесу створення. Абстрактний клас User визначає спільний інтерфейс для всіх похідних класів, гарантуючи консистентність поведінки.

Observer Pattern реалізований для системи сповіщень про зміни статусу транспортних засобів. Клас Vehicle виступає як Subject, який підтримує список зареєстрованих спостерігачів (VehicleObserver). Кожен раз, коли статус автомобіля змінюється, Vehicle автоматично повідомляє всіх підписаних спостерігачів, викликаючи їх метод update. BuyerObserver та ManagerObserver реалізують інтерфейс VehicleObserver і реагують на події відповідно до своєї бізнес-логіки. Цей патерн забезпечує слабку зв'язаність між об'єктами і дозволяє динамічно додавати або видаляти спостерігачів без змін в класі Vehicle.

Strategy Pattern застосований для генерації звітів у різних форматах. Інтерфейс ReportStrategy визначає спільний контракт для всіх стратегій генерації звітів. Конкретні реалізації (PDFReportStrategy, ExcelReportStrategy, HTMLReportStrategy) інкапсулюють алгоритми створення звітів у відповідних форматах. Клас ReportGenerator містить посилання на поточну стратегію і делегує їй генерацію звіту. Це дозволяє легко додавати нові формати звітів і динамічно змінювати стратегію під час виконання програми.

Decorator Pattern використовується для валідації даних з можливістю динамічного додавання нових перевірок. Базовий клас BaseValidator надає основну функціональність валідації, а декоратори (VINValidatorDecorator, PriceValidatorDecorator) розширюють його, додаючи специфічні перевірки. Кожен декоратор містить посилання на об'єкт валідатора і делегує йому базову валідацію, після чого виконує свою додаткову логіку. Такий підхід дозволяє комбінувати різні типи валідації без створення монолітних класів з усіма можливими перевірками.

Repository Pattern забезпечує абстракцію доступу до даних для сутності Vehicle. Інтерфейс VehicleRepository визначає контракт для всіх операцій зберігання та отримання даних, а VehicleRepositoryImpl надає конкретну реалізацію з використанням EntityManager. Цей патерн відокремлює бізнес-логіку від деталей зберігання даних, дозволяє легко міняти реалізацію персистентності (наприклад, перехід з SQL на NoSQL) і спрощує тестування за рахунок можливості використання мок-репозиторіїв.

Builder Pattern застосований для поступового створення складних об'єктів VehicleReport. Клас VehicleReportBuilder надає методи для налаштування різних аспектів звіту (включення фото, витрат, таймлайну) і повертає сам себе після кожного виклику, що дозволяє ланцюжкові виклики. Фінальний метод build() створює об'єкт VehicleReport з усіма заданими параметрами. Цей підхід спрощує створення об'єктів з багатьма опціональними параметрами і покращує читабельність коду.

Adapter Pattern використовується для інтеграції з різними API аукціонів. Інтерфейс AuctionApi визначає стандартний спосіб отримання даних про транспортні засоби, а адаптери (CopartApiAdapter, IAaiApiAdapter) трансформують специфічні формати даних від кожного аукціону у єдиний формат програми. Кожен адаптер інкапсулює знання про конкретний API і приховує його особливості від основної системи. Це дозволяє легко додавати підтримку нових аукціонів без змін в існуючому коді.

Singleton Pattern забезпечує глобальний доступ до AnalyticsService з гарантією існування лише одного екземпляра цього класу. Приватний конструктор забороняє створення екземплярів ззовні, а статичний метод getInstance() повертає єдиний екземпляр сервісу. Це корисно для сервісів, які керують спільними ресурсами або мають високі накладні витрати на створення. Патерн також забезпечує контрольовану точку доступу до глобального стану.

Composite Pattern використовується для побудови ієрархічних структур у звітах. Абстрактний клас ReportComponent визначає інтерфейс для всіх компонентів, а конкретні реалізації (TextComponent, ChartComponent) представляють лис-

тові елементи. Клас ReportSection може містити як прості компоненти, так і інші секції, утворюючи дерево. Метод render() викликається рекурсивно для всієї структури, що дозволяє обробляти складні ієрархії з єдиною точкою входу. Цей патерн спрощує роботу зі складними структурами даних і дозволяє однаково обробляти як окремі елементи, так і їх композиції.

Висновок: я набув практичних навичок щодо побудови та використання діаграм класів з урахуванням використання патернів проектування.